关于本文档

文档名称：《企业级WEB应用服务器TOMCAT》

使用协议：《知识共享公共许可协议(CCPL)》

贡献者

贡献者名称	贡献度	文档变更记录	个人主页
马哥（马永亮）	主编		http://github.com/iKubernetes/
王晓春	创始作者	248页	http://www.wangxiaochun.com

文档协议

致谢

本文档中，部分素材参考了相关项目的文档，以及通过搜索引擎获得的内容，这里先一并向相关的贡献者表示感谢。

企业级WEB应用服务器TOMCAT

内容概述

- **WEB 技术**
- **JAVA 基础**
- **TOMCAT 基础功能**
- **结合反向代理部署 TOMCAT 服务**
- **TOMCAT Session 复制集群**
- **Memcached**
- **MSM**
- **TOMCAT 性能优化**
- **Java 程序编译**

1 WEB技术

1.1 HTTP协议和B/S 结构

操作系统有进程子系统，使用多进程就可以充分利用硬件资源。进程中可以多个线程，每一个线程可以被CPU调度执行，这样就可以让程序并行的执行。这样一台主机就可以作为一个服务器为多个客户端提供计算服务。

客户端和服务端往往处在不同的物理主机上，它们分属不同的进程，这些进程间需要通信。跨主机的进程间通信需要使用网络编程。最常见的网络编程接口是Socket。

Socket称为套接字，本意是插座。也就是说网络通讯需要两端，如果一端被动的接收另一端请求并提供计算和数据的称为**服务器端**，另一端往往只是发起计算或数据请求，称为**客户端。**

这种编程模式称为Client/Server编程模式，简称**C/S编程。**开发的程序也称为C/S程序。C/S编程往往使用传输层协议（TCP/UDP），较为底层，比如：QQ，迅雷, 云音乐, 云盘, foxmail，xshell等

1990年HTTP协议和浏览器诞生。在应用层使用文本跨网络在不同进程间传输数据，最后在浏览器中将服务器端返回的HTML渲染出来。由此，诞生了网页开发。

网页是存储在WEB服务器端的文本文件，浏览器发起HTTP请求后，到达WEB服务程序后，服务程序根据URL读取对应的HTML文件，并封装成HTTP响应报文返回给浏览器端。

起初网页开发主要指的是HTML、CSS等文件制作，目的就是显示文字或图片，通过超级链接跳转到另一个HTML并显示其内容。

后来，网景公司意识到让网页动起来很重要，傍着SUN的Java的名气发布了JavaScript语言，可以在浏览器中使用JS引擎执行的脚本语言，可以让网页元素动态变化，网页动起来了。

为了让网页动起来，微软使用ActiveX技术、SUN的Applet都可以在浏览器中执行代码，但都有安全性问题。能不能直接把内容直接在WEB服务器端组织成HTML，然后把HTML返回给浏览器渲染呢?

最早出现了CGI（Common Gateway Interface）通用网关接口，通过浏览器中输入URL直接映射到一个服务器端的脚本程序执行，这个脚本可以查询数据库并返回结果给浏览器端。这种将用户请求使用程序动态生成的技术，称为动态网页技术。先后出现了ASP、PHP、JSP等技术，这些技术的使用不同语言编写的程序都运行在服务器端，所以称为**WEB后端编程。**有一部分程序员还是要编写HTML、CSS、

JavaScript，这些代码运行在浏览器端，称为**WEB前端编程**。合起来称为Browser/Server编程，即**B/S编程**。

1.2 前端三大核心技术

1.2.1 HTML

HTML（HyperText Markup Language）超文本标记语言，它不同于一般的编程语言。超文本即超出纯文本的范畴，例如：描述文本颜色、大小、字体等信息，或使用图片、音频、视频等非文本内容。

HTML由一个个的标签（标记）组成，这些标签各司其职，有的提供网页信息，有的负责文字，有的负责图片，有的负责网页布局，所以一个HTML文件，是由格式标签和数据组成。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
</head>
<body>
<h1>马哥教育欢迎您</h1>
</body>
</html>
```

超文本需要显示，就得有软件能够呈现超文本定义的排版格式，例如显示：图片、表格，显示字体的大小、颜色，这个软件就是浏览器。

超文本的诞生是为了解决纯文本不能格式显示的问题，是为了好看，但是只有通过网络才能分享超文本的内容，所以制定了HTTP协议。

1.2.2 CSS（Cascading Style Sheets）层叠样式表

HTML本身为了格式化显示文本，但是当网页呈现大家面前的时候，需求HTML提供更多样式能力。这使得HTML变得越来越臃肿。这促使了CSS的诞生。

1994年，W3C成立，CSS设计小组所有成员加入W3C，并努力研发CSS的标准，微软最终加入。

1996年12月发布CSS 1.0。

1998年5月发布CSS 2.0。

CSS 3采用了模块化思想，每个模块都在CSS 2基础上分别增强功能。所以，这些模块是陆续发布的。

不同厂家的浏览器使用的引擎，对CSS的支持不一样，导致网页布局、样式在不同浏览器不一样。因此，想要保证不同用户使用不同浏览器看到的网页效果一直非常困难。

1.2.3 JavaScript

Javascript 简称JS，是一种动态的弱类型脚本解释性语言，和HTML、CSS并称三大WEB核心技术，得到了几乎主流浏览器支持。

1994年，网景Netscape公司成立并发布了Netscape Navigator浏览器，占据了很大的市场份额，网景意识到WEB需要动态，需要一种技术来实现。

1995年9月网景浏览器2发布测试版本发布了LiveScript，随即在12月的测试版就更名为**JavaScript**。同时期，微软推出IE并支持JScript、VBScript，与之抗衡。

1997年，网景、微软、SUN、Borland公司和其他组织在ECMA（European Computer Manufacturers Association 欧洲计算机制造商协会）确定了ECMAScript的本程序设计语言的标准。JavaScript和JScript都成为ECMAScript标准的实现。

2008年后随着chrome浏览器的V8引擎发布。

V8 JS引擎不是解释执行，而是本地编译，在V8引擎做了很多优化，JS程序在其上运行堪比本地二进制程序。V8引擎使用C++开发，可以嵌入到任何C++程序中。基于V8引擎，2009年基于服务器javascript的运行环境Node.js诞生，创建了第一版npm (Node.js包管理器和开源库生态系统), 提供了大量的库供程序员使用。从此，便可以在服务器端真正大规模使用JavaScript编程了。也就是说 JavaScript 也可以真正称为服务器端编程语言了，成为目前唯一的前，后端通用的语言。

同步

交互式网页，用户提交了请求，就是想看到查询的结果。服务器响应到来后是一个全新的页面内容，哪怕URL不变，整个网页都需要重新渲染。例如，用户填写注册信息，只是2次密码不一致，提交后，整个注册页面重新刷新，所有填写项目重新填写(当然有办法让用户减少重填)。这种交互非常不友好。从代价的角度看，就是为了注册的一点点信息，结果返回了整个网页内容，不但浪费了网络带宽，还需要浏览器重新渲染网页，太浪费资源了，影响了用户体验和感受。上面这些请求的过程，就是同步过程，用户发起请求，页面整个刷新，直到服务器端响应的数据到来并重新渲染。

异步

1996年微软实现了iframe标签，可以在一个网页使用iframe标签局部异步加载内容。

1999年微软推出异步数据传输的ActiveX插件技术，太笨重了，但是也火了很多年。有一个组件XMLHttpRequest被大多数浏览器支持。

传统的网页如果需要更新内容，必需重载整个网页面。Ajax的出现，改变这一切，同时极大的促进了Javascript的发展。Ajax 即"Asynchronous Javascript And XML"（异步 JavaScript 和 XML），是指一种创建交互式、快速动态网页应用的网页开发技术，最早起源于1998年微软的Outlook Web Access开发团队。Ajax 通过在后台与服务器进行少量数据交换， 可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。Javascript 通过调用浏览器内置的WEB API中的**XMLHttpRequest** 对象实现Ajax 技术。早期Ajax结合数据格式XML，目前更多的使用JSON。利用AJAX可实现前后端开发的彻底分离，改变了传统的开发模式。

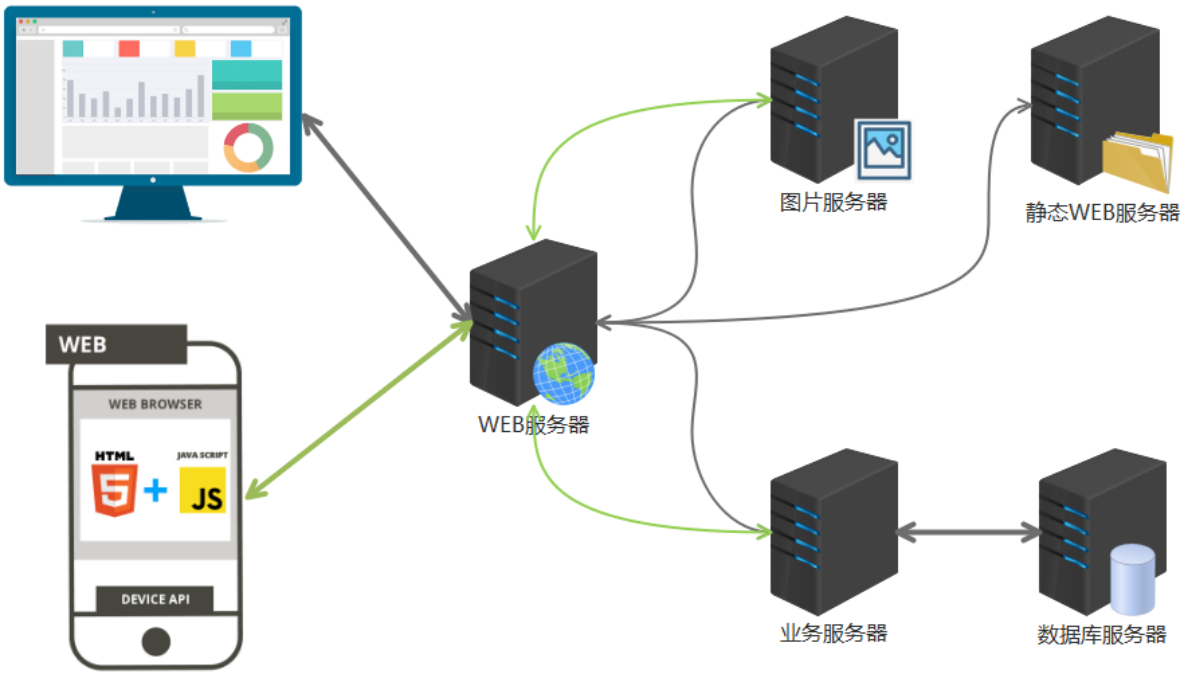
AJAX是一种技术的组合，技术的重新发现，而不是发明，但是它深远的影响了整个WEB开发。

参考资料：`https://www.w3school.com.cn/ajax/index.asp`

2 java 基础

2.1 WEB架构

2.1.1 web资源和访问



PC 端或移动端浏览器访问

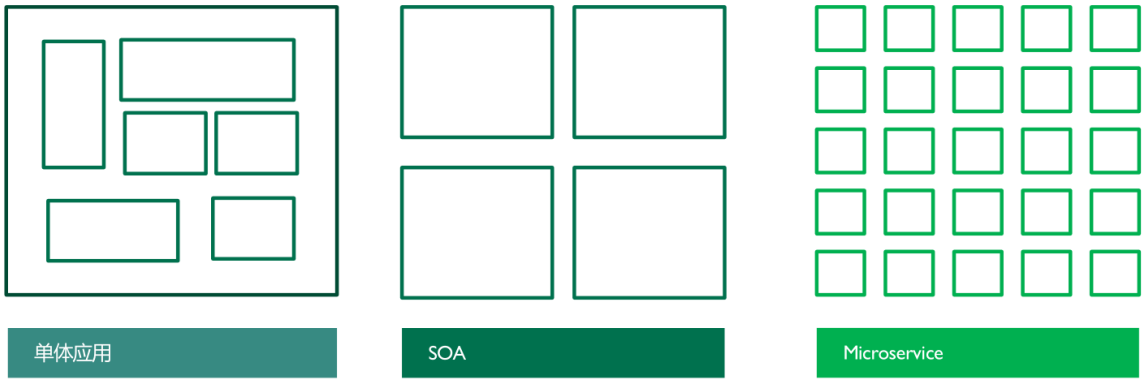
从静态服务器请求HTML、CSS、JS等文件发送到浏览器端，浏览器端接收后渲染在浏览器上
从图片服务器请求图片资源显示
从业务服务器访问动态内容，动态内容是请求后有后台服务访问数据库后得到的，最终返回到浏览器端

手机 App 访问

内置了HTML和JS文件，不需要从静态WEB服务器下载 JS 或 HTML。为的就是减少文件的发送，现代前端开发使用的JS文件太多或太大了
有必要就从图片服务器请求图片，从业务服务器请求动态数据

客户需求多样，更多的内容还是需要由业务服务器提供，业务服务器往往都是由一组服务器组成。

2.1.2 后台应用架构



2.1.2.1 单体架构

- 传统架构（单机系统），一个项目一个工程：比如商品、订单、支付、库存、登录、注册等等，统一部署，一个进程
- all in one的架构方式，把所有的功能单元放在一个应用里。然后把整个应用部署到一台服务器上。如果负载能力不行，将整个应用进行水平复制，进行扩展，然后通过负载均衡实现访问。
- Java实现：JSP、Servlet，打包成一个jar、war文件部署
- 易于开发和测试:也十分方便部署;当需要扩展时，只需要将war复制多份，然后放到多个服务器上，再做个负载均衡就可以了。

- 如果某个功能模块出问题，有可能全站不可访问，修改Bug后、某模块功能修改或升级后，需要停掉整个服务，重新整体重新打包、部署这个应用war包，功能模块相互之间耦合度高,相互影响,不适合当今互联网业务功能的快速迭代。
- 特别是对于一个大型应用，我们不可能吧所有内容都放在一个应用里面，我们如何维护、如何分工合作都是问题。如果项目庞大，管理难度大
- web应用服务器：开源的tomcat、jetty、glassfish。商用的有weblogic、websphere、Jboss

2.1.2.2 微服务

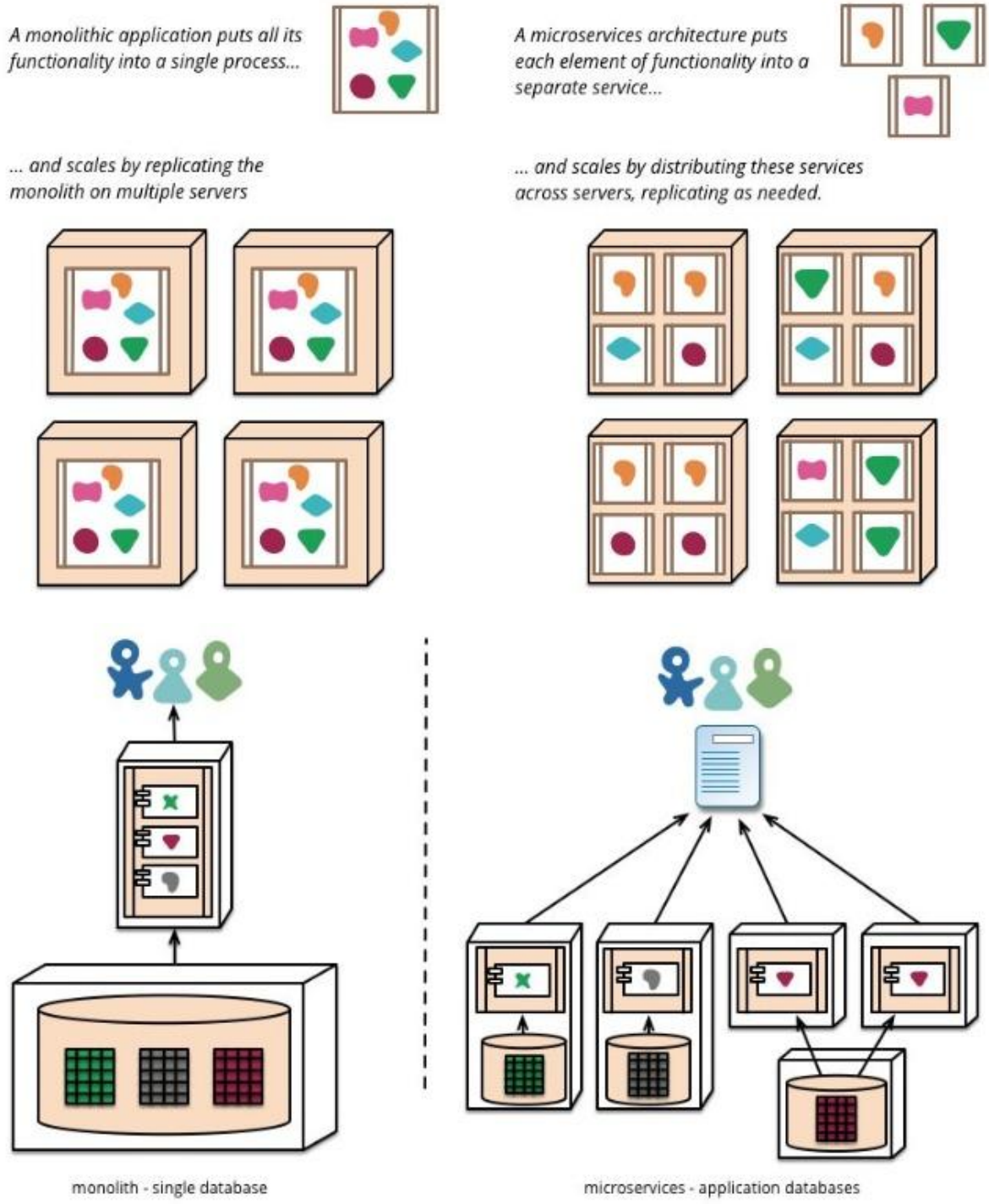
```
https://www.martinfowler.com/microservices/
```


```
In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.
简而言之，微服务架构风格是一种将单个应用程序开发为一组小服务的方法，每个小服务都在自己的进程中运行，并与轻量级机制（通常是 HTTP 资源 API）进行通信。 这些服务是围绕业务能力构建的，并且可以通过完全自动化的部署机制独立部署。 这些服务的集中管理极少，可以用不同的编程语言编写并使用不同的数据存储技术。
                                        -- James Lewis and Martin Fowler (2014)
```

```
亚马逊创始人 Jeff Bezos 说过一句话：“一个最好的团队用两个披萨可以喂饱”。
一个团队控制到6-10人左右
```

- 属于SOA（Service Oriented Architecture）的子集
- 微服务化的核心就是将传统的一站式应用，根据业务拆分成一个一个的服务，彻底去掉耦合，每一个微服务提供单个业务功能，一个服务只做一件事。每个服务都围绕着具体业务进行构建，并且能够被独立地部署到生产环境、类生产环境等
- 从技术角度讲就是一种小而独立的处理过程，类似与进程的概念，能够自行单独启动或销毁
- 微服务架构（分布式系统），各个模块/服务，各自独立出来，"让专业的人干专业的事"，独立部署。分布式系统中，不同的服务可以使用各自独立的数据库。
- 服务之间采用轻量级的通信机制（通常是基于HTTP的RESTful API）。
- 微服务设计的思想改变了原有的企业研发团队组织架构。传统的研发组织架构是水平架构，前端、后端、DBA、测试分别有自己对应的团队，属于水平团队组织架构。而微服务的设计思想对团队的划分有着一定的影响，使得团队组织架构的划分更倾向于垂直架构，比如用户业务是一个团队来负责，支付业务是一个团队来负责。但实际上在企业中并不会把团队组织架构拆分得这么绝对，垂直架构只是一种理想的架构
- 微服务的实现框架有多种，不同的应用架构，部署方式也有不同

2.1.2.3 单体架构和微服务比较



2.1.2.4 微服务的优缺点

微服务优点：

- 每个服务足够内聚，足够小，代码容易理解。这样能聚焦一个只当的业务功能或业务需求。
- 开发简单、开发效率提高，一个服务可能就是专业的只干一件事，微服务能够被小团队单独开发，这个小团队可以是2到5人的开发人员组成
- 微服务是松耦合的，是有功能意义的服务，无论是在开发阶段或部署阶段都是独立的。
- 微服务能使用不同的语言开发
- 易于和第三方集成，微服务运行容易且灵活的方式集成自动部署，通过持续集成工具，如: Jenkins、Hudson、Bamboo
- 微服务易于被一个开发人员理解、修改和维护，这样小团队能够更关注自己的工作成果，无需通过合作才能体现价值

- 微服务允许你利用融合最新技术。微服务只是业务逻辑的代码，不会和HTML/CSS或其他界面组件混合，即前后端分离
- 每个微服务都有自己的存储能力，可以有自己的数据库，也可以有统一数据库

微服务缺点：

- 微服务把原有的一个项目拆分成多个独立工程，增加了开发、测试、运维、监控等的复杂度
- 微服务架构需要保证不同服务之间的数据一致性，引入了分布式事务和异步补偿机制，为设计和开发带来一定挑战
- 开发人员和运维需要处理分布式系统的复杂性，需要更强的技术能力
- 微服务适用于复杂的大系统，对于小型应用使用微服务，进行盲目的拆分只会增加其维护和开发成本

2.1.2.5 常见的微服务框架

- Dubbo
    - 阿里开源贡献给了ASF，目前已经是Apache的顶级项目
    - 一款高性能的Java RPC服务框架，微服务生态体系中的一个重要组件
    - 将单体程序分解成多个功能服务模块，模块间使用Dubbo框架提供的高性能RPC通信
    - 内部协调使用Zookeeper，实现服务注册、服务发现和服务治理
- Spring cloud
    - 一个完整的微服务解决方案，相当于Dubbo的超集
    - 微服务框架，将单体应用拆分为粒度更小的单一功能服务
    - 基于HTTP协议的REST(Representational State Transfer 表述性状态转移）风格实现模块间通信

2.2 Java

2.2.1 Java历史

Java原指的是印度尼西亚的爪哇岛，人口众多，盛产咖啡、橡胶等。

Java语言最早是在1991年开始设计的，最初叫Oak项目，它初衷是跑在不同机顶盒设备中的。

1993年网景公司成立。Oak项目组很快他们发现了浏览器和动态网页技术这个巨大的市场，转向WEB方向。并首先发布了可以让网页动起来的Applet技术（浏览器中嵌入运行Java字节码的技术）。

在1995年，一杯爪哇岛咖啡成就了Java这个名字。

Sun公司第一个Java公开版本1.0发布于1996年。口号是"一次编写，到处运行"(Write once，Run anywhere)，跨平台运行。

1999年，SUN公司发布了第二代Java平台(Java2)。

2009年4月20日，Oracle甲骨文公司宣布将以每股9.50美元，总计**74亿美金**收购SUN（计算机系统）公司。2010年1月成功收购。

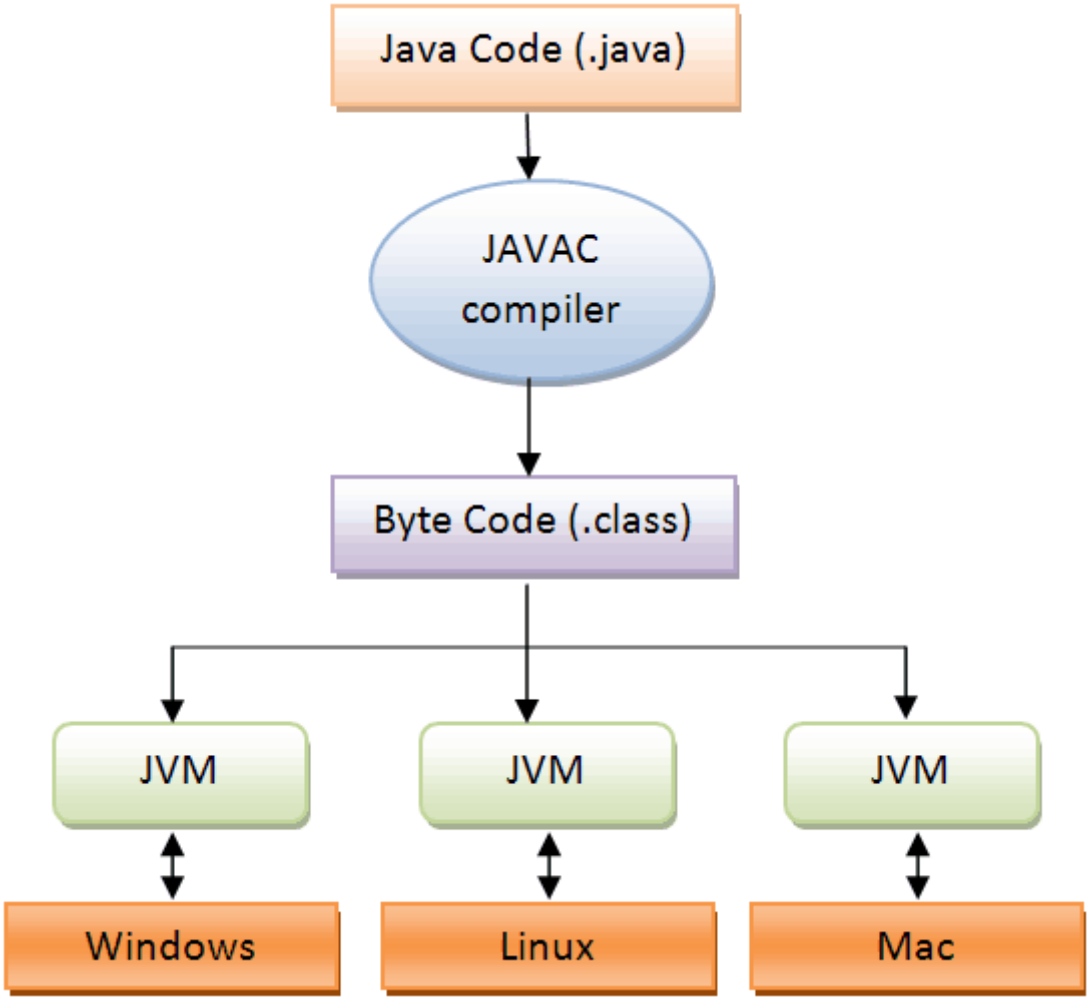
2010年，Java创始人之一的 **James Gosling** 离开了Oracle，去了Google。

> 2010年8月13日，Oracle在加利福尼亚地方法院起诉Google侵犯版权和专利权。Oracle声称Google侵犯了Java 37个API和部分专利。地方法院的陪审团认为未侵犯专利，且API无版权。
>
> 2016年5月26日，地方法院二审陪审团认定未侵犯版权，对37个JAVA API的重新实现受到**合理使用**的保护。
>
> 2017年Oracle上诉美国联邦巡回上诉法院，2018年3月27日判决Oracle胜诉，Google应赔偿近90亿美金。
>
> 2019年1月Google想让美国最高法院撤销联邦法院裁决。谷歌表示裁决是"对软件业的毁灭性一击"。现任特朗普政府支持Oracle公司，但微软、Mozilla、 红帽支持Google。目前案件已经受理，但由于疫情推迟。有更多的企业和组织加入进来，包括IBM、计算机和通信协会、互联网协会、超过150名学者和教授。
>
> 此案如果Oracle胜诉，将在美国形成判例，将深远广泛影响软件业。例如: POSIX接口， 是商用系统UNIX的兼容接口规范。
>
> 2021 年 4 月 5 日，美国最高法院就 Oracle 起诉 Google 知识产权侵权案作出了判决，这场两大科技巨头之间长达十几年的官司终于“一锤定音”：谷歌胜，甲骨文败。最高法院的判决中包含两项关键的决议：最高法院维持下级法院的原判，并裁定 API 受版权保护。 Google 从 Java 复制 11,500 行代码的行为属于“合理使用”。

2.2.2 java 组成

Java 包含下面部分：

- 语言、语法规范。关键字,如: if、for、class等
- 源代码 source code
- 依赖库，标准库(基础)、第三方库(针对某些应用)。底层代码太难使用且开发效率低，封装成现成的库
- JVM虚拟机。将源代码编译为中间码即字节码后,再运行在JVM之上



由于各种操作系统ABI不一样，采用编译方式，需要为不同操作系统编译成相应格式的二进制程序才能运行。

1995年，Java发布Applet技术，Java程序在后台编译成字节码，发送到浏览器端，在浏览器中运行一个Applet程序，这段程序是运行在另外一个JVM进程中的。

但是这种在客户端运行Java代码的技术，会有很大的安全问题。1997年CGI技术发展起来，动态网页技术开始向后端开发转移，在后端将动态内容组织好，拼成HTML发回到浏览器端。

2.2.3 Java动态网页技术

2.2.3.1 servlet

本质就是一段Java程序

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
  private String message;
  public void init() throws ServletException
  {
      message = "Hello World";
  }

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
```

```
    {
        response.setContentType("text/html"); //响应报文内容类型

        PrintWriter out = response.getWriter();  //构建响应报文内容
        out.println("<h1>" + message + "</h1>");
        out.println("<p><a href=http://www.magedu.com>马哥教育</a>欢迎你</p>");
    }

    public void destroy()
    {
    }
}
```

在Servlet中最大的问题是，HTML输出和Java代码混在一起，如果网页布局要调整，Java源代码就需要随之进行调整,对于开发人员来说就是个噩梦。

2.2.3.2 jsp（Java Server Pages）

JSP本质是提供一个HTML模板，也就是在网页中预留以后填充的空，后续将Java程序运行生成的数据对HTML进行填空就可以了。如果网页布局需要调整,JAVA源代码不需要很大的调整

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
本行后面的内容是服务器端动态生成字符串，最后拼接在一起
<%
out.println("你的 IP 地址 " + request.getRemoteAddr());
%>
</body>
</html>
```

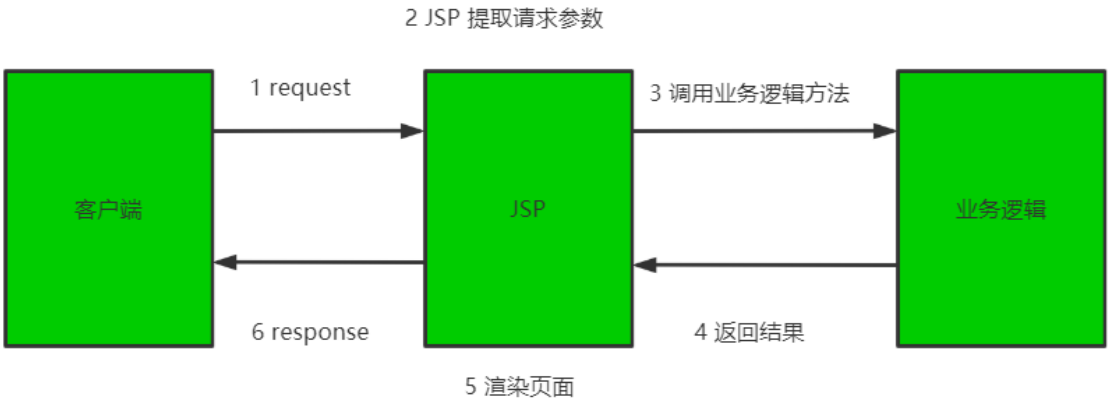
JSP是基于Servlet实现，JSP将表现和逻辑分离，这样页面开发人员更好的注重页面表现力更好服务客户。

不过最终 JSP 还需要先转换为 Servlet的源代码.java文件（Tomcat中使用Jasper转换），只不过这个转换过程无需人工完成,是通过工具自动实现的,然后再编译成.class文件，最后才可以在JVM中运行。

比如: 浏览器第一次请求test.jsp时, Tomcat服务器会自动将test.jsp转化成test.jsp.java这么一个类,并将该文件编译成class文件。编译完毕后再运行class文件来响应浏览器的请求。如果以后访问test.jsp就不再重新编译jsp文件了，直接调用class文件来响应浏览器。后续如果Tomcat检测到JSP页面改动了的话，会重新编译

JSP类似于PHP和ASP,前端代码和后端JAVA代码混写在一起,需要前端和后端工程师在一起协作才能完成,无法做到真正的前后端分离开发

在web早期的开发中，通常采用的分为两层，视图层和模型层。



优点：架构简单，比较适合小型项目开发

缺点：JSP职责不单一，职责过重，不便于维护

2.2.3.3 MVC

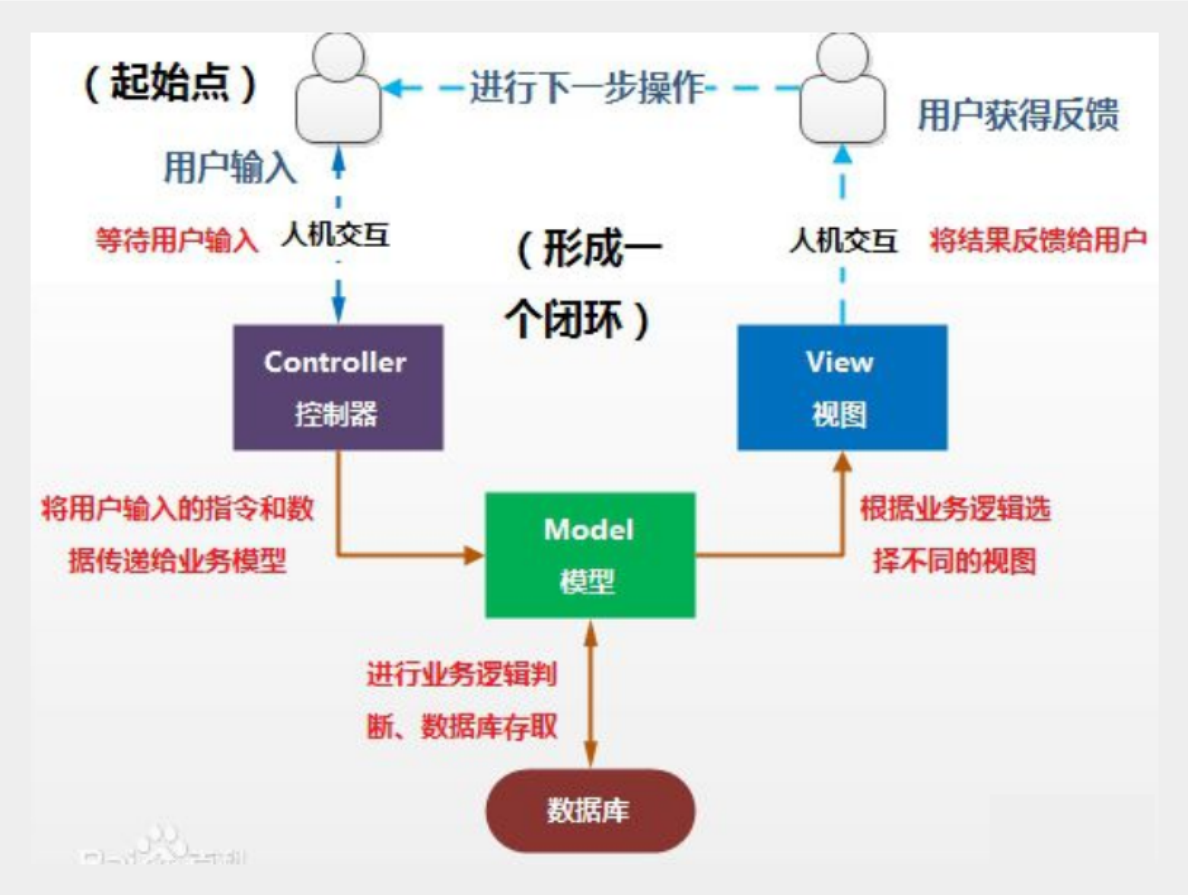
如果过度使用jsp技术,jsp中既写有大量的java代码，也有html，甚至还有javascript等，造成难以维护，难以实现前后端分工协作，后来java 的web开发借鉴了MVC（Model View Controller ）开发模式，

MVC是模型(Model)、视图(View)、控制器(Controller)的简写，是一种软件设计规范。是将业务逻辑、数据、显示分离的方法来组织代码。MVC主要作用是降低了视图与业务逻辑间的双向偶合。

MVC不是一种设计模式，MVC是一种架构模式。当然不同的MVC存在差异。

- Model（模型）：数据模型，提供要展示的数据，因此包含数据和行为，可以认为是领域模型或JavaBean组件（包含数据和行为），不过现在一般都分离开来：Value Object（数据Dao） 和 服务层（行为Service）。也就是模型提供了模型数据查询和模型数据的状态更新等功能，包括数据和业务。
- View（视图）：负责进行模型的展示，一般就是我们见到的用户界面，客户想看到的东西。可通过JSP实现
- Controller（控制器）：接收用户请求，委托给模型进行处理（状态改变），处理完毕后把返回的模型数据返回给视图，由视图负责展示。也就是说控制器做了个调度员的工作。最终表现为Servlet

最典型的MVC就是JSP + servlet + javabean的模式。



职责分析：

Controller：控制器

- 取得表单数据
- 调用业务逻辑
- 转向指定的页面

Model：模型

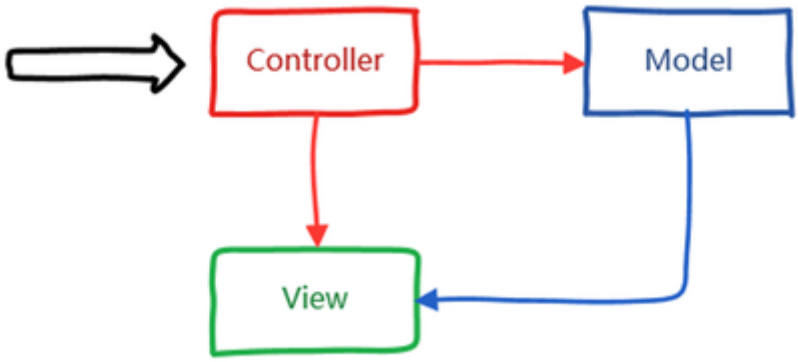
- 业务逻辑
- 保存数据的状态

View：视图

- 显示页面

处理流程

1. 用户发请求
2. Servlet接收请求数据，并调用对应的业务逻辑方法
3. 业务处理完毕，返回更新后的数据给servlet
4. servlet转向到JSP，由JSP来渲染页面
5. 响应给前端更新后的页面

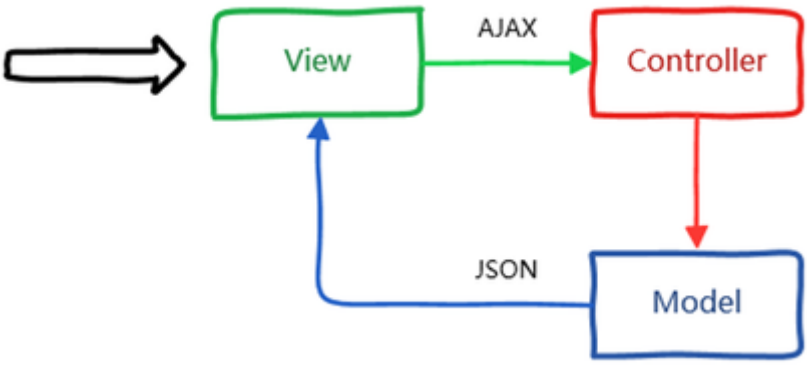


MVC模式也有以下不足：

- 每次请求必须经过"控制器->模型->视图"这个流程，用户才能看到最终的展现界面，这个过程似乎有些复杂
- 实际上视图是依赖于模型的，换句话说，如果没有模型，视图也无法呈现出最终的效果
- 渲染视图过程是在服务端来完成的，最终呈现给浏览器的是带有模型的视图页面，性能无法得到很好的优化

2.2.3.4 REST

为了使数据展现过程更加直接，并且提供更好的用户体验，对MVC模式进行改进。首先从浏览器发送AJAX（Asynchronous JavaScript and XML 异步的 JavaScript 和 XML））请求，然后服务端接受该请求并返回JSON数据返回给浏览器，最后在浏览器中进行界面渲染。改进后的MVC模式如下图所示：

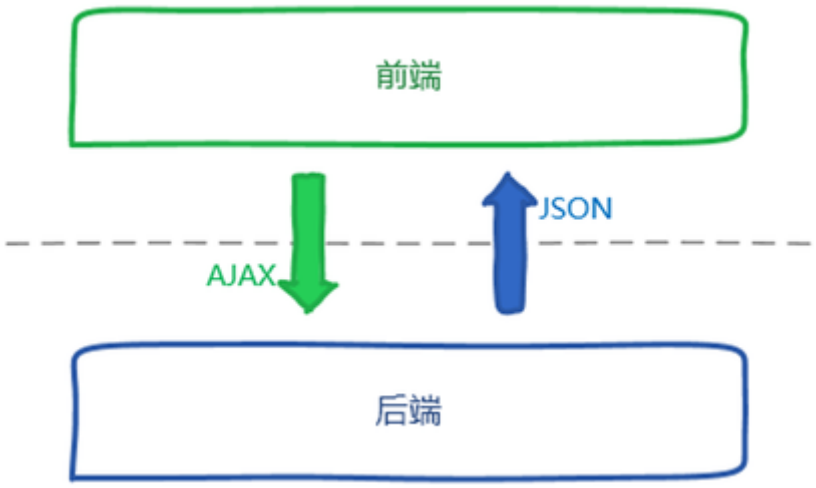


也就是说，我们输入的是AJAX请求，输出的是JSON数据，REST技术实现这样的功能。

REST全称是Representational State Transfer（表述性状态转移），全称是 Resource Representational State Transfer：通俗来讲就是：资源在网络中以某种表现形式进行状态转移。分解开来：Resource：资源，即数据；Representational：某种表现形式，比如用JSON，XML，JPEG等；State Transfer：状态变化。通过HTTP动词实现

它是Roy Fielding博士在2000年写的一篇关于软件架构风格的论文，后来国内外许多知名互联网公司纷纷开始采用这种轻量级的Web服务，大家习惯将其称为RESTful Web Services，或简称REST服务。

如果将浏览器这一端视为前端，而服务器那一端视为后端的话，可以将以上改进后的MVC模式简化为以下前后端分离模式，如下图所示：



可见，采用REST风格的架构可以使得前端关注界面展现，后端关注业务逻辑，分工明确，职责清晰。

在设计web接口的时候，REST主要是用于定义接口名，接口名一般是用名次写，不用动词，那怎么表达“获取”或者“删除”或者“更新”这样的操作呢——用请求类型来区分。

比如，我们有一个friends接口，对于“朋友”我们有增删改查四种操作，怎么定义REST接口？

增加一个朋友，uri: generalcode.cn/v1/friends 接口类型：POST

删除一个朋友，uri: generalcode.cn/va/friends 接口类型：DELETE

修改一个朋友，uri: generalcode.cn/va/friends 接口类型：PUT

查找朋友，uri: generalcode.cn/va/friends 接口类型：GET

上面我们定义的四个接口就是符合REST协议的，请注意，这几个接口都没有动词，只有名词friends，都是通过Http请求的接口类型来判断是什么业务操作。

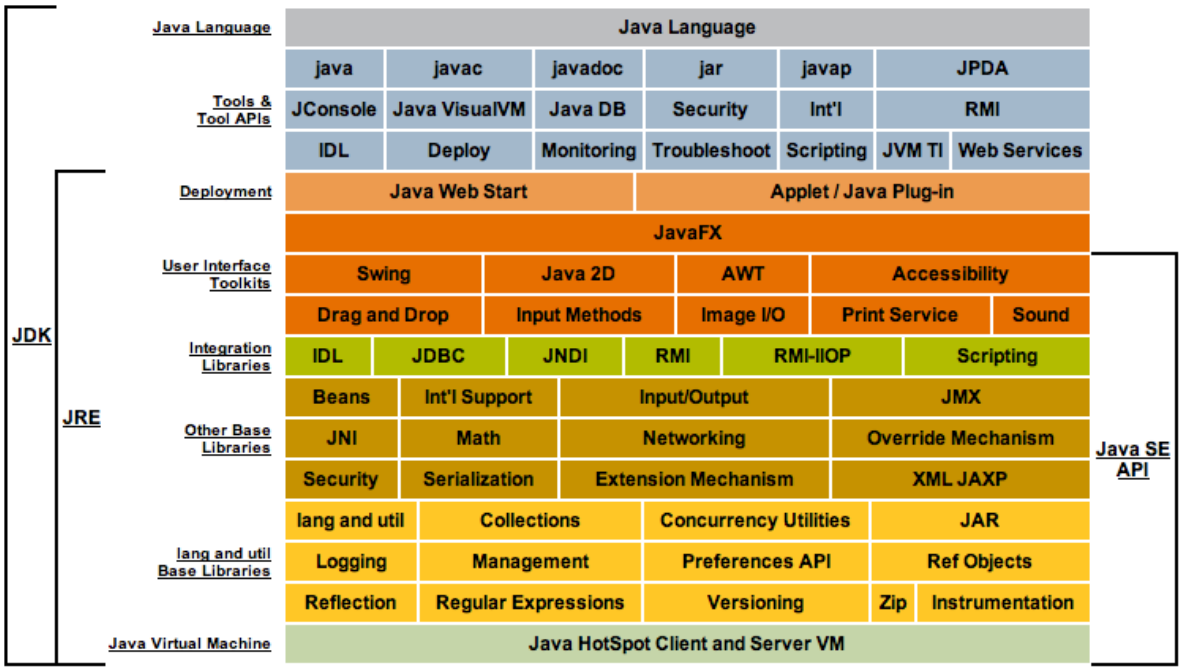
REST就是一种设计API的模式。最常用的数据格式是JSON。由于JSON能直接被JavaScript读取，所以，以JSON格式编写的REST风格的API具有简单、易读、易用的特点。

编写API有什么好处呢？由于API就是把Web App的功能全部封装了，所以，通过API操作数据，可以极大地把前端和后端的代码隔离，使得后端代码易于测试，前端代码编写更简单。离。前端拿到数据只负责展示和渲染，不对数据做任何处理。后端处理数据并以JSON格式传输出去，定义这样一套统一的接口，在web，ios，android三端都可以用相同的接口，通过客户端访问API，就可以完成通过浏览器页面提供的功能，而后端代码基本无需改动。

2.2.4 JDK

2.2.4.1 JDK和JRE

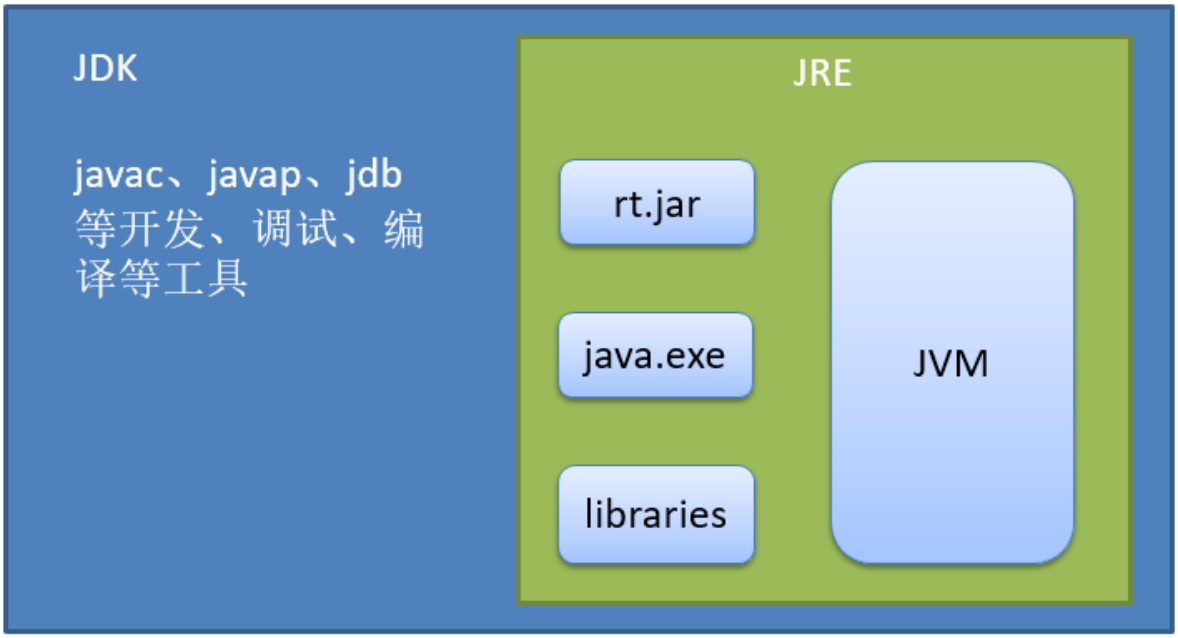
2.2.4.1.1 JDK 和 JRE 关系

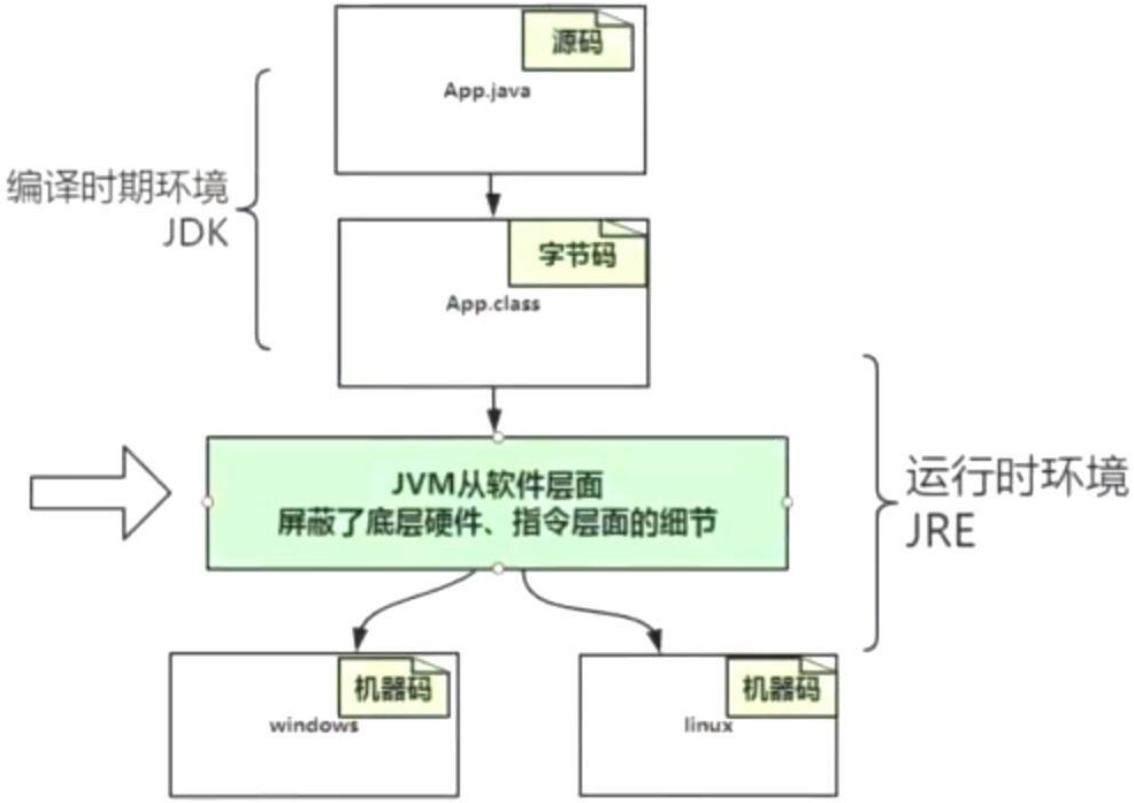


Java SE API: Java 基础类库开发接口

JRE：Java Runtime Environment缩写，指Java运行时环境， 包含 JVM + Java核心类库

JDK：Java Development Kit，即 Java 语言的软件开发工具包,JDK协议基于 JRL(JavaResearch License)协议





2.2.4.1.2 JVM 的各种版本

参考链接:

https://en.wikipedia.org/wiki/List_of_Java_virtual_machines

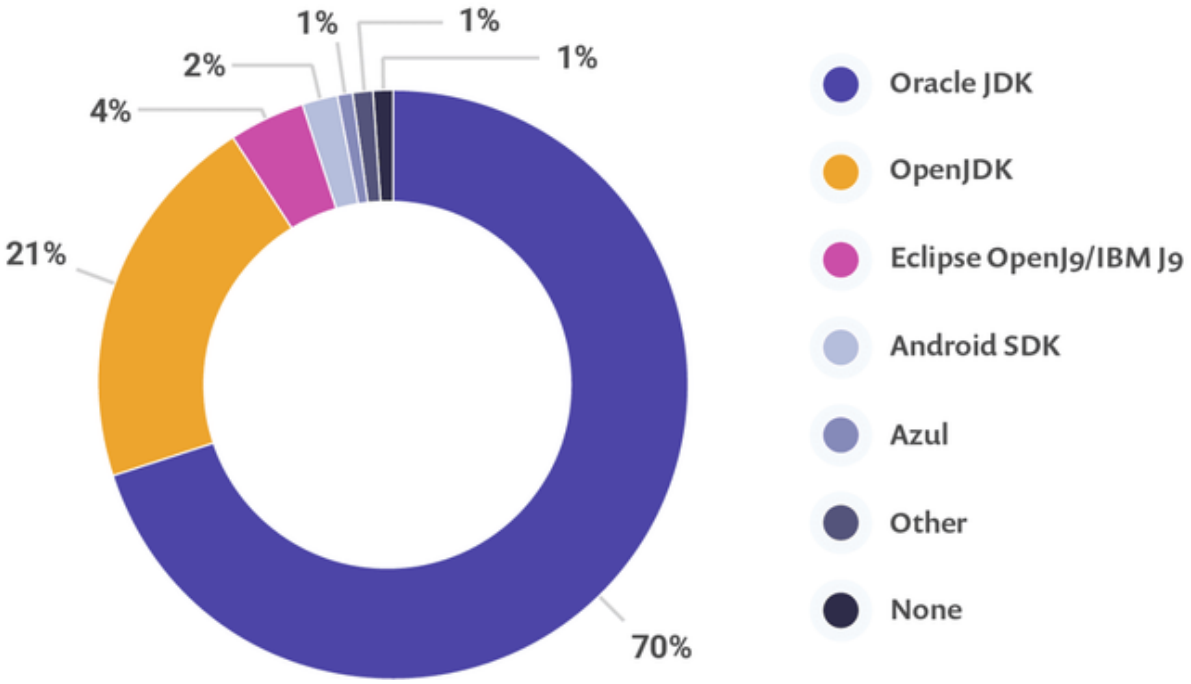
https://en.wikipedia.org/wiki/Comparison_of_Java_virtual_machines

各个公司和组织基于标准规范,开发了不同的JVM版本

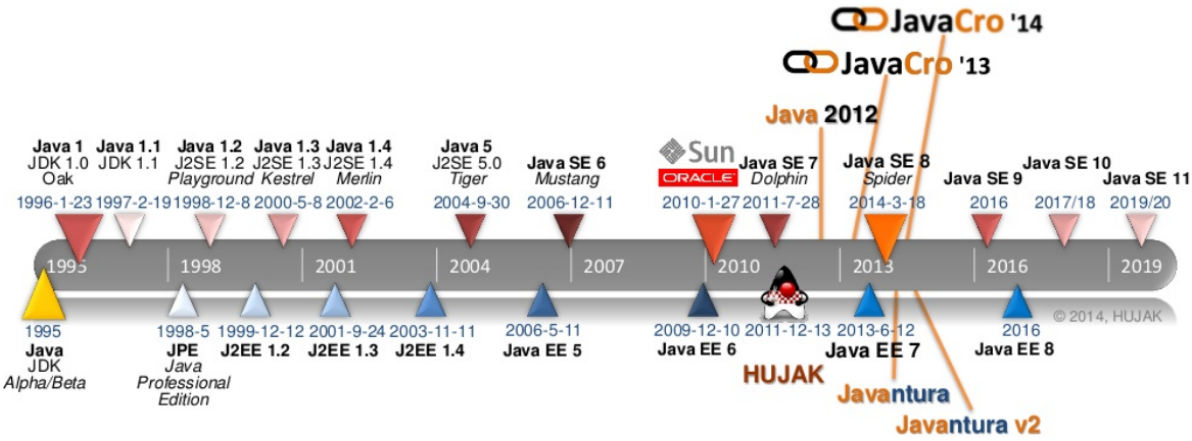
- SUN HotSpot
- IBM J9VM
- BEA JRockit

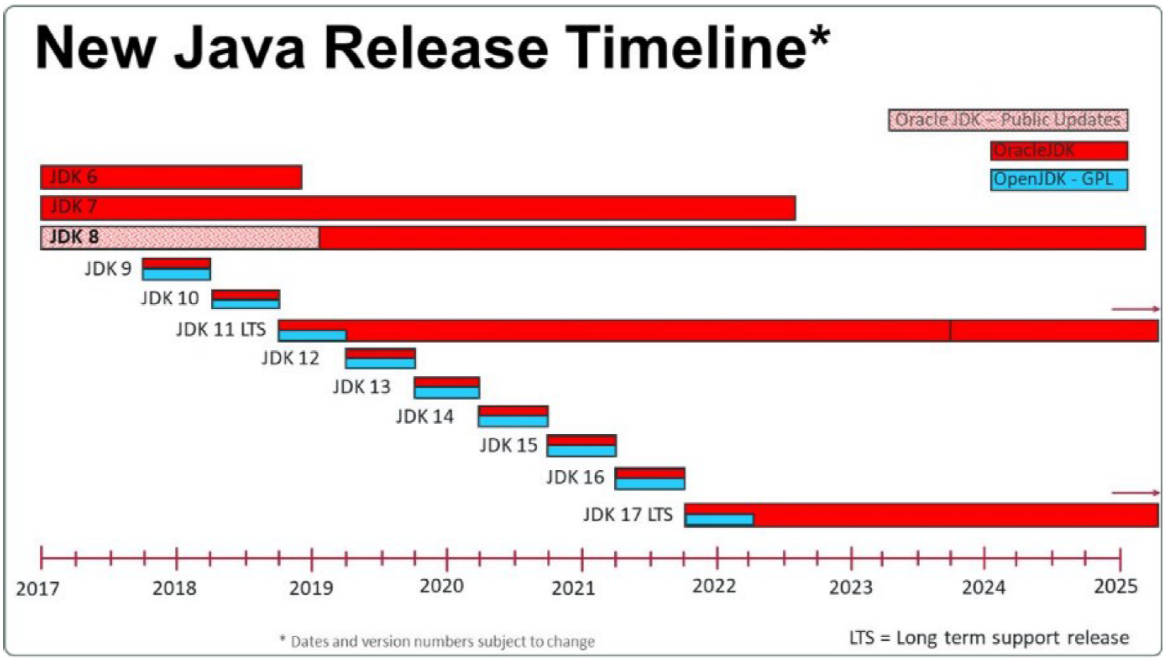
JVM 市场份额

2018年12月，由 Snyk 和 The Java Magazine 联合推出发布的 2018 JVM 生态调查报告



2.2.4.2 Oracle JDK版本





JDK也就是常说的J2SE，在1999年，正式发布了Java第二代平台，发布了三个版本：

- J2SE：标准版，适用于桌面平台
- J2EE：企业版，java在企业级开发所有规范的总和，共有13个大的规范,Servlet、Jsp都包含在JavaEE规范中
- J2ME：微型版，适用于移动、无线、机顶盒等设备环境

2005年，Java的版本又更名为JavaSE、JavaEE、JavaME

JDK7、JDK8、JDK11是LTS（Long Term Support）

JDK 历史版本

https://en.wikipedia.org/wiki/Java_version_history

Version	Release date	End of Free Public Updates[5][6]	Extended Support Until
JDK Beta	1995	?	?
JDK 1.0	January 1996	?	?
JDK 1.1	February 1997	?	?
J2SE 1.2	December 1998	?	?
J2SE 1.3	May 2000	?	?
J2SE 1.4	February 2002	October 2008	February 2013
J2SE 5.0	September 2004	November 2009	April 2015
Java SE 6	December 2006	April 2013	December 2018
Java SE 7	July 2011	April 2015	July 2022
Java SE 8 (LTS)	March 2014	**January 2019 for Oracle (commercial)** December 2020 for Oracle (personal use) At least May 2026 for AdoptOpenJDK At least June 2023[7] for Amazon Corretto	December 2030
Java SE 9	September 2017	March 2018 for OpenJDK	N/A
Java SE 10	March 2018	September 2018 for OpenJDK	N/A
Java SE 11 (LTS)	September 2018	At least August 2024[7] for Amazon Corretto October 2024 for AdoptOpenJDK	September 2026
Java SE 12	March 2019	September 2019 for OpenJDK	N/A
Java SE 13	September 2019	March 2020 for OpenJDK	N/A
Java SE 14	March 2020	September 2020 for OpenJDK	N/A
Java SE 15	September 2020	March 2021 for OpenJDK	N/A
Java SE 16	March 2021	September 2021 for OpenJDK	N/A
Java SE 17 (LTS)	September 2021	TBA	TBA

Legend: Old version | Older version, still maintained | **Latest version** | Future release

JDK 版本名称

版本	项目名称	发行日期
JDK 1.0	Oak(橡树)	1996-01-23
JDK 1.1		1997-02-19
JDK 1.1.4	Sparkler（宝石）	1997-09-12
JDK 1.1.5	Pumpkin（南瓜）	1997-12-13
JDK 1.1.6	Abigail（阿比盖尔–女子名）	1998-04-24
JDK 1.1.7	Brutus（布鲁图–古罗马政治家和将军）	1998-09-28
JDK 1.1.8	Chelsea（切尔西–城市名）	1999-04-08
J2SE 1.2	Playground（运动场）	1998-12-04
J2SE 1.2.1	none（无）	1999-03-30
J2SE 1.2.2	Cricket（蟋蟀）	1999-07-08
J2SE 1.3	Kestrel（美洲红隼）	2000-05-08
J2SE 1.3.1	Ladybird（瓢虫）	2001-05-17
J2SE 1.4.0	Merlin（灰背隼）	2002-02-13
J2SE 1.4.1	grasshopper（蚱蜢）	2002-09-16
J2SE 1.4.2	Mantis（螳螂）	2003-06-26
Java SE 5.0 (1.5.0)	Tiger（老虎）	2004-09-30
Java SE 6.0 (1.6.0)	Mustang（野马）	2006-12-11
Java SE 7.0 (1.7.0)	Dolphin（海豚）	2011-07-28
Java SE 8.0 (1.8.0)	Spider（蜘蛛）	2014-03-18
Java SE 9		2017-09-21
Java SE 10		2018-03-14
Java SE 11		2018-09-25
Java SE 12		2019-03-19
Java SE 13		2019-09-17
Java SE 14		2020-03-17

时间-事件轴

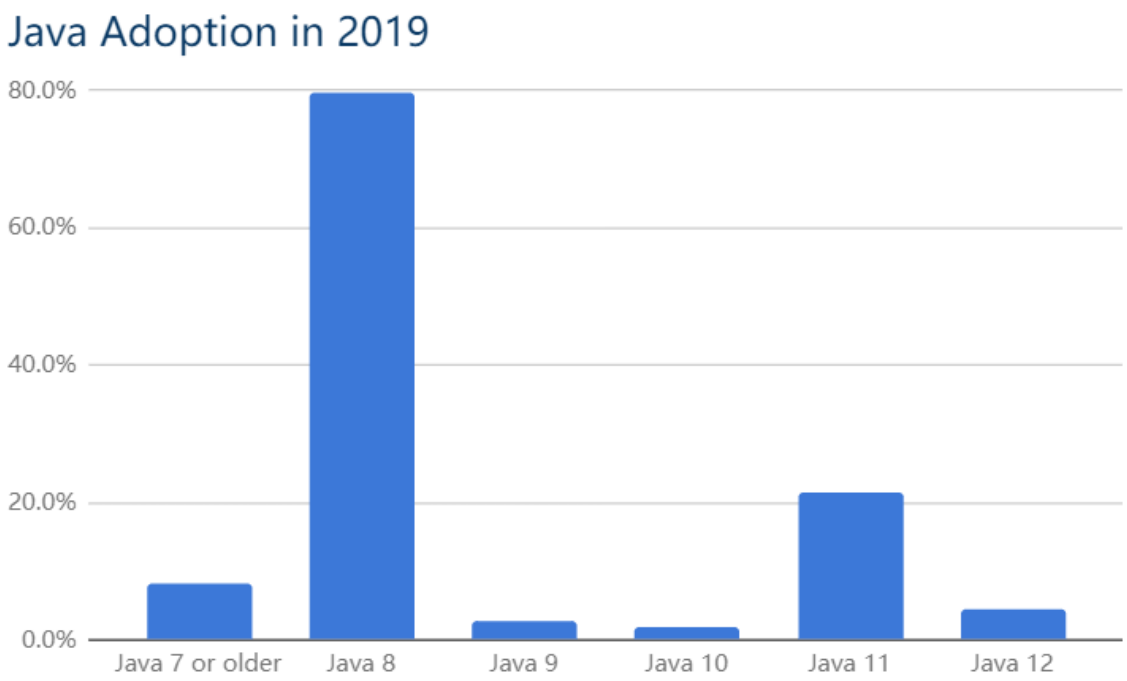
> 1995年5月23日，Java语言诞生
> 1996年1月，第一个JDK-JDK1.0诞生
> 1996年4月，10个最主要的操作系统供应商申明将在其产品中嵌入JAVA技术
> 1996年9月，约8.3万个网页应用了JAVA技术来制作
> 1997年2月18日，JDK1.1发布

> 1997年4月2日，JavaOne会议召开，参与者逾一万人，创当时全球同类会议规模之纪录
> 1997年9月，JavaDeveloperConnection社区成员超过十万
> 1998年2月，JDK1.1被下载超过2,000,000次
> 1998年12月8日，JAVA2企业平台J2EE发布
> 1999年6月，SUN公司发布Java的三个版本：标准版、企业版和微型版（J2SE、J2EE、J2ME）
> 2000年5月8日，JDK1.3发布
> 2000年5月29日，JDK1.4发布
> 2001年6月5日，NOKIA宣布，到2003年将出售1亿部支持Java的手机
> 2001年9月24日，J2EE1.3发布
> 2002年2月13日，J2SE1.4发布，自此Java的计算能力有了大幅提升。
> 2004年9月30日18:00PM，J2SE1.5发布，是Java语言的发展史上的又一里程碑事件。为了表示这个版本的重要性，J2SE1.5更名为J2SE5.0
> 2005年6月，JavaOne大会召开，SUN公司公开Java SE 6。此时，Java的各种版本已经更名以取消其中的数字"2"：J2EE更名为Java EE, J2SE更名为Java SE，J2ME更名为Java ME。
> 2006年11月13日，SUN公司宣布Java全线采纳GNU General Public License Version 2，从而公开了Java的源代码。

JDK 版本使用情况

数据来源

```
https://www.baeldung.com/java-in-2019
```



收费

从2019年1月份开始，Oracle JDK 开始对 Java SE 8 之后的版本开始进行商用收费，确切的说是 8u201/202 之后的版本。如果你用 Java 开发的功能如果是用作商业用途的，如果还不想花钱购买的话，能免费使用的最新版本是 8u201/202。当然如果是个人客户端或者个人开发者可以免费试用 Oracle JDK 所有的版本。

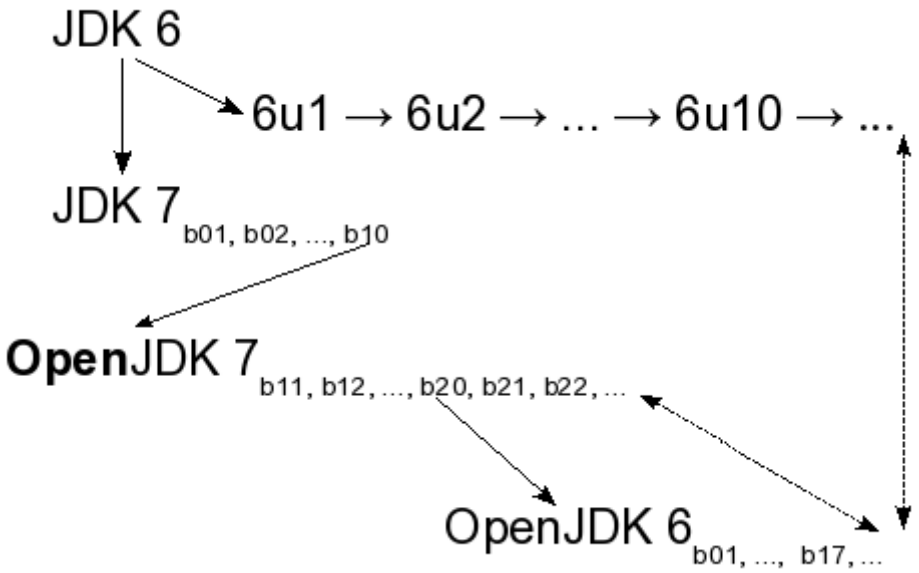
发版方式

在 JDK 9 发布之前，Oracle 的发版策略是以特性驱动的，只有重大的特性改变才会发布大版本，比如 JDK 7 到 JDK 8，中间会发多个更新版本。而从 JDK 9 开始变为以时间驱动的方式。发布周期为6个月一个大版本，比如 JDK 9 到 JDK 10，3个月一次补丁版，3年一个 LTS(长期支持版本)。

2.2.4.3 OpenJDK

2.2.4.3.1 OpenJDK 介绍

OpenJDK是Sun公司采用GPL v2协议发布的JDK开源版本，于2009年正式发布。



官方网站：https://openjdk.java.net/projects/jdk6/

OpenJDK 7是基于JDK7的beta版开发，但为了也将Java SE 6开源，从OpenJDK7的b20构建反向分支开发，从中剥离了不符合Java SE 6规范的代码，发布OpenJDK 6。所以OpenJDK6和JDK6没什么关系,只是API兼容而已

OpenJDK使用GPL v2可以用于商业用途。目前由红帽维护。OpenJDK也有在其基础上的众多发行版，比如阿里的Dragonwell。

相对来说,Oracle jDK具有更好的响应能力和JVM性能，更加稳定。

2.2.4.3.2 安装 openjdk

- 在CentOS中，可以使用 yum 仓库安装 openjdk

```
[root@centos8 ~]#dnf list "*jdk*"
BaseOS
 218 kB/s | 3.9 kB     00:00
AppStream
207 kB/s | 4.3 kB     00:00
EPEL
27 kB/s | 5.3 kB     00:00
EPEL
 6.9 MB/s | 5.6 MB     00:00
extras
 6.8 kB/s | 1.5 kB     00:00
Available Packages
copy-jdk-configs.noarch                          3.7-1.el8
AppStream
```

```
java-1.8.0-openjdk.x86_64                          1:1.8.0.232.b09-0.el8_0
AppStream
java-1.8.0-openjdk-accessibility.x86_64            1:1.8.0.232.b09-0.el8_0
AppStream
java-1.8.0-openjdk-demo.x86_64                     1:1.8.0.232.b09-0.el8_0
AppStream
java-1.8.0-openjdk-devel.x86_64                    1:1.8.0.232.b09-0.el8_0
AppStream
java-1.8.0-openjdk-headless.x86_64                 1:1.8.0.232.b09-0.el8_0
AppStream
java-1.8.0-openjdk-javadoc.noarch                  1:1.8.0.232.b09-0.el8_0
AppStream
java-1.8.0-openjdk-javadoc-zip.noarch              1:1.8.0.232.b09-0.el8_0
AppStream
java-1.8.0-openjdk-src.x86_64                      1:1.8.0.232.b09-0.el8_0
AppStream
java-11-openjdk.x86_64                             1:11.0.5.10-0.el8_0
AppStream
java-11-openjdk-demo.x86_64                        1:11.0.5.10-0.el8_0
AppStream
java-11-openjdk-devel.x86_64                       1:11.0.5.10-0.el8_0
AppStream
java-11-openjdk-headless.x86_64                    1:11.0.5.10-0.el8_0
AppStream
java-11-openjdk-javadoc.x86_64                     1:11.0.5.10-0.el8_0
AppStream
java-11-openjdk-javadoc-zip.x86_64                 1:11.0.5.10-0.el8_0
AppStream
java-11-openjdk-jmods.x86_64                       1:11.0.5.10-0.el8_0
AppStream
java-11-openjdk-src.x86_64                         1:11.0.5.10-0.el8_0
AppStream
java-latest-openjdk.x86_64                         1:13.0.2.8-1.rolling.el8
epel
java-latest-openjdk-demo.x86_64                    1:13.0.2.8-1.rolling.el8
epel
java-latest-openjdk-demo-slowdebug.x86_64          1:13.0.2.8-1.rolling.el8
epel
java-latest-openjdk-devel.x86_64                   1:13.0.2.8-1.rolling.el8
epel
java-latest-openjdk-devel-slowdebug.x86_64         1:13.0.2.8-1.rolling.el8
epel
java-latest-openjdk-headless.x86_64                1:13.0.2.8-1.rolling.el8
epel
java-latest-openjdk-headless-slowdebug.x86_64      1:13.0.2.8-1.rolling.el8
epel
java-latest-openjdk-javadoc.x86_64                 1:13.0.2.8-1.rolling.el8
epel
java-latest-openjdk-javadoc-zip.x86_64             1:13.0.2.8-1.rolling.el8
epel
java-latest-openjdk-jmods.x86_64                   1:13.0.2.8-1.rolling.el8
epel
java-latest-openjdk-jmods-slowdebug.x86_64         1:13.0.2.8-1.rolling.el8
epel
java-latest-openjdk-slowdebug.x86_64               1:13.0.2.8-1.rolling.el8
epel
java-latest-openjdk-src.x86_64                     1:13.0.2.8-1.rolling.el8
epel
```

```
java-latest-openjdk-src-slowdebug.x86_64        1:13.0.2.8-1.rolling.el8
epel

[root@centos8 ~]#dnf -y install java-1.8.0-openjdk.x86_64 java-1.8.0-openjdk-
devel

[root@centos8 ~]#java -version
openjdk version "1.8.0_232"
OpenJDK Runtime Environment (build 1.8.0_232-b09)
OpenJDK 64-Bit Server VM (build 25.232-b09, mixed mode)

[root@centos8 ~]#which java
/usr/bin/java
[root@centos8 ~]#ll /usr/bin/java
lrwxrwxrwx 1 root root 22 Feb  8 20:03 /usr/bin/java -> /etc/alternatives/java

[root@centos8 ~]#ll /etc/alternatives/java
lrwxrwxrwx 1 root root 73 Feb  8 20:03 /etc/alternatives/java ->
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.232.b09-0.el8_0.x86_64/jre/bin/java

[root@centos8 ~]#rpm -qf /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.232.b09-
0.el8_0.x86_64/jre/bin/java
java-1.8.0-openjdk-headless-1.8.0.232.b09-0.el8_0.x86_64
```

范例: 安装 java-11-openjdk

```
OpenJDK 64-Bit Server VM 18.9 (build 11.0.7+10-LTS, mixed mode, sharing)
[root@centos8 ~]#dnf -y install java-11-openjdk.x86_64
[root@centos8 ~]#java --version
openjdk 11.0.7 2020-04-14 LTS
OpenJDK Runtime Environment 18.9 (build 11.0.7+10-LTS)
```

范例: 编译运行java程序

```
[root@centos8 ~]#dnf -y install  java-1.8.0-openjdk-devel

[root@centos8 ~]#cat Hello.java
class Hello{
    public static void main(String[] args)
    {
        System.out.println("hello, world");
    }
}

#编译成字节码
[root@centos8 ~]#javac Hello.java
[root@centos8 ~]#ll  Hello.*
-rw-r--r-- 1 root root 416 Oct 24 13:00 Hello.class
-rw-r--r-- 1 root root 130 Aug 22 23:38 Hello.java

[root@centos8 ~]#file Hello.class
Hello.class: compiled Java class data, version 52.0 (Java 1.8)

#运行java程序
[root@centos8 ~]#java Hello
hello, world
```

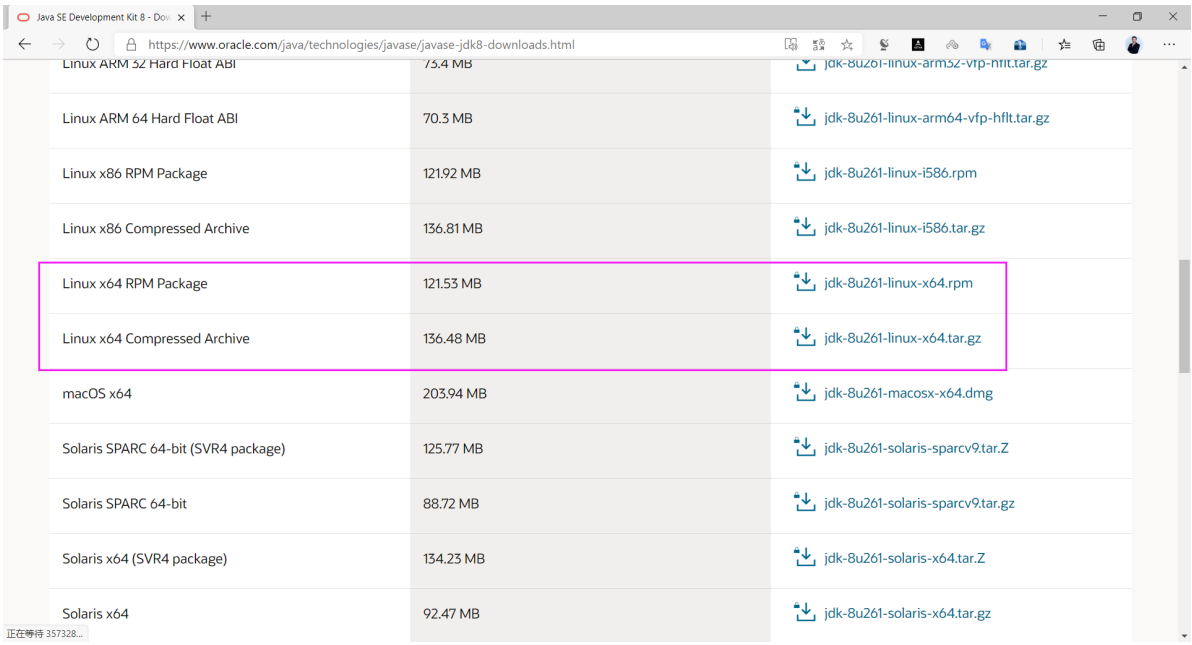
- ubuntu 安装 openjdk

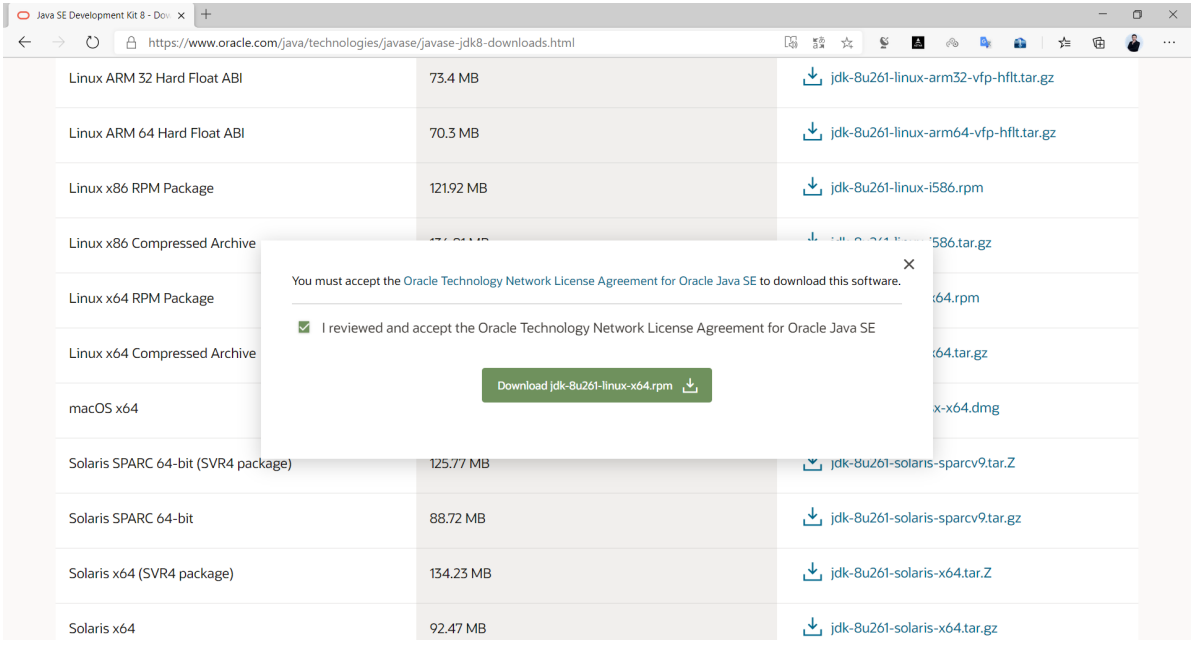
```
[root@ubuntu1804 ~]#apt update
[root@ubuntu1804 ~]#apt -y install openjdk-8-jdk
[root@ubuntu1804 ~]#java -version
openjdk version "1.8.0_242"
OpenJDK Runtime Environment (build 1.8.0_242-8u242-b08-0ubuntu3~18.04-b08)
OpenJDK 64-Bit Server VM (build 25.242-b08, mixed mode)

root@ubuntu2004:~# apt -y install openjdk-11-jdk
root@ubuntu2004:~# java -version
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-Ubuntu-0ubuntu1.20.04)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-Ubuntu-0ubuntu1.20.04, mixed mode,
sharing)
```

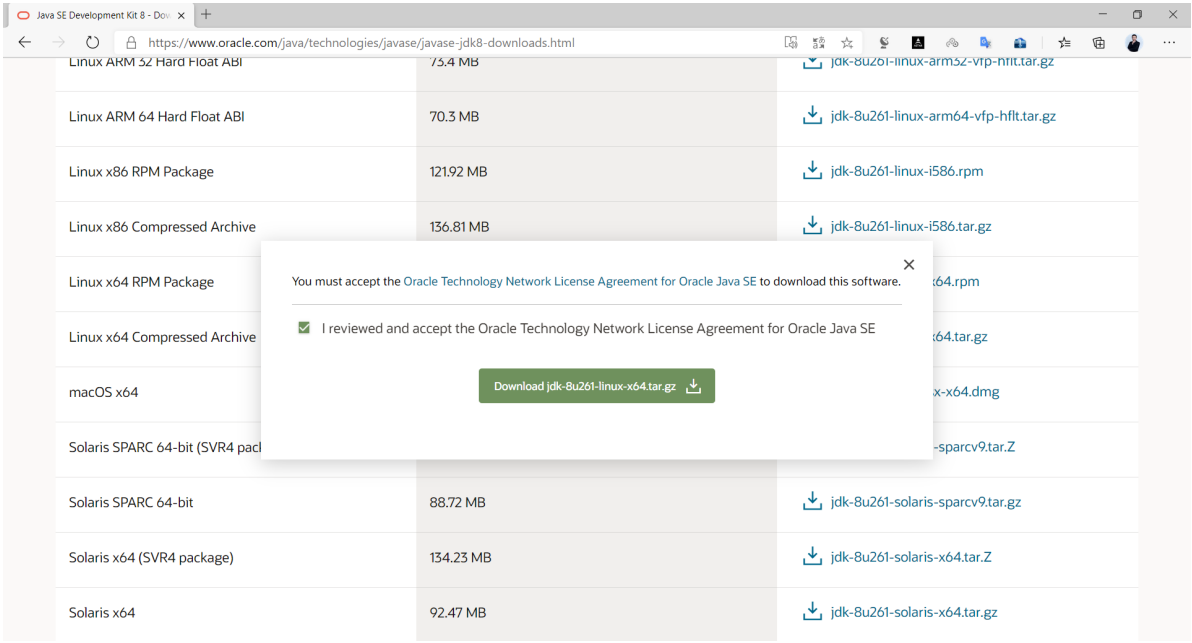
2.2.4.4 安装oracle官方 JDK

官方下载链接:

```
#注意需要注册登录后,才能下载JDK
https://www.oracle.com/java/technologies/downloads/#java8
https://www.oracle.com/java/technologies/downloads/#java11
https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html
```







2.2.4.4.1 Oracle JDK 的 rpm安装

```
#需要登录下载：https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html

[root@centos8 ~]#ls -lh jdk-8u241-linux-x64.rpm
-rw-r--r-- 1 root root 171M Feb  8 18:29 jdk-8u241-linux-x64.rpm

#安装jdk，无相关依赖包
[root@centos8 ~]#dnf -y install jdk-8u241-linux-x64.rpm
[root@centos8 ~]#java -version
java version "1.8.0_241"
Java(TM) SE Runtime Environment (build 1.8.0_241-b07)
Java HotSpot(TM) 64-Bit Server VM (build 25.241-b07, mixed mode)

#初始化环境变量
[root@centos8 ~]#vim /etc/profile.d/jdk.sh
[root@centos8 ~]#cat /etc/profile.d/jdk.sh
export JAVA_HOME=/usr/java/default
export PATH=$JAVA_HOME/bin:$PATH
#以下两项非必须项
```

```
export JRE_HOME=$JAVA_HOME/jre
export CLASSPATH=$JAVA_HOME/lib/:$JRE_HOME/lib/

[root@centos8 ~]#. /etc/profile.d/jdk.sh

#查看jdk信息
[root@centos8 ~]#which java
/usr/bin/java
[root@centos8 ~]#ll /usr/bin/java
lrwxrwxrwx 1 root root 22 Feb  8 18:35 /usr/bin/java -> /etc/alternatives/java
[root@centos8 ~]#ll /etc/alternatives/java
lrwxrwxrwx 1 root root 41 Feb  8 18:35 /etc/alternatives/java ->
/usr/java/jdk1.8.0_241-amd64/jre/bin/java

[root@centos8 ~]#rpm -q --scripts jdk-8u241-linux-x64.rpm

#查看到安装目录为/user/java下
[root@centos8 ~]#rpm -ql jdk-8u241-linux-x64.rpm |less
warning: jdk-8u241-linux-x64.rpm: Header V3 RSA/SHA256 Signature, key ID
ec551f03: NOKEY
/usr
/usr/java
/usr/java/jdk1.8.0_241-amd64
/usr/java/jdk1.8.0_241-amd64/.java
/usr/java/jdk1.8.0_241-amd64/.java/.systemPrefs
/usr/java/jdk1.8.0_241-amd64/.java/.systemPrefs/.system.lock
/usr/java/jdk1.8.0_241-amd64/.java/.systemPrefs/.systemRootModFile
/usr/java/jdk1.8.0_241-amd64/.java/init.d
......

[root@centos8 ~]#ll /usr/java/
total 0
lrwxrwxrwx 1 root root  16 Feb  8 18:35 default -> /usr/java/latest
drwxr-xr-x 8 root root 258 Feb  8 18:35 jdk1.8.0_241-amd64
lrwxrwxrwx 1 root root  28 Feb  8 18:35 latest -> /usr/java/jdk1.8.0_241-amd64
```

2.2.4.4.2 Oracle JDK的二进制文件安装

```
#下载安装包：https://www.oracle.com/java/technologies/javase/javase-jdk8-
downloads.html
[root@centos8 ~]#tar xvf jdk-8u241-linux-x64.tar.gz  -C /usr/local/
[root@centos8 ~]#cd  /usr/local/
[root@centos8 ~]#ln -s jdk1.8.0_241/ jdk

#初始化环境变量
[root@centos8 ~]#vim /etc/profile.d/jdk.sh
[root@centos8 ~]#cat /etc/profile.d/jdk.sh
export JAVA_HOME=/usr/local/jdk
export PATH=$PATH:$JAVA_HOME/bin
#以下两项非必须项
#export JRE_HOME=$JAVA_HOME/jre
#export CLASSPATH=.:$JAVA_HOME/lib/:$JRE_HOME/lib/

[root@centos8 ~]#.  /etc/profile.d/jdk.sh

#注意:JAVA_HOME变量必须设置,否则tomcat启动时会出下面错误
[root@centos8 ~]#catalina.sh
```

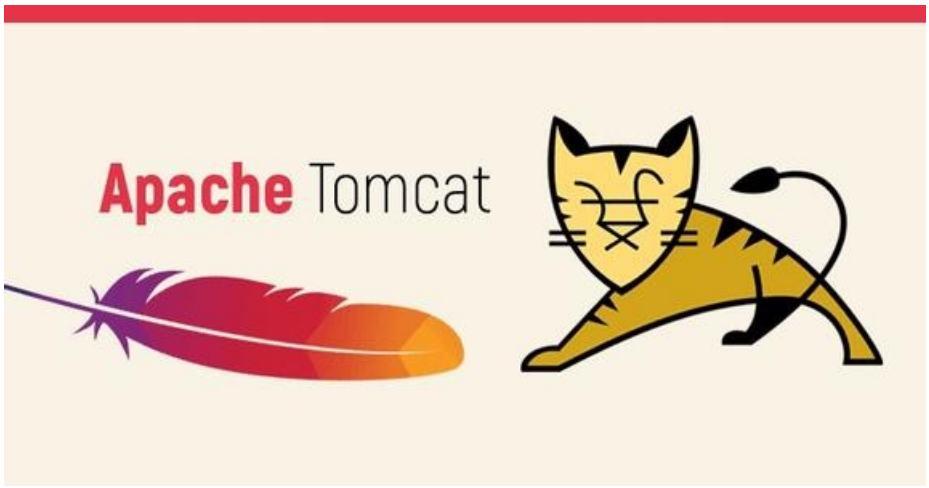
```
Neither the JAVA_HOME nor the JRE_HOME environment variable is defined
At least one of these environment variable is needed to run this program
[root@centos8 ~]#startup.sh
Neither the JAVA_HOME nor the JRE_HOME environment variable is defined
At least one of these environment variable is needed to run this program

#验证安装
[root@centos8 ~]#java -version
java version "1.8.0_241"
Java(TM) SE Runtime Environment (build 1.8.0_241-b07)
Java HotSpot(TM) 64-Bit Server VM (build 25.241-b07, mixed mode)
[root@centos8 ~]#which java
/usr/local/jdk/bin/java
```

2.3.4.4.3 一键安装二进制的JDK

```
[root@ubuntu1804 ~]#cat install_jdk.sh
#!/bin/bash
#**************************************************************************
#Author:            wangxiaochun
#QQ:                29308620
#Date:              2021-03-15
#FileName:          install_jdk.sh
#URL:               http://www.wangxiaochun.com
#Description:       The test script
#Copyright (C):     2021 All rights reserved
#**************************************************************************

DIR=`pwd`
JDK_FILE="jdk-8u281-linux-x64.tar.gz"
JDK_DIR="/usr/local"

color () {
    RES_COL=60
    MOVE_TO_COL="echo -en \\033[${RES_COL}G"
    SETCOLOR_SUCCESS="echo -en \\033[1;32m"
    SETCOLOR_FAILURE="echo -en \\033[1;31m"
    SETCOLOR_WARNING="echo -en \\033[1;33m"
    SETCOLOR_NORMAL="echo -en \E[0m"
    echo -n "$2" && $MOVE_TO_COL
    echo -n "["
    if [ $1 = "success" -o $1 = "0" ] ;then
        ${SETCOLOR_SUCCESS}
        echo -n $"  OK  "
    elif [ $1 = "failure" -o $1 = "1"  ] ;then
        ${SETCOLOR_FAILURE}
        echo -n $"FAILED"
    else
        ${SETCOLOR_WARNING}
        echo -n $"WARNING"
    fi
    ${SETCOLOR_NORMAL}
    echo -n "]"
    echo

}
```

```
install_jdk(){
if !  [  -f "$DIR/$JDK_FILE" ];then
    color 1  "$JDK_FILE 文件不存在"
    exit;
elif [ -d $JDK_DIR/jdk ];then
        color 1  "JDK 已经安装"
    exit
else
        [ -d "$JDK_DIR" ] || mkdir -pv $JDK_DIR
fi
tar xvf $DIR/$JDK_FILE  -C $JDK_DIR
cd  $JDK_DIR && ln -s jdk* jdk

cat >  /etc/profile.d/jdk.sh <<EOF
export JAVA_HOME=$JDK_DIR/jdk
export PATH=\$PATH:\$JAVA_HOME/bin
#export JRE_HOME=\$JAVA_HOME/jre
#export CLASSPATH=.:\$JAVA_HOME/lib/:\$JRE_HOME/lib/
EOF
.  /etc/profile.d/jdk.sh
java -version && color 0  "JDK 安装完成" || { color 1  "JDK 安装失败" ; exit; }

}

install_jdk
```

3 Tomcat 基础功能



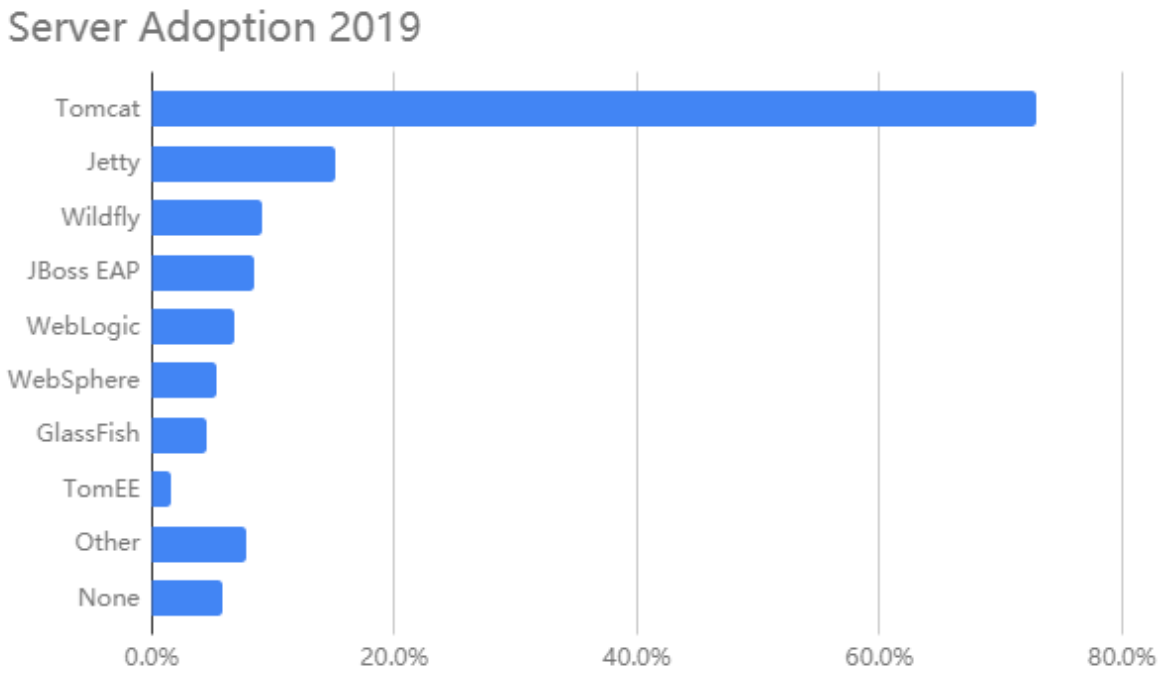
3.1 Tomcat历史和介绍

3.1.1 WEB应用服务器

Web 应用服务器的使用

数据来源

```
https://www.baeldung.com/java-in-2019
```



- 商用：IBM WebSphere、Oracle WebLogic（原属于BEA公司）、Oracle Oc4j、RedHat JBoss等
- 开源：Tomcat、Jetty、Resin、Glassfish

3.1.2 Tomcat 介绍

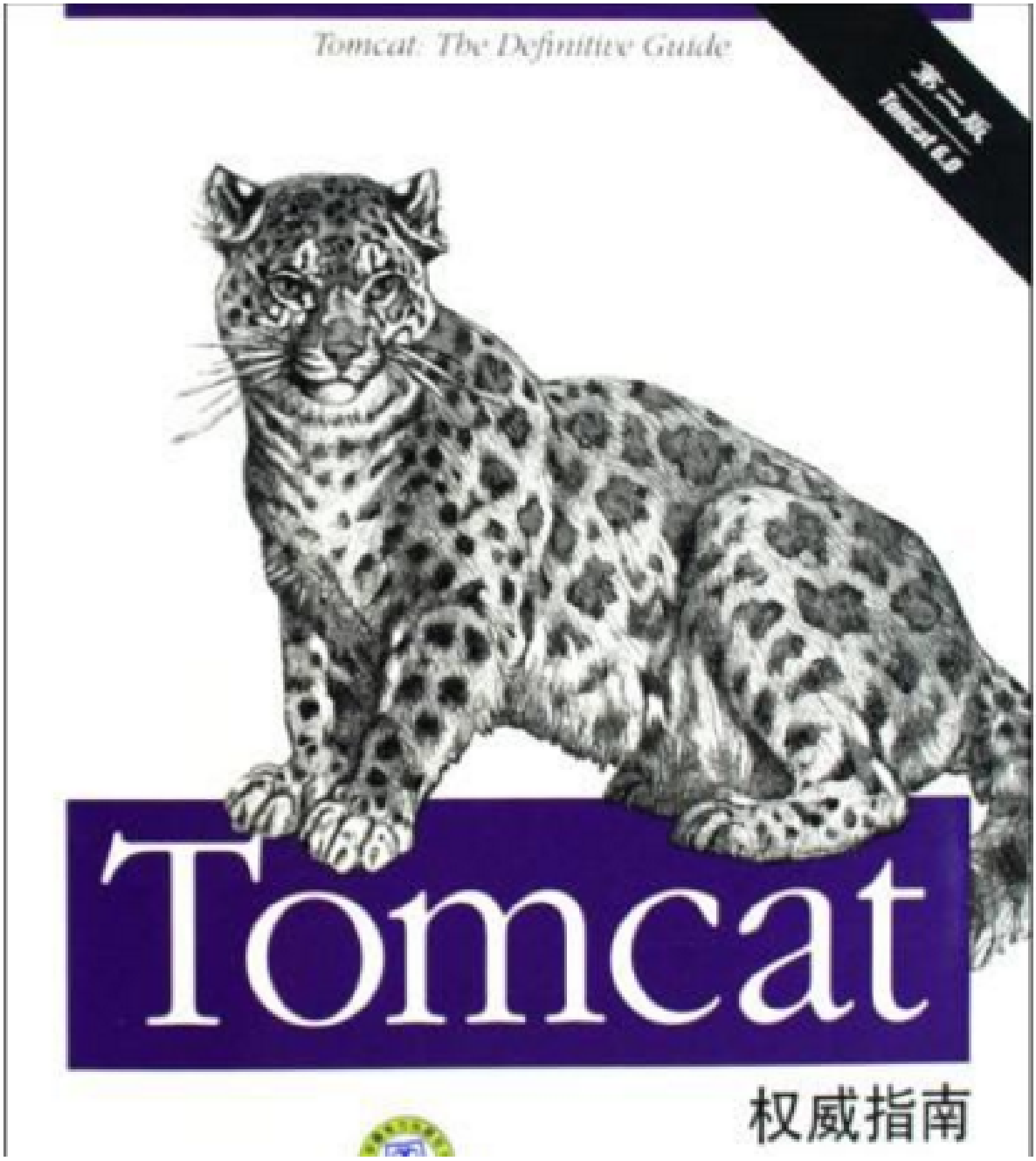
Tomcat 服务器是一个免费的开放源代码的Web 应用服务器，属于轻量级应用服务器，在中小型系统和并发访问用户不是很多的场合下被普遍使用，Tomcat 具有处理HTML静态资源页面的功能，它还是一个Servlet和JSP容器

起始于SUN 公司的一个Servlet的参考实现项目 Java Web Server，开发者是 James Duncan Davidson，在1999年，将项目贡献给了apache软件基金会（ASF），和ASF现有的项目 JServ 合并，并开源成为顶级项目

Tomcat 仅仅实现了Java EE规范中与Servlet、JSP相关的类库，是JavaEE不完整实现。

著名图书出版商O'Reilly约稿该项目成员Davidson希望使用一个公猫作为封面，但是公猫已经被使用，书出版后封面是一只雪豹。

《Tomcat权威指南》封面如下



1999年发布初始版本是Tomcat 3.0，实现了Servlet 2.2 和 JSP 1.1规范。

Tomcat 4.x发布时，内建了Catalina（Servlet容器）和 Jasper（JSP engine）等

当前 Tomcat 的正式版本已经更新到 9.0.x 版本，但当前企业中主流版本为 8.x 和 7.x

官网： http://tomcat.apache.org/

官网文档: https://tomcat.apache.org/tomcat-8.5-doc/index.html

帮助文档:

https://cwiki.apache.org/confluence/display/tomcat/

https://cwiki.apache.org/confluence/display/tomcat/FAQ

3.1.3 Tomcat 各版本区别

官方文档：https://tomcat.apache.org/whichversion.html

Servlet Spec	JSP Spec	EL Spec	WebSocket Spec	Authentication (JASIC) Spec	Apache Tomcat Version	Latest Released Version	Supported Java Versions
5.0	3.0	4.0	2.0	2.0	10.0.x	10.0.0-M1	8 and later
4.0	2.3	3.0	1.1	1.1	9.0.x	9.0.31	8 and later
3.1	2.3	3.0	1.1	1.1	8.5.x	8.5.51	7 and later
3.1	2.3	3.0	1.1	N/A	8.0.x (superseded)	8.0.53 (superseded)	7 and later
3.0	2.2	2.2	1.1	N/A	7.0.x	7.0.100	6 and later (7 and later for WebSocket)
2.5	2.1	2.1	N/A	N/A	6.0.x (archived)	6.0.53 (archived)	5 and later
2.4	2.0	N/A	N/A	N/A	5.5.x (archived)	5.5.36 (archived)	1.4 and later
2.3	1.2	N/A	N/A	N/A	4.1.x (archived)	4.1.40 (archived)	1.3 and later
2.2	1.1	N/A	N/A	N/A	3.3.x (archived)	3.3.2 (archived)	1.1 and later

3.2 安装 Tomcat

3.2.1 基于包安装 Tomcat

3.2.1.1 CentOS 包安装 tomcat

CentOS 8 包仓库中目前还没有提供tomcat相关包

```
[root@centos8 ~]#yum list tomcat
Last metadata expiration check: 1:25:35 ago on Wed 15 Jul 2020 09:01:28 AM CST.
Error: No matching Packages to list
```

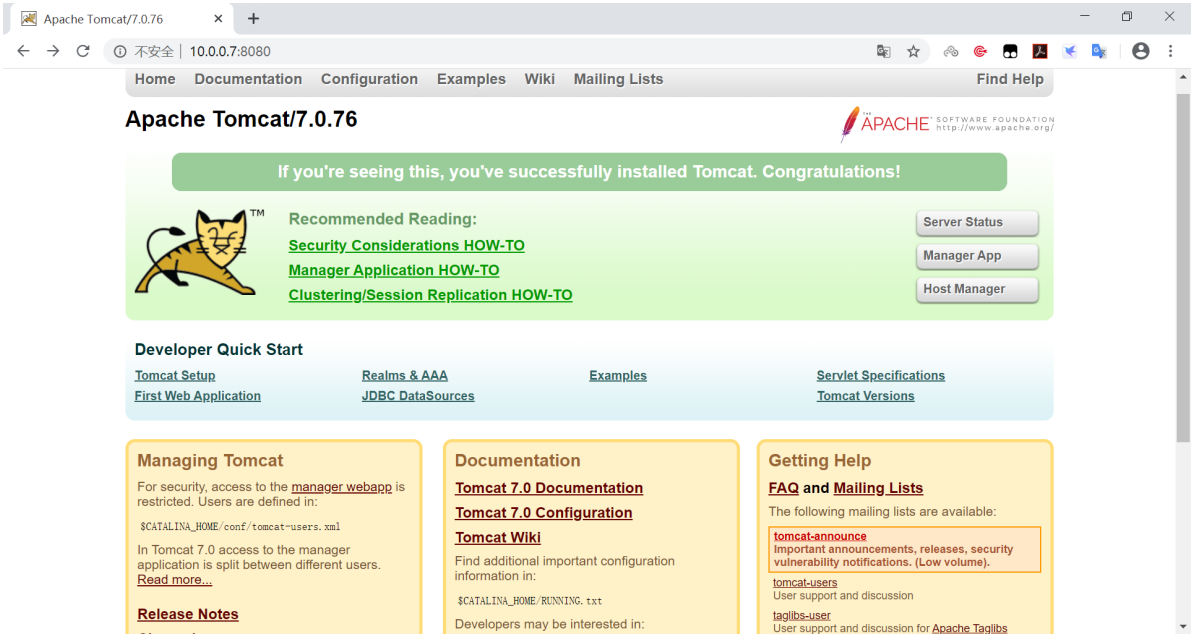
CentOS 7 yum仓库源中自带的Tomcat 7.0版本安装，此方式安装tomcat版本较低，不推荐

范例：在CentOS 7 上安装 tomcat

```
[root@centos7 ~]#yum list tomcat*
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base:
Available Packages
tomcat.noarch                                           7.0.76-9.el7_6
                          base
tomcat-admin-webapps.noarch                             7.0.76-9.el7_6
                          base
tomcat-docs-webapp.noarch                               7.0.76-9.el7_6
                          base
```

```
tomcat-el-2.2-api.noarch                                    7.0.76-9.el7_6
                          base
tomcat-javadoc.noarch                                       7.0.76-9.el7_6
                          base
tomcat-jsp-2.2-api.noarch                                   7.0.76-9.el7_6
                          base
tomcat-jsvc.noarch                                          7.0.76-9.el7_6
                          base
tomcat-lib.noarch                                           7.0.76-9.el7_6
                          base
tomcat-native.x86_64                                        1.2.21-1.el7
                          epel
tomcat-servlet-3.0-api.noarch                               7.0.76-9.el7_6
                          base
tomcat-webapps.noarch                                       7.0.76-9.el7_6
                          base
tomcatjss.noarch                                            7.2.1-8.el7_6
                           base
[root@centos7 ~]#yum -y install tomcat tomcat-webapps tomcat-admin-webapps
tomcat-docs-webapp
[root@centos7 ~]#systemctl enable --now tomcat
Created symlink from /etc/systemd/system/multi-user.target.wants/tomcat.service
to /usr/lib/systemd/system/tomcat.service.

[root@centos7 ~]#ss -ntl
State      Recv-Q Send-Q          Local Address:Port                         Peer
Address:Port
LISTEN     0      100                 127.0.0.1:25
       *:*
LISTEN     0      128                       *:22
       *:*
LISTEN     0      100                    [::1]:25
    [::]:*
LISTEN     0      1          [::ffff:127.0.0.1]:8005
    [::]:*
LISTEN     0      100                     [::]:8009
    [::]:*
LISTEN     0      100                     [::]:8080
    [::]:*
LISTEN     0      128                     [::]:22
    [::]:*

[root@centos7 ~]#getent passwd tomcat
tomcat:x:53:53:Apache Tomcat:/usr/share/tomcat:/sbin/nologin
[root@centos7 ~]#ps aux|grep tomcat
tomcat     1328  0.4 11.2 2298188 112004 ?      Ssl  21:32   0:11
/usr/lib/jvm/jre/bin/java -classpath
/usr/share/tomcat/bin/bootstrap.jar:/usr/share/tomcat/bin/tomcat-
juli.jar:/usr/share/java/commons-daemon.jar -Dcatalina.base=/usr/share/tomcat -
Dcatalina.home=/usr/share/tomcat -Djava.endorsed.dirs= -
Djava.io.tmpdir=/var/cachetomcat/temp -
Djava.util.logging.config.file=/usr/share/tomcat/conf/logging.properties -
Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager
org.apache.catalina.startup.Bootstrap start
root       1521  0.0  0.0 112712   960 pts/0    R+   22:15   0:00 grep --
color=auto tomcat
```

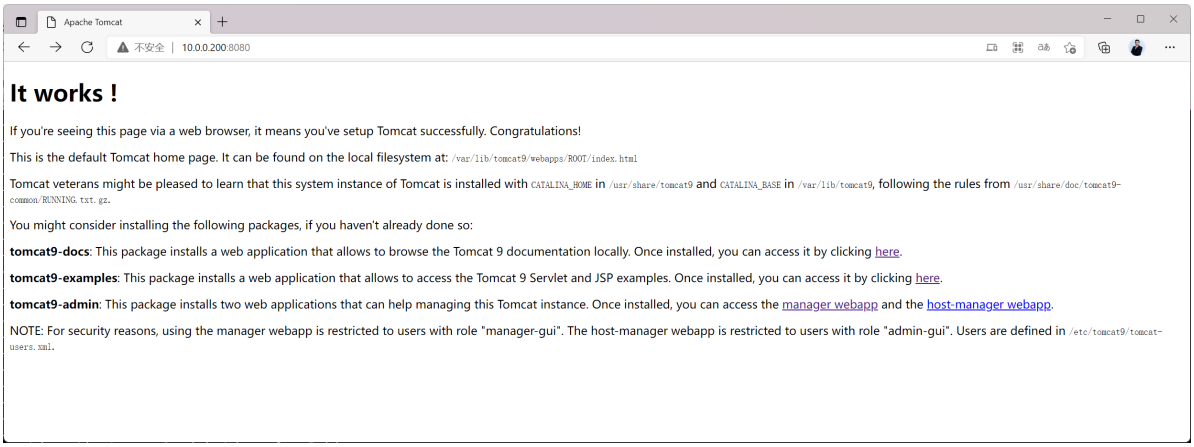
打开浏览器访问：http://tomcat:8080/



3.2.1.2 Ubuntu 包安装 tomcat

范例：Ubuntu20.04安装tomcat9

```
[root@ubuntu2004 ~]#apt  update
[root@ubuntu2004 ~]#apt list tomcat*
Listing... Done
tomcat9-admin/focal-security,focal-updates,now 9.0.31-1ubuntu0.1 all [installed]
tomcat9-common/focal-security,focal-updates,now 9.0.31-1ubuntu0.1 all
[installed,automatic]
tomcat9-docs/focal-security,focal-updates,now 9.0.31-1ubuntu0.1 all [installed]
tomcat9-examples/focal-security,focal-updates,now 9.0.31-1ubuntu0.1 all
[installed]
tomcat9-user/focal-security,focal-updates 9.0.31-1ubuntu0.1 all
tomcat9/focal-security,focal-updates,now 9.0.31-1ubuntu0.1 all [installed]
[root@ubuntu2004 ~]#apt -y install tomcat9 tomcat9-admin tomcat9-docs tomcat9-
examples
```



范例: Ubuntu180.4安装 tomcat8

```
[root@ubuntu1804 ~]#apt  update
[root@ubuntu1804 ~]#apt -y install tomcat8 tomcat8-admin tomcat8-docs
```

3.2.2 二进制安装 Tomcat

CentOS 7 的yum源的tomcat版本老旧,而CentOS8 yum源里无tomcat

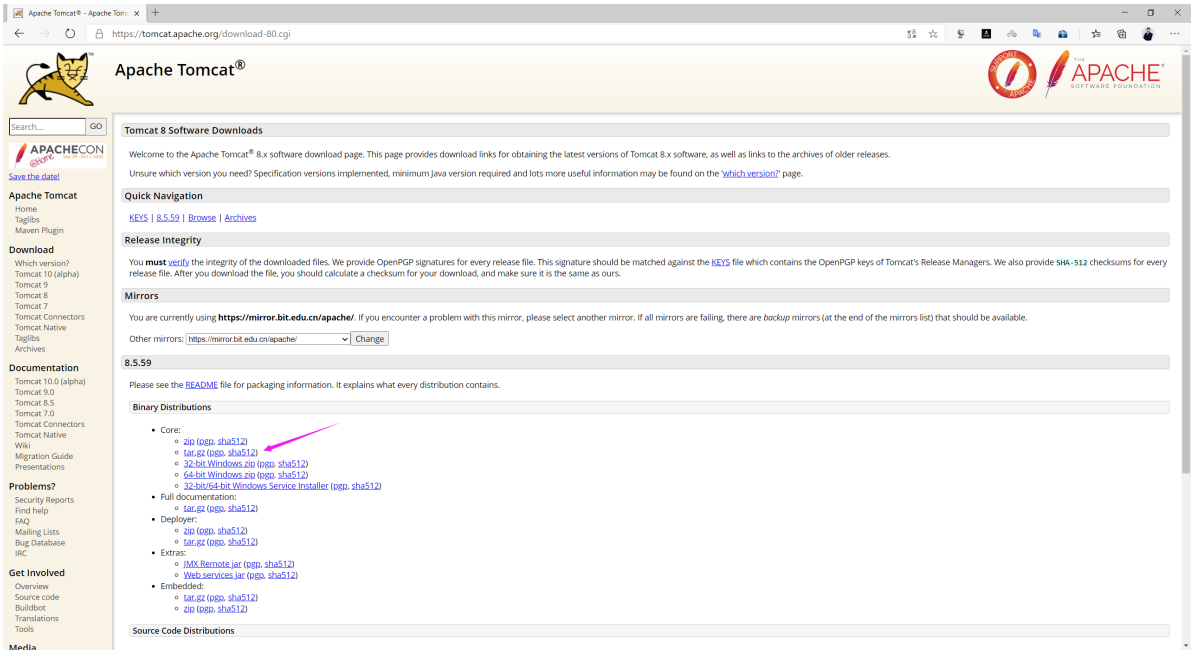
目前比较主流的Tomcat是8.5.X版本,推荐从Apache官网下载二进制tomcat包进行安装,此为生产常用方式

3.2.2.1 下载并安装

注意: 安装tomcat 前必须先部署JDK

官方和镜像站点下载:

```
https://tomcat.apache.org/download-80.cgi
https://mirrors.tuna.tsinghua.edu.cn/apache/tomcat/
```



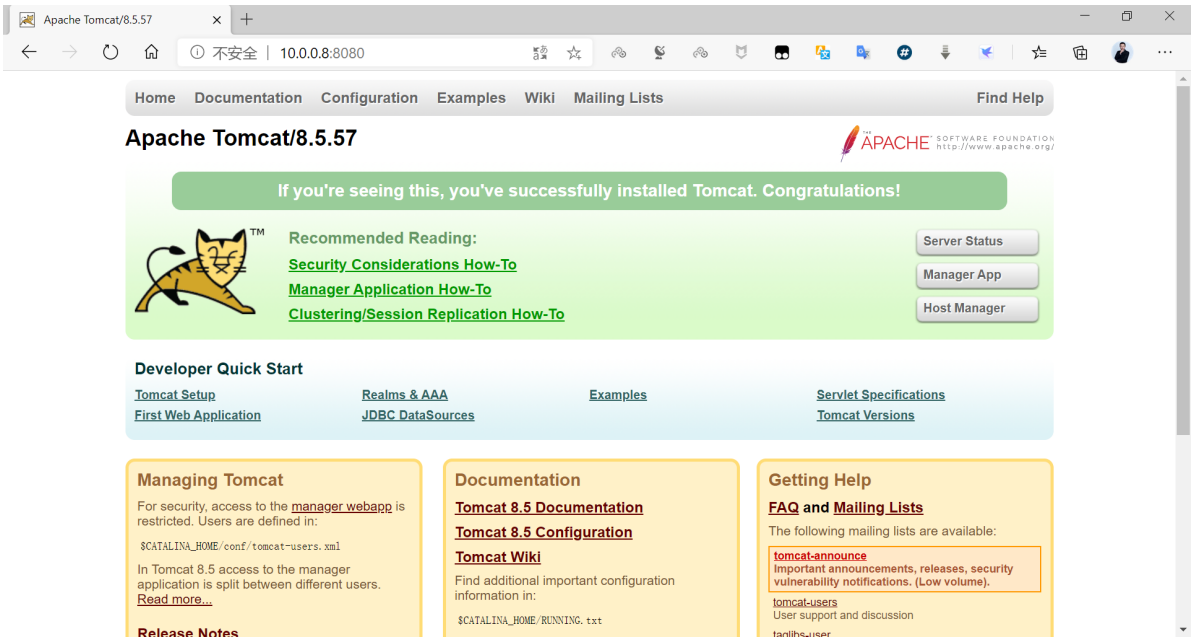

```
#官网或镜像网站下载:
[root@centos8 ~]#wget http://mirrors.tuna.tsinghua.edu.cn/apache/tomcat/tomcat-8/v8.5.50/bin/apache-tomcat-8.5.50.tar.gz

[root@centos8 ~]#tar xf apache-tomcat-8.5.50.tar.gz  -C /usr/local/
[root@centos8 ~]#cd /usr/local/
[root@centos8 local]#ln -s apache-tomcat-8.5.50/ tomcat

#指定PATH变量
[root@centos8 ~]#echo 'PATH=/usr/local/tomcat/bin:$PATH' > /etc/profile.d/tomcat.sh
[root@centos8 ~]#. /etc/profile.d/tomcat.sh
[root@centos8 ~]#echo $PATH
/usr/local/tomcat/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/usr/local/jdk/bin:/root/bin

#查看当前变量设置和命令用法
[root@centos8 ~]#catalina.sh
Using CATALINA_BASE:   /usr/local/tomcat
Using CATALINA_HOME:   /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME:        /usr/local/jdk
```

```
Using CLASSPATH:
/usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar
Usage: catalina.sh ( commands ... )
commands:
  debug             Start Catalina in a debugger
  debug -security   Debug Catalina with a security manager
  jpda start        Start Catalina under JPDA debugger
  run               Start Catalina in the current window
  run -security     Start in the current window with security manager
  start             Start Catalina in a separate window
  start -security   Start in a separate window with security manager
  stop              Stop Catalina, waiting up to 5 seconds for the process to
end
  stop n            Stop Catalina, waiting up to n seconds for the process to
end
  stop -force       Stop Catalina, wait up to 5 seconds and then use kill -KILL
if still running
  stop n -force     Stop Catalina, wait up to n seconds and then use kill -KILL
if still running
  configtest        Run a basic syntax check on server.xml - check exit code for
result
  version           What version of tomcat are you running?
Note: Waiting for the process to end and use of the -force option require that
$CATALINA_PID is defined

#查看环境变量和版本信息
[root@centos8 ~]#catalina.sh version
Using CATALINA_BASE:   /usr/local/tomcat
Using CATALINA_HOME:   /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME:        /usr/local/jdk/jre
Using CLASSPATH:
/usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar
Using CATALINA_OPTS:
Server version: Apache Tomcat/8.5.59
Server built:   Oct 6 2020 16:57:18 UTC
Server number:  8.5.59.0
OS Name:        Linux
OS Version:     4.18.0-193.el8.x86_64
Architecture:   amd64
JVM Version:    1.8.0_261-b12
JVM Vendor:     Oracle Corporation

#启动tomcat
[root@centos8 ~]#startup.sh
Using CATALINA_BASE:   /usr/local/tomcat
Using CATALINA_HOME:   /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME:        /usr/local/jdk/jre
Using CLASSPATH:
/usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar
Tomcat started.

#查看端口
[root@centos8 ~]#ss -ntl
State       Recv-Q Send-Q        Local Address:Port     Peer Address:Port
LISTEN      0      128                 0.0.0.0:22            0.0.0.0:*
LISTEN      0      100                       *:8080                *:*
```

```
LISTEN      0      128                    [::]:22                [::]:*
LISTEN      0      1         [::ffff:127.0.0.1]:8005                 *:*
LISTEN      0      100                     *:8009                 *:*

#查看进程是以root启动的
[root@centos8 ~]#ps aux|grep tomcat
root       12994 34.1  9.4 2155140 76912 pts/0   Sl   22:38   0:02
/usr/local/jdk/jre/bin/java -
Djava.util.logging.config.file=/usr/local/tomcat/conf/logging.properties -
Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -
Djdk.tls.ephemeralDHKeySize=2048 -
Djava.protocol.handler.pkgs=org.apache.catalina.webresources -
Dorg.apache.catalina.security.SecurityListener.UMASK=0027 -
Dignore.endorsed.dirs= -classpath
/usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar -
Dcatalina.base=/usr/local/tomcat -Dcatalina.home=/usr/local/tomcat -
Djava.io.tmpdir=/usr/local/tomcat/temp org.apache.catalina.startup.Bootstrap
start
root       13039  0.0  0.1  12108  1076 pts/0    R+   22:38   0:00 grep --
color=auto tomcat

#关闭tomcat
[root@centos8 ~]#shutdown.sh
Using CATALINA_BASE:   /usr/local/tomcat
Using CATALINA_HOME:   /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME:        /usr/local/jdk/jre
Using CLASSPATH:
/usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar
#或者以下也可以,指定10s后停止,默认5s
[root@centos8 ~]#catalina.sh  stop 10

[root@centos8 ~]#ss -ntl
State        Recv-Q        Send-Q                Local Address:Port
    Peer Address:Port
LISTEN       0             128                         0.0.0.0:22
         0.0.0.0:*
LISTEN       0             128                            [::]:22
            [::]:*

#再次用不同方式启动tomcat
[root@centos8 ~]#catalina.sh start
Using CATALINA_BASE:   /usr/local/tomcat
Using CATALINA_HOME:   /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME:        /usr/local/jdk/jre
Using CLASSPATH:
/usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar
Tomcat started.
[root@centos8 ~]#ss -ntl
State       Recv-Q        Send-Q                    Local Address:Port
    Peer Address:Port      LISTEN      0             128
     0.0.0.0:22                       0.0.0.0:*
LISTEN      0             100                              *:8080
                 *:*
LISTEN      0             128                           [::]:22
              [::]:*
```

```
LISTEN      0           1                    [::ffff:127.0.0.1]:8005
                *:*
LISTEN      0           100                        *:8009
                *:*

#再次用不同方式关闭tomcat
[root@centos8 ~]#catalina.sh stop
Using CATALINA_BASE:   /usr/local/tomcat
Using CATALINA_HOME:   /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME:        /usr/local/jdk/jre
Using CLASSPATH:
/usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar
```

打开浏览器访问：http://tomcat:8080/，正常可以看到以下界面



扩展知识：tomcat 和 catalina 关系

Tomcat的servlet容器在4.X版本中被Craig McClanahan(Apache Struts项目的创始人,也是Tomcat的 Catalina 的架构师)重新设计为Catalina.即Catalina就是servlet容器。

Tomcat的核心分为3个部分：
（1）Web容器:处理静态页面；
（2）JSP容器:把jsp页面翻译成一般的 servlet
（3）catalina: 是一个servlet容器,用于处理servlet

Catalina是美国西海岸靠近洛杉矶22英里的一个小岛，因为其风景秀丽而著名，曾被评为全美最漂亮的小岛。Servlet运行模块的最早开发者Craig McClanahan因为喜欢Catalina岛,故以Catalina命名他所开这个模块，另外在开发的早期阶段，Tomcat是被搭建在一个叫Avalon的服务器框架上，而Avalon则是Catalina岛上的一个小镇的名字，于是想一个与小镇名字相关联的单词也是自然而然。设计者估计是想把tomcat设计成最美的轻量级容器吧。下图为该小岛。

3.2.2.2 配置 tomcat自启动的 service 文件

```
#创建tomcat专用帐户
[root@centos8 ~]#useradd -r -s /sbin/nologin tomcat

#准备service文件中相关环境文件
[root@centos8 ~]#vim /usr/local/tomcat/conf/tomcat.conf
[root@centos8 ~]#cat /usr/local/tomcat/conf/tomcat.conf
#两个变量至少设置一项才能启动 tomcat
JAVA_HOME=/usr/local/jdk
#JRE_HOME=/usr/local/jdk/jre
#如果不指定上面变量,/var/log/messages文件中会出现下面无法启动错误提示
Mar 15 14:30:09 centos8 startup.sh[1530]: Neither the JAVA_HOME nor the JRE_HOME
environment variable is defined
Mar 15 14:30:09 centos8 startup.sh[1530]: At least one of these environment
variable is needed to run this program


[root@centos8 ~]#chown -R tomcat.tomcat /usr/local/tomcat/

#创建tomcat.service文件
[root@centos8 ~]#vim /lib/systemd/system/tomcat.service
[root@centos8 ~]#cat /lib/systemd/system/tomcat.service
[Unit]
Description=Tomcat
#After=syslog.target network.target remote-fs.target nss-lookup.target
After=syslog.target network.target

[Service]
Type=forking
#以下二选一
EnvironmentFile=/usr/local/tomcat/conf/tomcat.conf
#或者，如果没有创建上面的/usr/local/tomcat/conf/tomcat.conf文件，可以加下面一行也可
Environment=JAVA_HOME=/usr/local/jdk
```

```
ExecStart=/usr/local/tomcat/bin/startup.sh
ExecStop=/usr/local/tomcat/bin/shutdown.sh
PrivateTmp=true
User=tomcat
Group=tomcat

[Install]
WantedBy=multi-user.target

[root@centos8 ~]#systemctl daemon-reload
[root@centos8 ~]#systemctl enable --now tomcat
Created symlink /etc/systemd/system/multi-user.target.wants/tomcat.service →
/usr/lib/systemd/system/tomcat.service.
[root@centos8 ~]#systemctl status tomcat
● tomcat.service - Tomcat
   Loaded: loaded (/usr/lib/systemd/system/tomcat.service; enabled; vendor
preset: disabled)
   Active: active (running) since Sat 2020-02-08 23:37:02 CST; 5s ago
  Process: 14312 ExecStart=/usr/local/tomcat/bin/startup.sh (code=exited,
status=0/SUCCESS)
 Main PID: 14320 (java)
    Tasks: 43 (limit: 4895)
   Memory: 64.2M
   CGroup: /system.slice/tomcat.service
           └─14320 /usr/local/jdk/jre/bin/java -
Djava.util.logging.config.file=/usr/local/tomcat/conf/logging>

Feb 08 23:37:02 centos8.localdomain systemd[1]: Starting Tomcat...
Feb 08 23:37:02 centos8.localdomain systemd[1]: Started Tomcat.

#查看日志
[root@centos8 ~]#tail  /var/log/messages
Mar 15 14:32:13 centos8 systemd[1]: Reloading.
Mar 15 14:32:23 centos8 systemd[1]: Starting Tomcat...
Mar 15 14:32:23 centos8 startup.sh[1575]: Tomcat started.
Mar 15 14:32:23 centos8 systemd[1]: Started Tomcat.
```

3.2.3 实战案例： 一键安装 tomcat 脚本

```
[root@ubuntu1804 ~]#cat install_tomcat.sh
#!/bin/bash
#
#**************************************************************************
#Author:            wangxiaochun
#QQ:                29308620
#Date:              2021-03-15
#FileName:          install_tomcat.sh
#URL:               http://www.wangxiaochun.com
#Description:       The test script
#Copyright (C):     2021 All rights reserved
#**************************************************************************

JDK_FILE="jdk-11.0.14_linux-x64_bin.tar.gz"
#JDK_FILE="jdk-8u281-linux-x64.tar.gz"
TOMCAT_FILE="apache-tomcat-9.0.59.tar.gz"
#TOMCAT_FILE="apache-tomcat-8.5.64.tar.gz"
```

```
JDK_DIR="/usr/local"
TOMCAT_DIR="/usr/local"
DIR=`pwd`

color () {
    RES_COL=60
    MOVE_TO_COL="echo -en \\033[${RES_COL}G"
    SETCOLOR_SUCCESS="echo -en \\033[1;32m"
    SETCOLOR_FAILURE="echo -en \\033[1;31m"
    SETCOLOR_WARNING="echo -en \\033[1;33m"
    SETCOLOR_NORMAL="echo -en \E[0m"
    echo -n "$2" && $MOVE_TO_COL
    echo -n "["
    if [ $1 = "success" -o $1 = "0" ] ;then
        ${SETCOLOR_SUCCESS}
        echo -n $"  OK  "
    elif [ $1 = "failure" -o $1 = "1"  ] ;then
        ${SETCOLOR_FAILURE}
        echo -n $"FAILED"
    else
        ${SETCOLOR_WARNING}
        echo -n $"WARNING"
    fi
    ${SETCOLOR_NORMAL}
    echo -n "]"
    echo

}



install_jdk(){
if !  [  -f "$DIR/$JDK_FILE" ];then
    color 1 "$JDK_FILE 文件不存在"
    exit;
elif [ -d $JDK_DIR/jdk ];then
    color 1  "JDK 已经安装"
    exit
else
    [ -d "$JDK_DIR" ] || mkdir -pv $JDK_DIR
fi
tar xvf $DIR/$JDK_FILE  -C $JDK_DIR
cd  $JDK_DIR && ln -s jdk* jdk

cat >  /etc/profile.d/jdk.sh <<EOF
export JAVA_HOME=$JDK_DIR/jdk
export PATH=\$PATH:\$JAVA_HOME/bin
#export JRE_HOME=\$JAVA_HOME/jre
#export CLASSPATH=\$JAVA_HOME/lib/:\$JRE_HOME/lib/
EOF
.  /etc/profile.d/jdk.sh
java -version && color 0 "JDK 安装完成" || { color 1  "JDK 安装失败" ; exit; }

}

install_tomcat(){
if ! [ -f "$DIR/$TOMCAT_FILE" ];then
    color 1 "$TOMCAT_FILE 文件不存在"
```

```
    exit;
elif [ -d $TOMCAT_DIR/tomcat ];then
    color 1 "TOMCAT 已经安装"
    exit
else
    [ -d "$TOMCAT_DIR" ] || mkdir -pv $TOMCAT_DIR
fi
tar xf $DIR/$TOMCAT_FILE -C $TOMCAT_DIR
cd  $TOMCAT_DIR && ln -s apache-tomcat-*/  tomcat
echo "PATH=$TOMCAT_DIR/tomcat/bin:"'$PATH' > /etc/profile.d/tomcat.sh
id tomcat &> /dev/null || useradd -r -s /sbin/nologin tomcat

cat > $TOMCAT_DIR/tomcat/conf/tomcat.conf <<EOF
JAVA_HOME=$JDK_DIR/jdk
EOF

chown -R tomcat.tomcat $TOMCAT_DIR/tomcat/

cat > /lib/systemd/system/tomcat.service  <<EOF
[Unit]
Description=Tomcat
#After=syslog.target network.target remote-fs.target nss-lookup.target
After=syslog.target network.target

[Service]
Type=forking
EnvironmentFile=$TOMCAT_DIR/tomcat/conf/tomcat.conf
ExecStart=$TOMCAT_DIR/tomcat/bin/startup.sh
ExecStop=$TOMCAT_DIR/tomcat/bin/shutdown.sh
RestartSec=3
PrivateTmp=true
User=tomcat
Group=tomcat

[Install]
WantedBy=multi-user.target
EOF
systemctl daemon-reload
systemctl enable --now tomcat.service &> /dev/null
systemctl is-active tomcat.service &> /dev/null &&  color 0 "TOMCAT 安装完成" || {
color 1 "TOMCAT 安装失败" ; exit; }

}

install_jdk

install_tomcat
```

3.3 tomcat的文件结构和组成

3.3.1 目录结构

目录	说明
bin	服务启动、停止等相关程序和文件
conf	配置文件
lib	库目录
logs	日志目录
webapps	应用程序，应用部署目录
work	jsp编译后的结果文件，建议提前预热访问，升级应用后，删除此目录数据才能更新

范例：查看tomcat相关目录和文件

```
[root@centos8 tomcat]#pwd
/usr/local/tomcat
[root@centos8 tomcat]#ls
bin           conf             lib      logs    README.md      RUNNING.txt
webapps
BUILDING.txt  CONTRIBUTING.md  LICENSE  NOTICE  RELEASE-NOTES  temp         work
[root@centos8 tomcat]#ls bin
bootstrap.jar       ciphers.sh                   daemon.sh         shutdown.bat
    tomcat-native.tar.gz
catalina.bat        commons-daemon.jar           digest.bat        shutdown.sh
   tool-wrapper.bat
catalina.sh         commons-daemon-native.tar.gz  digest.sh         startup.bat
   tool-wrapper.sh
catalina-tasks.xml  configtest.bat               setclasspath.bat  startup.sh
    version.bat
ciphers.bat         configtest.sh                setclasspath.sh   tomcat-
juli.jar  version.sh
[root@centos8 tomcat]#ls conf
Catalina             context.xml           logging.properties  tomcat-users.xml
catalina.policy      jaspic-providers.xml  server.xml          tomcat-users.xsd
catalina.properties  jaspic-providers.xsd  tomcat.conf         web.xml
[root@centos8 tomcat]#ls lib
annotations-api.jar       ecj-4.6.3.jar   servlet-api.jar     tomcat-i18n-fr.jar
    tomcat-jni.jar
catalina-ant.jar          el-api.jar      tomcat-api.jar      tomcat-i18n-ja.jar
    tomcat-util.jar
catalina-ha.jar           jasper-el.jar   tomcat-coyote.jar   tomcat-i18n-ko.jar
    tomcat-util-scan.jar
catalina.jar              jasper.jar      tomcat-dbcp.jar     tomcat-i18n-ru.jar
    tomcat-websocket.jar
catalina-storeconfig.jar  jaspic-api.jar  tomcat-i18n-de.jar  tomcat-i18n-zh-
CN.jar  websocket-api.jar
catalina-tribes.jar       jsp-api.jar     tomcat-i18n-es.jar  tomcat-jdbc.jar
[root@centos8 tomcat]#ls logs
catalina.2020-02-09.log  host-manager.2020-02-09.log  localhost_access_log.2020-
02-09.txt
catalina.out             localhost.2020-02-09.log     manager.2020-02-09.log
[root@centos8 tomcat]#ls webapps/
docs  examples  host-manager  manager  ROOT
```

```
[root@centos8 tomcat]#ls work/
Catalina
[root@centos8 tomcat]#ls work/Catalina/
localhost
[root@centos8 tomcat]#ls work/Catalina/localhost/
docs  examples  host-manager  manager  ROOT
[root@centos8 tomcat]#ll -i work/Catalina/localhost/
total 0
 68039883 drwxr-x--- 2 tomcat tomcat 6 Feb  9 11:02 docs
135579640 drwxr-x--- 2 tomcat tomcat 6 Feb  9 11:02 examples
202681358 drwxr-x--- 2 tomcat tomcat 6 Feb  9 11:02 host-manager
   571365 drwxr-x--- 2 tomcat tomcat 6 Feb  9 11:02 manager
   571364 drwxr-x--- 2 tomcat tomcat 6 Feb  9 11:02 ROOT
[root@centos8 tomcat]#ll -i webapps/
total 4
202681088 drwxr-x--- 15 tomcat tomcat 4096 Feb  9 11:02 docs
202681094 drwxr-x---  6 tomcat tomcat   83 Feb  9 11:02 examples
   571165 drwxr-x---  5 tomcat tomcat   87 Feb  9 11:02 host-manager
 68039687 drwxr-x---  5 tomcat tomcat  103 Feb  9 11:02 manager
 68039663 drwxr-x---  3 tomcat tomcat  283 Feb  9 11:02 ROOT

[root@centos8 tomcat]#tree work/Catalina/localhost/
work/Catalina/localhost/
├── docs
├── examples
├── host-manager
├── manager
└── ROOT

5 directories, 0 files


[root@centos8 tomcat]#curl http://10.0.0.8:8080/
#当访问过后，work目录中生成新文件
[root@centos8 tomcat]#tree work/Catalina/localhost/
work/Catalina/localhost/
├── docs
├── examples
├── host-manager
├── manager
└── ROOT
    └── org
        └── apache
            └── jsp
                ├── index_jsp.class #字节码文件
                └── index_jsp.java  #servlet文件

8 directories, 2 files

#tomcat会自动的将jsp文件生成java源文件，再编译成class文件
[root@centos8 tomcat]#less
work/Catalina/localhost/ROOT/org/apache/jsp/index_jsp.java
/*
 * Generated by the Jasper component of Apache Tomcat
 * Version: Apache Tomcat/8.5.50
 * Generated at: 2020-02-09 03:20:20 UTC
 * Note: The last modified time of this file was set to
 *       the last modified time of the source file after
```

```
 *       generation to assist with modification tracking.
 */
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class index_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent,
                 org.apache.jasper.runtime.JspSourceImports {

  private static final javax.servlet.jsp.JspFactory _jspxFactory =
          javax.servlet.jsp.JspFactory.getDefaultFactory();

  private static java.util.Map<java.lang.String,java.lang.Long>
_jspx_dependants;
```

3.3.2 配置文件

3.3.2.1 配置文件说明

官方帮助文档：http://tomcat.apache.org/tomcat-8.5-doc/index.html

在tomcat安装目录下的 conf 子目录中，有以下的 tomcat 的配置文件

文件名	说明
server.xml	主配置文件
web.xml	每个webapp只有“部署”后才能被访问，它的部署方式通常由web.xml进行定义，其存放位置为WEB-INF/目录中；此文件为所有的webapps提供默认部署相关的配置,每个web应用也可以使用专用配置文件,来覆盖全局文件
context.xml	用于定义所有web应用均需加载的Context配置，此文件为所有的webapps提供默认配置，每个web应用也可以使用自己专用的配置，它通常由专用的配置文件context.xml来定义，其存放位置为WEB-INF/目录中，覆盖全局的文件
tomcat-users.xml	用户认证的账号和密码文件
catalina.policy	当使用security选项启动tomcat时，用于为tomcat设置安全策略
catalina.properties	Tomcat 环境变量的配置，用于设定类加载器路径，以及一些与JVM调优相关参数
logging.properties	Tomcat 日志系统相关的配置，可以修改日志级别和日志路径等

注意：配置文件大小写敏感

范例：查看配置文件

```
[root@centos8 conf]#pwd
/usr/local/tomcat/conf
[root@centos8 conf]#ls
Catalina             context.xml           logging.properties  tomcat-users.xml
catalina.policy      jaspic-providers.xml  server.xml          tomcat-users.xsd
```

```
catalina.properties  jaspic-providers.xsd  tomcat.conf         web.xml
[root@centos8 conf]#wc -l server.xml web.xml context.xml tomcat-users.xml
catalina.policy  catalina.properties  logging.properties
   167 server.xml
  4726 web.xml
    30 context.xml
    44 tomcat-users.xml
   271 catalina.policy
   214 catalina.properties
    75 logging.properties
  5527 total
[root@centos8 conf]#
```

范例: 主要配置文件内容

```
[root@centos8 ~]#grep -v '\-\-' /usr/local/tomcat/conf/server.xml
<?xml version="1.0" encoding="UTF-8"?>
  Licensed to the Apache Software Foundation (ASF) under one or more
  contributor license agreements.  See the NOTICE file distributed with
  this work for additional information regarding copyright ownership.
  The ASF licenses this file to You under the Apache License, Version 2.0
  (the "License"); you may not use this file except in compliance with
  the License.  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
     define subcomponents such as "Valves" at this level.
     Documentation at /docs/config/server.html
<Server port="8005" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.startup.VersionLoggerListener" />
  <Listener className="org.apache.catalina.security.SecurityListener" />
  <Listener className="org.apache.catalina.core.AprLifecycleListener"
SSLEngine="on" />
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener"
/>
  <Listener
className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
  <Listener
className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

      Documentation at /docs/jndi-resources-howto.html
  <GlobalNamingResources>
        UserDatabaseRealm to authenticate users
    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
              description="User database that can be updated and saved"
              factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />
  </GlobalNamingResources>

      a single "Container" Note:  A "Service" is not itself a "Container",
      so you may not define subcomponents such as "Valves" at this level.
```

```
      Documentation at /docs/config/service.html
  <Service name="Catalina">

    <Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
        maxThreads="150" minSpareThreads="4"/>


        and responses are returned. Documentation at :
        Java HTTP Connector: /docs/config/http.html
        Java AJP  Connector: /docs/config/ajp.html
        APR (HTTP/AJP) Connector: /docs/apr.html
        Define a non-SSL/TLS HTTP/1.1 Connector on port 8080
    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
    <Connector executor="tomcatThreadPool"
               port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
        This connector uses the NIO implementation. The default
        SSLImplementation will depend on the presence of the APR/native
        library and the useOpenSSL attribute of the
        AprLifecycleListener.
        Either JSSE or OpenSSL style configuration may be used regardless of
        the SSLImplementation selected. JSSE style configuration is used below.
    <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
               maxThreads="150" SSLEnabled="true">
        <SSLHostConfig>
            <Certificate certificateKeystoreFile="conf/localhost-rsa.jks"
                         type="RSA" />
        </SSLHostConfig>
    </Connector>
        This connector uses the APR/native implementation which always uses
        OpenSSL for TLS.
        Either JSSE or OpenSSL style configuration may be used. OpenSSL style
        configuration is used below.
    <Connector port="8443" protocol="org.apache.coyote.http11.Http11AprProtocol"
               maxThreads="150" SSLEnabled="true" >
        <UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol" />
        <SSLHostConfig>
            <Certificate certificateKeyFile="conf/localhost-rsa-key.pem"
                         certificateFile="conf/localhost-rsa-cert.pem"
                         certificateChainFile="conf/localhost-rsa-chain.pem"
                         type="RSA" />
        </SSLHostConfig>
    </Connector>

    <Connector protocol="AJP/1.3"
               address="::1"
               port="8009"
               redirectPort="8443" />

        every request.  The Engine implementation for Tomcat stand alone
        analyzes the HTTP headers included with the request, and passes them
        on to the appropriate Host (virtual host).

    <Engine name="Catalina" defaultHost="localhost" jvmRoute="jvm1">
    <Engine name="Catalina" defaultHost="localhost">
```

```
          /docs/cluster-howto.html  (simple how to)
      <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>

      <Realm className="org.apache.catalina.realm.LockOutRealm">
             resources under the key "UserDatabase".  Any edits
             that are performed against this UserDatabase are immediately
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
               resourceName="UserDatabase"/>
      </Realm>

      <Host name="localhost"  appBase="webapps"
            unpackWARs="true" autoDeploy="true">

        <Valve className="org.apache.catalina.authenticator.SingleSignOn" />

             Documentation at: /docs/config/valve.html
        <Valve className="org.apache.catalina.valves.AccessLogValve"
directory="logs"
               prefix="localhost_access_log" suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />

      </Host>
    </Engine>
  </Service>
</Server>

[root@centos8 ~]#grep -v '\-\-' /usr/local/tomcat/conf/context.xml
<?xml version="1.0" encoding="UTF-8"?>
  Licensed to the Apache Software Foundation (ASF) under one or more
  contributor license agreements.  See the NOTICE file distributed with
  this work for additional information regarding copyright ownership.
  The ASF licenses this file to You under the Apache License, Version 2.0
  (the "License"); you may not use this file except in compliance with
  the License.  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
<Context>

    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>

    <Manager pathname="" />
</Context>

[root@centos8 ~]#grep -v '\-\-' /usr/local/tomcat/conf/tomcat-users.xml
<?xml version="1.0" encoding="UTF-8"?>
  Licensed to the Apache Software Foundation (ASF) under one or more
  contributor license agreements.  See the NOTICE file distributed with
  this work for additional information regarding copyright ownership.
  The ASF licenses this file to You under the Apache License, Version 2.0
  (the "License"); you may not use this file except in compliance with
```

```
  the License.  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
<tomcat-users xmlns="http://tomcat.apache.org/xml"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
              version="1.0">
  NOTE:  By default, no user is included in the "manager-gui" role required
  to operate the "/manager/html" web application.  If you wish to use this app,
  you must define such a user - the username and password are arbitrary. It is
  strongly recommended that you do NOT use one of the users in the commented out
  section below since they are intended for use with the examples web
  application.
  NOTE:  The sample user and role entries below are intended for use with the
  examples web application. They are wrapped in a comment and thus are ignored
  when reading this file. If you wish to configure these users for use with the
  examples web application, do not forget to remove the <!.. ..> that surrounds
  them. You will also need to set the passwords to something appropriate.
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
  <user username="both" password="<must-be-changed>" roles="tomcat,role1"/>
  <user username="role1" password="<must-be-changed>" roles="role1"/>
</tomcat-users>

[root@centos8 ~]#cat /usr/local/tomcat/conf/tomcat.conf
JAVA_HOME=/usr/local/jdk
```

3.3.2.2 日志文件

参考文档: https://cwiki.apache.org/confluence/display/TOMCAT/Logging

日志格式: https://tomcat.apache.org/tomcat-9.0-doc/config/valve.html#Access_Logging

```
%a - Remote IP address
%A - Local IP address
%b - Bytes sent, excluding HTTP headers, or '-' if zero
%B - Bytes sent, excluding HTTP headers
%h - Remote host name (or IP address if enableLookups for the connector is
false)
%H - Request protocol
%l - Remote logical username from identd (always returns '-')
%m - Request method (GET, POST, etc.)
%p - Local port on which this request was received. See also %{xxx}p below.
%q - Query string (prepended with a '?' if it exists)
%r - First line of the request (method and request URI)
%s - HTTP status code of the response
%S - User session ID
%t - Date and time, in Common Log Format
%u - Remote user that was authenticated (if any), else '-'
%U - Requested URL path
```

```
%v - Local server name
%D - Time taken to process the request in millis. Note: In httpd %D is
microseconds. Behaviour will be aligned to httpd in Tomcat 10 onwards.
%T - Time taken to process the request, in seconds. Note: This value has
millisecond resolution whereas in httpd it has second resolution. Behaviour will
be align to httpd in Tomcat 10 onwards.
%F - Time taken to commit the response, in millis
%I - Current request thread name (can compare later with stacktraces)
%X - Connection status when response is completed:
X = Connection aborted before the response completed.
+ = Connection may be kept alive after the response is sent.
- = Connection will be closed after the response is sent.
There is also support to write information incoming or outgoing headers, cookies,
session or request attributes and special timestamp formats. It is modeled after
the Apache HTTP Server log configuration syntax. Each of them can be used
multiple times with different xxx keys:

%{xxx}i write value of incoming header with name xxx
%{xxx}o write value of outgoing header with name xxx
%{xxx}c write value of cookie with name xxx
%{xxx}r write value of ServletRequest attribute with name xxx
%{xxx}s write value of HttpSession attribute with name xxx
%{xxx}p write local (server) port (xxx==local) or remote (client) port
(xxx=remote)
%{xxx}t write timestamp at the end of the request formatted using the enhanced
SimpleDateFormat pattern xxx
```

范例: tomcat中的日志文件

```
[root@tomcat ~]#cat /usr/local/tomcat/conf/logging.properties
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

handlers = 1catalina.org.apache.juli.AsyncFileHandler,
2localhost.org.apache.juli.AsyncFileHandler,
3manager.org.apache.juli.AsyncFileHandler, 4host-
manager.org.apache.juli.AsyncFileHandler, java.util.logging.ConsoleHandler

.handlers = 1catalina.org.apache.juli.AsyncFileHandler,
java.util.logging.ConsoleHandler

############################################################
# Handler specific properties.
# Describes specific configuration info for Handlers.
############################################################
```

```
1catalina.org.apache.juli.AsyncFileHandler.level = FINE
1catalina.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
1catalina.org.apache.juli.AsyncFileHandler.prefix = catalina.
1catalina.org.apache.juli.AsyncFileHandler.encoding = UTF-8

2localhost.org.apache.juli.AsyncFileHandler.level = FINE
2localhost.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
2localhost.org.apache.juli.AsyncFileHandler.prefix = localhost.
2localhost.org.apache.juli.AsyncFileHandler.encoding = UTF-8

3manager.org.apache.juli.AsyncFileHandler.level = FINE
3manager.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
3manager.org.apache.juli.AsyncFileHandler.prefix = manager.
3manager.org.apache.juli.AsyncFileHandler.encoding = UTF-8

4host-manager.org.apache.juli.AsyncFileHandler.level = FINE
4host-manager.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
4host-manager.org.apache.juli.AsyncFileHandler.prefix = host-manager.
4host-manager.org.apache.juli.AsyncFileHandler.encoding = UTF-8

java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = org.apache.juli.OneLineFormatter
java.util.logging.ConsoleHandler.encoding = UTF-8


############################################################
# Facility specific properties.
# Provides extra control for each logger.
############################################################

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].level = INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].handlers =
2localhost.org.apache.juli.AsyncFileHandler

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager].level =
INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager].handlers
= 3manager.org.apache.juli.AsyncFileHandler

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/host-
manager].level = INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/host-
manager].handlers = 4host-manager.org.apache.juli.AsyncFileHandler

# For example, set the org.apache.catalina.util.LifecycleBase logger to log
# each component that extends LifecycleBase changing state:
#org.apache.catalina.util.LifecycleBase.level = FINE

# To see debug messages in TldLocationsCache, uncomment the following line:
#org.apache.jasper.compiler.TldLocationsCache.level = FINE

# To see debug messages for HTTP/2 handling, uncomment the following line:
#org.apache.coyote.http2.level = FINE

# To see debug messages for WebSocket handling, uncomment the following line:
#org.apache.tomcat.websocket.level = FINE
```

```
[root@centos8 ~]#ls /usr/local/tomcat/logs/ -1
catalina.2020-07-14.log  #tomcat服务日志
catalina.out               #tomcat服务日志
host-manager.2020-07-14.log   #host manager管理日志
localhost.2020-07-14.log       #默认主机日志
localhost_access_log.2020-07-14.txt  ##默认主机访问日志
manager.2020-07-14.log        #manager 管理日志
```

范例: tomcat的访问日志格式

```
#查看访问日志格式
[root@centos8 ~]#tail /usr/local/tomcat/conf/server.xml
             Documentation at: /docs/config/valve.html
             Note: The pattern used is equivalent to using pattern="common" -->
        <Valve className="org.apache.catalina.valves.AccessLogValve"
directory="logs"
               prefix="localhost_access_log" suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />   #说明: &amp;quot;在html
中表示双引号"符号

      </Host>
    </Engine>
  </Service>
</Server>

#查看访问日志
[root@centos8 ~]#tail /usr/local/tomcat/logs/localhost_access_log.2020-07-14.txt

10.0.0.1 - - [14/Jul/2020:09:08:46 +0800] "GET / HTTP/1.1" 200 11215
10.0.0.1 - - [14/Jul/2020:09:08:46 +0800] "GET /tomcat.css HTTP/1.1" 200 5581
10.0.0.1 - - [14/Jul/2020:09:08:46 +0800] "GET /tomcat.png HTTP/1.1" 200 5103
10.0.0.1 - - [14/Jul/2020:09:08:46 +0800] "GET /bg-nav.png HTTP/1.1" 200 1401
10.0.0.1 - - [14/Jul/2020:09:08:46 +0800] "GET /bg-upper.png HTTP/1.1" 200 3103
10.0.0.1 - - [14/Jul/2020:09:08:46 +0800] "GET /asf-logo-wide.svg HTTP/1.1" 200
27235
10.0.0.1 - - [14/Jul/2020:09:08:46 +0800] "GET /bg-middle.png HTTP/1.1" 200 1918
10.0.0.1 - - [14/Jul/2020:09:08:46 +0800] "GET /bg-button.png HTTP/1.1" 200 713
10.0.0.1 - - [14/Jul/2020:09:08:46 +0800] "GET /favicon.ico HTTP/1.1" 200 21630
```

范例: tomcat日志实现json格式的访问日志

```
[root@centos8 ~]#vim /usr/local/tomcat/conf/server.xml
.......
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
               prefix="localhost_access_log" suffix=".txt"
#########################添加下面一行,注意是一行,不要换行
#####################################
pattern="
{&amp;quot;clientip&amp;quot;:&amp;quot;%h&amp;quot;,&amp;quot;ClientUser&amp;quot;:&amp;quot;%l&amp;quot;,&amp;quot
;authenticated&amp;quot;:&amp;quot;%u&amp;quot;,&amp;quot;AccessTime&amp;quot;:&amp;quot;%t&amp;quot;,&amp;quot;
method&amp;quot;:&amp;quot;%r&amp;quot;,&amp;quot;status&amp;quot;:&amp;quot;%s&amp;quot;,&amp;quot;SendBytes&amp;qu
ot;:&amp;quot;%b&amp;quot;,&amp;quot;Query?
string&amp;quot;:&amp;quot;%q&amp;quot;,&amp;quot;partner&amp;quot;:&amp;quot;%
{Referer}i&amp;quot;,&amp;quot;AgentVersion&amp;quot;:&amp;quot;%{User-Agent}i&amp;quot;}"/>
#########################################################################
<!-- pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" /> -->
```

```
      </Host>
    </Engine>
  </Service>
</Server>
[root@centos8 ~]#systemctl restart tomcat

[root@centos8 ~]#tail   /usr/local/tomcat/logs/localhost_access_log.2020-08-
18.txt
10.0.0.1 - - [18/Aug/2020:17:59:06 +0800] "GET
/examples/jsp/jsp2/simpletag/hello.html HTTP/1.1" 200 1450
10.0.0.1 - - [18/Aug/2020:17:59:08 +0800] "GET
/examples/jsp/jsp2/simpletag/hello.jsp.html HTTP/1.1" 200 1416
10.0.0.1 - - [18/Aug/2020:17:59:23 +0800] "GET
/examples/jsp/jsp2/simpletag/HelloWorldSimpleTag.java.html HTTP/1.1" 200 1312
10.0.0.1 - - [18/Aug/2020:17:59:38 +0800] "GET
/examples/jsp/jsp2/tagfiles/hello.html HTTP/1.1" 200 1399
10.0.0.1 - - [18/Aug/2020:17:59:40 +0800] "GET
/examples/jsp/jsp2/tagfiles/hello.jsp.html HTTP/1.1" 200 1611
{"clientip":"10.0.0.100","ClientUser":"-","authenticated":"-","AccessTime":"
[18/Aug/2020:18:03:46 +0800]","method":"GET /
HTTP/1.0","status":"200","SendBytes":"11136","Query?string":"","partner":"-
","AgentVersion":"ApacheBench/2.3"}
{"clientip":"10.0.0.1","ClientUser":"-","authenticated":"-","AccessTime":"
[18/Aug/2020:18:08:43 +0800]","method":"GET
/examples/jsp/jsp2/tagfiles/hello.jsp.html
HTTP/1.1","status":"304","SendBytes":"-","Query?
string":"","partner":"http://10.0.0.28:8080/examples/jsp/jsp2/tagfiles/hello.htm
l","AgentVersion":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36 Edg/92.0.902.73"}
{"clientip":"10.0.0.1","ClientUser":"-","authenticated":"-","AccessTime":"
[18/Aug/2020:18:08:43 +0800]","method":"GET /favicon.ico
HTTP/1.1","status":"200","SendBytes":"21630","Query?
string":"","partner":"http://10.0.0.28:8080/examples/jsp/jsp2/tagfiles/hello.jsp
.html","AgentVersion":"Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36
Edg/92.0.902.73"}
{"clientip":"10.0.0.1","ClientUser":"-","authenticated":"-","AccessTime":"
[18/Aug/2020:18:08:51 +0800]","method":"GET /
HTTP/1.1","status":"200","SendBytes":"11156","Query?string":"","partner":"-
","AgentVersion":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36 Edg/92.0.902.73"}
{"clientip":"10.0.0.1","ClientUser":"-","authenticated":"-","AccessTime":"
[18/Aug/2020:18:08:51 +0800]","method":"GET /favicon.ico
HTTP/1.1","status":"200","SendBytes":"21630","Query?
string":"","partner":"http://10.0.0.28:8080/","AgentVersion":"Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/92.0.4515.131 Safari/537.36 Edg/92.0.902.73"}

#安装json工具jq
[root@rocky8 ~]#yum -y install jq

#利用jq解析json格式
[root@rocky8 ~]#echo '{"clientip":"10.0.0.1","ClientUser":"-","authenticated":"-
","AccessTime":"[11/Mar/2022:09:26:53 +0800]","method":"GET /
HTTP/1.1","status":"200","SendBytes":"11165","Query?string":"","partner":"-
","AgentVersion":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36 Edg/99.0.1150.36"}' |jq
{
```

```
  "clientip": "10.0.0.1",
  "ClientUser": "-",
  "authenticated": "-",
  "AccessTime": "[11/Mar/2022:09:26:53 +0800]",
  "method": "GET / HTTP/1.1",
  "status": "200",
  "SendBytes": "11165",
  "Query?string": "",
  "partner": "-",
  "AgentVersion": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36 Edg/99.0.1150.36"
}
```

3.3.3 组件

3.3.3.1 组件分层和分类

顶级组件

Server，代表整个Tomcat容器，一台主机可以启动多tomcat实例，需要确保端口不要产生冲突

服务类组件

Service，实现组织Engine和Connector，建立两者之间关联关系, service 里面只能包含一个Engine

连接器组件

Connector，有HTTP（默认端口8080/tcp）、HTTPS（默认端口8443/tcp）、AJP（默认端口8009/tcp）协议的连接器，AJP（Apache Jserv protocol）是一种基于TCP的二进制通讯协议。

容器类

Engine、Host（虚拟主机）、Context(上下文件,解决路径映射)都是容器类组件，可以嵌入其它组件，内部配置如何运行应用程序。

内嵌类

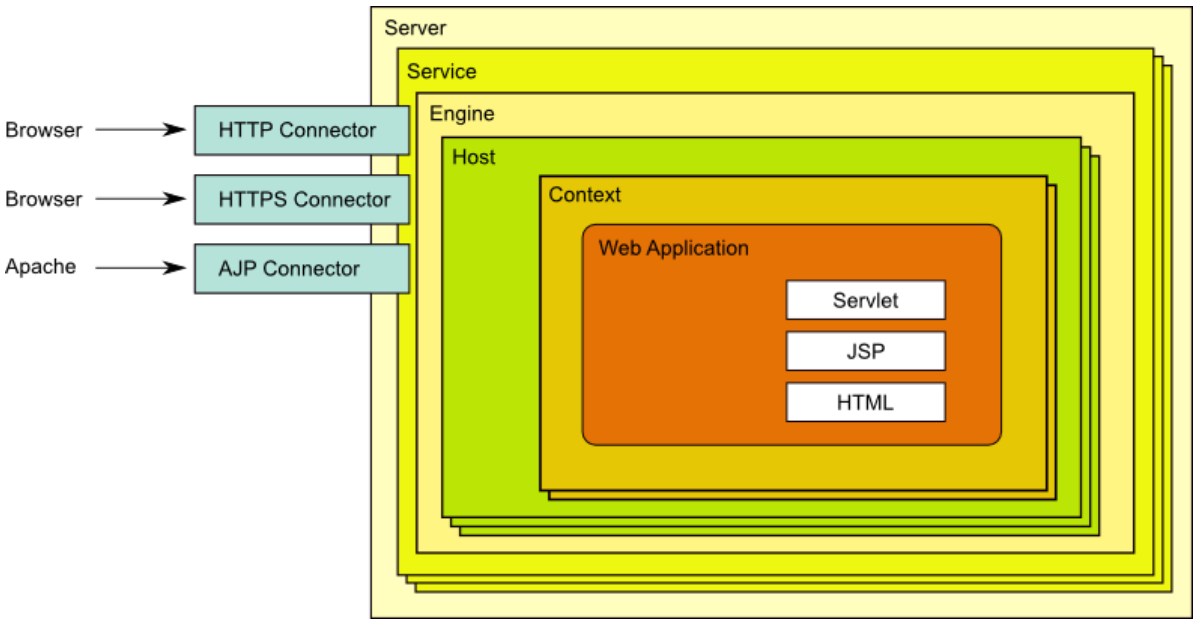
可以内嵌到其他组件内，valve、logger、realm、loader、manager等。以logger举例，在不同容器组件内分别定义。

集群类组件

listener、cluster

3.3.3.2 Tomcat 内部组成

由上述组件就构成了Tomcat，如下图



名称	说明
Server	服务器，Tomcat 运行的进程实例，一个Server中可以有多个service，但通常就一个
Service	服务，用来组织Engine和Connector的对应关系，一个service中只有一个Engine
Connector	连接器，负责客户端的HTTP、HTTPS、AJP等协议连接。一个Connector只属于某一个Engine
Engine	即引擎，用来响应并处理用户请求。一个Engine上可以绑定多个Connector
Host	即虚拟主机,可以实现多虚拟主机,例如使用不同的主机头区分
Context	应用的上下文，配置特定url路径映射和目录的映射关系：url => directory

每一个组件都由一个Java“类”实现，这些组件大体可分为以下几个类型：

```
顶级组件：Server
服务类组件：Service
连接器组件：http, https, ajp（apache jserv protocol）
容器类：Engine, Host, Context
被嵌套类：valve, logger, realm, loader, manager, ...
集群类组件：listener, cluster, ...
```

范例: 查看类

```
[root@centos8 ~]#grep className  /usr/local/tomcat/conf/server.xml
  <Listener className="org.apache.catalina.startup.VersionLoggerListener" />
  <Listener className="org.apache.catalina.security.SecurityListener" />
  <Listener className="org.apache.catalina.core.AprLifecycleListener"
SSLEngine="on" />
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener"
/>
  <Listener
className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
  <Listener
className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />
        <UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol" />
      <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
      <Realm className="org.apache.catalina.realm.LockOutRealm">
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
        <Valve className="org.apache.catalina.authenticator.SingleSignOn" />
        <Valve className="org.apache.catalina.valves.AccessLogValve"
directory="logs"
```

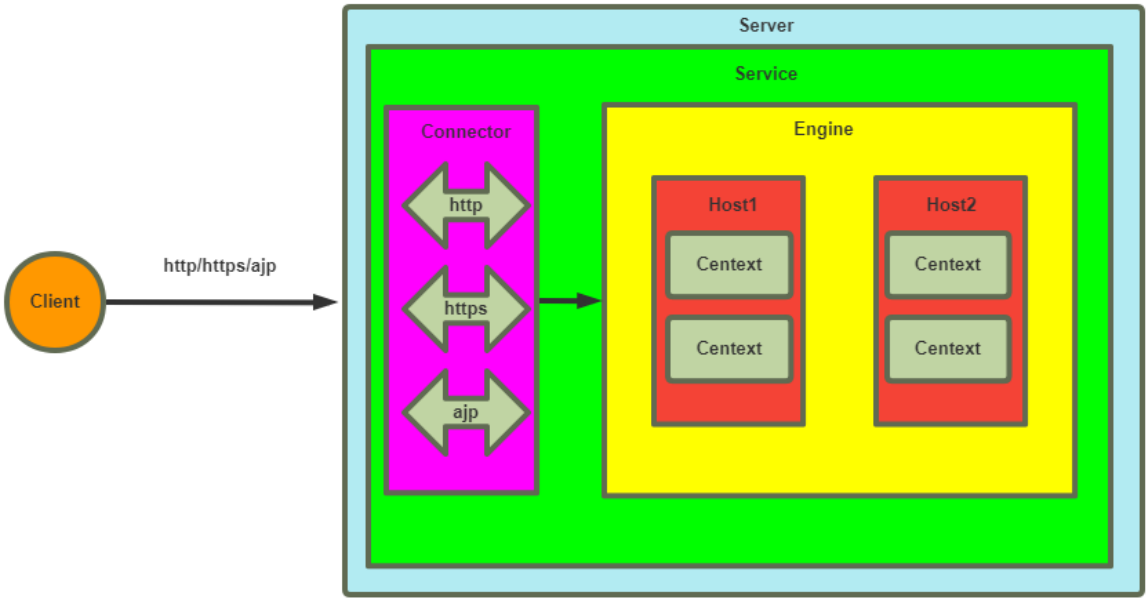
3.3.3.3 核心组件

- Tomcat启动一个Server进程。可以启动多个Server，即tomcat的多实例, 但一般只启动一个
- 创建一个Service提供服务。可以创建多个Service，但一般也只创建一个
  - 每个Service中，是Engine和其连接器Connector的关联配置
- 可以为这个Service提供多个连接器Connector，这些Connector使用了不同的协议，绑定了不同的端口。其作用就是处理来自客户端的不同的连接请求或响应
- Service 内部还定义了Engine，引擎才是真正的处理请求的入口，其内部定义多个虚拟主机Host
  - Engine对请求头做了分析，将请求发送给相应的虚拟主机
  - 如果没有匹配，数据就发往Engine上的defaultHost缺省虚拟主机
  - Engine上的缺省虚拟主机可以修改
- Host 定义虚拟主机，虚拟主机有name名称，通过名称匹配
- Context 定义应用程序单独的路径映射和配置

范例：多个组件关系 conf/server.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Server port="8005" shutdown="SHUTDOWN">
  <Service name="Catalina">
    <Connector port="8080" protocol="HTTP/1.1"connectionTimeout="20000"
               redirectPort="8443" />
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
    <Engine name="Catalina" defaultHost="localhost">
         <Host name="localhost"  appBase="webapps" unpackWARs="true"
autoDeploy="true">
            <Context >
            <Context />
         </Host>
    </Engine>
  </Service>
</Server>
```

3.3.3.4 tomcat 处理请求过程



假设来自客户的请求为：http://localhost:8080/test/index.jsp

- 浏览器端的请求被发送到服务端端口8080，Tomcat进程监听在此端口上。通过侦听的HTTP/1.1 Connector获得此请求。
- Connector把该请求交给它所在的Service的Engine来处理，并等待Engine的响应
- Engine获得请求localhost:8080/test/index.jsp，遍历它所有虚拟主机Host
- Engine匹配到名为localhost的Host。如果匹配不到,就把请求交给该Engine中的defaultHost处理
- localhost Host获得请求/test/index.jsp，匹配它所拥有的所有Context
- Host匹配到路径为/test的Context
- path=/test的Context获得请求index.jsp，在它的mapping table中寻找对应的servlet
- Context匹配到URL PATTERN为`*.jsp`的servlet，对应于JspServlet类构造HttpServletRequest对象和HttpServletResponse对象，作为参数调用JspServlet的doGet或doPost方法。
- Context把执行完了之后的HttpServletResponse对象返回给Host
- Host把HttpServletResponse对象返回给Engine
- Engine把HttpServletResponse对象返回给Connector
- Connector把HttpServletResponse对象返回给浏览器端

3.4 应用部署

3.4.1 tomcat的根目录结构

Tomcat中默认网站根目录是**$CATALINA_BASE**/webapps/

在Tomcat中部署主站应用程序和其他应用程序，和之前WEB服务程序不同。

nginx

假设在nginx中部署2个网站应用eshop、forum，假设网站根目录是/data/nginx/html，那么部署可以是这样的。
eshop解压缩所有文件放到`/data/nginx/html/`目录下,forum 的文件放在`/data/nginx/html/forum/`下。

最终网站链接有以下对应关系

```
http://localhost/ 对应于eshop的应用，即 /data/nginx/html/
http://localhost/forum/ 对应于forum的应用，即/data/nginx/html/forum/
```

Tomcat

Tomcat中默认网站根目录是**$CATALINA_BASE**/webapps/

在Tomcat的webapps目录中，有个非常特殊的目录ROOT，它就是网站默认根目录。

将eshop解压后的文件放到这个**$CATALINA_BASE**/webapps/ROOT中。

bbs解压后文件都放在**$CATALINA_BASE**/webapps/forum目录下。

$CATALINA_BASE/webapps下面的每个目录都对应一个Web应用,即WebApp

最终网站链接有以下对应关系

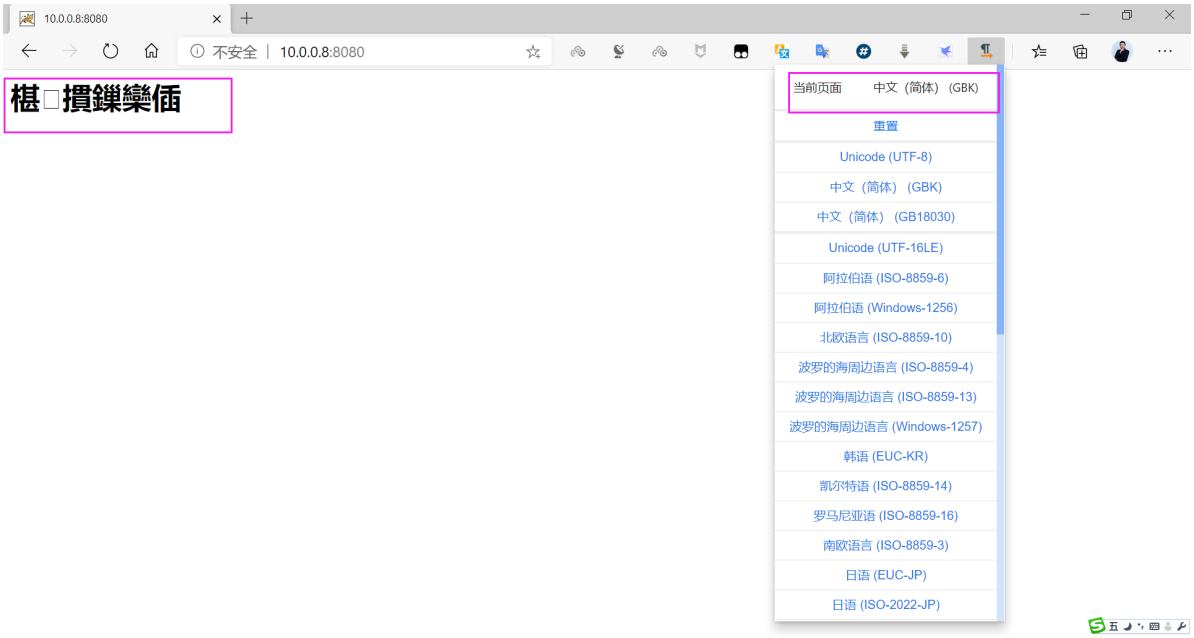
```
http://localhost/    对应于eshop的应用WebApp，即$CATALINA_BASE/webapps/ROOT/目录，
http://localhost/forum/ 对应于forum的应用WebApp，即$CATALINA_BASE/webapps/forum/
```

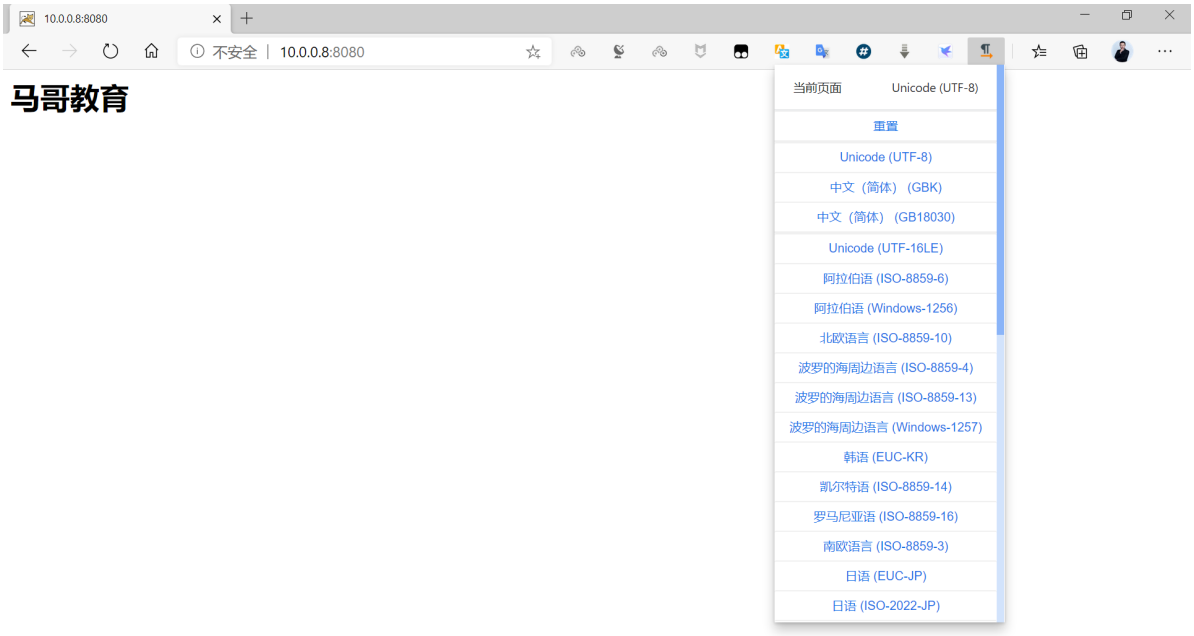
如果同时存在**$CATALINA_BASE**`/webapps/ROOT/forum`，仍以`$CATALINA_BASE/webapps/forum/`优先生效

每一个虚拟主机都可以使用**appBase**指令配置自己的站点目录，使用appBase目录下的ROOT目录作为主站目录。

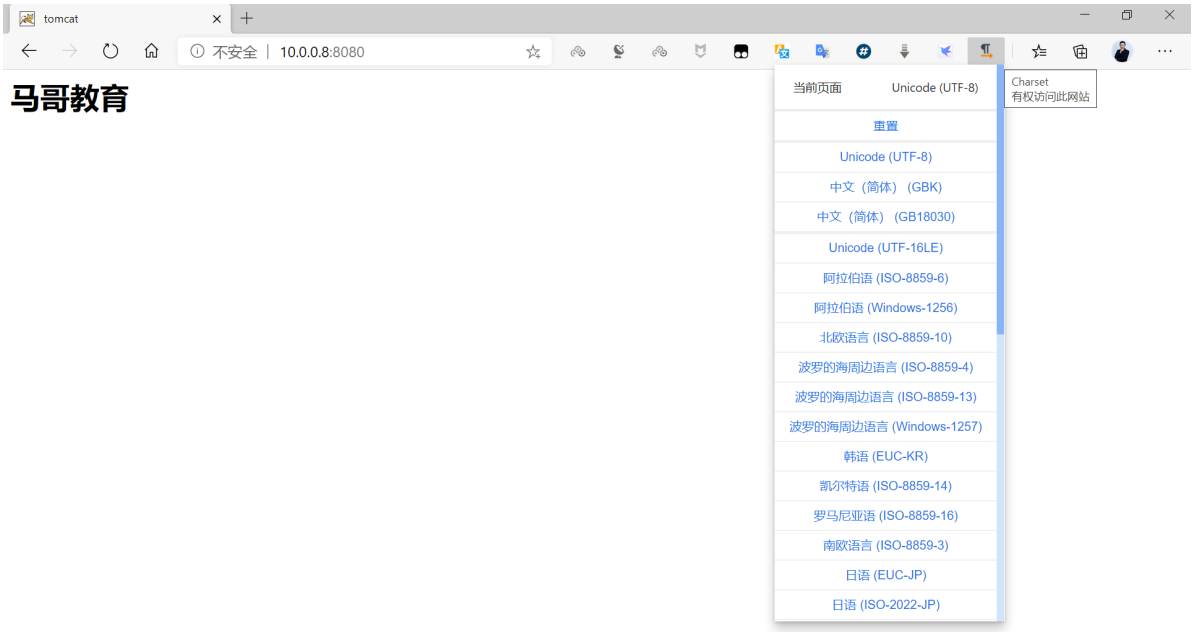
范例: 主页目录和编码

```
[root@centos8 ~]#cat /usr/local/tomcat/webapps/ROOT/index.html
<h1>马哥教育</h1>

[root@centos8 ~]#curl 10.0.0.8:8080/index.html  -I
HTTP/1.1 200
Accept-Ranges: bytes
ETag: W/"22-1594212097000"
Last-Modified: Wed, 08 Jul 2020 12:41:37 GMT
Content-Type: text/html  #tomcat无指定编码,浏览器自动识别为GBK,可能会导致乱码
Content-Length: 22
Date: Wed, 08 Jul 2020 13:06:03 GMT

#httpd服务器默认指定编码为UTF-8,因为服务器本身不会出现乱码
#nginx服务器默认在响应头部没有批定编码,也会出现乱码
[root@centos8 ~]#curl 10.0.0.18/index.html  -I
HTTP/1.1 200 OK
Date: Wed, 08 Jul 2020 13:07:57 GMT
Server: Apache/2.4.37 (centos)
Last-Modified: Wed, 08 Jul 2020 12:59:55 GMT
ETag: "16-5a9edaf39d274"
Accept-Ranges: bytes
Content-Length: 22
Content-Type: text/html; charset=UTF-8

#浏览器的设置默认不是UTF-8,可能会导致乱码
```






```
#修改网页指定编码
[root@centos8 ~]#cat /usr/local/tomcat/webapps/ROOT/index.html
<html>
<head>
<meta http-equiv=Content-Type content="text/html;charset=utf-8">
<title>tomcat</title>
</head>
<h1>马哥教育</h1>
```



3.4.2 JSP WebApp目录结构

$CATALINA_BASE/webapps下面的每个目录对应的WebApp,可能有以下子目录,但下面子目录是非必须的

- 主页配置：默认按以下顺序查找主页文件`index.html，index.htm、index.jsp`
- WEB-INF/：当前目录WebApp的私有资源路径，通常存储当前应用使用的web.xml和context.xml配置文件
- META-INF/：类似于WEB-INF，也是私有资源的配置信息，和WEB-INF/目录一样浏览器无法访问
- classes/：类文件，当前webapp需要的类
- lib/：当前应用依赖的jar包

3.4.3 主页设置

3.4.3.1 全局配置实现修改默认主页文件

默认情况下 tomcat 会在**$CATALINA_BASE**/webapps/ROOT/目录下按以下次序查找文件,找到第一个则进行显示

- index.html
- index.htm
- index.jsp

可以通过修改 **$CATALINA_BASE**/conf/web.xml 中的下面`<welcome-file-list>标签`内容修改默认页文件

范例: 修改默认主页文件

```
[root@centos8 tomcat]#pwd
/usr/local/tomcat
[root@centos8 tomcat]#echo '<h1>www.magedu.org</h1>' > webapps/ROOT/index.html
[root@centos8 tomcat]#curl http://127.0.0.1:8080/
<h1>www.magedu.org</h1>

[root@centos8 tomcat]#tail conf/web.xml
  <!-- here, so be sure to include any of the default values that you wish  -->
  <!-- to use within your application.                                      -->

    <welcome-file-list>
```

```
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

</web-app>
[root@centos8 tomcat]#vim conf/web.xml
[root@centos8 tomcat]#tail conf/web.xml
  <!-- here, so be sure to include any of the default values that you wish  -->
  <!-- to use within your application.                                       -->

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>

</web-app>
[root@centos8 tomcat]#systemctl restart tomcat
[root@centos8 tomcat]#curl http://127.0.0.1:8080/
```

3.4.3.2 WebApp的专用配置文件

将上面主配置文件conf/web.xml中的`<welcome-file-list>`标签内容，复制到/usr/local/tomcat/webapps/ROOT/WEB-INF/web.xml中，如下所示:

范例: 针对主站点根目录设置专用配置文件

```
[root@centos8 tomcat]#vim webapps/ROOT/WEB-INF/web.xml
[root@centos8 tomcat]#cat webapps/ROOT/WEB-INF/web.xml
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                      http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1"
  metadata-complete="true">

  <display-name>Welcome to Tomcat</display-name>
  <description>
     Welcome to Tomcat
  </description>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>    #修改三个文件的顺序
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
#配置修改后，无需重启tomcat服务，即可观察首页变化
[root@centos8 tomcat]#curl http://127.0.0.1:8080/
```

范例: 针对特定APP目录设置专用配置文件

```
[root@centos8 tomcat]#cp -a  webapps/ROOT/WEB-INF/   webapps/magedu/
[root@centos8 tomcat]#echo /usr/local/tomcat/webapps/magedu/test.html >
webapps/magedu/test.html
```

```
[root@centos8 tomcat]#tree webapps/magedu/
webapps/magedu/
├── index.htm
├── index.html
├── test.html
└── WEB-INF
    └── web.xml

1 directory, 4 files

[root@centos8 tomcat]#cat webapps/magedu/WEB-INF/web.xml
......
  <description>
     Welcome to Tomcat
  </description>
   <welcome-file-list>
    <welcome-file>test.html</welcome-file>   #修改默认页面文件的顺序
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.html</welcome-file>
    </welcome-file-list>
</web-app>

#注意修改属性
[root@centos8 tomcat]#chown -R tomcat.tomcat webapps/magedu/

[root@centos7 ~]#curl http://www.magedu.org:8080/magedu/
/usr/local/tomcat/webapps/magedu/test.html
```

配置规则：

- webApp的专有配置优先于系统的全局配置
- 修改系统的全局配置文件，需要重新启动服务生效
- 修改 webApp的专有配置，无需重启即可生效

3.4.4 应用部署实现

3.4.4.1 WebApp应用的归档格式

- .war：WebApp打包,类zip格式文件,通常包括一个应用的所有资源,比如jsp,html,配置文件等
- .jar：EJB类文件的打包压缩类zip格式文件，,包括很多的class文件, 网景公司发明
- .rar：资源适配器类打包文件，目前已不常用
- .ear：企业级WebApp打包，目前已不常用

传统应用开发测试后，通常打包为war格式，这种文件部署到Tomcat的webapps目录下，并默认会自动解包展开和部署上线。

```
#conf/server.xml中文件配置
<Host name="localhost"  appBase="webapps" unpackWARs="true" autoDeploy="true">
```

3.4.4.2 部署方式

- 部署Deploy：将webapp的源文件放置到目标目录，通过web.xml和context.xml文件中配置的路径就可以访问该webapp，通过类加载器加载其特有的类和依赖的类到JVM上，即：最终用户可以通过浏览器访问该应用
    - 自动部署：Tomcat一旦发现多了一个web应用APP.war包，默认会自动把它解压缩，加载并启动起来

    - 手动部署
        - 冷部署：将webapp放到指定目录，才去启动Tomcat服务
        - 热部署：Tomcat服务不停止，需要依赖manager、ant脚本、tcd（tomcat client deployer）等工具
- 反部署undeploy：停止webapp运行，并从JVM上清除已经加载的类，从Tomcat应用目录中移除部署的文件
- 启动start：是webapp能够访问
- 停止stop：webapp不能访问，不能提供服务，但是JVM并不清除它

3.4.4.3 部署WebApp的目录结构

常见开发项目目录组成

```
#目录结构一般由开发用工具自动生成，以下模拟生成相关目录
mkdir projects/myapp/{WEB-INF,META-INF,classes,lib} -pv
mkdir: 已创建目录 "projects"
mkdir: 已创建目录 "projects/myapp"
mkdir: 已创建目录 "projects/myapp/WEB-INF"
mkdir: 已创建目录 "projects/myapp/META-INF"
mkdir: 已创建目录 "projects/myapp/classes"
mkdir: 已创建目录 "projects/myapp/lib"

#常见应用首页，内容就用前面的test.jsp内部
vi projects/myapp/index.jsp

#手动复制项目目录到webapps目录下去
cp -r projects/myapp/ /usr/local/tomcat/webapps/

#注意权限和属性
chown -R tomcat.tomcat /usr/local/tomcat/webapps/myapp

#访问http://YourIP:8080/myapp/
```

3.4.4.4 实战案例：手动的应用部署

3.4.4.4.1 部署主页目录下的应用WebApp

```
[root@centos8 tomcat]#vim webapps/ROOT/test.jsp
[root@centos8 tomcat]#cat webapps/ROOT/test.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
<%
out.println("hello jsp");
%>
<br>
<%=request.getRequestURL()%>
</body>
```

```
</html>

[root@centos8 tomcat]#curl http://127.0.0.1:8080/test.jsp

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
hello jsp

<br>
http://127.0.0.1:8080/test.jsp
</body>
</html>

[root@centos8 tomcat]#tree work/Catalina/localhost/ROOT/
work/Catalina/localhost/ROOT/
└── org
    └── apache
        └── jsp
            ├── test_jsp.class
            └── test_jsp.java

3 directories, 2 files
[root@centos8 tomcat]#
```

3.4.4.4.2 部署一个子目录的应用WebApp

```
[root@centos8 tomcat]#pwd
/usr/local/tomcat
[root@centos8 tomcat]#mkdir webapps/app1/

#利用之前实验的文件生成新应用
[root@centos8 tomcat]#cp -p webapps/ROOT/test.jsp  webapps/app1/
[root@centos8 tomcat]#chown -R tomcat.tomcat webapps/app1/
[root@centos8 tomcat]#tree webapps/app1/
webapps/testapp1/
└── test.jsp

0 directories, 1 file
[root@centos8 tomcat]#curl http://127.0.0.1:8080/app1/test.jsp

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
hello jsp
<br>
<%=request.getRequestURL()%>
```

```
</body>
</html>
[root@centos8 tomcat]#tree work/Catalina/localhost/app1/
work/Catalina/localhost/app1/
└── org
    └── apache
        └── jsp
            ├── test_jsp.class
            └── test_jsp.java

3 directories, 2 files
[root@centos8 tomcat]#

#删除应用
[root@centos8 tomcat]#rm -rf webapps/app1/
[root@centos8 tomcat]#ls webapps/
docs  examples  host-manager  manager  ROOT
[root@centos8 tomcat]#ls work/Catalina/localhost/
docs  examples  host-manager  manager  ROOT
```

3.4.4.5 实战案例：自动的应用部署war包

3.4.4.5.1 制作应用的war包文件

```
[root@centos8 ~]#ls /data/app2/
test.html  test.jsp
[root@centos8 ~]#cat /data/app2/test.html
<h1>This is test html </h1>
[root@centos8 ~]#cat /data/app2/test.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
<%
out.println("test jsp");
%>
<br>
<%=request.getRequestURL()%>
</body>
</html>
[root@centos8 ~]#cd /data/app2/

#生成war包文件app2.war,此文件的名称决定了tomcat子目录的名称
[root@centos8 app2]#jar cvf /data/app2.war *
added manifest
adding: test.html(in = 28) (out= 27)(deflated 3%)
adding: test.jsp(in = 329) (out= 275)(deflated 16%)
[root@centos8 app2]#cd /data/app2/
[root@centos8 app2]#ls
test.html  test.jsp
[root@centos8 app2]#cd /data/
```

```
[root@centos8 data]#ls
app2  app2.war
[root@centos8 data]#file app2.war
app2.war: Java archive data (JAR)
[root@centos8 data]#chown tomcat.tomcat /data/app2.war
```

3.4.4.5.2 自动应用部署上面的war包

```
[root@centos8 tomcat]#pwd
/usr/local/tomcat
[root@centos8 tomcat]#ls webapps/
docs  examples  host-manager  manager  ROOT
[root@centos8 tomcat]#ls work/Catalina/localhost/
docs  examples  host-manager  manager  ROOT
[root@centos8 tomcat]#cp -p /data/app2.war webapps/


#再次查看，tomcat将app2.war自动解压缩
[root@centos8 tomcat]#ll webapps/
total 8
drwxr-x--- 15 tomcat tomcat 4096 Feb  9 11:02 docs
drwxr-x---  6 tomcat tomcat   83 Feb  9 11:02 examples
drwxr-x---  5 tomcat tomcat   87 Feb  9 11:02 host-manager
drwxr-x---  5 tomcat tomcat  103 Feb  9 11:02 manager
drwxr-x---  3 tomcat tomcat  300 Feb  9 19:59 ROOT
drwxr-x---  3 tomcat tomcat   55 Feb  9 20:14 app2
-rw-r--r--  1 tomcat tomcat  862 Feb  9 20:05 app2.war
[root@centos8 tomcat]#ll webapps/app2
total 8
drwxr-x--- 2 tomcat tomcat  44 Feb  9 20:14 META-INF
-rw-r----- 1 tomcat tomcat  28 Feb  9 20:03 test.html
-rw-r----- 1 tomcat tomcat 329 Aug 30 02:30 test.jsp

#work目录会自动生成对应的app2的子目录，但目录内无内容
[root@centos8 tomcat]#tree work/Catalina/localhost/app2/
work/Catalina/localhost/app2/

0 directories, 0 files

#访问jsp文件后，tomcat会自动将jsp转换和编译生成work目录下对应的java和class文件
[root@centos8 tomcat]#curl http://127.0.0.1:8080/app2/test.jsp

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
hello jsp

</body>
</html>
[root@centos8 tomcat]#tree work/Catalina/localhost/app2/
work/Catalina/localhost/app2/
└── org
```

```
    └── apache
        └── jsp
            ├── test_jsp.class
            └── test_jsp.java

3 directories, 2 files
[root@centos8 tomcat]#curl http://127.0.0.1:8080/app2/test.html
<h1>This is test html </h1>
[root@centos8 tomcat]#tree work/Catalina/localhost/app2/
work/Catalina/localhost/estapp2/
└── org
    └── apache
        └── jsp
            ├── test_jsp.class
            └── test_jsp.java

3 directories, 2 files

#自动删除（反部署）
#[root@centos8 tomcat]#rm -f webapps/app2.war
[root@centos8 tomcat]#ls webapps/
docs  examples  host-manager  manager  ROOT  app2
#过几秒再查看，发现app2目录也随之删除
[root@centos8 tomcat]#ls webapps/
docs  examples  host-manager  manager  ROOT
[root@centos8 tomcat]#ls webapps/
docs  examples  host-manager  manager  ROOT
[root@centos8 tomcat]#ls  work/Catalina/localhost/
docs  examples  host-manager  manager  ROOT
```

3.4.4.6 实站案例: 部署基于JAVA的博客系统 JPress

JPress 是一个使用Java开发的类似WordPress的产品的建站神器，目前已有超过10万+网站使用 JPress 搭建，其中包括多个政府机构，200+上市公司，中科院、红十字会等。

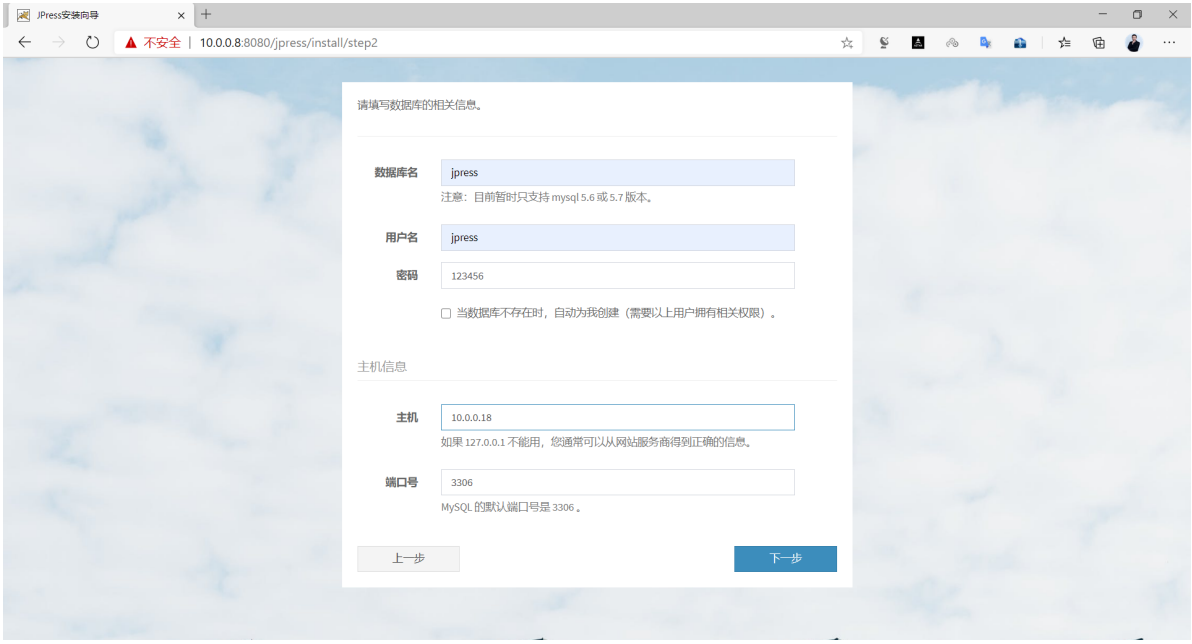
和JPress 相类似的基于 java 开发的博客系统还有zrlog,Halo等

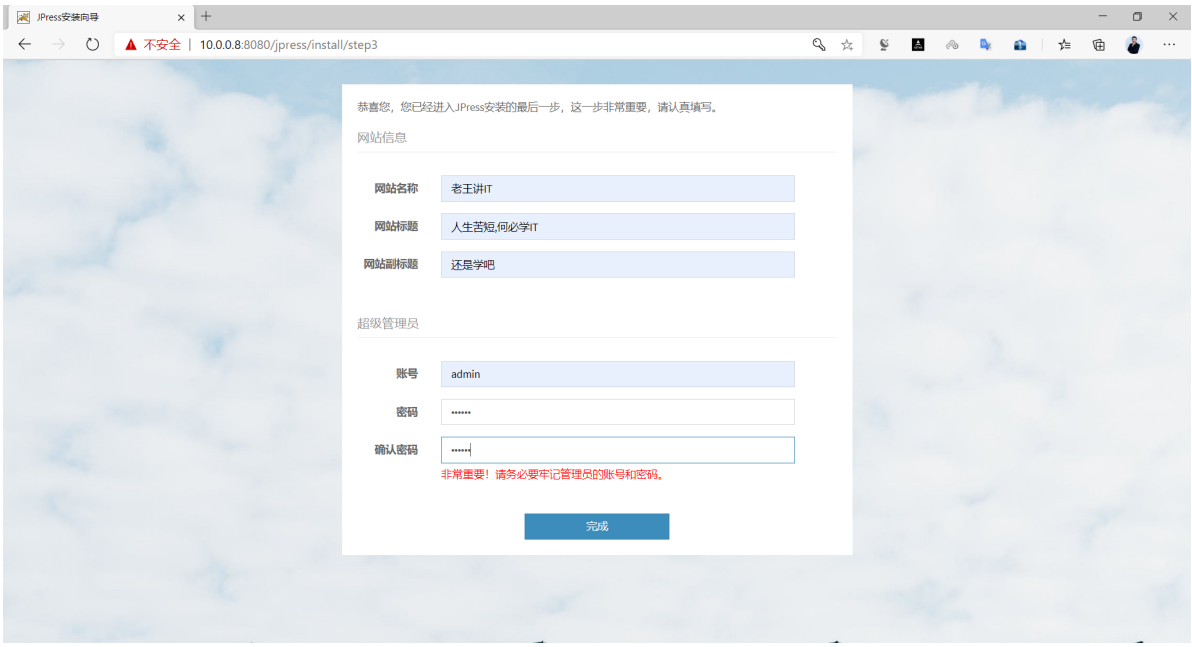
开源协议: LGPL-3.0

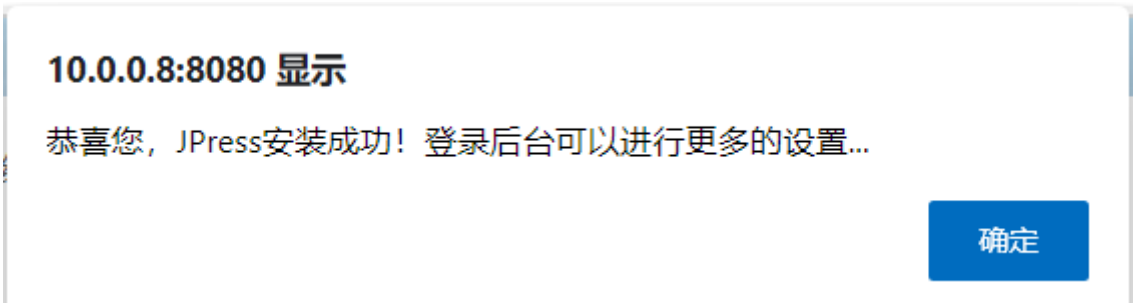
官方网站: http://www.jpress.io/

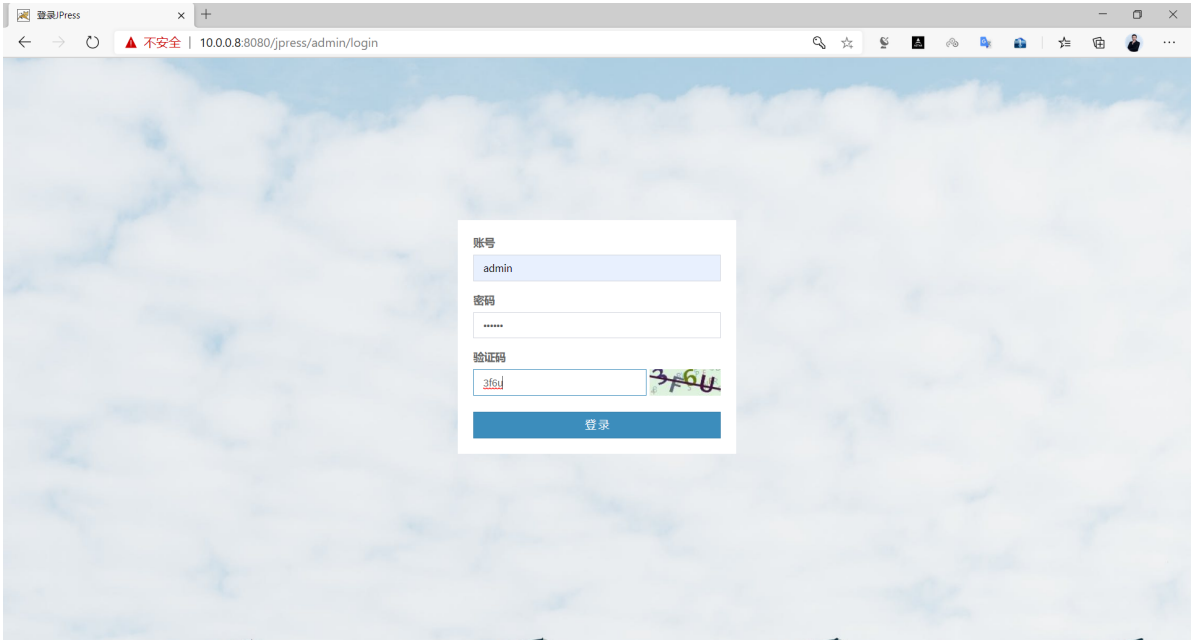
```
[root@centos8 ~]#cd /usr/local/tomcat/webapps/
[root@centos8 webapps]#ls
docs  examples  host-manager  jpress-v3.2.1.war  manager  ROOT
```

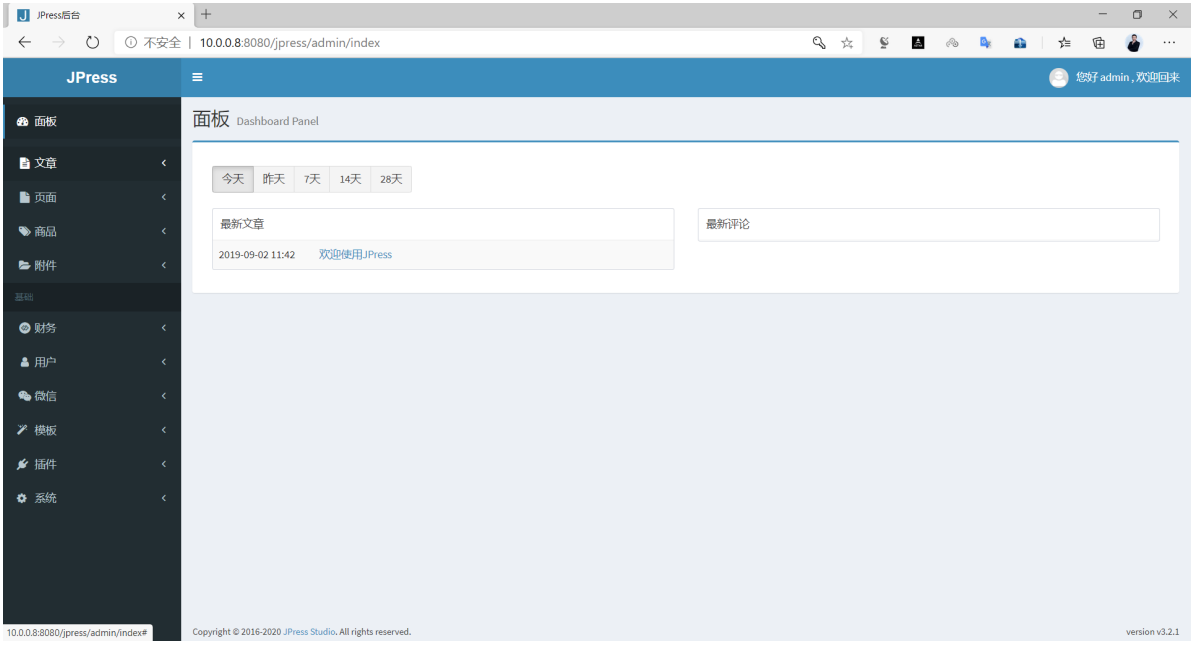
```
#自动解压缩生成jpress-v3.2.1目录
[root@centos8 webapps]#ls
docs  examples  host-manager  jpress-v3.2.1  jpress-v3.2.1.war  manager  ROOT

#生成软链接
[root@centos8 webapps]#ln -s jpress-v3.2.1 jpress
[root@centos8 webapps]#ls
docs  examples  host-manager  jpress  jpress-v3.2.1  jpress-v3.2.1.war  manager
ROOT
[root@centos8 webapps]#

#准备数据库和用户授权
[root@centos8 ~]#yum -y install mysql-server
[root@centos8 ~]#;systemctl enable --now mysqld
[root@centos8 ~]#mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.17 Source distribution

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create user jpress@'10.0.0.%' identified by '123456';
mysql> create database jpress;
mysql> grant all on jpress.* to jpress@'10.0.0.%';
```

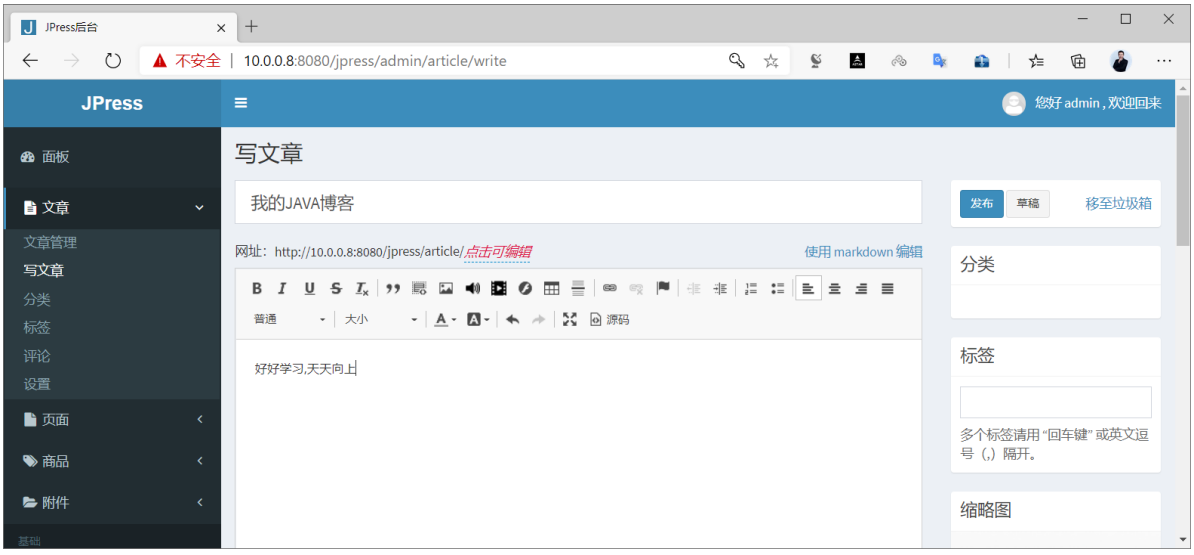

JPress安装向导
不安全 | 10.0.0.8:8080/jpress/install/step2
请填写数据库的相关信息。
数据库名
jpress
注意：目前暂时只支持 mysql 5.6 或 5.7 版本。
用户名
jpress
密码
123456
当数据库不存在时，自动为我创建（需要以上用户拥有相关权限）。
主机信息
主机
10.0.0.18
如果 127.0.0.1 不能用，您通常可以从网站服务商得到正确的信息。
端口号
3306
MySQL 的默认端口号是 3306。
上一步
下一步

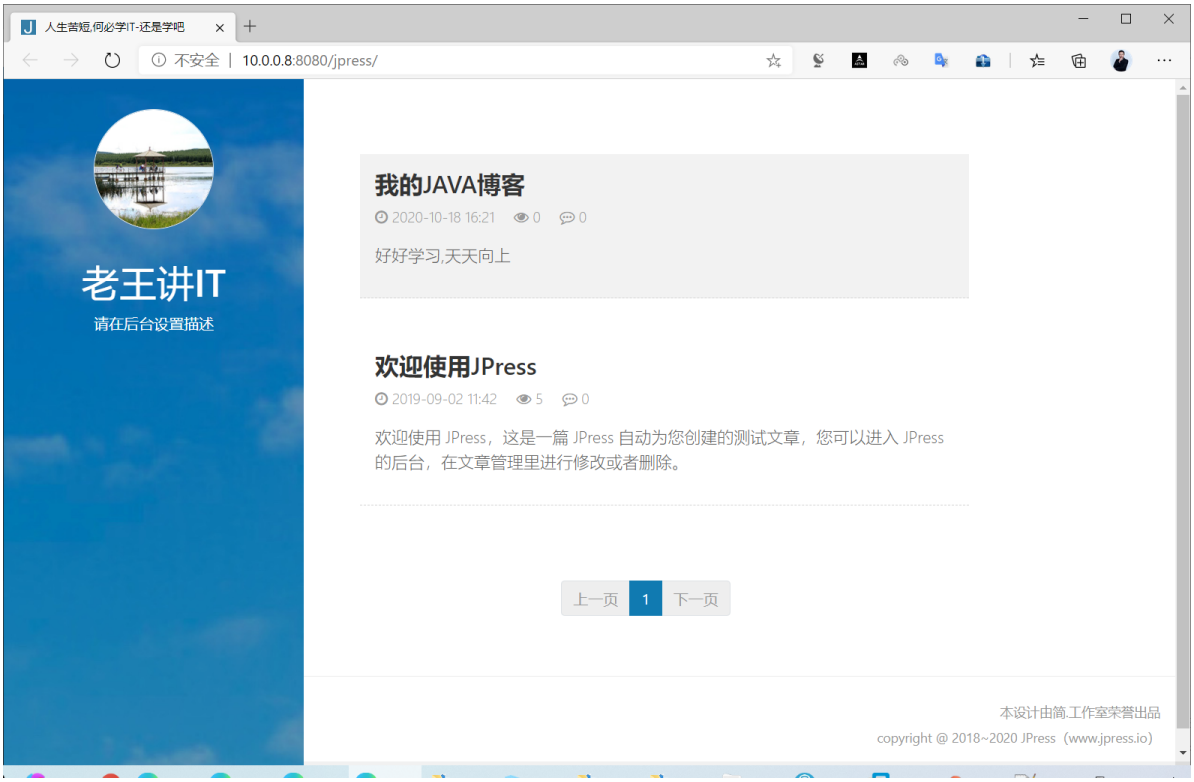

JPress安装向导
不安全 | 10.0.0.8:8080/jpress/install/step3
恭喜您，您已经进入JPress安装的最后一步，这一步非常重要，请认真填写。
网站信息
网站名称
老王讲IT
网站标题
人生苦短,何必学IT
网站副标题
还是学吧
超级管理员
账号
admin
密码
确认密码
非常重要！请务必要牢记管理员的账号和密码。
完成


10.0.0.8:8080 显示
恭喜您，JPress安装成功！登录后台可以进行更多的设置...
确定


登录JPress
不安全 | 10.0.0.8:8080/jpress/admin/login
账号
admin
密码
验证码
3f6u
登录


JPress后台
不安全 | 10.0.0.8:8080/jpress/admin/index
JPress
您好 admin，欢迎回来
面板
文章
页面
商品
附件
财务
用户
微信
模板
插件
系统
面板 Dashboard Panel
今天
昨天
7天
14天
28天
最新文章
2019-09-02 11:42
欢迎使用JPress
最新评论
Copyright © 2016-2020 JPress Studio. All rights reserved.
version v3.2.1


JPress后台
不安全 | 10.0.0.8:8080/jpress/admin/article/write
JPress
您好 admin，欢迎回来
面板
文章
文章管理
写文章
分类
标签
评论
设置
页面
商品
附件
写文章
我的JAVA博客
网址：http://10.0.0.8:8080/jpress/article/点击可编辑
使用 markdown 编辑
普通
大小
源码
好好学习,天天向上
发布
草稿
移至垃圾箱
分类
标签
多个标签请用“回车键”或英文逗号（,）隔开。
缩略图



上传的图片存放路径

```
[root@centos8 webapps]#tree jpress/attachment/
jpress/attachment/
└── 20210110
    └── 2169440a4e75459d8a420f4b245565d4.jpg

1 directory, 1 file
```

3.4.4.7 实战案例: 部署基于JAR包的博客系统Halo

Halo 是一款现代化的基于JAVA实现的博客/CMS系统

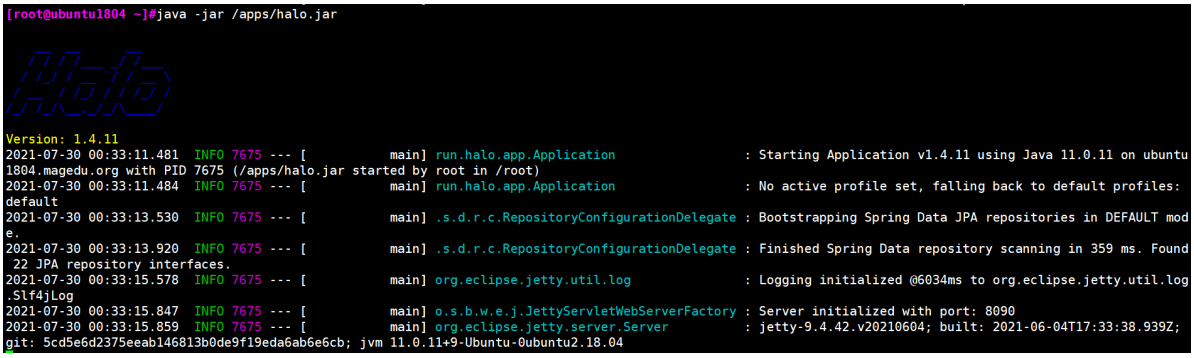
Halo 官网: https://halo.run/

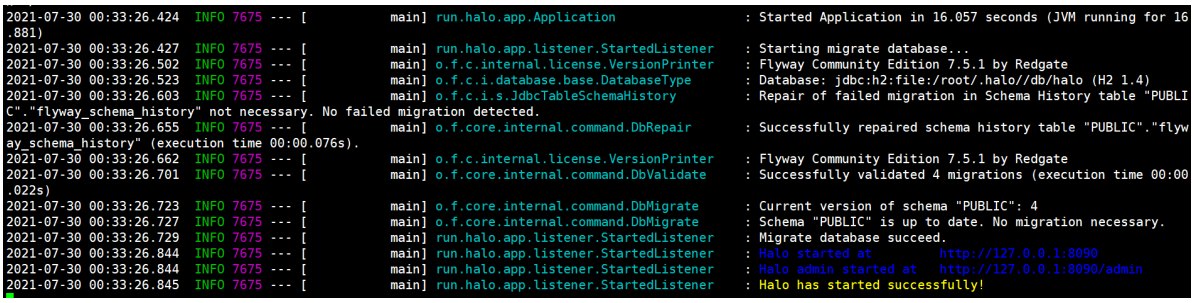
Halo 部署: https://docs.halo.run/install/linux

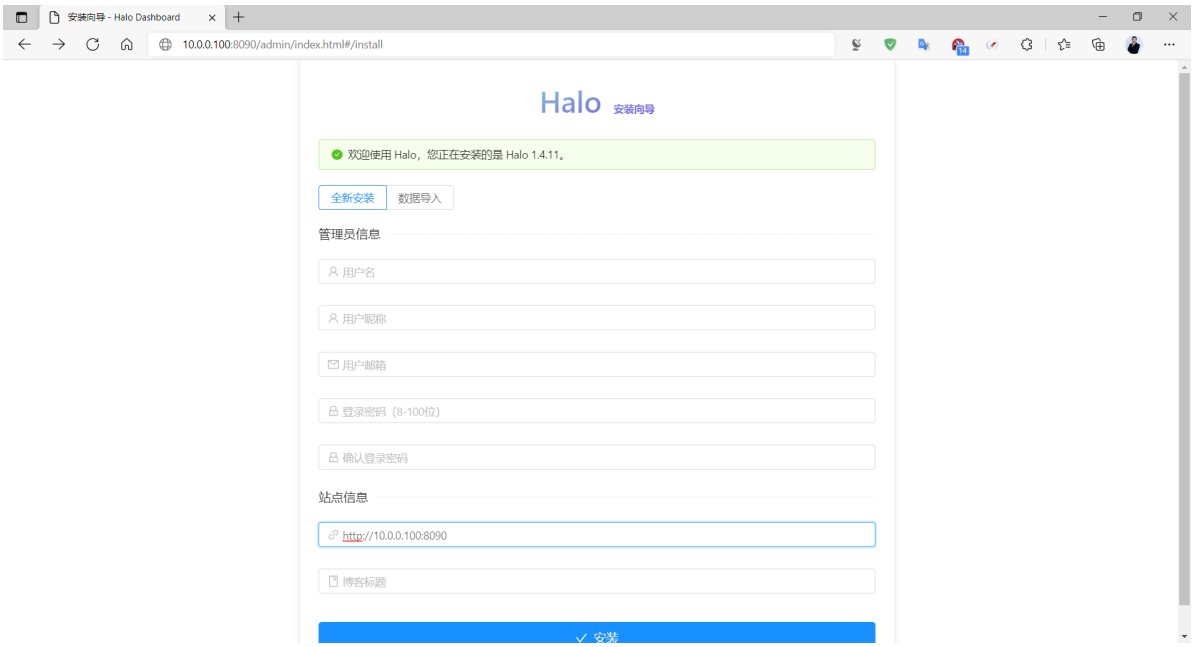
docker 部署: https://docs.halo.run/install/docker

范例: 部署 Halo 博客系统

```
#注意:运行当前版本Halo的最低依赖要求为 JRE 11，目前介绍两种 Linux 发行版的安装方式，均为 OpenJRE，不推荐 Oracle 版本。
[root@ubuntu1804 ~]#apt -y install openjdk-11-jdk
[root@ubuntu1804 ~]#mkdir /apps
[root@ubuntu1804 ~]#wget https://dl.halo.run/release/halo-1.4.11.jar -O /apps/halo.jar
#或者
[root@ubuntu1804 ~]#curl -L https://github.com/halo-dev/halo/releases/download/v1.4.11/halo-1.4.11.jar --output /apps/halo.jar
#运行Halo
[root@ubuntu1804 ~]#java -jar /apps/halo.jar
```







范例: docker 启动 halo

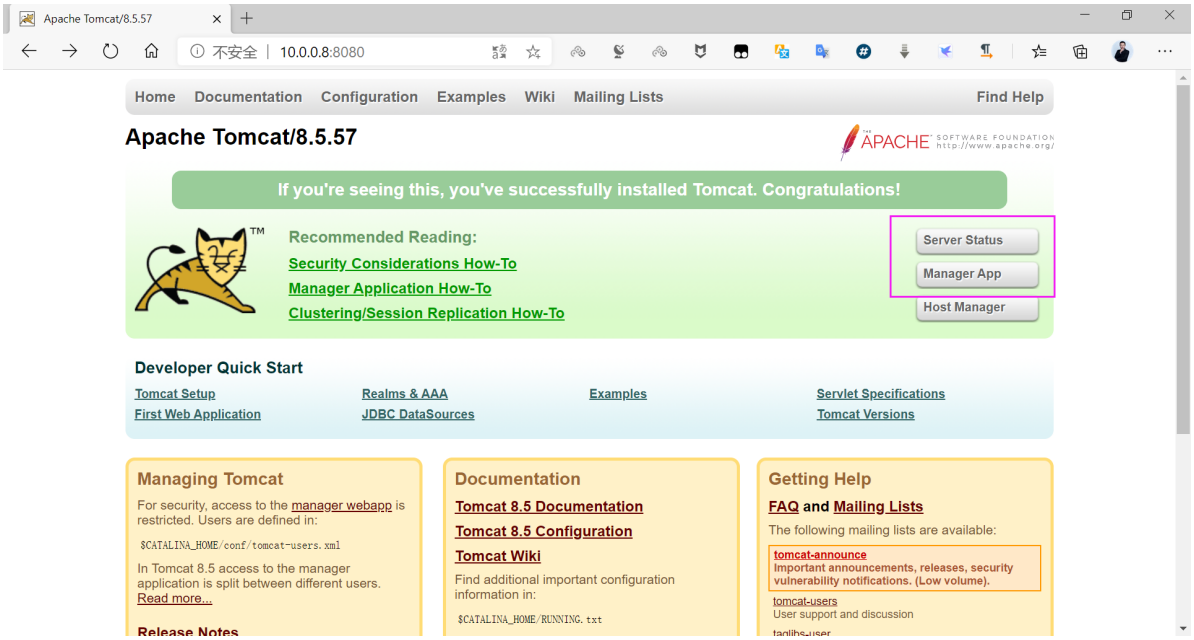
```
[root@ubuntu1804 ~]#docker run -it -d --name halo -p 8090:8090 -v
~/.halo:/root/.halo --restart=always halohub/halo
```

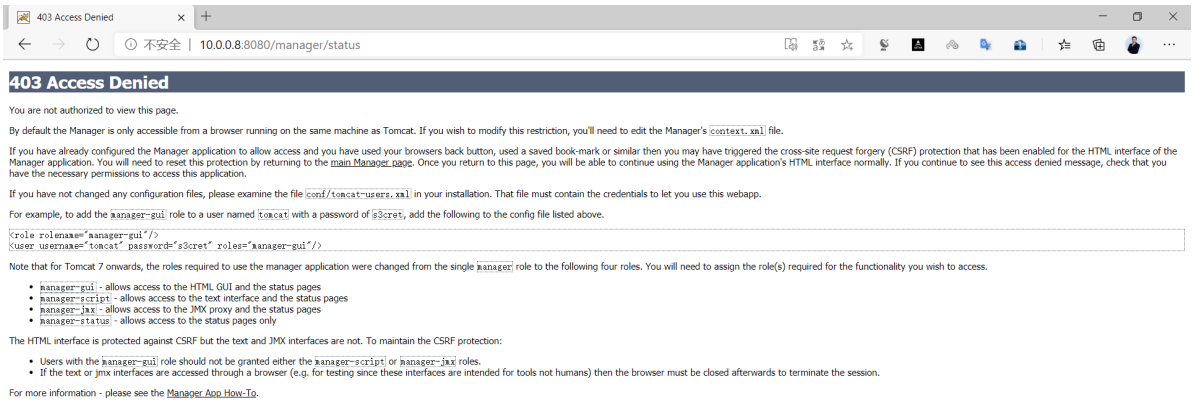
3.4.4.8 基于WEB的管理Server status和Manager APP实现应用部署

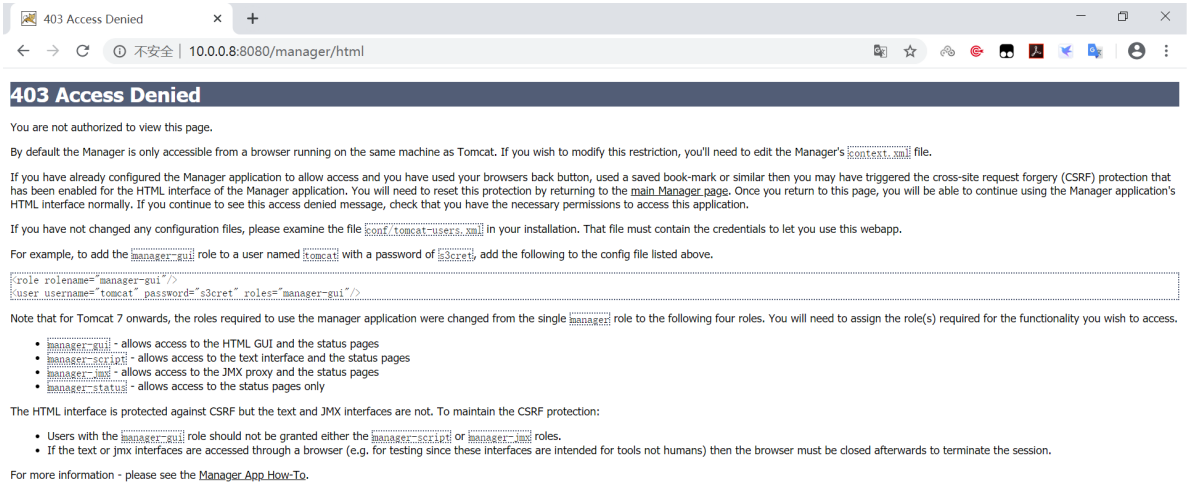
tomcat 提供了基于WEB的管理页面,默认由 tomcat-admin-webapps.noarch包提供相关文件

3.4.4.8.1 实现WEB的管理Server status和Manager APP

打开浏览器可以访问tomcat管理的默认管理页面，点击下图两个按钮都会出现下面提示403的错误提示







默认的管理页面被禁用，启用方法如下

- **修改conf/conf/tomcat-users.xml**

```
[root@centos8 tomcat]#ls conf/
Catalina             context.xml           logging.properties  tomcat-users.xml
catalina.policy      jaspic-providers.xml  server.xml          tomcat-users.xsd
catalina.properties  jaspic-providers.xsd  tomcat.conf         web.xml

#查看配置信息
[root@centos8 tomcat]#cat conf/server.xml
<GlobalNamingResources>
    <!-- Editable user database that can also be used by
         UserDatabaseRealm to authenticate users
    -->
    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
              description="User database that can be updated and saved"
              factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />   #由此文件指定授权用户信息
  </GlobalNamingResources>
```

用户认证，配置文件是conf/tomcat-users.xml。打开tomcat-users.xml，我们需要一个角色manager-gui。

```
[root@centos8 tomcat]#vim conf/tomcat-users.xml
<tomcat-users xmlns="http://tomcat.apache.org/xml"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
              version="1.0">
#加下面两行，指定用户和密码
  <role rolename="manager-gui"/>
  <user username="admin" password="123456" roles="manager-gui"/>
</tomcat-users>

#修改全局配置文件需要重启服务生效
[root@centos8 tomcat]#systemctl restart tomcat
```

- **修改webapps/manager/META-INF/context.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<Context antiResourceLocking="false" privileged="true" >
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
  <Manager sessionAttributeValueClassNameFilter="java\.lang\.
(?:Boolean|Integer|Long|Number|String)|org\.apache\.catalina\.filters\.CsrfPreve
ntionFilter\$LruCache(?:\$1)?|java\.util\.(?:Linked)?HashMap"/>
</Context>
```

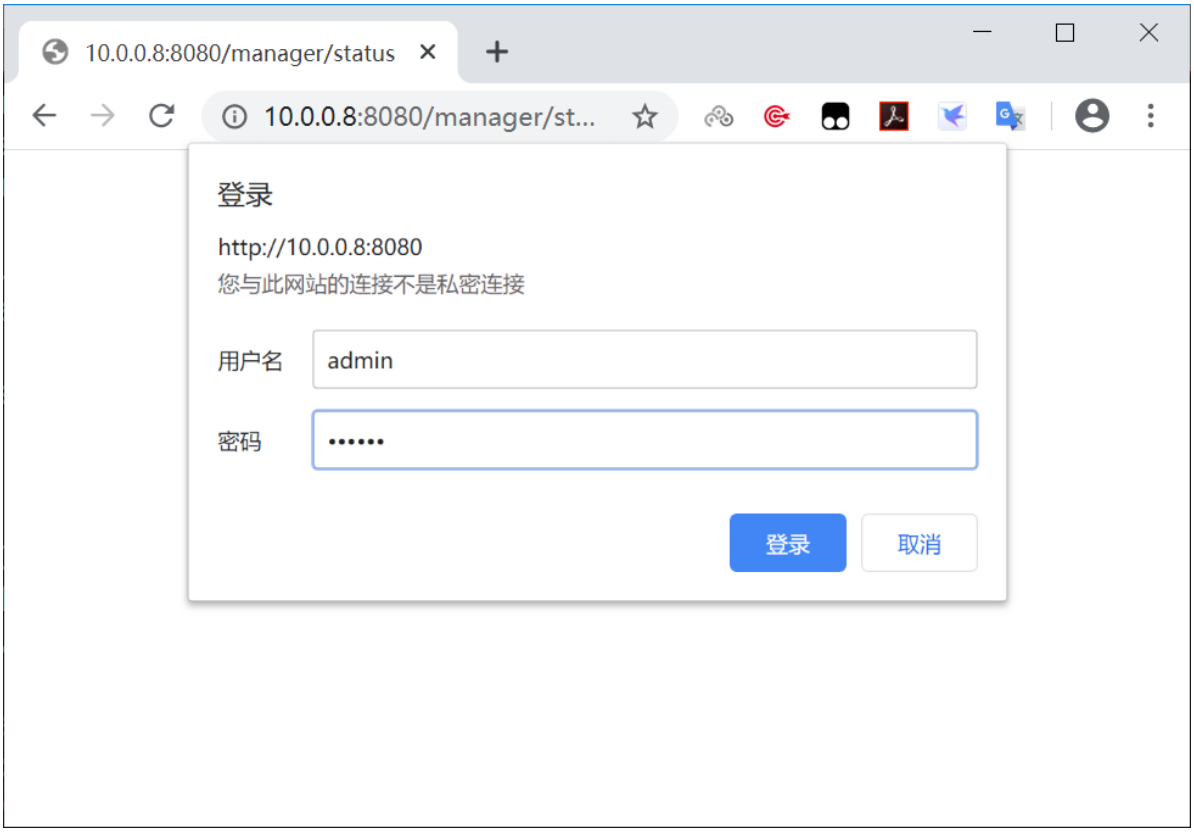
查看正则表达式就知道是本地访问了，由于当前访问地址是192.168.x.x，可以修改正则表达式为

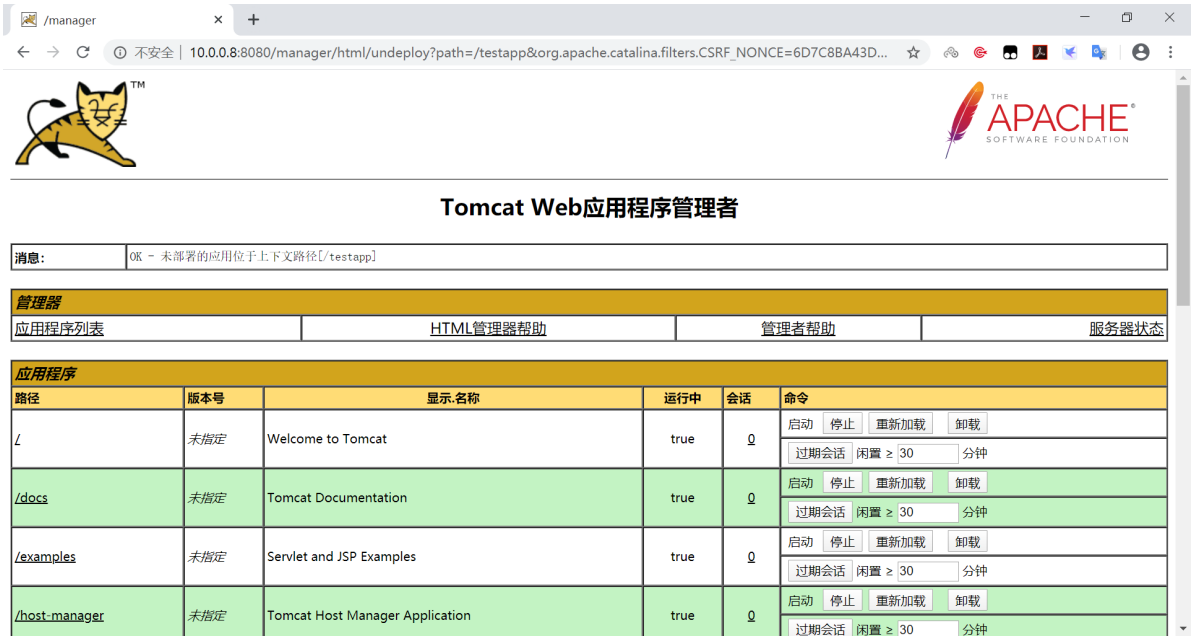
```
allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1|192\.168\.\d+\.\d+"
```

范例：

```
[root@centos8 tomcat]#vim webapps/manager/META-INF/context.xml
<Context antiResourceLocking="false" privileged="true" >
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1|10\.0\.0\.\d+" />
  <Manager sessionAttributeValueClassNameFilter="java\.lang\.
(?:Boolean|Integer|Long|Number|String)|org\.apach
e\.catalina\.filters\.CsrfPreventionFilter\$LruCache(?:\$1)?|java\.util\.
(?:Linked)?HashMap"/>
</Context>

#修改webApp的配置无需重启服务即可生效
```

- 再次通过浏览器访问两个按钮Server Status和Manager App，可以看到以下管理界面，输入前面的用户和密码进行登录





3.4.4.8.2 基于WEB应用程序管理器实现APP的部署

Web 应用程序管理界面可以实现以下功能

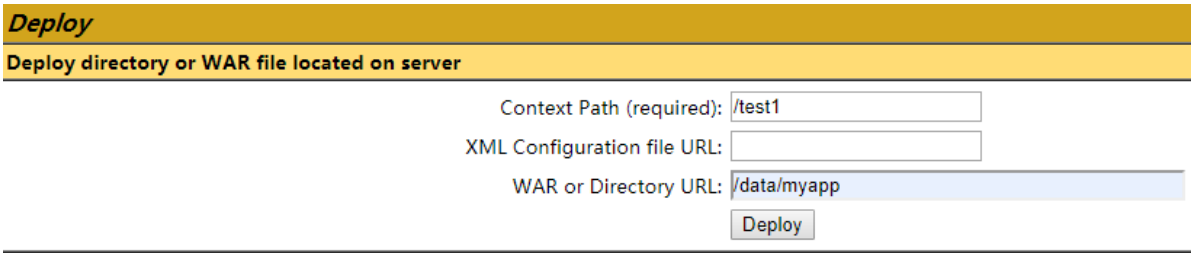
Applications 应用程序管理，可以启动、停止、重加载、反部署、清理过期session

Deploy 可以热部署，也可以部署war文件。

方式1: 指定目录部署软件

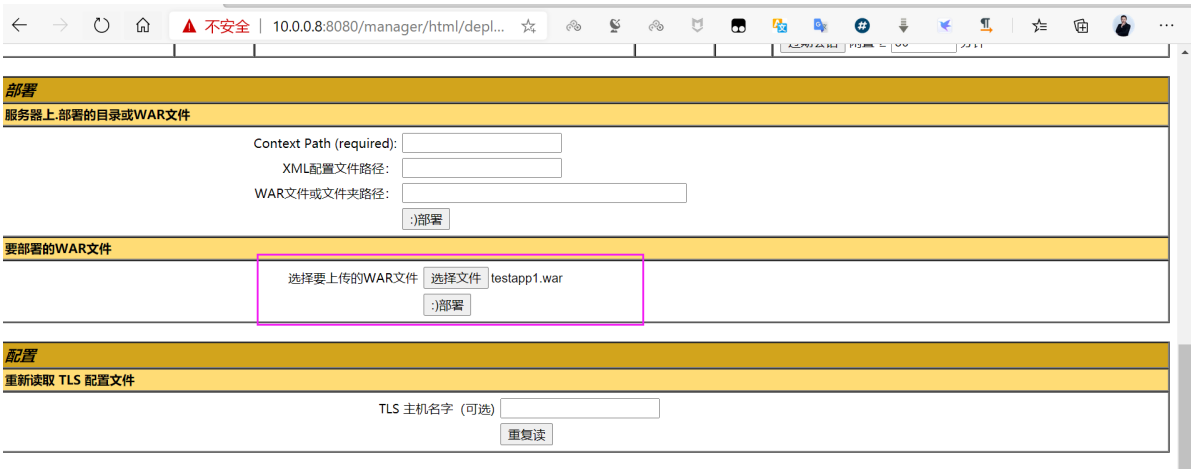
```
[root@centos8 ~]#mkdir -p /data/myapp/
[root@centos8 ~]#echo /data/myapp/index.html > /data/myapp/index.html

#按下面信息添写,实现下面链接的访问
http://10.0.0.8:8080/test1/
```

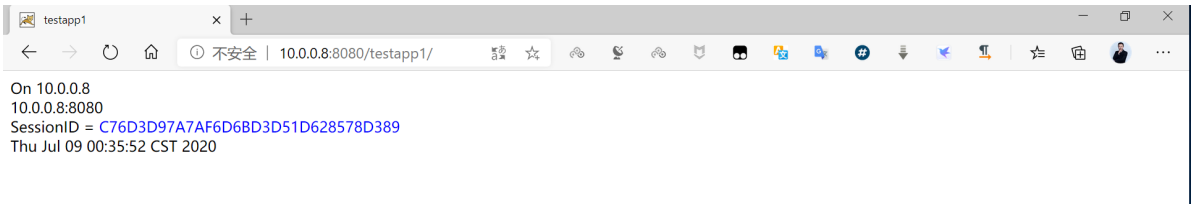
```
Context Path (required): 指定通过浏览器访问的虚拟目录
WAR or Directory URL: 指定真正存放文件的实际磁盘目录路径
```




```
#自动将/data/myapp目录下的数据复制到webapps/test1下面
[root@centos8 ~]#tree /usr/local/tomcat/webapps/test1/
/usr/local/tomcat/webapps/test1/
└── index.html

0 directories, 1 file
[root@centos8 ~]#cat /usr/local/tomcat/webapps/test1/index.html
/data/myapp/index.html
```

方式2: 部署war包文件





3.4.5 常见配置详解

3.4.5.1 端口8005/tcp 安全配置管理

在conf/server.xml 有以下内容

```
<?xml version="1.0" encoding="UTF-8"?>
<Server port="8005" shutdown="SHUTDOWN">
  <Service name="Catalina">
    <Connector port="8080" protocol="HTTP/1.1"
```

```
               connectionTimeout="20000"
               redirectPort="8443" />
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

    <Engine name="Catalina" defaultHost="localhost">
      <Host name="localhost"  appBase="webapps"
            unpackWARs="true" autoDeploy="true">
      </Host>
    </Engine>
  </Service>
</Server>
```

```
<Server port="8005" shutdown="SHUTDOWN">
```

8005是Tomcat的管理端口，默认监听在127.0.0.1上。无需验证就可发送SHUTDOWN (大小写敏感)这个字符串，tomcat接收到后就会关闭此Server。

此管理功能建议禁用，可将SHUTDOWN改为一串猜不出的字符串实现

或者port修改成 0, 会使用随机端口,如:36913

port设为-1等无效端口,将关闭此功能，注意：-2等不支持

此行不能被注释,否则无法启动tomcat服务

范例:

```
<Server port="8005" shutdown="44ba3c71d57f494992641b258b965f28">
```

范例：修改8005/tcp端口管理命令

```
[root@centos8 ~]#ss -ntl
State       Recv-Q        Send-Q                    Local Address:Port
    Peer Address:Port
LISTEN      0             128                            0.0.0.0:22
          0.0.0.0:*
LISTEN      0             100                                *:8080
               *:*
LISTEN      0             128                               [::]:22
              [::]:*          LISTEN      0             1
[::ffff:127.0.0.1]:8005                          *:*          LISTEN      0
        100                                   *:8009
*:*
[root@centos8 ~]#telnet 127.0.0.1 8005
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
SHUTDOWN                            #执行命令关闭tomcat
Connection closed by foreign host.

[root@centos8 ~]#ss -ntl
State        Recv-Q        Send-Q                Local Address:Port
    Peer Address:Port
LISTEN       0             128                        0.0.0.0:22
         0.0.0.0:*
LISTEN       0             128                           [::]:22
            [::]:*
```

```
[root@centos8 tomcat]#vim conf/server.xml
<Server port="8005" shutdown="magedu">
[root@centos8 tomcat]#systemctl start tomcat
[root@centos8 tomcat]#telnet 127.0.0.1 8005
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
SHUTDOWN
Connection closed by foreign host.
[root@centos8 tomcat]#ss -ntl
State       Recv-Q        Send-Q                    Local Address:Port
    Peer Address:Port
LISTEN      0             128                             0.0.0.0:22
          0.0.0.0:*
LISTEN      0             100                                 *:8080
                *:*
LISTEN      0             128                               [::]:22
              [::]:*
LISTEN      0             1                    [::ffff:127.0.0.1]:8005
                *:*
LISTEN      0             100                                 *:8009
                *:*

[root@centos8 tomcat]#telnet 127.0.0.1 8005
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
magedu
Connection closed by foreign host.
[root@centos8 tomcat]#ss -ntl
State        Recv-Q        Send-Q                Local Address:Port
    Peer Address:Port
LISTEN       0             128                        0.0.0.0:22
         0.0.0.0:*
LISTEN       0             128                           [::]:22
            [::]:*

[root@centos8 tomcat]#
```

3.4.5.2 显示指定的http服务器版本信息

默认不显示tomcat的http的Server头信息, 可以指定tomcat的http的Server头信息为相应的值

```
#conf/server.xml
<Connector port="8080" protocol="HTTP/1.1"  connectionTimeout="20000"
redirectPort="8443" Server="SOME STRING"/>
```

范例:

```
[root@centos8 ~]#curl -I 127.0.0.1:8080
HTTP/1.1 200
Content-Type: text/html;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 17 Jul 2020 08:32:52 GMT
```

```
#修改配置,指定想显示的tomcat版本
[root@centos8 ~]#vim /usr/local/tomcat/conf/server.xml
......
  <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" Server="WangServer"/>
......

[root@centos8 ~]#systemctl restart tomcat
[root@centos8 ~]#curl 127.0.0.1:8080 -I
HTTP/1.1 200
Content-Type: text/html;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 17 Jul 2020 08:34:17 GMT
Server: WangServer
```

3.4.5.3 其它配置

conf/server.xml中可以配置service，connector, Engine,Host等

- service配置

一般情况下，一个Server实例配置一个Service，name属性相当于该Service的ID。

```
<Service name="Catalina">
```

- 连接器配置

```
    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
```

redirectPort，如果访问HTTPS协议，自动转向这个连接器。但大多数时候，Tomcat并不会开启HTTPS，因为Tomcat往往部署在内部，HTTPS性能较差

- 引擎配置

```
<Engine name="Catalina" defaultHost="localhost">
```

- defaultHost 配置

defaultHost指向内部定义某虚拟主机。缺省虚拟主机可以改动，默认localhost。

```
<Host name="localhost"  appBase="webapps" unpackWARs="true" autoDeploy="true">
```

3.4.5.4 多虚拟主机配置

3.4.5.4.1 多虚拟主机配置说明

- name 必须是主机名，用主机名来匹配
- appBase 当前主机的网页根目录，是相对于`$CATALINA_HOME`，也可以使用绝对路径
- unpackWARs 是否自动解压war格式
- autoDeploy 热部署，自动加载并运行应用

虚拟主机配置过程

- 再添加和配置一个新的虚拟主机，并将myapp部署到/data/webapps目录下

```
vim conf/server.xml
#在文件最后面增加下面内容
<Host name="web1.magedu.org" appBase="/data/webapps/" unpackWARs="True"
autoDeploy="false">

#虚拟主机专有访问日志
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
prefix="web1_access_log" suffix=".txt" pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s
%b" />
</Host>

#以下行是自带的不需要修改
</Engine>
</Service>
</Server>

#或者如果不加日志也可以用下面简化写法
<Host name="web1.magedu.org" appBase="/data/webapps/" unpackWARs="True"
autoDeploy="false"/>
```

- 准备虚拟主机的数据目录

```
常见虚拟主机根目录
# mkdir /data/webapps/ROOT -pv
# chown -R tomcat.tomcat /data/webapps
# echo web1.magedu.org > /data/webapps/ROOT/index.html
```

- 测试

刚才在虚拟主机中主机名定义node1.magedu.org，所以需要主机在本机手动配置一个**域名解析**。

如果是windows，修改在C:\Windows\System32\drivers\etc下的hosts文件，需要管理员权限。

使用http://web1.magedu.org:8080/访问查看

3.4.5.4.2 实战案例：tomcat实现多虚拟主机

```
[root@centos8 tomcat]#pwd
/usr/local/tomcat
[root@centos8 tomcat]#vim conf/server.xml
[root@centos8 tomcat]#tail conf/server.xml
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
      </Host>
#添加了以下行
       <Host name="node1.magedu.org"  appBase="/data/webapps1/"
unpackWARs="true" autoDeploy="true">
```

```
            <Valve className="org.apache.catalina.valves.AccessLogValve"
directory="logs"
               prefix="node1_access_log" suffix=".txt"
                pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
       </Host>
       <Host name="node2.magedu.org"  appBase="/data/webapps2/"
unpackWARs="true" autoDeploy="true">
       <Valve className="org.apache.catalina.valves.AccessLogValve"
directory="logs"
               prefix="node2_access_log" suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
       </Host>
    </Engine>
  </Service>
</Server>

#对每个虚拟主机，准备数据
[root@centos8 ~]#mkdir /data/webapps{1,2}/ROOT -pv
mkdir: created directory '/data/webapps1/ROOT'
mkdir: created directory '/data/webapps2/ROOT'

[root@centos8 ~]#cat /data/webapps1/ROOT/index.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
<br>
<%=request.getRequestURL()%>
</body>
</html>
[root@centos8 ~]#cat /data/webapps2/ROOT/index.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
<br>
<%=request.getRequestURL()%>
</body>
</html>
[root@centos8 ~]#

#设置权限
[root@centos8 ~]#chown -R tomcat.tomcat /data/webapps{1,2}/

#准备虚拟主机的名称解析
[root@centos8 ~]#cat /etc/hosts
```

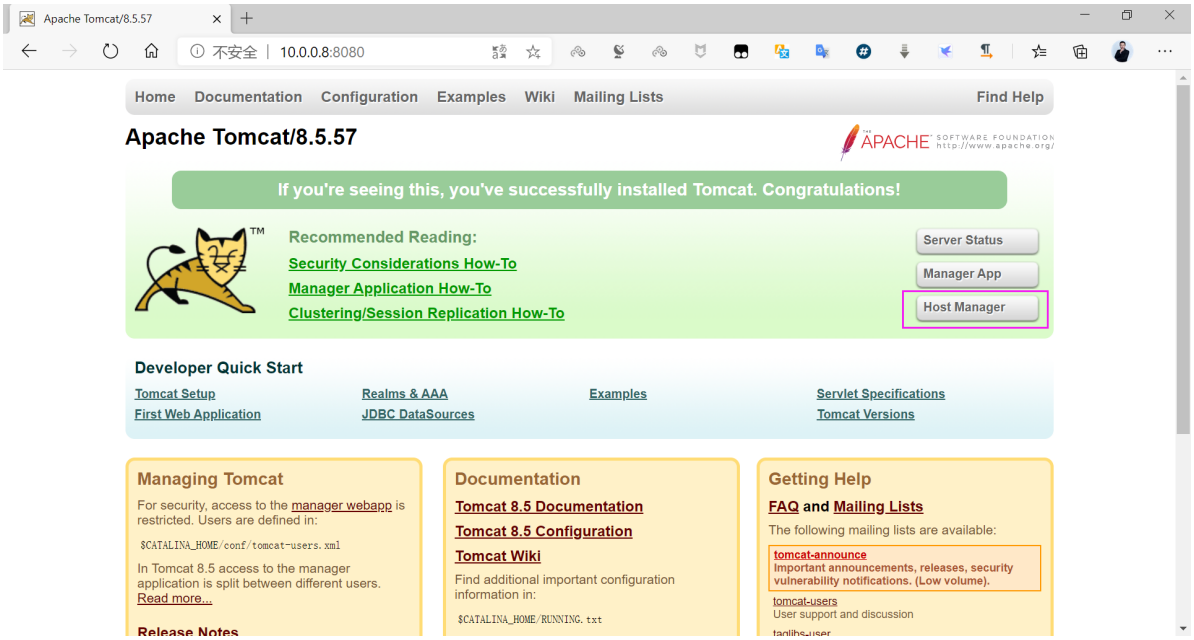
```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
centos8.localdomain
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.0.0.8 node1.magedu.org node2.magedu.org

[root@centos8 ~]#curl http://node1.magedu.org:8080/
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
http://node1.magedu.org:8080/

</body>
</html>
[root@centos8 ~]#curl http://node2.magedu.org:8080/

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
http://node2.magedu.org:8080/

</body>
</html>
```

3.4.5.4.3 实战案例：修改tomcat实现多虚拟主机的端口为80

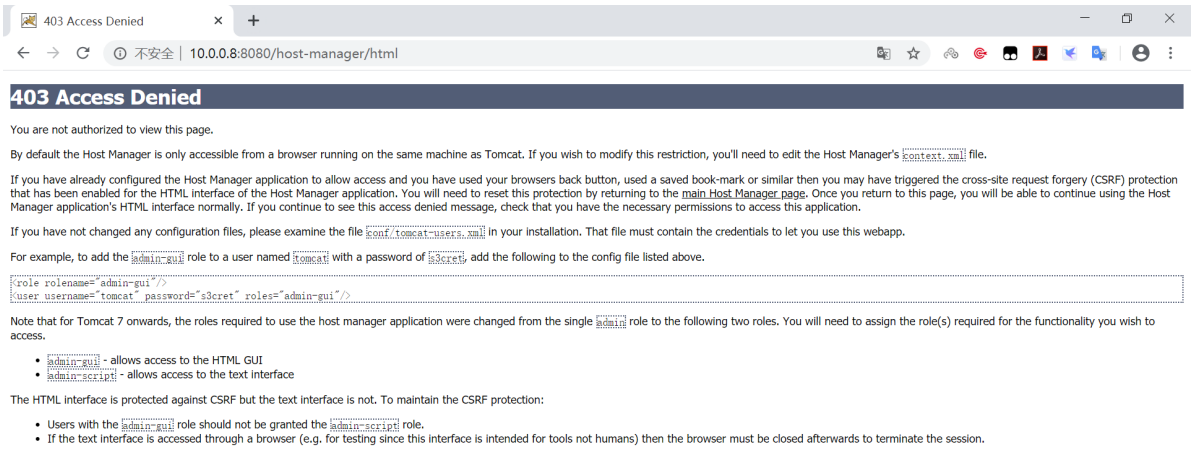
```
 [root@centos8 ~]#vim /usr/local/tomcat/conf/server.xml
 <Connector port="80" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />

#注意：因为以tomcat用户运行,不能直接使用1024以下的端口,需要修改tomcat的运行身份,否则会出现下面错误
[root@centos8 ~]#tail -f /usr/local/tomcat/logs/catalina.out
    Caused by: java.net.SocketException: Permission denied


[root@centos8 ~]#vim /lib/systemd/system/tomcat.service
[Service]
.....
#User=tomcat
#Group=tomcat

[root@centos8 ~]#systemctl daemon-reload
[root@centos8 ~]#systemctl restart tomcat
```

3.4.5.5 基于web方式的Host Manager虚拟主机管理

可以通过tomcat的管理页面点下面Host Manager按钮进入管理虚拟主机的页面



默认Host Manager 管理页被禁用，会出现下面提示，解决方法类似于**3.4.4.6**

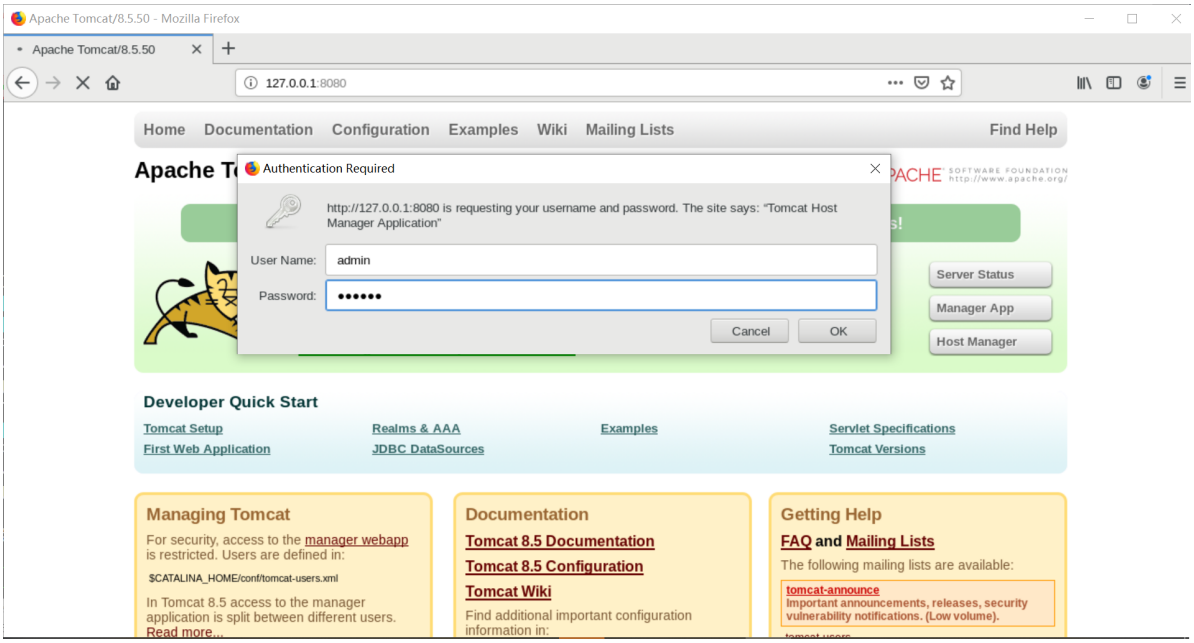


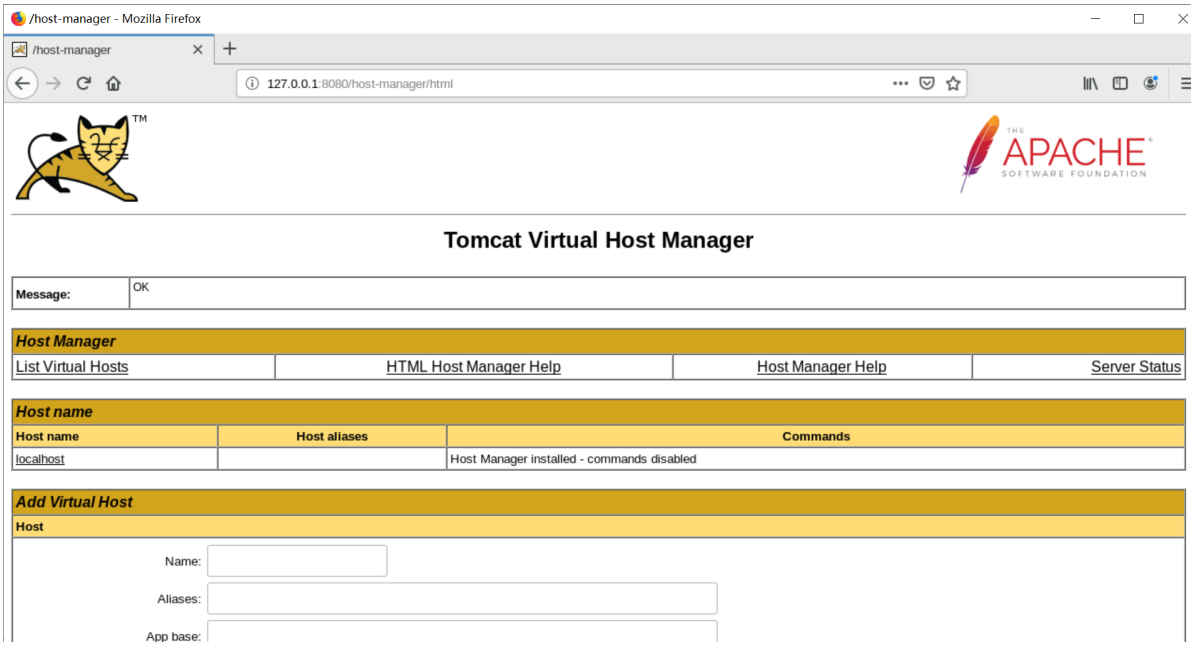
3.4.5.5.1 允许本机访问

配置如下

```
[root@centos8 tomcat]#vim conf/tomcat-users.xml
<tomcat-users xmlns="http://tomcat.apache.org/xml"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
              version="1.0">
    <role rolename="manager-gui"/>  #3.4.4.6添加的内容
    <role rolename="admin-gui" />   #添加新的role
    <user username="admin" password="123456" roles="manager-gui,admin-gui"/> #再
加新role

</tomcat-users>
[root@centos8 tomcat]#systemctl restart tomcat
```

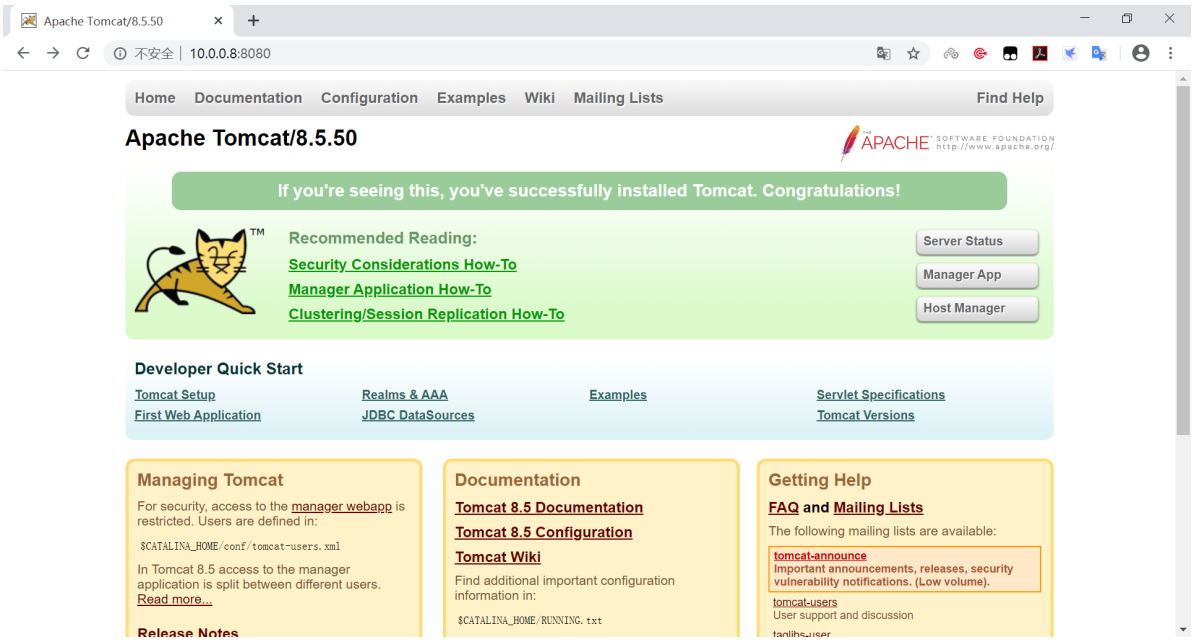
重启Tomcat后，点击"Host Manager"按钮



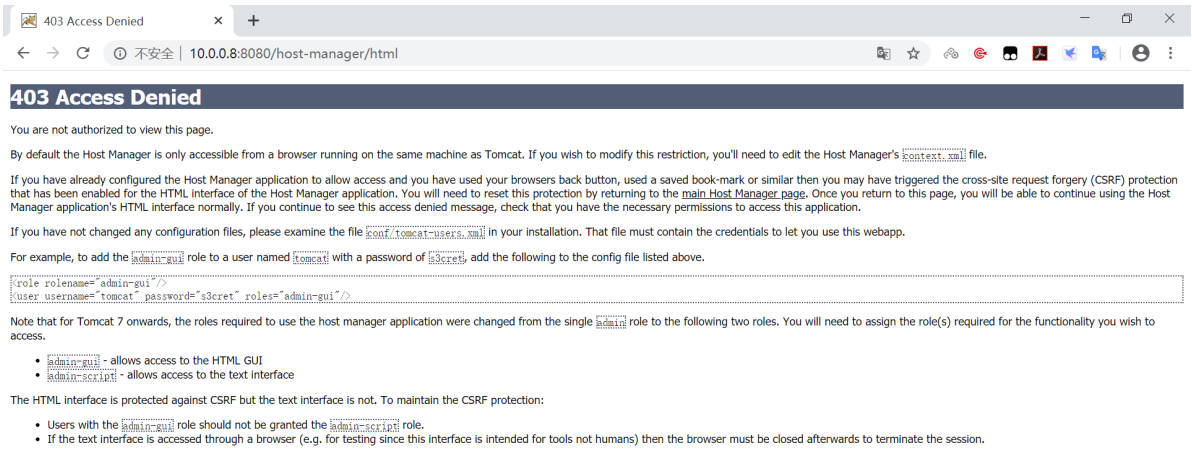


3.4.5.5.2 允许远程主机访问

但通过远程访问地址仍无法访问Host Manager管理页面

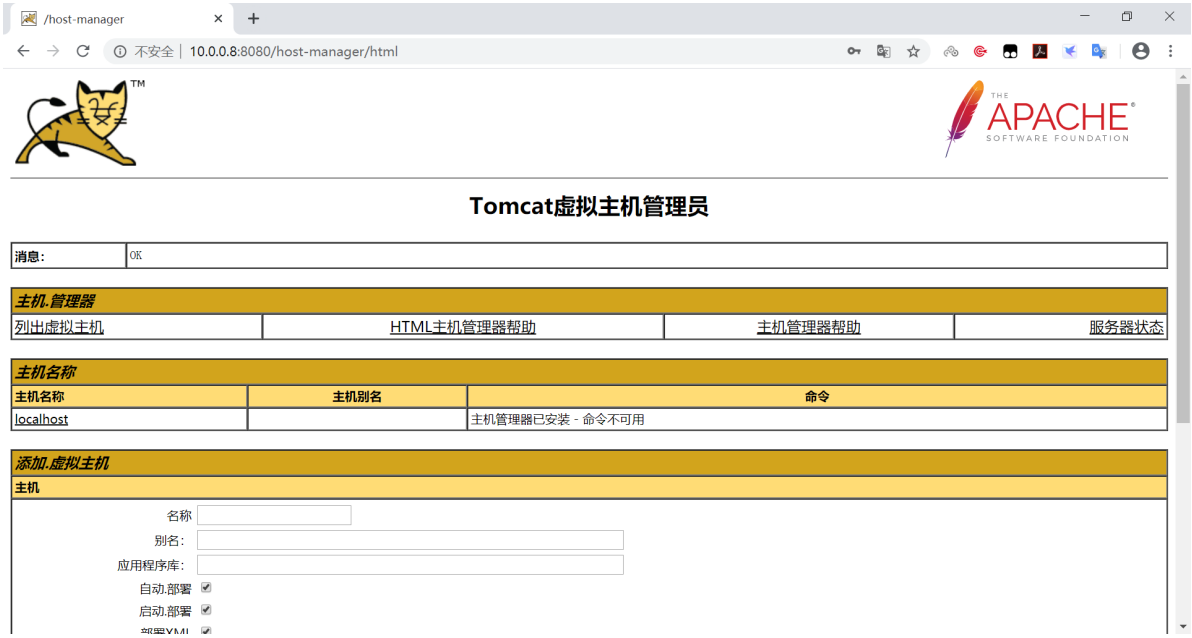


默认无法通过网络远程访问Host Manager管理页面，默认会出现以下界面




```
[root@centos8 tomcat]#vim webapps/host-manager/META-INF/context.xml
<Context antiResourceLocking="false" privileged="true" >
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1|10\.0\.0\.\d+" />
  <Manager sessionAttributeValueClassNameFilter="java\.lang\.
(?:Boolean|Integer|Long|Number|String)|org\.apach
e\.catalina\.filters\.CsrfPreventionFilter\$LruCache(?:\$1)?|java\.util\.
(?:Linked)?HashMap"/>
</Context>
```

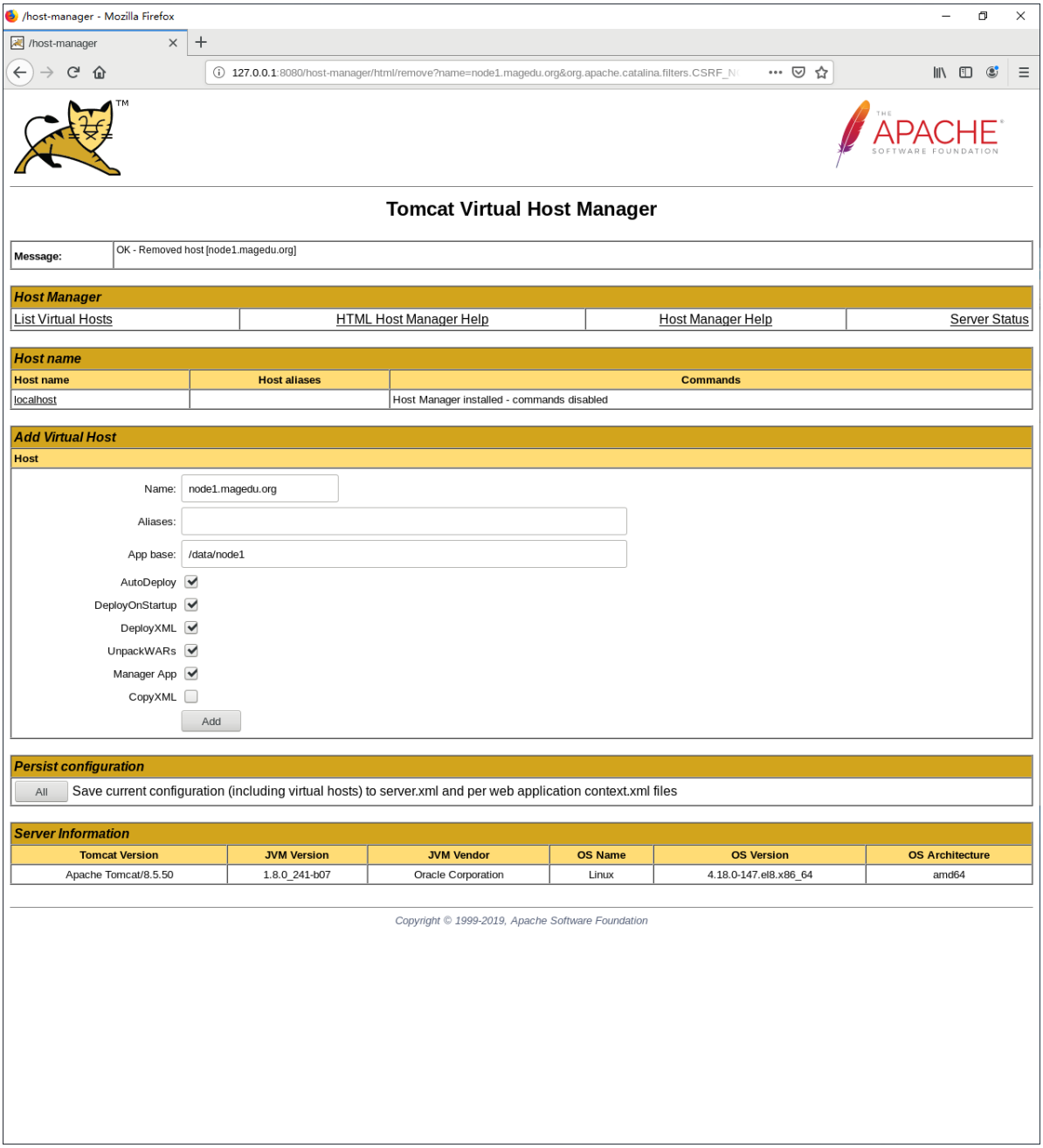
无需重启服务，直接访问，输入前面的用户和密码，即可登录成功

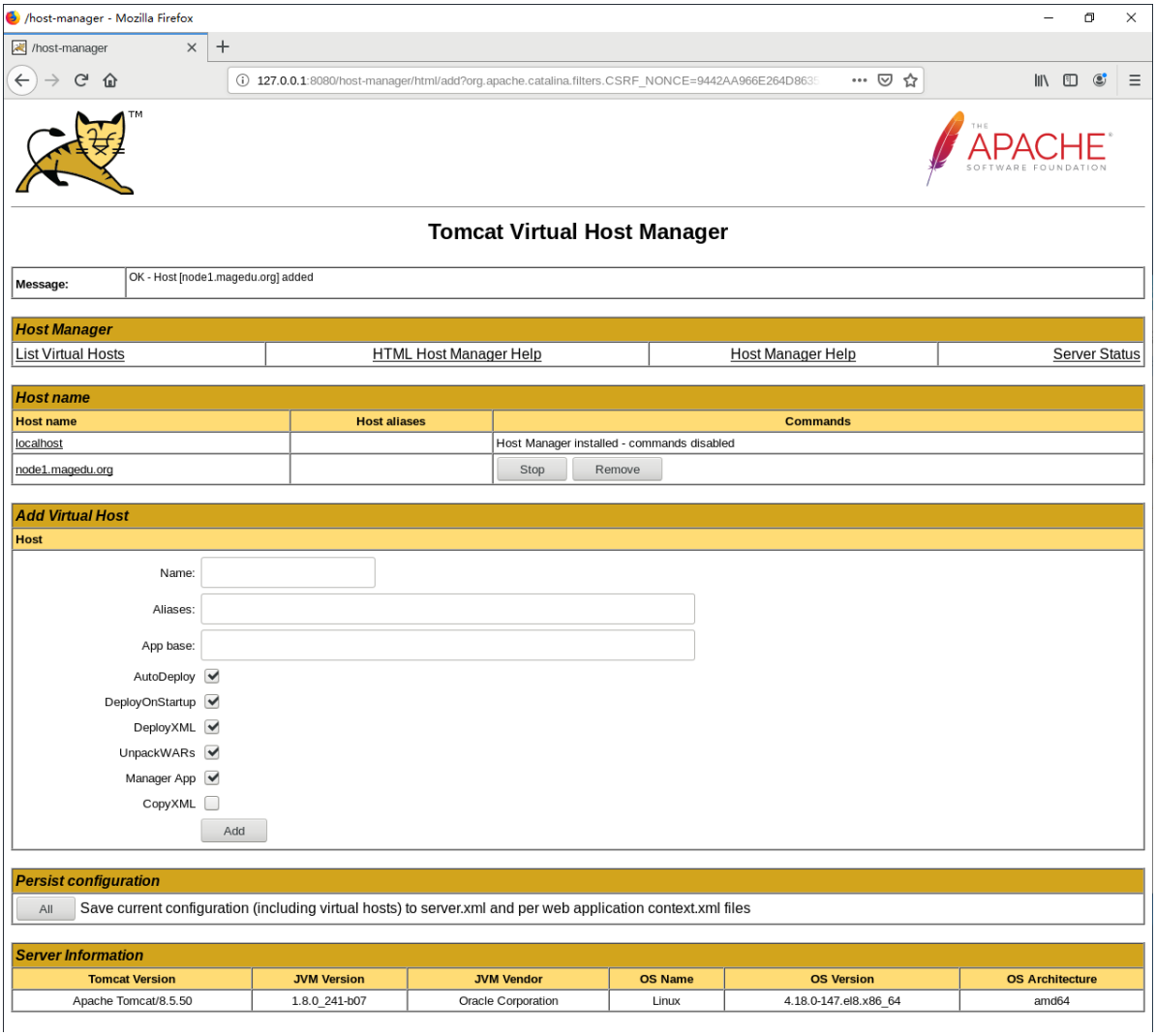


3.4.5.5.3 创建新的虚拟主机

可以管理虚拟主机

```
#创建虚拟主机前,必须先创建相关目录,否则创建虚拟机不成功
[root@centos8 ~]#mkdir /data/node1/ROOT/
[root@centos8 ~]#echo node1.magedu.org >  /data/node1/ROOT/index.html
[root@centos8 ~]#chown -R tomcat.tomcat /data/node1/
```


/host-manager - Mozilla Firefox
/host-manager
127.0.0.1:8080/host-manager/html/remove?name=node1.magedu.org&org.apache.catalina.filters.CSRF_N
Tomcat Virtual Host Manager
Message:
OK - Removed host [node1.magedu.org]
Host Manager
List Virtual Hosts
HTML Host Manager Help
Host Manager Help
Server Status
Host name
Host name
Host aliases
Commands
localhost
Host Manager installed - commands disabled
Add Virtual Host
Host
Name:
node1.magedu.org
Aliases:
App base:
/data/node1
AutoDeploy
DeployOnStartup
DeployXML
UnpackWARs
Manager App
CopyXML
Add
Persist configuration
All
Save current configuration (including virtual hosts) to server.xml and per web application context.xml files
Server Information
Tomcat Version
JVM Version
JVM Vendor
OS Name
OS Version
OS Architecture
Apache Tomcat/8.5.50
1.8.0_241-b07
Oracle Corporation
Linux
4.18.0-147.el8.x86_64
amd64
Copyright © 1999-2019, Apache Software Foundation



3.4.5.5 Context 配置

3.4.5.5.1 Centext 配置方式

Context作用：

- 路径映射：将url映射至指定路径，而非使用appBase下的物理目录，实现虚拟目录功能
- 应用独立配置，例如单独配置应用日志、单独配置应用访问控制

```
#映射指定路径
<Context path="/test" docBase="/data/test" reloadable="true" />

#映射站点的根目录
<Context path="/" docBase="/data/website" reloadable="true" />

#还可以添加日志等独立的配置
<Context path="/test" docBase="/data/test" reloadable="true" >
  <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
              prefix="localhost_test_log" suffix=".txt"
              pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
</Context>
```

说明：

- path：指的是访问的URL路径，如果path与appBase下面的子目录同名，context的docBase路径优先级更高
- docBase：可以是磁盘文件的绝对路径，也可以是相对路径（相对于Host的appBase）
- reloadable：true表示如果WEB-INF/classes或META-INF/lib目录下.class文件有改动，就会将WEB应用重新加载。生产环境中，建议使用false来禁用。

Centext实现过程

- 将~/projects/myapp/下面的项目文件复制到/data/下，可以修改一下index.jsp 区别一下

```
# cp -r ~/projects/myapp /data/myapp-v1
# vim /data/myappv1/index.jsp
# cd /data
# ln -sv myapp-v1 test
```

注意：这里特别使用了软链接，原因方便后期版升级或回滚，如是是版本升级，需要将软链接指向myappv2，重新启动。如果新版上线后，出现问题，重新修改软链接到上一个版本的目录，并重启，就可以实现回滚

- 修改conf/server.xml设置context

Tomcat的配置文件server.xml中修改如下，重启Tomcat生效

```
      <Host name="node1.magedu.com"  appBase="/data/webapps"
            unpackWARs="true" autoDeploy="true" >
        <Context path="/test" docBase="/data/test" reloadable="true" />
      </Host>
```

- 测试

  使用http://node1.magedu.com:8080/test/

3.4.5.5.2 Valve组件

日志格式说明

```
http://tomcat:8080/docs/config/valve.html#Access_Logging
```

valve(阀门)组件可以定义日志

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
            prefix="localhost_access_log" suffix=".txt"
            pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
```

valve存在多种类型：

```
定义访问日志：org.apache.catalina.valves.AccessLogValve
定义访问控制：org.apache.catalina.valves.RemoteAddrValve
```

示例:

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
deny="10\.0\.0\.\d+"/>
```

3.4.5.5.3 实战案例

范例：虚拟主机上利用context实现虚拟目录

```
#在前面范例的基础上实现，继续创建node1.magedu.org虚拟主机下的物理子目录
[root@centos8 ~]#mkdir /data/webapps1/app1/
[root@centos8 ~]#echo /data/webapps1/app1/index.html >
/data/webapps1/app1/index.html
[root@centos8 ~]#curl http://node1.magedu.org:8080/app1/
/data/webapps1/app1/index.html

#利用context实现node1.magedu.org虚拟主机下的虚拟子目录
[root@centos8 tomcat]#vim conf/server.xml
[root@centos8 tomcat]#tail conf/server.xml
      </Host>
      <Host  name="node1.magedu.org"  appBase="/data/webapps1">
      #加下面六行
      <Context path="/app1" docBase="/data/app1" reloadable="true" >
        <Valve className="org.apache.catalina.valves.AccessLogValve"
directory="logs"
               prefix="node1.magedu.org_app1" suffix=".log"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
       <Valve className="org.apache.catalina.valves.RemoteAddrValve"
deny="10\.0\.0\.7"/>
       </Context>

      </Host>
      <Host  name="node2.magedu.org"  appBase="/data/webapps2">
      </Host>
    </Engine>
  </Service>
</Server>
[root@centos8 tomcat]#systemctl restart tomcat

#因数据没有准备好，出现下面错误
[root@centos8 tomcat]#curl http://node1.magedu.org:8080/app1/
curl: (7) Failed to connect to node1.magedu.org port 8080: Connection refused

#准备数据目录
[root@centos8 tomcat]#mkdir /data/app1-v1
```

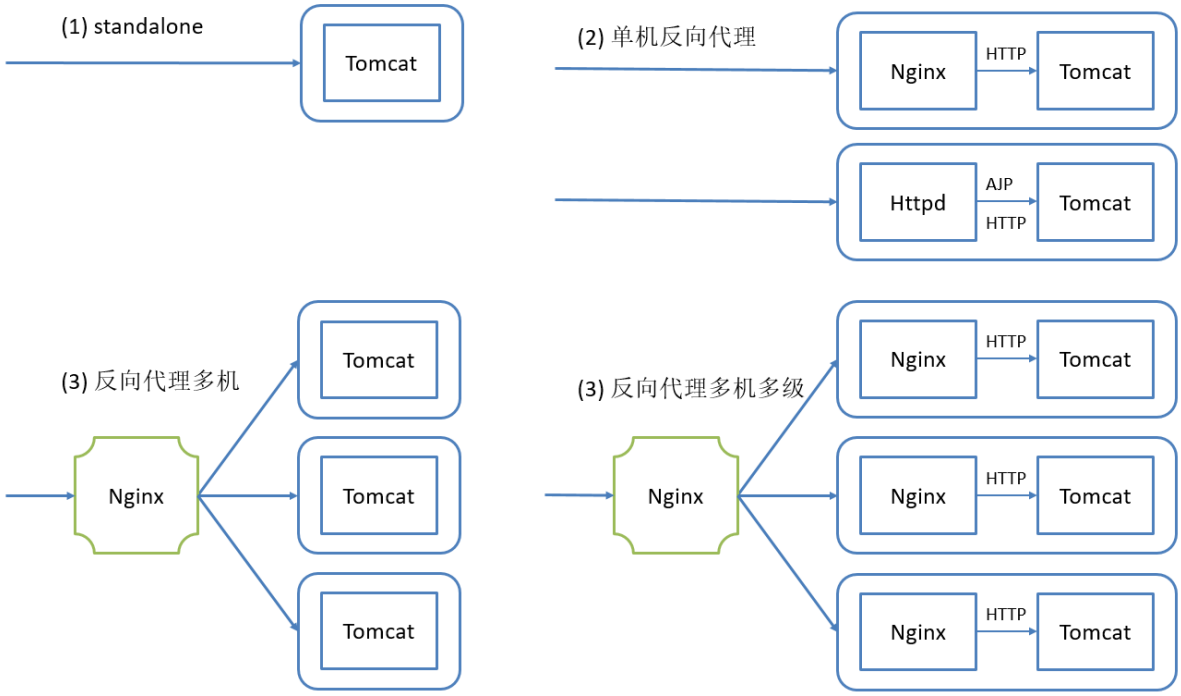
```
[root@centos8 tomcat]#echo /data/app1-v1/index.html > /data/app1-v1/index.html
[root@centos8 tomcat]#ln -s /data/app1-v1/ /data/app1
[root@centos8 tomcat]#curl http://node1.magedu.org:8080/app1/
curl: (7) Failed to connect to node1.magedu.org port 8080: Connection refused

#数据目录准备好，还需要重新启动服务，才能访问
[root@centos8 tomcat]#systemctl restart tomcat
[root@centos8 tomcat]#curl http://node1.magedu.org:8080/app1/
/data/app1-v1/index.html
[root@centos7 ~]#curl -I  http://node1.magedu.org:8080/app1/
HTTP/1.1 403
Content-Type: text/html;charset=utf-8
Content-Language: en
Transfer-Encoding: chunked
Date: Tue, 14 Jul 2020 06:48:54 GMT


#可以看到此目录单独的访问日志
[root@centos8 ~]#cat /usr/local/tomcat/logs/node1.magedu.org_app1.2020-07-14.log

10.0.0.8 - - [14/Jul/2020:14:36:01 +0800] "GET /app1/ HTTP/1.1" 200 330
10.0.0.7 - - [14/Jul/2020:14:48:07 +0800] "GET /app1/ HTTP/1.1" 403 618
```

范例: 基于前面环境,实现软件升级和回滚功能

```
#升级版本
[root@centos8 tomcat]#mkdir /data/app1-v2
[root@centos8 tomcat]#echo /data/app1-v2/index.html > /data/app1-v2/index.html
[root@centos8 tomcat]#rm -f /data/app1

#删除软链接，仍然可以访问旧版本
[root@centos8 tomcat]#curl http://node1.magedu.org:8080/app1/
/data/app1-v1/index.html

#重新服务后，出现错误
[root@centos8 tomcat]#systemctl restart tomcat
[root@centos8 tomcat]#curl http://node1.magedu.org:8080/app1/
curl: (7) Failed to connect to node1.magedu.org port 8080: Connection refused

#新建软链接，指向新版，仍需重启服务才生效
[root@centos8 tomcat]#ln -s /data/app1-v2/ /data/app1
[root@centos8 tomcat]#curl http://node1.magedu.org:8080/app1/
curl: (7) Failed to connect to node1.magedu.org port 8080: Connection refused
[root@centos8 tomcat]#systemctl restart tomcat
[root@centos8 tomcat]#curl http://node1.magedu.org:8080/app1/
/data/app1-v2/index.html

#软件降级或回滚
[root@centos8 tomcat]#rm -f /data/app1
[root@centos8 tomcat]#ln -s /data/app1-v1/ /data/app1
[root@centos8 tomcat]#systemctl restart tomcat
[root@centos8 tomcat]#curl http://node1.magedu.org:8080/app1/
/data/app1-v1/index.html
```

4 结合反向代理实现tomcat部署

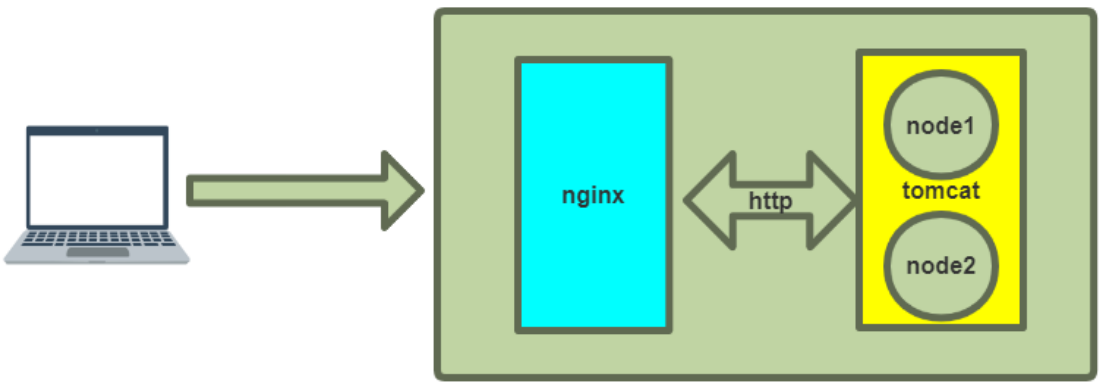
4.1 常见部署方式介绍



- standalone模式，Tomcat单独运行，直接接受用户的请求，不推荐。
- 反向代理，单机运行，提供了一个Nginx作为反向代理，可以做到静态由nginx提供响应，动态jsp代理给Tomcat
  - LNMT：Linux + Nginx + MySQL + Tomcat
  - LAMT：Linux + Apache（Httpd）+ MySQL + Tomcat
- 前置一台Nginx，给多台Tomcat实例做反向代理和负载均衡调度，Tomcat上部署的纯动态页面更适合
  - LNMT：Linux + Nginx + MySQL + Tomcat
- 多级代理
  - LNNMT：Linux + Nginx + Nginx + MySQL + Tomcat

4.2 利用 nginx 反向代理实现全部转发置指定同一个虚拟主机

4.2.1 配置说明



利用nginx反向代理功能，实现上图的代理功能，将用户请求全部转发至指定的同一个tomcat主机

利用nginx指令proxy_pass 可以向后端服务器转发请求报文,并且在转发时会保留客户端的请求报文中的host首部

```
#从yum源安装nginx
#yum install nginx -y
#vim /etc/nginx/nginx.conf
#全部反向代理测试
location / {
    # proxy_pass http://127.0.0.1:8080; # 不管什么请求，都会访问后面的localhost虚拟主机
    proxy_pass http://node1.magedu.com:8080; # 此项将用户访问全部请求转发到node1的虚拟主机上
    #proxy_pass http://node2.magedu.com:8080; #此项将用户访问全部请求转发到node2的虚拟主机上
    #proxy_set_header Host  $http_host; #转发主机头至后端服务器

    #以上两项都需要修改nginx服务器的/etc/hosts,实现node1.magedu.com和node2.magedu.com到IP的解析
}

#nginx -t
#systemctl restart nginx

#说明: proxy_pass http://FQDN/ 中的FQDN 决定转发至后端哪个虚拟主机,而与用户请求的URL无关
#如果转到后端的哪个服务器由用户请求决定,可以向后端服务转发请求的主机头实现,示例:
proxy_set_header Host  $http_host;
```

4.2.2 实战案例1

环境说明:

```
一台主机,实现nginx和tomcat
tomcat上有两个虚拟主机node1和node2
```

```
#先按3.4.5.4介绍方式在同一下主机上建立两个tomcat虚拟主机，node1.magedu.org和node2.magedu.org
#修改/etc/hosts文件，实现名称解析
[root@centos8 ~]#vim /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4 centos8.localdomain
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.0.0.100 node1.magedu.org node2.magedu.org

#安装nginx
[root@centos8 ~]#yum -y install nginx

#修改nginx.conf配置文件
[root@centos8 ~]#vim /etc/nginx/nginx.conf
......
#修改location / 此行，添加以下内容
location / {
            #proxy_pass http://127.0.0.1:8080;
            proxy_pass http://node1.magedu.org:8080;
            #proxy_set_header Host  $http_host; #转发主机头至后端服务器
            }
......
```

```
[root@centos8 ~]#systemctl enable --now  nginx

#先别访问node1,node2和IP都可以看到一样的node1的虚拟主机页面
[root@centos8 ~]#curl http://node1.magedu.org/

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
node1.magedu.org

</body>
</html>
[root@centos8 ~]#curl http://node2.magedu.org/

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
node2.magedu.org

</body>
</html>
[root@centos8 ~]#curl http://127.0.0.1/

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
node1.magedu.org

</body>
</html>
[root@centos8 ~]#curl http://10.0.0.100/

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
node1.magedu.org
```

```
</body>
</html>


[root@centos8 ~]#systemctl restart nginx
#再次修改nginx.conf配置文件
[root@centos8 ~]#vim /etc/nginx/nginx.conf
......
#修改location /  行，添加以下内容
location / {
            #proxy_pass http://127.0.0.1:8080;
            proxy_pass http://node2.magedu.org:8080;
            #proxy_set_header Host  $http_host; #转发主机头至后端服务器
            }
......


#先别访问node1,node2和IP都可以看到一样的node2的虚拟主机页面
[root@centos8 ~]#curl http://node1.magedu.org/
[root@centos8 ~]#curl http://node2.magedu.org/
[root@centos8 ~]#curl http://127.0.0.1/
[root@centos8 ~]#curl http://10.0.0.100/

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
node2.magedu.org

</body>
</html>
```

4.2.3 实战案例2

tomcat 实现 https的参考文档

```
https://help.aliyun.com/document_detail/98576.html?
spm=5176.b657008.0.0.5a471b48Cyahpi
```

范例： 实现 http自动跳转至 tomcat的https

```
server {
    listen 80;
    server_name blog.magedu.org;
    return 302 https://$server_name$request_uri;
}
server {
    listen 443 ssl;
    server_name blog.magedu.org;
    ssl_certificate    /etc/nginx/ssl/www.magedu.org.pem;
    ssl_certificate_key /etc/nginx/ssl/www.magedu.org.key;
    location / {
```

```
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header Host  $http_host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

4.3 利用nginx实现动静分离代理

4.3.1 配置说明

可以利用nginx实现动静分离

```
vim  nginx.conf
root /usr/share/nginx/html;
#下面行可不加
#location / {
#    root /data/webapps/ROOT;
#    index index.html;
#}

# ~* 不区分大小写
location ~* \.jsp$ {
    proxy_pass http://node1.magedu.com:8080; #注意: 8080后不要加/,需要在nginx服务器修改 /etc/hosts
}
```

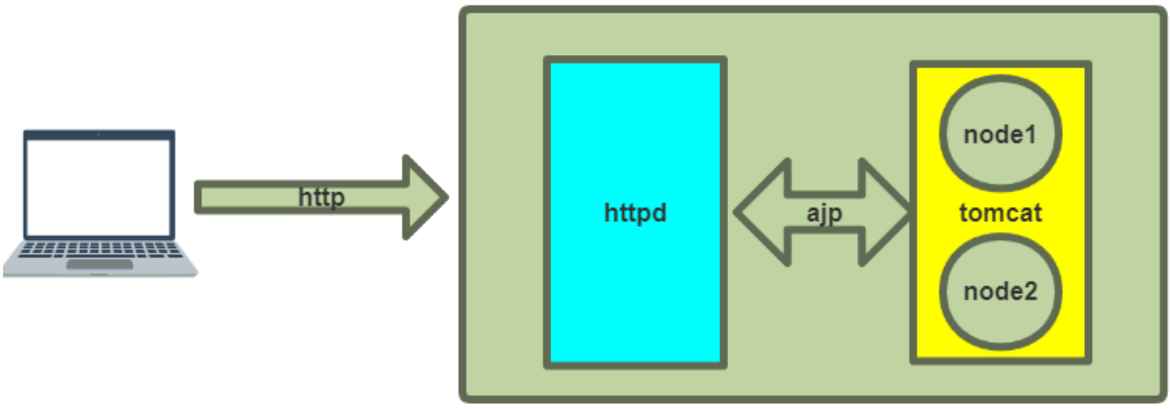
以上设置，可以将jsp的请求反向代理到tomcat，而其它文件仍由nginx处理，从而实现所谓动静分离。但由于jsp文件中实际上是由静态资源和动态组成，所以无法彻底实现动静分离。实际上Tomcat不太适合做动静分离，用它来管理程序的图片不好做动静分离部署

4.3.2 实战案例

```
#准备三个不同的资源文件
[root@centos8 ~]#echo /usr/local/tomcat/webapps/ROOT/test.html  >
/usr/local/tomcat/webapps/ROOT/test.html
[root@centos8 ~]#echo /usr/local/tomcat/webapps/ROOT/test.jsp  >
/usr/local/tomcat/webapps/ROOT/test.jsp
[root@centos8 ~]#echo /usr/share/nginx/html/test.html >
/usr/share/nginx/html/test.html

[root@centos8 ~]#vim /etc/nginx/nginx.conf
......
root         /usr/share/nginx/html;
#location / {
#
#}
location ~* \.jsp$ {
    proxy_pass http://127.0.0.1:8080;
         }
......


[root@centos8 ~]#systemctl restart nginx
```

```
#访问test.html，观察结果都一样访问的是nginx自身的资源
[root@centos8 ~]#curl http://node1.magedu.org/test.html
[root@centos8 ~]#curl http://node2.magedu.org/test.html
[root@centos8 ~]#curl http://127.0.0.1/test.html
[root@centos8 ~]#curl http://10.0.0.8/test.html
/usr/share/nginx/html/test.html


#访问test.jsp，观察结果都一样访问的是tomcat的默认主机资源
[root@centos8 ~]#curl http://node1.magedu.org/test.jsp
[root@centos8 ~]#curl http://node2.magedu.org/test.jsp
[root@centos8 ~]#curl http://127.0.0.1/test.jsp
[root@centos8 ~]#curl http://10.0.0.8/test.jsp
/usr/local/tomcat/webapps/ROOT/test.jsp
```

4.4 利用 httpd 实现基于AJP协议的反向代理至后端Tomcat服务器



4.4.1 AJP 协议说明

AJP（Apache JServ Protocol）是定向包协议，是一个二进制的TCP传输协议，相比HTTP这种纯文本的协议来说，效率和性能更高，也做了很多优化。但是浏览器并不能直接支持AJP13协议，只支持HTTP协议。所以实际情况是，通过Apache的proxy_ajp模块进行反向代理，暴露成http协议给客户端访问

4.4.2 启用和禁用 AJP

注意: Tomcat/8.5.51之后版本基于安全需求默认禁用AJP协议

范例: Tomcat/8.5.51之后版启用支持AJP协议

```
[root@centos8 tomcat]#vim conf/server.xml
#取消前面的注释,并修改下面行,修改address和secretRequired
<Connector protocol="AJP/1.3"  address="0.0.0.0" port="8009"
 redirectPort="8443" secretRequired="" />

[root@centos8 tomcat]#systemctl restart tomcat
[root@centos8 tomcat]#ss -ntl
State         Recv-Q        Send-Q                    Local Address:Port
      Peer Address:Port
LISTEN        0             128                             0.0.0.0:22
          0.0.0.0:*
```

```
LISTEN       0            100                        127.0.0.1:25
          0.0.0.0:*
LISTEN       0            128                            [::]:22
             [::]:*
LISTEN       0            100                           [::1]:25
             [::]:*
LISTEN       0            1                  [::ffff:127.0.0.1]:8005
                *:*
LISTEN       0            100                [::ffff:127.0.0.1]:8009
                *:*
LISTEN       0            100                             *:8080
                *:*
LISTEN       0            128                             *:80
                *:*
```

注意：secretRequired="" 必须加上,否则出现以下错误提示

```
[root@centos8 tomcat]#cat logs/catalina.log
Caused by: java.lang.IllegalArgumentException: The AJP Connector is configured
with secretRequired="true" but the secret attribute is either null or "". This
combination is not valid.
```

除httpd外，其它支持AJP代理的服务器非常少，比如Nginx就不支持AJP，所以目前一般都禁用AJP协议端口

范例：禁用AJP协议

```
#Tomcat/8.5.50版本之前默认支持AJP协议
[root@centos8 ~]#ss -ntl
State       Recv-Q       Send-Q                  Local Address:Port
    Peer Address:Port
LISTEN      0            128                           0.0.0.0:22
          0.0.0.0:*
LISTEN      0            100                             *:8080
               *:*
LISTEN      0            128                             *:80
               *:*
LISTEN      0            128                          [::]:22
             [::]:*
LISTEN      0            1                  [::ffff:127.0.0.1]:8005
               *:*
LISTEN      0            100                             *:8009
               *:*

#配置tomcat配置文件，删除下面一行
[root@centos8 ~]#vim /usr/local/tomcat/conf/server.xml
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

[root@centos8 ~]#systemctl restart tomcat
[root@centos8 ~]#ss -ntl
State       Recv-Q       Send-Q                  Local Address:Port
    Peer Address:Port
LISTEN      0            128                           0.0.0.0:22
          0.0.0.0:*
LISTEN      0            100                             *:8080
               *:*
```

```
LISTEN     0          128                                    *:80
                *:*
LISTEN     0          128                                 [::]:22
              [::]:*
LISTEN     0          1                    [::ffff:127.0.0.1]:8005
                *:*
```

4.4.3 httpd 实现 AJP 反向代理

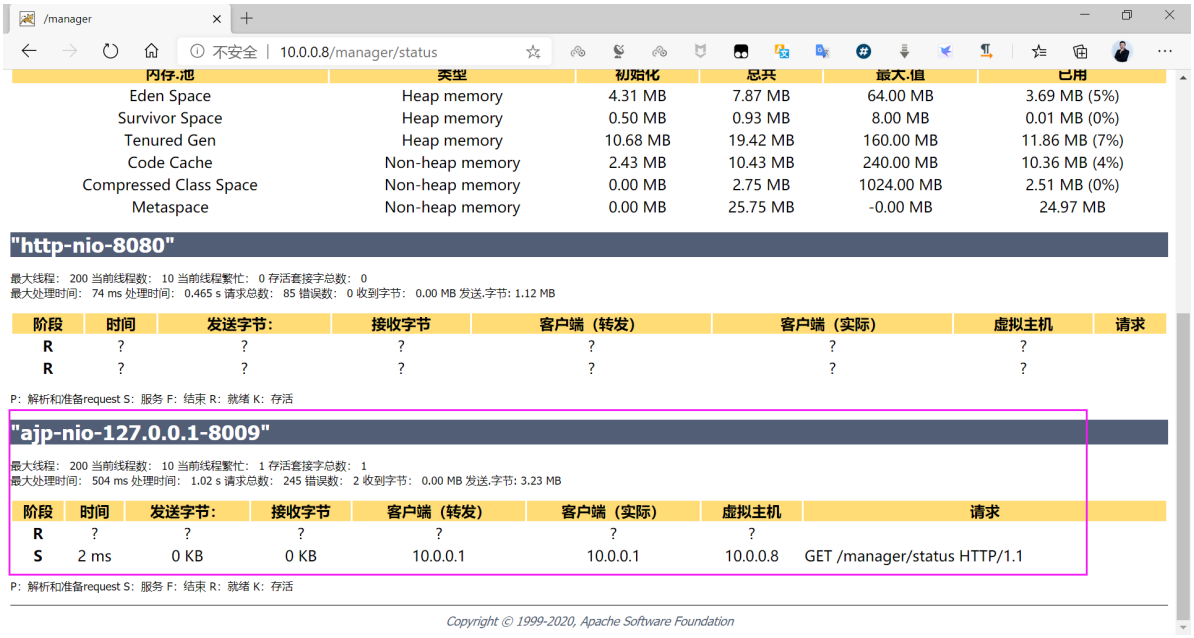
4.4.3.1 配置说明

相对来讲，AJP协议基于二进制比使用HTTP协议的连接器效率高些。

proxy_ajp_module模块代理配置

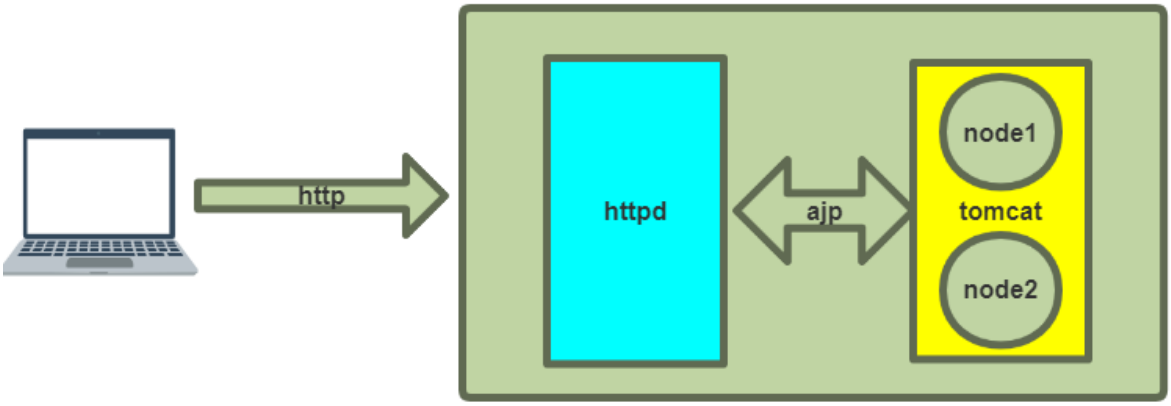
```
<VirtualHost *:80>
    ServerName        node1.magedu.org
    ProxyRequests     Off
    ProxyVia          On
    ProxyPreserveHost On
    ProxyPass         /   ajp://127.0.0.1:8009/
</VirtualHost>
```

查看Server Status可以看到确实使用的是ajp连接了。



4.5.3.2 实战案例

范例：启用httpd的AJP反向代理功能




```
[root@centos8 ~]#vim /etc/httpd/conf.d/tomcat.conf
[root@centos8 ~]#cat /etc/httpd/conf.d/tomcat.conf
<VirtualHost *:80>
    ServerName        node1.magedu.org
    ProxyRequests     Off
    ProxyVia          On   #此项对AJP无效
    ProxyPreserveHost On   #此项对AJP无效
    ProxyPass        /   ajp://127.0.0.1:8009/
</VirtualHost>


[root@centos8 ~]#systemctl restart httpd

#再次用用下面不同URL访问，可以看以下结果
[root@centos8 ~]#curl http://node1.magedu.org/test.html
/data/node1/ROOT/test.html
[root@centos8 ~]#curl http://node2.magedu.org/test.html
/data/node2/ROOT/test.html
[root@centos8 ~]#curl http://10.0.0.8/test.html
/usr/local/tomcat/webapps/ROOT/test.html
[root@centos8 ~]#curl http://127.0.0.1/test.html
/usr/local/tomcat/webapps/ROOT/test.html

[root@centos8 ~]#vim /etc/httpd/conf.d/tomcat.conf
#只修改下面一行,关闭向后端转发请求的host首部
 ProxyPreserveHost Off

#再次用用下面不同URL访问，可以看到和上面一样的结果，说明AJP协议和Http不同，自动转发所有首部信息
[root@centos8 ~]#curl http://node1.magedu.org/test.html
/data/node1/ROOT/test.html
[root@centos8 ~]#curl http://node2.magedu.org/test.html
/data/node2/ROOT/test.html
[root@centos8 ~]#curl http://10.0.0.8/test.html
/usr/local/tomcat/webapps/ROOT/test.html
[root@centos8 ~]#curl http://127.0.0.1/test.html
/usr/local/tomcat/webapps/ROOT/test.html
```

可以通过status页面看到下面AJP的信息


```
#用iptables禁用AJP的访问
[root@centos8 ~]#iptables -A INPUT -p tcp --dport 8009 -j REJECT
[root@centos8 ~]#curl http://node1.magedu.org/test.html
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>503 Service Unavailable</title>
</head><body>
<h1>Service Unavailable</h1>
<p>The server is temporarily unable to service your
request due to maintenance downtime or capacity
problems. Please try again later.</p>
</body></html>
```

4.5 实现 tomcat 负载均衡

动态服务器的问题，往往就是并发能力太弱，往往需要多台动态服务器一起提供服务。如何把并发的压力分摊，这就需要调度，采用一定的调度策略，将请求分发给不同的服务器，这就是Load Balance负载均衡。

当单机Tomcat，演化出多机多级部署的时候，一个问题便凸显出来，这就是Session。而这个问题的由来，都是由于HTTP协议在设计之初没有想到未来的发展。

4.5.1 HTTP的无状态，有连接和短连接

- 无状态：指的是服务器端无法知道2次请求之间的联系，即使是前后2次请求来自同一个浏览器，也没有任何数据能够判断出是同一个浏览器的请求。后来可以通过cookie、session机制来判断。
    - 浏览器端第一次HTTP请求服务器端时，在服务器端使用session这种技术，就可以在服务器端产生一个随机值即SessionID发给浏览器端，浏览器端收到后会保持这个SessionID在Cookie当中，这个Cookie值一般不能持久存储，浏览器关闭就消失。浏览器在每一次提交HTTP请求的时候会把这个SessionID传给服务器端，服务器端就可以通过比对知道是谁了
    - Session通常会保存在服务器端内存中，如果没有持久化，则易丢失
    - Session会定时过期。过期后浏览器如果再访问，服务端发现没有此ID，将给浏览器端重新发新的SessionID
    - 更换浏览器也将重新获得新的SessionID
- 有连接：是因为它基于TCP协议，是面向连接的，需要3次握手、4次断开。
- 短连接：Http 1.1之前，都是一个请求一个连接，而Tcp的连接创建销毁成本高，对服务器有很大的影响。所以，自Http 1.1开始，支持keep-alive，默认也开启，一个连接打开后，会保持一段时间（可设置），浏览器再访问该服务器就使用这个Tcp连接，减轻了服务器压力，提高了效率。

服务器端如果故障，即使Session被持久化了，但是服务没有恢复前都不能使用这些SessionID。

如果使用HAProxy或者Nginx等做负载均衡器，调度到了不同的Tomcat上，那么也会出现找不到SessionID的情况。

4.5.2 会话保持方式

4.5.2.1 session sticky 会话黏性

Session绑定

- nginx：source ip, cookie
- HAProxy：source ip, cookie

优点：简单易配置

缺点：如果目标服务器故障后，如果没有做sessoin持久化，就会丢失session,此方式生产很少使用

4.5.2.2 Session 复制集群

Tomcat自己的提供的多播集群，通过多播将任何一台的session同步到其它节点。

缺点

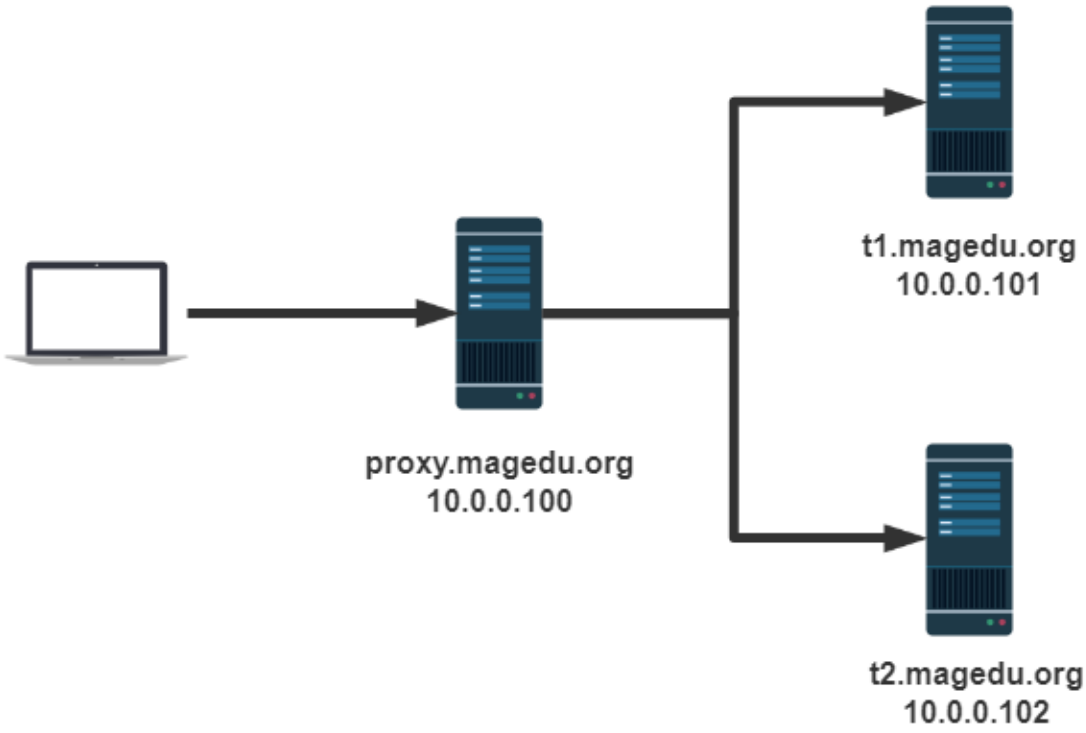
- Tomcat的同步节点不宜过多，互相即时通信同步session需要太多带宽
- 每一台都拥有全部session，内存损耗太多

4.5.2.3 Session Server

session 共享服务器，使用memcached、redis做共享的Session服务器，此为推荐方式

4.5.3 负载均衡规划和准备

4.5.3.1 负载均衡主机和网络地址规划



IP	主机名	服务	软件
10.0.0.100	proxy.magedu.org	调度器	Nginx、HTTPD
10.0.0.101	t1.magedu.org	tomcat1	JDK8、Tomcat8
10.0.0.102	t2.magedu.org	tomcat2	JDK8、Tomcat8

```
#只需在10.0.0.100的nginx主机上实现域名解析
vim /etc/hosts
#添加以下三行
10.0.0.100 proxy.magedu.org proxy
10.0.0.101 t1.magedu.org t1
10.0.0.102 t2.magedu.org t2
```

4.5.3.2 负载均衡tomcat主机准备

修改tomcat的虚拟机主机为自定义的主机名,并设为默认的虚拟主机

t1虚拟主机配置conf/server.xml

```
<Engine name="Catalina" defaultHost="t1.magedu.org">
    <Host name="t1.magedu.org" appBase="/data/webapps" autoDeploy="true" >
    </Host>
</Engine>
```

t2虚拟主机配置conf/server.xml

```
<Engine name="Catalina" defaultHost="t2.magedu.org">
    <Host name="t2.magedu.org" appBase="/data/webapps" autoDeploy="true" >
    </Host>
</Engine>
```

4.5.3.3 准备负载均衡规划测试用的jsp文件

在t1和 t2节点创建相同的文件/data/webapps/ROOT/index.jsp

```
#项目路径配置
mkdir -pv /data/webapps/ROOT

#编写测试jsp文件，内容在下面
vim /data/webapps/ROOT/index.jsp
<%@ page import="java.util.*" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<div>On  <%=request.getServerName() %></div>
<div><%=request.getLocalAddr() + ":" + request.getLocalPort() %></div>
<div>SessionID = <span style="color:blue"><%=session.getId() %></span></div>
<%=new Date()%>
</body>
</html>

#设置权限
chown -R tomcat.tomcat /data/webapps/
```

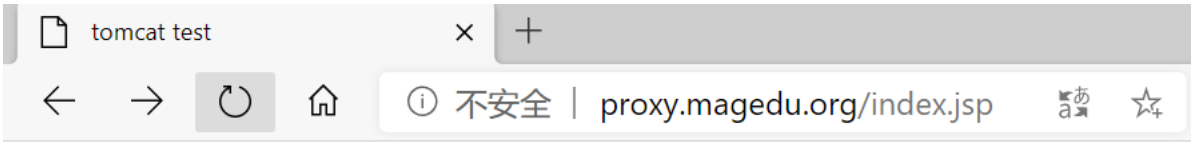
4.5.4 Nginx 实现后端 tomcat 的负载均衡调度

4.5.4.1 Nginx 实现后端 tomcat 的负载均衡

nginx 配置如下

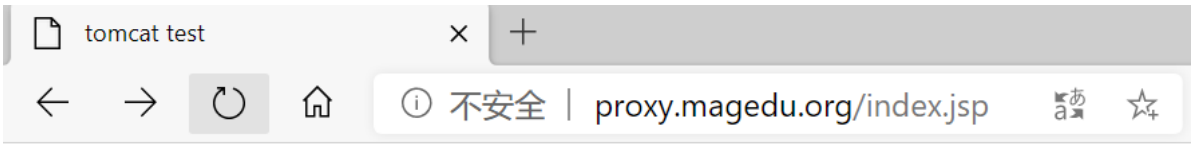
```
vim /etc/nginx/nginx.conf
#在http块中加以下内容
#注意名称不要用下划线
upstream tomcat-server {
        #ip_hash;                    # 先禁用看看轮询，之后开启开黏性
        #hash $cookie_JSESSIONID;    # 先禁用看看轮询，之后开启开黏性
        server t1.magedu.org:8080;
        server t2.magedu.org:8080;
}

server {
        location ~* \.(jsp|do)$ {
            proxy_pass http://tomcat-server;
            #proxy_set_header Host  $http_host; #转发主机头至后端服务器
        }
}
```

测试 http://proxy.magedu.com/index.jsp，可以看到轮询调度效果,每次刷新后端主机和SessionID都会变化



tomcat website

On tomcat-server
10.0.0.101:8080
SessionID = 498976507DD31157EF0A89D26AB698CD
Thu Jul 09 17:56:05 CST 2020



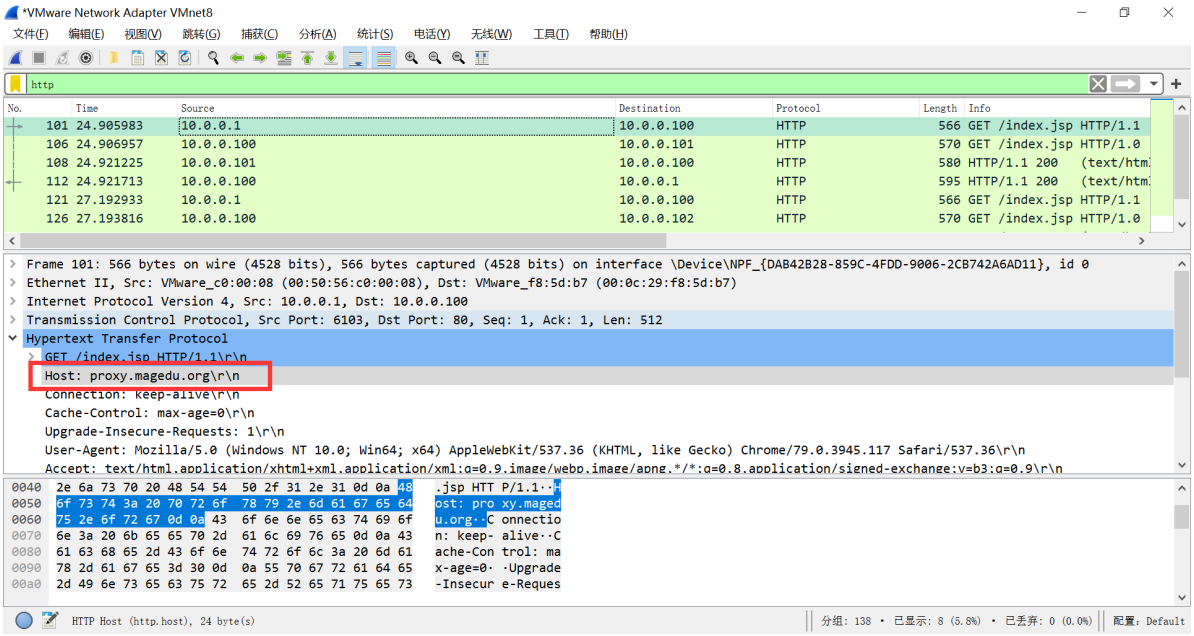
tomcat website

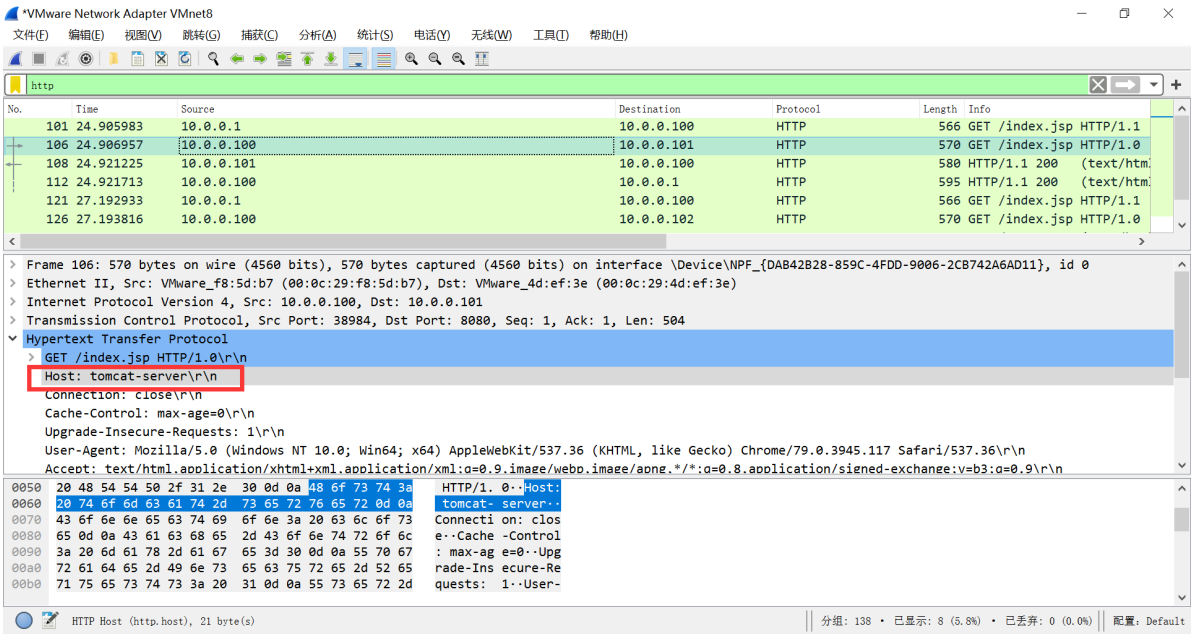
On tomcat-server
10.0.0.102:8080
SessionID = B48A610CA70832CF4698504CDFE1A886
Thu Jul 09 17:56:56 CST 2020

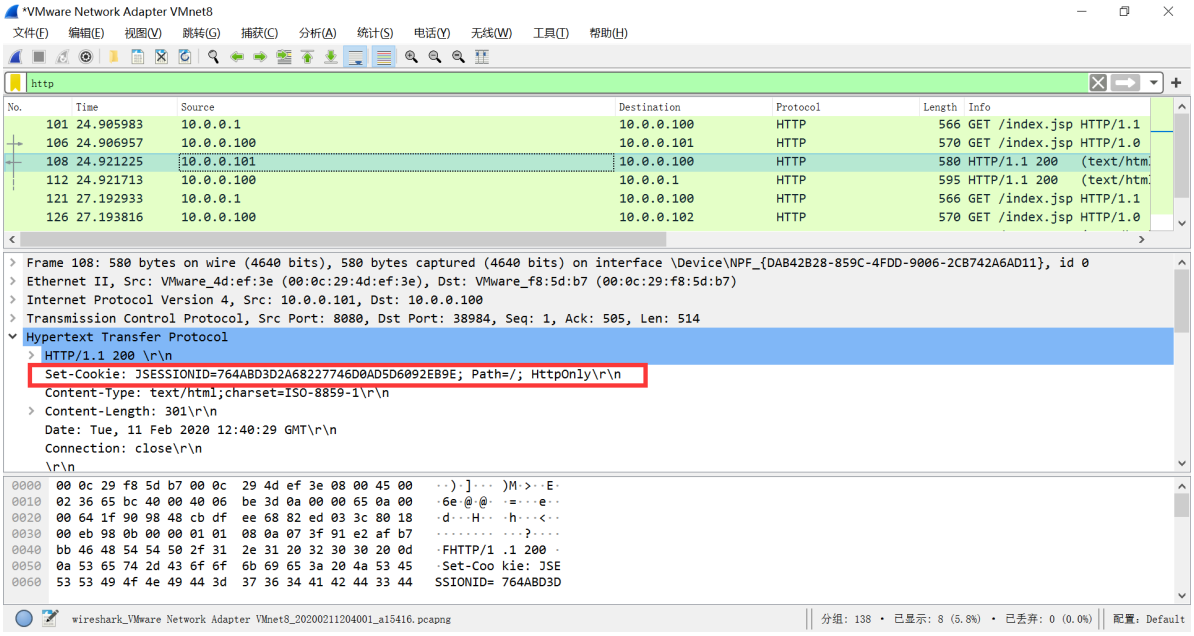
```
[root@proxy ~]#curl http://proxy.magedu.org/index.jsp

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<h1> tomcat website </h1>
<div>On  tomcat-server</div>
<div>10.0.0.101:8080</div>
<div>SessionID = <span
style="color:blue">2E4BFA5135497EA3628F1EBDAE62493E</span></div>
Thu Jul 09 17:58:06 CST 2020
</body>
</html>
[root@proxy ~]#curl http://proxy.magedu.org/index.jsp

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<h1> tomcat website </h1>
```

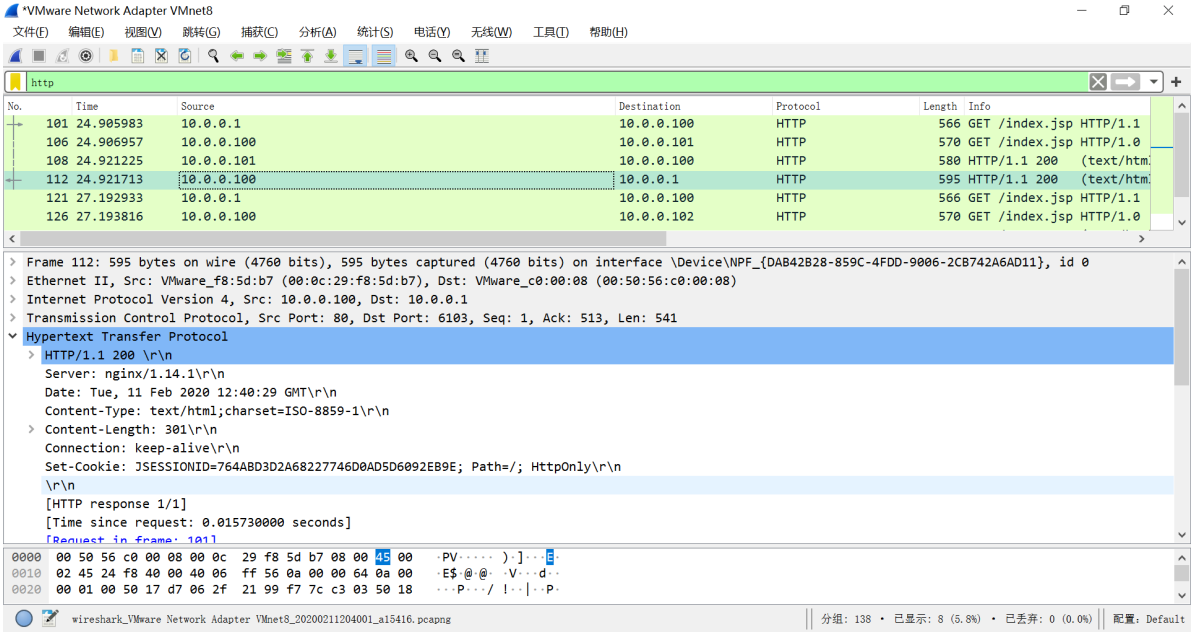
```
<div>On  tomcat-server</div>
<div>10.0.0.102:8080</div>
<div>SessionID = <span
style="color:blue">C5CC437BC05EE5A8620822CB07E71B7C</span></div>
Thu Jul 09 17:58:07 CST 2020
</body>
</html>
```

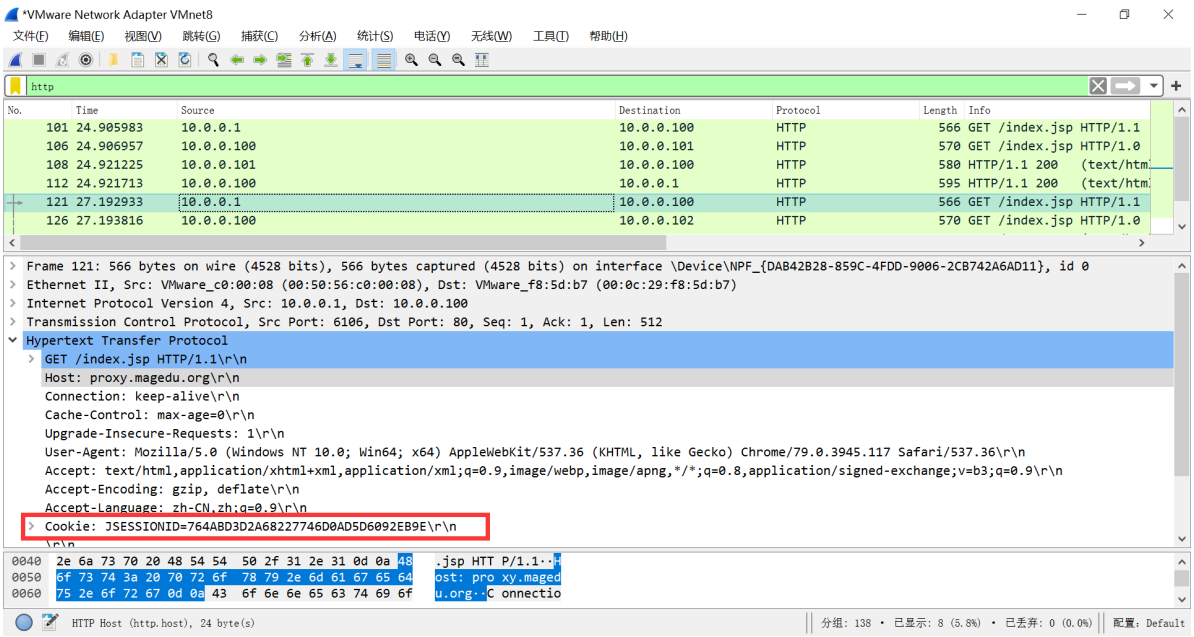
使用抓包wireshark工具可以看到下面信息






*VMware Network Adapter VMnet8
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
http
No. Time Source Destination Protocol Length Info
101 24.905983 10.0.0.1 10.0.0.100 HTTP 566 GET /index.jsp HTTP/1.1
106 24.906957 10.0.0.100 10.0.0.101 HTTP 570 GET /index.jsp HTTP/1.0
108 24.921225 10.0.0.101 10.0.0.100 HTTP 580 HTTP/1.1 200 (text/htm
112 24.921713 10.0.0.100 10.0.0.1 HTTP 595 HTTP/1.1 200 (text/htm
121 27.192933 10.0.0.1 10.0.0.100 HTTP 566 GET /index.jsp HTTP/1.1
126 27.193816 10.0.0.100 10.0.0.102 HTTP 570 GET /index.jsp HTTP/1.0
Frame 108: 580 bytes on wire (4640 bits), 580 bytes captured (4640 bits) on interface \Device\NPF_{DAB42B28-859C-4FDD-9006-2CB742A6AD11}, id 0
Ethernet II, Src: VMware_4d:ef:3e (00:0c:29:4d:ef:3e), Dst: VMware_f8:5d:b7 (00:0c:29:f8:5d:b7)
Internet Protocol Version 4, Src: 10.0.0.101, Dst: 10.0.0.100
Transmission Control Protocol, Src Port: 8080, Dst Port: 38984, Seq: 1, Ack: 505, Len: 514
Hypertext Transfer Protocol
HTTP/1.1 200 \r\n
Set-Cookie: JSESSIONID=764ABD3D2A68227746D0AD5D6092EB9E; Path=/; HttpOnly\r\n
Content-Type: text/html;charset=ISO-8859-1\r\n
Content-Length: 301\r\n
Date: Tue, 11 Feb 2020 12:40:29 GMT\r\n
Connection: close\r\n
\r\n
0000 00 0c 29 f8 5d b7 00 0c 29 4d ef 3e 08 00 45 00
0010 02 36 65 bc 40 00 40 06 be 3d 0a 00 00 65 0a 00
0020 00 64 1f 90 98 48 cb df ee 68 82 ed 03 3c 80 18
0030 00 eb 98 0b 00 00 01 01 08 0a 07 3f 91 e2 af b7
0040 bb 46 48 54 54 50 2f 31 2e 31 20 32 30 30 20 0d
0050 0a 53 65 74 2d 43 6f 6f 6b 69 65 3a 20 4a 53 45
0060 53 53 49 4f 4e 49 44 3d 37 36 34 41 42 44 33 44
wireshark_VMware Network Adapter VMnet8_20200211204001_a15416.pcapng
分组: 138 · 已显示: 8 (5.8%) · 已丢弃: 0 (0.0%)
配置: Default


*VMware Network Adapter VMnet8
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
http
No. Time Source Destination Protocol Length Info
101 24.905983 10.0.0.1 10.0.0.100 HTTP 566 GET /index.jsp HTTP/1.1
106 24.906957 10.0.0.100 10.0.0.101 HTTP 570 GET /index.jsp HTTP/1.0
108 24.921225 10.0.0.101 10.0.0.100 HTTP 580 HTTP/1.1 200 (text/htm
112 24.921713 10.0.0.100 10.0.0.1 HTTP 595 HTTP/1.1 200 (text/htm
121 27.192933 10.0.0.1 10.0.0.100 HTTP 566 GET /index.jsp HTTP/1.1
126 27.193816 10.0.0.100 10.0.0.102 HTTP 570 GET /index.jsp HTTP/1.0
Frame 112: 595 bytes on wire (4760 bits), 595 bytes captured (4760 bits) on interface \Device\NPF_{DAB42B28-859C-4FDD-9006-2CB742A6AD11}, id 0
Ethernet II, Src: VMware_f8:5d:b7 (00:0c:29:f8:5d:b7), Dst: VMware_c0:00:08 (00:50:56:c0:00:08)
Internet Protocol Version 4, Src: 10.0.0.100, Dst: 10.0.0.1
Transmission Control Protocol, Src Port: 80, Dst Port: 6103, Seq: 1, Ack: 513, Len: 541
Hypertext Transfer Protocol
HTTP/1.1 200 \r\n
Server: nginx/1.14.1\r\n
Date: Tue, 11 Feb 2020 12:40:29 GMT\r\n
Content-Type: text/html;charset=ISO-8859-1\r\n
Content-Length: 301\r\n
Connection: keep-alive\r\n
Set-Cookie: JSESSIONID=764ABD3D2A68227746D0AD5D6092EB9E; Path=/; HttpOnly\r\n
\r\n
[HTTP response 1/1]
[Time since request: 0.015730000 seconds]
[Request in frame: 101]
0000 00 50 56 c0 00 08 00 0c 29 f8 5d b7 08 00 45 00
0010 02 45 24 f8 40 00 40 06 ff 56 0a 00 00 64 0a 00
0020 00 01 00 50 17 d7 06 2f 21 99 f7 7c c3 03 50 18
wireshark_VMware Network Adapter VMnet8_20200211204001_a15416.pcapng
分组: 138 · 已显示: 8 (5.8%) · 已丢弃: 0 (0.0%)
配置: Default


*VMware Network Adapter VMnet8
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
http
No. Time Source Destination Protocol Length Info
101 24.905983 10.0.0.1 10.0.0.100 HTTP 566 GET /index.jsp HTTP/1.1
106 24.906957 10.0.0.100 10.0.0.101 HTTP 570 GET /index.jsp HTTP/1.0
108 24.921225 10.0.0.101 10.0.0.100 HTTP 580 HTTP/1.1 200 (text/htm
112 24.921713 10.0.0.100 10.0.0.1 HTTP 595 HTTP/1.1 200 (text/htm
121 27.192933 10.0.0.1 10.0.0.100 HTTP 566 GET /index.jsp HTTP/1.1
126 27.193816 10.0.0.100 10.0.0.102 HTTP 570 GET /index.jsp HTTP/1.0
Frame 121: 566 bytes on wire (4528 bits), 566 bytes captured (4528 bits) on interface \Device\NPF_{DAB42B28-859C-4FDD-9006-2CB742A6AD11}, id 0
Ethernet II, Src: VMware_c0:00:08 (00:50:56:c0:00:08), Dst: VMware_f8:5d:b7 (00:0c:29:f8:5d:b7)
Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.100
Transmission Control Protocol, Src Port: 6106, Dst Port: 80, Seq: 1, Ack: 1, Len: 512
Hypertext Transfer Protocol
GET /index.jsp HTTP/1.1\r\n
Host: proxy.magedu.org\r\n
Connection: keep-alive\r\n
Cache-Control: max-age=0\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.117 Safari/537.36\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: zh-CN,zh;q=0.9\r\n
Cookie: JSESSIONID=764ABD3D2A68227746D0AD5D6092EB9E\r\n
\r\n
0040 2e 6a 73 70 20 48 54 54 50 2f 31 2e 31 0d 0a 48
0050 6f 73 74 3a 20 70 72 6f 78 79 2e 6d 61 67 65 64
0060 75 2e 6f 72 67 0d 0a 43 6f 6e 6e 65 63 74 69 6f
HTTP Host (http.host), 24 byte(s)
分组: 138 · 已显示: 8 (5.8%) · 已丢弃: 0 (0.0%)
配置: Default

4.5.4.2 实现 session 黏性

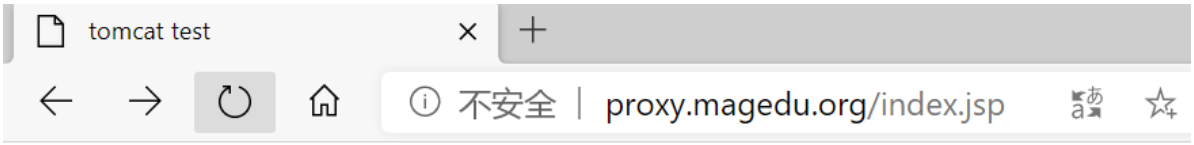
在upstream中使用ip_hash指令，使用客户端IP地址Hash。

```
[root@proxy ~]#vim /etc/nginx/nginx.conf
#只添加ip_hash;这一行
upstream tomcat-server {
        ip_hash;                    #启动源地址hash
        #hash $cookie_JSESSIONID    #启动基于cookie的hash
        server t1.magedu.org:8080;
        server t2.magedu.org:8080;
}
```

配置完reload nginx服务。curl 测试一下看看效果。

```
#用curl访问每次都调度到10.0.0.102主机上，但因为curl每次请求不会自动携带之前获取的cookie,所有SessionID每次都在变化
[root@proxy ~]#curl http://proxy.magedu.org/index.jsp

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<h1> tomcat website </h1>
<div>On  tomcat-server</div>
<div>10.0.0.102:8080</div>
<div>SessionID = <span
style="color:blue">C471641C26865B08B2FDA970BE7C71A6</span></div>
Thu Jul 09 18:02:48 CST 2020
</body>
</html>
[root@proxy ~]#curl http://proxy.magedu.org/index.jsp

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<h1> tomcat website </h1>
<div>On  tomcat-server</div>
<div>10.0.0.102:8080</div>
<div>SessionID = <span
style="color:blue">3F61232DFD791A94D60D0D2E9561309A</span></div>
Thu Jul 09 18:02:52 CST 2020
</body>
</html>
```

通过图形浏览器看到主机不变，sessionID不变



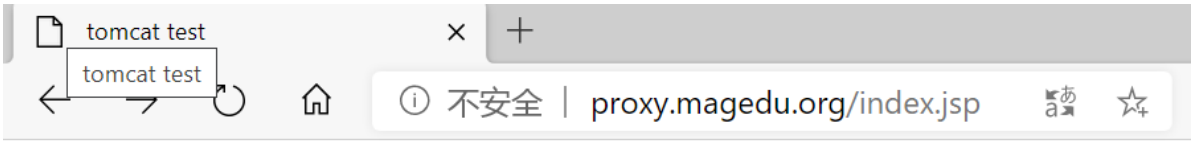
tomcat website

On tomcat-server
10.0.0.102:8080
SessionID = D474DE3C3A67F6EAD1CFCE0AA3D6E6A2
Thu Jul 09 18:01:41 CST 2020

关闭Session对应的Tomcat服务，再重启启动它，看看Session的变化。

```
[root@t2 ~]#systemctl restart tomcat
```

通过浏览器看到主机不变，但sessionID和上一次变化，但后续刷新不再变化



tomcat website

On tomcat-server
10.0.0.102:8080
SessionID = 94EDBE6B19FBFC028816F2D06708BF19
Thu Jul 09 18:04:10 CST 2020

4.5.4.3 实现 https 的负载均衡

范例： 实现 https 的负载均衡

```
[root@rocky8 ~]#vim /etc/nginx/conf.d/blog.wangxiaochun.com.conf
upstream blog {
    ip_hash;
    server 10.0.0.101:8080;
    server 10.0.0.102:8080;
}
server {
    listen 80;
    server_name blog.wangxiaochun.com;
    return 302 https://$server_name$request_uri; #$server_name 来自于上面的
server_name,即blog.wangxiaochun.com
}
server {
    listen 443 ssl;
    server_name blog.wangxiaochun.com;
    ssl_certificate     /etc/nginx/ssl/blog.wangxiaochun.com.pem;
    ssl_certificate_key /etc/nginx/ssl/blog.wangxiaochun.com.key;
    client_max_body_size 20m;
```

```
    location / {
        proxy_pass http://blog;
        proxy_set_header Host  $http_host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}

#配置后端tomcat服务器
[root@rocky8 ~]#vim /usr/local/tomcat/conf/server.xml
...
#添加下面行
    <Host name="blog.wangxiaochun.com"  appBase="/data/blog"
            unpackWARs="true" autoDeploy="true">
    </Host>
#添加上面行
    </Engine>
  </Service>
</Server>

[root@rocky8 ~]#mkdir /data/blog -p
[root@rocky8 ~]#mv jpress-4.1.2.war  /data/blog/ROOT.war
[root@rocky8 ~]#chown -R tomcat.tomcat /data/blog

#注意：先只在第一台tomcat服务器上配置完成初始化连接数据库和上传图片发表文章后，然后复制相关web
文件至第二台主机即可，无需多次初始化

#配置后端数据库
[root@rocky8 ~]#yum -y install mysql-server
[root@rocky8 ~]#systemctl enable --now mysqld
mysql> create user blog@'10.0.0.%' identified by '123456';
mysql> create database jpress;
mysql> grant all on jpress.* to blog@'10.0.0.%';

#配置后端NFS共享存储
[root@rocky8 ~]#yum -y install nfs-utils
[root@rocky8 ~]#groupadd -g 80 www
[root@rocky8 ~]#useradd -g www -u 80 -s /sbin/nlogin www
[root@rocky8 ~]#mkdir /data/blog/ROOT/attachment -p
[root@rocky8 ~]#vim /etc/exports
[root@rocky8 ~]#cat /etc/exports
/data/blog/ROOT/attachment *(rw,all_squash,anonuid=80,anongid=80)
[root@rocky8 ~]#systemctl enable --now nfs-server.service
[root@rocky8 ~]#chown -R www.www /data/blog/ROOT/attachment

#tomcat服务器挂载 NFS共享
[root@rocky8 ~]#vim /etc/fstab
NFS服务器IP:/data/blog/ROOT/attachment /data/blog/ROOT/attachment   nfs    _netdev
0 0
[root@rocky8 ~]#yum -y install nfs-utils
[root@rocky8 ~]#mount -a
```

5 Tomcat Session Replication Cluster

Tomcat 官方实现了 Session 的复制集群,将每个Tomcat的Session进行相互的复制同步,从而保证所有Tomcat都有相同的Session信息.

5.1 配置说明

官方文档:

```
https://tomcat.apache.org/tomcat-10.0-doc/cluster-howto.html
https://tomcat.apache.org/tomcat-9.0-doc/cluster-howto.html
https://tomcat.apache.org/tomcat-8.5-doc/cluster-howto.html
```

说明

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
         channelSendOptions="8">

  <Manager className="org.apache.catalina.ha.session.DeltaManager"
           expireSessionsOnShutdown="false"
           notifyListenersOnReplication="true"/>

  <Channel className="org.apache.catalina.tribes.group.GroupChannel">
    <Membership className="org.apache.catalina.tribes.membership.McastService"
                address="228.0.0.4"          #指定的多播地址
                port="45564"                 #45564/UDP
                frequency="500"              #间隔500ms发送
                dropTime="3000"/>            #故障阈值3s
    <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
              address="auto"                 #监听地址,此项建议修改为当前主机的IP
              port="4000"                    #监听端口
              autoBind="100"                 #如果端口冲突,自动绑定其它端口,范围是4000-
4100
              selectorTimeout="5000"         #自动绑定超时时长5s
              maxThreads="6"/>

    <Sender
className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
```

```
  <Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
    </Sender>
<Interceptor
className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/>
<Interceptor
className="org.apache.catalina.tribes.group.interceptors.MessageDispatchIntercep
tor"/>
</Channel>

  <Valve className="org.apache.catalina.ha.tcp.ReplicationValve" filter=""/>
  <Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>

  <Deployer className="org.apache.catalina.ha.deploy.FarmWarDeployer"
            tempDir="/tmp/war-temp/"
            deployDir="/tmp/war-deploy/"
            watchDir="/tmp/war-listen/"
            watchEnabled="false"/>

  <ClusterListener
className="org.apache.catalina.ha.session.ClusterSessionListener"/>
</Cluster>


#注意:tomcat7的官方文档此处有错误
http://tomcat.apache.org/tomcat-7.0-doc/cluster-howto.html
......
<ClusterListener
className="org.apache.catalina.ha.session.JvmRouteSessionIDBinderListener">
          <ClusterListener
className="org.apache.catalina.ha.session.ClusterSessionListener">
        </Cluster>
```

配置说明

- Cluster 集群配置
- Manager 会话管理器配置
- Channel 信道配置
    - Membership 成员判定。使用什么多播地址、端口多少、间隔时长ms、超时时长ms。同一个多播地址和端口认为同属一个组。使用时修改这个多播地址，以防冲突
    - Receiver 接收器，多线程接收多个其他节点的心跳、会话信息。默认会从4000到4100依次尝试可用端口
        - address="auto"，auto可能绑定到127.0.0.1上，所以一定要改为当前主机可用的IP
    - Sender 多线程发送器，内部使用了tcp连接池。
    - Interceptor 拦截器
- Valve
    - ReplicationValve 检测哪些请求需要检测Session，Session数据是否有了变化，需要启动复制过程
- ClusterListener
    - ClusterSessionListener 集群session侦听器

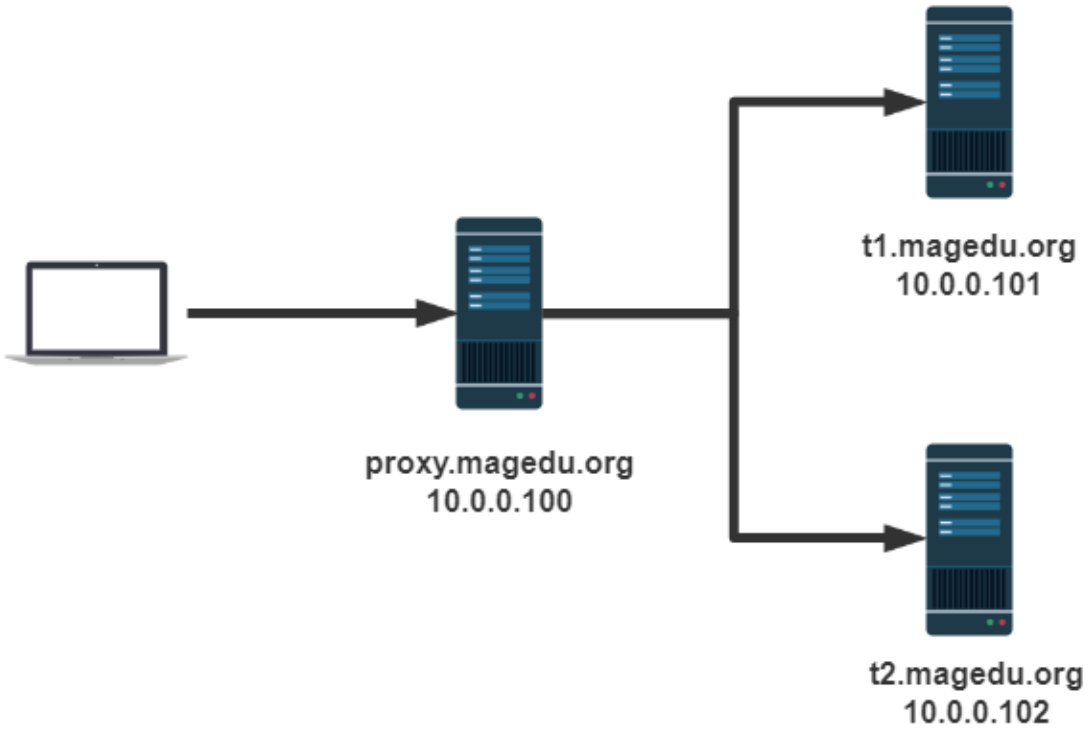
使用`<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>`

添加到`<Engine>`所有虚拟主机都可以启用Session复制

添加到`<Host>`，该虚拟主机可以启用Session复制

最后，在应用程序内部启用了才可以使用

5.2 实战案例: 实现 Tomcat Session 集群



环境准备：

- 时间同步，确保NTP或Chrony服务正常运行
- 防火墙规则

IP	主机名	服务	
10.0.0.100	proxy.magedu.org	调度器	Nginx、HTTPD
10.0.0.101	t1.magedu.org	tomcat1	JDK8、Tomcat8
10.0.0.102	t2.magedu.org	tomcat2	JDK8、Tomcat8

5.2.1 在 proxy 主机设置 httpd (或nginx)实现后端tomcat主机轮询

```
[root@proxy ~]#cat /etc/httpd/conf.d/tomcat.conf
<Proxy balancer://tomcat-server>
    BalancerMember http://t1.magedu.org:8080 loadfactor=1
    BalancerMember http://t2.magedu.org:8080 loadfactor=1
</Proxy>

<VirtualHost *:80>
    ServerName        proxy.magedu.org
    ProxyRequests     Off
    ProxyVia          On
    ProxyPreserveHost On
    ProxyPass        / balancer://tomcat-server/
    ProxyPassReverse / balancer://tomcat-server/
```

```
</VirtualHost>

[root@proxy ~]#systemctl restart httpd
```

5.2.1 在所有后端tomcat主机上修改conf/server.xml

本次把多播复制的配置放到t1.magedu.org和t2.magedu.org虚拟主机里面， 即Host块中。

特别注意**修改Receiver的address属性**为一个本机可对外的IP地址。

5.2.1.1 修改 t1 主机的 conf/server.xml

```
#将5.1 内容复制到conf/server.xml的Host块内或Engine块(针对所有主机)
[root@t1 ~]#vim /usr/local/tomcat/conf/server.xml
[root@t1 ~]#cat  /usr/local/tomcat/conf/server.xml
.....以上省略.....
   <Host name="t1.magedu.org"  appBase="/data/webapps" unpackWARs="true"
autoDeploy="true">
####################在<Host> </host>块中间加下面一段内容
##############################
     <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
         channelSendOptions="8">

  <Manager className="org.apache.catalina.ha.session.DeltaManager"
           expireSessionsOnShutdown="false"
           notifyListenersOnReplication="true"/>

  <Channel className="org.apache.catalina.tribes.group.GroupChannel">
    <Membership className="org.apache.catalina.tribes.membership.McastService"
                address="230.100.100.100"   #指定不冲突的多播地址
                port="45564"
                frequency="500"
                dropTime="3000"/>
    <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
              address="10.0.0.101"          #指定网卡的IP
              port="4000"
              autoBind="100"
              selectorTimeout="5000"
              maxThreads="6"/>

    <Sender
className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
  <Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
    </Sender>
<Interceptor
className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/>
<Interceptor
className="org.apache.catalina.tribes.group.interceptors.MessageDispatchIntercep
tor"/>
</Channel>

  <Valve className="org.apache.catalina.ha.tcp.ReplicationValve"
         filter=""/>
  <Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>

  <Deployer className="org.apache.catalina.ha.deploy.FarmWarDeployer"
```

```
            tempDir="/tmp/war-temp/"
            deployDir="/tmp/war-deploy/"
            watchDir="/tmp/war-listen/"
            watchEnabled="false"/>

  <ClusterListener
className="org.apache.catalina.ha.session.ClusterSessionListener"/>
</Cluster>
########################################以上内容是新加的
##################################
      </Host>
    </Engine>
  </Service>
</Server>
[root@t1 ~]#systemctl restart tomcat

[root@t1 ~]#ss -ntl
State            Recv-Q    Send-Q       Local Address:Port Peer
Address:Port
LISTEN           0         128                      0.0.0.0:22
 0.0.0.0:*
LISTEN           0         100                    127.0.0.1:25
 0.0.0.0:*
LISTEN           0         128                        [::]:22          [::]:*

LISTEN           0         100                       [::1]:25          [::]:*

LISTEN           0         50           [::ffff:10.0.0.101]:4000           *:*

LISTEN           0         1              ::ffff:127.0.0.1]:8005           *:*

LISTEN           0         100                           *:8009           *:*

LISTEN           0         100                           *:8080           *:*
```

简化说明

t1的conf/server.xml中，如下

```
<Host name="t1.magedu.com" appBase="/data/webapps" autoDeploy="true" >
    #其他略去
    <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
              address="10.0.0.101"  #只改此行
              port="4000"
              autoBind="100"
              selectorTimeout="5000"
              maxThreads="6"/>
```

5.2.1.2 修改 t2 主机的 conf/server.xml

```
[root@t2 ~]#vim /usr/local/tomcat/conf/server.xml
[root@t2 ~]#cat  /usr/local/tomcat/conf/server.xml
.....以上省略.....
   <Host name="localhost"  appBase="webapps"
             unpackWARs="true" autoDeploy="true">

        <!-- SingleSignOn valve, share authentication between web applications
             Documentation at: /docs/config/valve.html -->
        <!--
        <Valve className="org.apache.catalina.authenticator.SingleSignOn" />
        -->

        <!-- Access log processes all example.
             Documentation at: /docs/config/valve.html
             Note: The pattern used is equivalent to using pattern="common" -->
        <Valve className="org.apache.catalina.valves.AccessLogValve"
directory="logs"
               prefix="localhost_access_log" suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />

      </Host>
          <Host name="t2.magedu.org" appBase="/data/webapps" autoDeploy="true" >

####################在<Host> </host>块中间加下面一段内容
##############################
     <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
         channelSendOptions="8">

  <Manager className="org.apache.catalina.ha.session.DeltaManager"
           expireSessionsOnShutdown="false"
           notifyListenersOnReplication="true"/>

  <Channel className="org.apache.catalina.tribes.group.GroupChannel">
    <Membership className="org.apache.catalina.tribes.membership.McastService"
                address="230.100.100.100"
                port="45564"
                frequency="500"
                dropTime="3000"/>
    <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
              address="10.0.0.102"            #此行指定当前主机的IP,其它和T1节点配置相
同
              port="4000"
              autoBind="100"
              selectorTimeout="5000"
              maxThreads="6"/>

    <Sender
className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
  <Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
    </Sender>
<Interceptor
className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/>
```

```
<Interceptor
className="org.apache.catalina.tribes.group.interceptors.MessageDispatchIntercep
tor"/>
</Channel>

  <Valve className="org.apache.catalina.ha.tcp.ReplicationValve"
         filter=""/>
  <Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>

  <Deployer className="org.apache.catalina.ha.deploy.FarmWarDeployer"
            tempDir="/tmp/war-temp/"
            deployDir="/tmp/war-deploy/"
            watchDir="/tmp/war-listen/"
            watchEnabled="false"/>

  <ClusterListener
className="org.apache.catalina.ha.session.ClusterSessionListener"/>
</Cluster>
#######################################以上内容是新加的
#################################
              </Host>
    </Engine>
  </Service>
</Server>
[root@t2 ~]#systemctl restart tomcat
[root@t2 ~]#ss -ntl
State            Recv-Q   Send-Q       Local Address:Port    Peer Address:Port

LISTEN           0        128                0.0.0.0:22           0.0.0.0:*
LISTEN           0        100              127.0.0.1:25           0.0.0.0:*
LISTEN           0        128                   [::]:22              [::]:*
LISTEN           0        100                  [::1]:25              [::]:*
LISTEN           0        50     [::ffff:10.0.0.102]:4000              *:*
LISTEN           0        1       [::ffff:127.0.0.1]:8005              *:*
LISTEN           0        100                      *:8009               *:*
LISTEN           0        100                      *:8080               *:*
```

简化说明

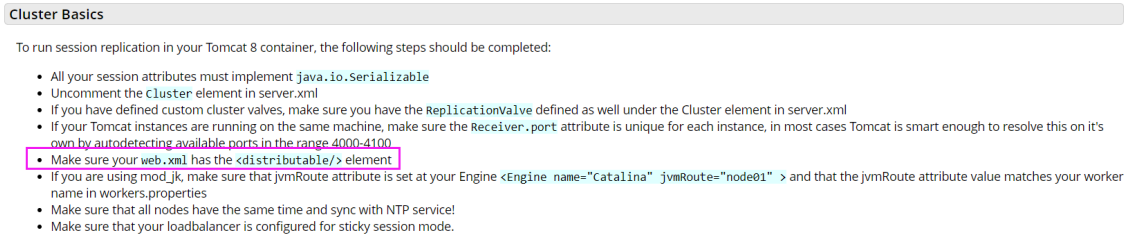
t2主机的server.xml中，如下

```
<Host name="t2.magedu.com" appBase="/data/webapps" autoDeploy="true" >
    其他略去
    <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
              address="10.0.0.102"  #只改此行
              port="4000"
              autoBind="100"
              selectorTimeout="5000"
              maxThreads="6"/>
```

尝试使用刚才配置过得负载均衡(移除Session黏性)，测试发现Session还是变来变去。

5.2.3 修改应用的web.xml文件开启该应用程序的分布式

参考官方说明: https://tomcat.apache.org/tomcat-8.5-doc/cluster-howto.html



Cluster Basics

To run session replication in your Tomcat 8 container, the following steps should be completed:

- All your session attributes must implement `java.io.Serializable`
- Uncomment the `Cluster` element in server.xml
- If you have defined custom cluster valves, make sure you have the `ReplicationValve` defined as well under the Cluster element in server.xml
- If your Tomcat instances are running on the same machine, make sure the `Receiver.port` attribute is unique for each instance, in most cases Tomcat is smart enough to resolve this on it's own by autodetecting available ports in the range 4000-4100
- Make sure your `web.xml` has the `<distributable/>` element
- If you are using mod_jk, make sure that jvmRoute attribute is set at your Engine `<Engine name="Catalina" jvmRoute="node01" >` and that the jvmRoute attribute value matches your worker name in workers.properties
- Make sure that all nodes have the same time and sync with NTP service!
- Make sure that your loadbalancer is configured for sticky session mode.

```
Make sure your web.xml has the <distributable/> element
```

为所有tomcat主机应用web.xml的`<web-app>`标签增加子标签`<distributable/>`来开启该应用程序的分布式。

5.2.3.1 修改t1主机的应用的web.xml文件

```
[root@t1 ~]#ll /usr/local/tomcat/webapps/ROOT/WEB-INF/
total 4
-rw-r----- 1 tomcat tomcat 1227 Jul  1 05:53 web.xml
[root@t1 ~]#cp -a /usr/local/tomcat/webapps/ROOT/WEB-INF/ /data/webapps/ROOT/
[root@t1 ~]#tree /data/webapps/ROOT/
/data/webapps/ROOT/
├── index.jsp
└── WEB-INF
    └── web.xml

1 directory, 2 files

#在倒数第二行加一行
[root@t1 ~]##vim /data/webapps/ROOT/WEB-INF/web.xml
[root@t1 ~]#tail -n3  /data/webapps/ROOT/WEB-INF/web.xml
  </description>
<distributable/>  #添加此行
</web-app>

#注意权限
[root@t1 ~]#ll /data/webapps/ROOT/WEB-INF/
total 4
-rw-r----- 1 tomcat tomcat 1243 Jan 17 09:37 web.xml


[root@t1 ~]#systemctl restart tomcat

#同时观察日志
[root@t1 ~]#tail -f /usr/local/tomcat/logs/catalina.out
15-Jul-2020 11:29:10.998 INFO [Membership-MemberAdded.]
org.apache.catalina.ha.tcp.SimpleTcpCluster.memberAdded Replication member added:
[org.apache.catalina.tribes.membership.MemberImpl[tcp://{10, 0, 0, 102}:4000,
{10, 0, 0, 102},4000, alive=1022, securePort=-1, UDP Port=-1, id={89 -26 -30 -99
16 80 65 95 -65 14 -33 124 -55 -123 -30 82 }, payload={}, command={}, domain=
{}]]
```

5.2.3.2 修改t2主机的应用的web.xml文件

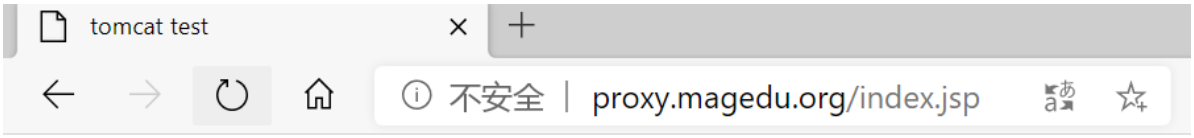
```
#与5.2.3.1上的t1相同的操作
[root@t2 ~]#cp -a /usr/local/tomcat/webapps/ROOT/WEB-INF/ /data/webapps/ROOT/
[root@t2 ~]#vim /data/webapps/ROOT/WEB-INF/web.xml
[root@t2 ~]#tail -n3  /data/webapps/ROOT/WEB-INF/web.xml
  </description>
<distributable/>  #添加此行
</web-app>

#注意权限
[root@t2 ~]#ll /data/webapps/ROOT/WEB-INF/
total 4
-rw-r----- 1 tomcat tomcat 1243 Jan 17 09:38 web.xml


[root@t2 ~]#systemctl restart tomcat

#同时观察日志
[root@t2 ~]#tail -f /usr/local/tomcat/logs/catalina.out
15-Jul-2020 11:29:12.088 INFO [t2.magedu.org-startStop-1]
org.apache.catalina.ha.session.DeltaManager.getAllClusterSessions Manager [],
requesting session state from
[org.apache.catalina.tribes.membership.MemberImpl[tcp://{10, 0, 0, 101}:4000,
{10, 0, 0, 101},4000, alive=208408, securePort=-1, UDP Port=-1, id={118 -108
-116 119 58 22 73 113 -123 -96 -94 111 -65 -90 -87 -107 }, payload={}, command=
{}, domain={}]]. This operation will timeout if no session state has been
received within [60] seconds.
```

5.2.4 测试访问

重启全部Tomcat，通过负载均衡调度到不同节点，返回的SessionID不变了。

用浏览器访问,并刷新多次,发现SessionID 不变,但后端主机在轮询

但此方式当后端tomcat主机较多时,会重复占用大量的内存,并不适合后端服务器众多的场景

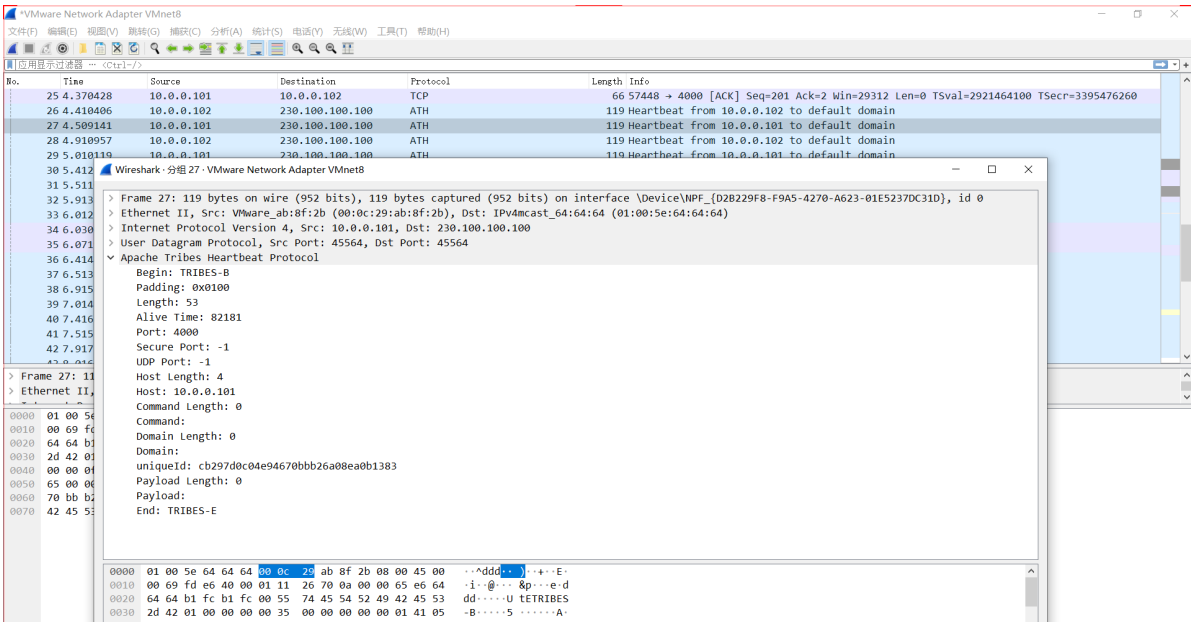


tomcat website

On proxy.magedu.org
10.0.0.102:8080
SessionID = 2E62503B033DCB8C46B2671313D8341E.Tomcat2
Fri Jul 10 00:56:28 CST 2020

```
#修改t1和t2的配置项,删除jvmRoute配置项
[root@t1 tomcat]#vim conf/server.xml
 <Engine name="Catalina" defaultHost="t1.magedu.org" >
[root@t1 tomcat]#systemctl restart tomcat


#多次执行下面操作,可以看到SessionID不变
[root@centos7 ~]#curl -b 'JSESSIONID=1A3E7EED14F3E44FAF7469F8693E1CB6'
proxy.magedu.org/index.jsp

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<h1> tomcat website </h1>
<div>On  tomcat-server</div>
<div>10.0.0.102:8080</div>
<div>SessionID = <span
style="color:blue">1A3E7EED14F3E44FAF7469F8693E1CB6</span></div>
Wed Jul 15 11:33:09 CST 2020
</body>
</html>
[root@centos7 ~]#curl -b 'JSESSIONID=1A3E7EED14F3E44FAF7469F8693E1CB6'
proxy.magedu.org/index.jsp

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<h1> tomcat website </h1>
<div>On  tomcat-server</div>
<div>10.0.0.101:8080</div>
<div>SessionID = <span
style="color:blue">1A3E7EED14F3E44FAF7469F8693E1CB6</span></div>
Wed Jul 15 11:33:10 CST 2020
</body>
```

```
</html>
[root@centos7 ~]#
```



5.2.5 故障模拟

```
#模拟t2节点故障
[root@t2 ~]#systemctl stop tomcat

#多次访问SessionID不变
[root@centos7 ~]#curl -b 'JSESSIONID=1A3E7EED14F3E44FAF7469F8693E1CB6'
proxy.magedu.org/index.jsp

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<h1> tomcat website </h1>
<div>On  tomcat-server</div>
<div>10.0.0.101:8080</div>
<div>SessionID = <span
style="color:blue">1A3E7EED14F3E44FAF7469F8693E1CB6</span></div>
Wed Jul 15 12:01:16 CST 2020
</body>
</html>
```

5.2.6 恢复实验环境

本小节结束,为学习后面的内容,删除此节相关配置,为后续内容准备

```
#恢复t1环境
[root@t1 ~]#vim /usr/local/tomcat/conf/server.xml
[root@t1 ~]#tail  /usr/local/tomcat/conf/server.xml
```

```
        <Valve className="org.apache.catalina.valves.AccessLogValve"
directory="logs"
               prefix="localhost_access_log" suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />

      </Host>
      <Host name="t1.magedu.org"  appBase="/data/webapps" unpackWARs="true"
autoDeploy="true">
      </Host>
    </Engine>
  </Service>
</Server>
[root@t1 ~]#rm -f   /data/webapps/ROOT/WEB-INF/web.xml
[root@t1 ~]#systemctl restart tomcat


#恢复t2环境
[root@t2 ~]#vim /usr/local/tomcat/conf/server.xml
[root@t2 ~]#tail /usr/local/tomcat/conf/server.xml
               prefix="localhost_access_log" suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />

      </Host>
          <Host name="t2.magedu.org" appBase="/data/webapps" autoDeploy="true" >

              </Host>
    </Engine>
  </Service>
</Server>
[root@t2 ~]#rm -f   /data/webapps/ROOT/WEB-INF/web.xml
[root@t2 ~]#systemctl restart tomcat
```

6 Memcached

6.1 NoSQL介绍

NoSQL是对 Not Only SQL、非传统关系型数据库的统称。

NoSQL一词诞生于1998年，2009年这个词汇被再次提出指非关系型、分布式、不提供ACID的数据库设计模式。

随着互联网时代的到来，数据爆发式增长，数据库技术发展日新月异，要适应新的业务需求。

而随着移动互联网、物联网的到来，大数据的技术中NoSQL也同样重要。

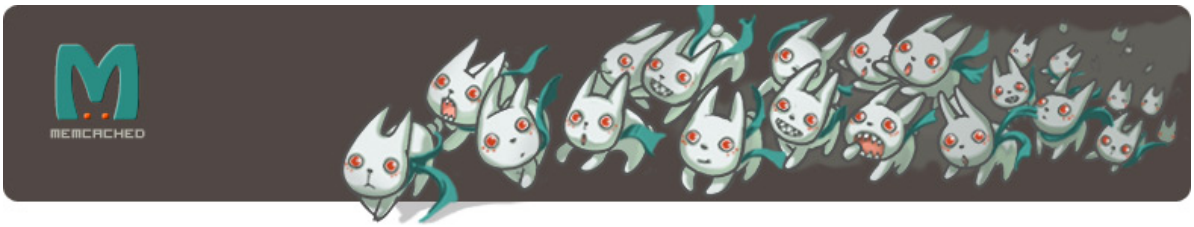
数据库排名：https://db-engines.com/en/ranking

NoSQL 分类

- Key-value Store k/v数据库
  - 性能好 O(1) , 如: redis、memcached
- Document Store 文档数据库
  - mongodb、CouchDB
- Column Store 列存数据库，Column-Oriented DB
  - HBase、Cassandra，大数据领域应用广泛

- Graph DB 图数据库
    - Neo4j
- Time Series 时序数据库
    - InfluxDB、Prometheus

6.2 Memcached



Memcached 只支持能序列化的数据类型，不支持持久化，基于Key-Value的内存缓存系统

memcached 虽然没有像redis所具备的数据持久化功能，比如RDB和AOF都没有，但是可以通过做集群同步的方式，让各memcached服务器的数据进行同步，从而实现数据的一致性，即保证各memcached的数据是一样的，即使有任何一台 memcached 发生故障，只要集群中有一台 memcached 可用就不会出现数据丢失，当其他memcached 重新加入到集群的时候,可以自动从有数据的memcached 当中自动获取数据并提供服务。

Memcached 借助了操作系统的 libevent 工具做高效的读写。libevent是个程序库，它将Linux的epoll、BSD类操作系统的kqueue等事件处理功能封装成统一的接口。即使对服务器的连接数增加，也能发挥高性能。memcached使用这个libevent库，因此能在Linux、BSD、Solaris等操作系统上发挥其高性能

Memcached 支持最大的内存存储对象为1M，超过1M的数据可以使用客户端压缩或拆分报包放到多个key中，比较大的数据在进行读取的时候需要消耗的时间比较长，memcached 最适合保存用户的session实现session共享

Memcached存储数据时, Memcached会去申请1MB的内存, 把该块内存称为一个slab, 也称为一个page

Memcached 支持多种开发语言，包括：JAVA,C,Python,PHP,C#,Ruby,Perl等

Memcached 官网：http://memcached.org/

6.3 Memcached 和 Redis 比较

比较类别	Redis	memcached
支持的数据结构	哈希、列表、集合、有序集合	纯kev-value
持久化支持	有	无
高可用支持	redis支持集群功能，可以实现主动复制，读写分离。官方也提供了sentinel集群管理工具，能够实现主从服务监控，故障自动转移，这一切，对于客户端都是透明的，无需程序改动，也无需人工介入	需要二次开发
存储value容量	最大512M	最大1M
内存分配	临时申请空间，可能导致碎片	预分配内存池的方式管理内存，能够省去内存分配时间
虚拟内存使用	有自己的VM机制，理论上能够存储比物理内存更多的数据，当数据超量时，会引发swap，把冷数据刷到磁盘上	所有的数据存储在物理内存里
网络模型	非阻塞IO复用模型,提供一些非KV存储之外的排序，聚合功能，在执行这些功能时，复杂的CPU计算，会阻塞整个IO调度	非阻塞IO复用模型
水平扩展的支持	redis cluster 可以横向扩展	暂无
多线程	Redis6.0之前是只支持单线程	Memcached支持多线程,CPU利用方面Memcache优于Redis
过期策略	有专门线程，清除缓存数据	懒淘汰机制：每次往缓存放入数据的时候，都会存一个时间，在读取的时候要和设置的时间做TTL比较来判断是否过期
单机QPS	约10W	约60W
源代码可读性	代码清爽简洁	可能是考虑了太多的扩展性，多系统的兼容性，代码不清爽
适用场景	复杂数据结构、有持久化、高可用需求、value存储内容较大	纯KV，数据量非常大，并发量非常大的业务

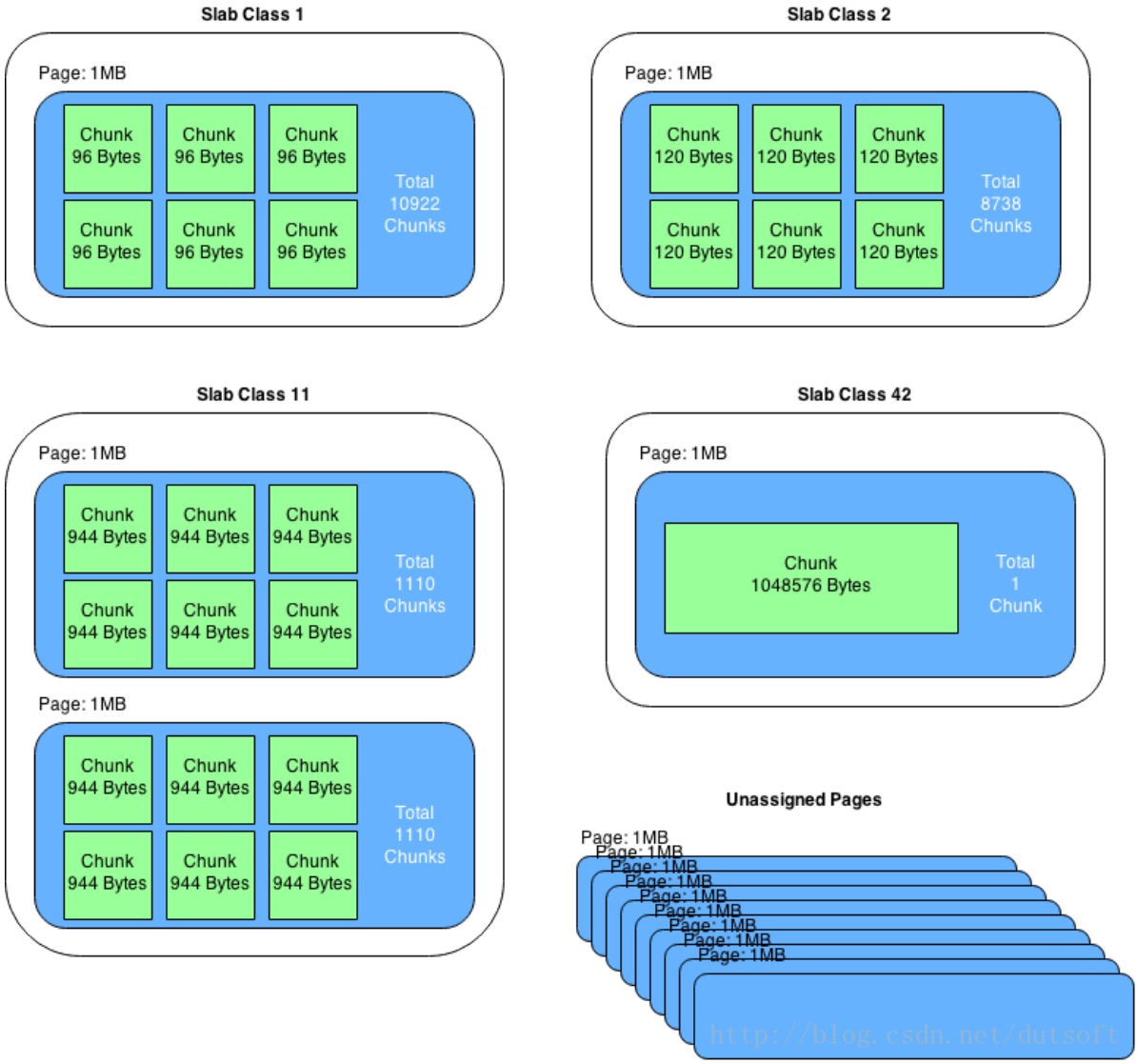
6.4 Memcached 工作机制

6.4.1 内存分配机制

应用程序运行需要使用内存存储数据，但对于一个缓存系统来说，申请内存、释放内存将十分频繁，非常容易导致大量内存碎片，最后导致无连续可用内存可用。

Memcached采用了Slab Allocator机制来分配、管理内存。

- Page：分配给Slab的内存空间，默认为1MB，分配后就得到一个Slab。Slab分配之后内存按照固定字节大小等分成chunk。
- Chunk：用于缓存记录k/v值的内存空间。Memcached会根据数据大小选择存到哪一个chunk中，假设chunk有128bytes、64bytes等多种，数据只有100bytes存储在128bytes中，存在少许浪费。
    - Chunk最大就是Page的大小，即一个Page中就一个Chunk
- Slab Class：Slab按照Chunk的大小分组，就组成不同的Slab Class, 第一个Chunk大小为 96B的Slab为Class1,Chunk 120B为Class 2,如果有100bytes要存，那么Memcached会选择下图中Slab Class 2 存储，因为它是120bytes的Chunk。Slab之间的差异可以使用Growth Factor 控制，默认1.25。



范例：查看Slab Class

```
#-f, --slab-growth-factor=<num> chunk size growth factor (default: 1.25)
[root@centos8 ~]#memcached -u memcached  -f 2 -vv
slab class   1: chunk size        96 perslab   10922
slab class   2: chunk size       192 perslab    5461
slab class   3: chunk size       384 perslab    2730
slab class   4: chunk size       768 perslab    1365
slab class   5: chunk size      1536 perslab     682
slab class   6: chunk size      3072 perslab     341
slab class   7: chunk size      6144 perslab     170
slab class   8: chunk size     12288 perslab      85
slab class   9: chunk size     24576 perslab      42
slab class  10: chunk size     49152 perslab      21
slab class  11: chunk size     98304 perslab      10
slab class  12: chunk size    196608 perslab       5
slab class  13: chunk size    524288 perslab       2
<27 server listening (auto-negotiate)
<28 server listening (auto-negotiate)
```

6.4.2 懒过期 Lazy Expiration

memcached不会监视数据是否过期，而是在取数据时才看是否过期，如果过期,把数据有效期限标识为0，并不清除该数据。以后可以覆盖该位置存储其它数据。

6.4.3 LRU

当内存不足时，memcached会使用LRU（Least Recently Used）机制来查找可用空间，分配给新记录使用。

6.4.4 集群

Memcached集群，称为基于**客户端**的分布式集群，即由客户端实现集群功能，即Memcached本身不支持集群

Memcached集群内部并不互相通信，一切都需要客户端连接到Memcached服务器后自行组织这些节点，并决定数据存储的节点。

6.5 安装和启动

官方安装说明

```
https://github.com/memcached/memcached/wiki/Install
```

6.5.1 yum安装

范例: CentOS 8 安装 memcached

```
[root@centos8 ~]#dnf info memcached
Last metadata expiration check: 0:16:45 ago on Wed 15 Jul 2020 03:07:47 PM CST.
Available Packages
Name         : memcached
Version      : 1.5.9
Release      : 3.el8
Architecture : x86_64
Size         : 132 k
Source       : memcached-1.5.9-3.el8.src.rpm
```

```
Repository   : AppStream
Summary      : High Performance, Distributed Memory Object Cache
URL          : https://www.memcached.org/
License      : BSD
Description  : memcached is a high-performance, distributed memory object
caching
             : system, generic in nature, but intended for use in speeding up
dynamic
             : web applications by alleviating database load.

[root@centos8 ~]#dnf -y install memcached
[root@centos8 ~]#rpm -ql memcached
/etc/sysconfig/memcached
/usr/bin/memcached
/usr/bin/memcached-tool
/usr/lib/.build-id
/usr/lib/.build-id/25
/usr/lib/.build-id/25/2528fb78bbe5b14596eb4ee8c88120b5cc6b59
/usr/lib/systemd/system/memcached.service
/usr/share/doc/memcached
/usr/share/doc/memcached/AUTHORS
/usr/share/doc/memcached/CONTRIBUTORS
/usr/share/doc/memcached/COPYING
/usr/share/doc/memcached/ChangeLog
/usr/share/doc/memcached/NEWS
/usr/share/doc/memcached/README.md
/usr/share/doc/memcached/new_lru.txt
/usr/share/doc/memcached/protocol.txt
/usr/share/doc/memcached/readme.txt
/usr/share/doc/memcached/storage.txt
/usr/share/doc/memcached/threads.txt
/usr/share/man/man1/memcached-tool.1.gz
/usr/share/man/man1/memcached.1.gz

[root@t1 ~]#cat /etc/sysconfig/memcached
PORT="11211"      #监听端口
USER="memcached" #启动用户
MAXCONN="1024"  #最大连接数
CACHESIZE="64"  #最大使用内存
OPTIONS="-l 127.0.0.1,::1" #其他选项

[root@centos8 ~]#grep -Ev "^#|^$" /usr/lib/systemd/system/memcached.service
[Unit]
Description=memcached daemon
Before=httpd.service
After=network.target
[Service]
EnvironmentFile=/etc/sysconfig/memcached
ExecStart=/usr/bin/memcached -p ${PORT} -u ${USER} -m ${CACHESIZE} -c ${MAXCONN}
$OPTIONS
PrivateTmp=true
ProtectSystem=full
NoNewPrivileges=true
PrivateDevices=true
CapabilityBoundingSet=CAP_SETGID CAP_SETUID CAP_SYS_RESOURCE
RestrictAddressFamilies=AF_INET AF_INET6 AF_UNIX
[Install]
WantedBy=multi-user.target
```

```
[root@centos8 ~]#getent passwd memcached
memcached:x:992:989:Memcached daemon:/run/memcached:/sbin/nologin

[root@centos8 ~]#systemctl enable --now memcached
[root@centos8 ~]#pstree -p |grep memcached
           |-memcached(25582)-+-{memcached}(25584)
           |                  |-{memcached}(25585)
           |                  |-{memcached}(25586)
           |                  |-{memcached}(25587)
           |                  |-{memcached}(25588)
           |                  |-{memcached}(25589)
           |                  |-{memcached}(25590)
           |                  |-{memcached}(25591)
           |                  `-{memcached}(25592)

[root@centos8 ~]#ss -ntlup|grep memcached
tcp    LISTEN  0       128                127.0.0.1:11211         0.0.0.0:*
users:(("memcached",pid=25453,fd=27))
tcp    LISTEN  0       128                    [::1]:11211            [::]:*
users:(("memcached",pid=25453,fd=28))


#修改端口绑定的IP为当前主机的所有IP
[root@centos8 ~]#vim /etc/sysconfig/memcached
[root@centos8 ~]#cat  /etc/sysconfig/memcached
PORT="11211"
USER="memcached"
MAXCONN="1024"
CACHESIZE="64"
#OPTIONS="-l 127.0.0.1,::1"  #注释此行
OPTIONS=""

[root@centos8 ~]#systemctl restart memcached.service
[root@centos8 ~]#ss -ntul
Netid     State   Recv-Q Send-Q Local Address:Port   Peer Address:Port
udp       UNCONN  0      0          127.0.0.1:323         0.0.0.0:*
udp       UNCONN  0      0              [::1]:323            [::]:*
tcp       LISTEN  0      128          0.0.0.0:22          0.0.0.0:*
tcp       LISTEN  0      100        127.0.0.1:25          0.0.0.0:*
tcp       LISTEN  0      128          0.0.0.0:11211       0.0.0.0:*
tcp       LISTEN  0      128             [::]:22             [::]:*
tcp       LISTEN  0      100            [::1]:25             [::]:*
tcp       LISTEN  0      128             [::]:11211          [::]:*
```

范例: CentOS 7 安装 memcached

```
[root@centos7 ~]#yum info memcached
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base:
Installed Packages
Name        : memcached
Arch        : x86_64
Version     : 1.4.15
Release     : 10.el7_3.1
Size        : 176 k
```

```
Repo        : installed
From repo   : base
Summary     : High Performance, Distributed Memory Object Cache
URL         : http://www.memcached.org/
License     : BSD
Description : memcached is a high-performance, distributed memory object caching
            : system, generic in nature, but intended for use in speeding up
dynamic
            : web applications by alleviating database load.

[root@centos7 ~]# yum install memcached
[root@centos7 ~]# rpm -ql memcached
/etc/sysconfig/memcached
/usr/bin/memcached
/usr/bin/memcached-tool
/usr/lib/systemd/system/memcached.service
/usr/share/doc/memcached-1.4.15
/usr/share/doc/memcached-1.4.15/AUTHORS
/usr/share/doc/memcached-1.4.15/CONTRIBUTORS
/usr/share/doc/memcached-1.4.15/COPYING
/usr/share/doc/memcached-1.4.15/ChangeLog
/usr/share/doc/memcached-1.4.15/NEWS
/usr/share/doc/memcached-1.4.15/README.md
/usr/share/doc/memcached-1.4.15/protocol.txt
/usr/share/doc/memcached-1.4.15/readme.txt
/usr/share/doc/memcached-1.4.15/threads.txt
/usr/share/man/man1/memcached-tool.1.gz
/usr/share/man/man1/memcached.1.gz

[root@centos7 ~]#getent passwd memcached
memcached:x:997:995:Memcached daemon:/run/memcached:/sbin/nologin

[root@centos7 ~]# cat /usr/lib/systemd/system/memcached.service
[Service]
Type=simple
EnvironmentFile=-/etc/sysconfig/memcached
ExecStart=/usr/bin/memcached -u $USER -p $PORT -m $CACHESIZE -c $MAXCONN
$OPTIONS

[root@centos7 ~]# cat /etc/sysconfig/memcached
PORT="11211"
USER="memcached"
MAXCONN="1024"
CACHESIZE="64"
OPTIONS=""

#前台显示看看效果
[root@centos7 ~]# memcached -u memcached -p 11211 -f 1.25 -vv

[root@centos7 ~]# systemctl start memcached

[root@centos7 ~]#ss -ntlpu|grep memcached
udp    UNCONN     0      0         *:11211                 *:*          users:
(("memcached",pid=2591,fd=28))
udp    UNCONN     0      0      [::]:11211              [::]:*          users:
(("memcached",pid=2591,fd=29))
tcp    LISTEN     0      128       *:11211                 *:*          users:
(("memcached",pid=2591,fd=26))
```

```
tcp    LISTEN     0      128    [::]:11211              [::]:*          users:
(("memcached",pid=2591,fd=27)) tcp    LISTEN     0      100    [::1]:25
       [::]:*
```

6.5.2 编译安装

```
[root@centos7 ~]#yum -y install gcc  libevent-devel
[root@ubuntu1804 ~]#apt -y install gcc  make libevent-dev

[root@centos7 ~]#wget http://memcached.org/files/memcached-1.6.6.tar.gz
[root@centos7 ~]#tar xvf memcached-1.6.6.tar.gz
[root@centos7 ~]#cd memcached-1.6.6/
[root@centos7 memcached-1.6.6]#./configure  --prefix=/apps/memcached
[root@centos7 memcached-1.6.6]#make && make install

[root@centos7 ~]#tree /apps/memcached/
/apps/memcached/
├── bin
│   └── memcached
├── include
│   └── memcached
│       └── protocol_binary.h
└── share
    └── man
        └── man1
            └── memcached.1

6 directories, 3 files
[root@centos7 ~]#echo 'PATH=/apps/memcached/bin:$PATH' >
/etc/profile.d/memcached.sh
[root@centos7 ~]#. /etc/profile.d/memcached.sh

#准备用户
[root@centos7 ~]#useradd -r -s /sbin/nologin memcached

[root@centos7 ~]#cat /etc/sysconfig/memcached
PORT="11211"
USER="memcached"
MAXCONN="1024"
CACHESIZE="64"
OPTIONS="-l 127.0.0.1,::1"

#默认前台执行
[root@centos7 ~]#memcached -u memcached  -m 2048 -c 65536 -f 2  -vv

#以后台方式执行
[root@centos7 ~]#memcached -u memcached  -m 2048 -c 65536 -d

#准备service文件
[root@centos7 ~]#cat  /lib/systemd/system/memcached.service
[Unit]
Description=memcached daemon
Before=httpd.service
After=network.target
```

```
[Service]
EnvironmentFile=/etc/sysconfig/memcached
ExecStart=/apps/memcached/bin/memcached -p ${PORT} -u ${USER} -m ${CACHESIZE} -c
${MAXCONN} $OPTIONS

[Install]
WantedBy=multi-user.target

[root@centos7 ~]#systemctl daemon-reload
[root@centos7 ~]#systemctl enable --now memcached.service

[root@centos7 ~]#ss -ntl
State      Recv-Q Send-Q  Local Address:Port    Peer Address:Port
LISTEN     0      128         127.0.0.1:11211      *:*
LISTEN     0      128                 *:22         *:*
LISTEN     0      100         127.0.0.1:25         *:*
LISTEN     0      128             [::1]:11211      [::]:*
LISTEN     0      128              [::]:22         [::]:*
LISTEN     0      100             [::1]:25         [::]:*
[root@centos7 ~]#memcached --version
memcached 1.6.6
```

6.5.3 memcached 启动程序说明

修改memcached 运行参数，可以使用下面的选项修改/etc/sysconfig/memcached文件

memcached 常见选项

```
-u username memcached运行的用户身份，必须普通用户
-p 绑定的端口，默认11211
-m num 最大内存，单位MB，默认64MB
-c num 最大连接数，缺省1024
-d 守护进程方式运行
-f 增长因子Growth Factor，默认1.25
-v 详细信息，-vv能看到详细信息
-M 使用内存直到耗尽，不许LRU
-U 设置UDP监听端口，0表示禁用UDP
```

范例:

```
[root@centos8 ~]#memcached -u memcached -p 11211 -f 2 -vv
slab class   1: chunk size        96 perslab   10922
slab class   2: chunk size       192 perslab    5461
slab class   3: chunk size       384 perslab    2730
slab class   4: chunk size       768 perslab    1365
slab class   5: chunk size      1536 perslab     682
slab class   6: chunk size      3072 perslab     341
slab class   7: chunk size      6144 perslab     170
slab class   8: chunk size     12288 perslab      85
slab class   9: chunk size     24576 perslab      42
slab class  10: chunk size     49152 perslab      21
slab class  11: chunk size     98304 perslab      10
slab class  12: chunk size    196608 perslab       5
slab class  13: chunk size    524288 perslab       2
<27 server listening (auto-negotiate)
<28 server listening (auto-negotiate)
```

```
[root@centos8 ~]#memcached -u memcached -p 11211 -f 1.25 -vv
slab class   1: chunk size        96 perslab   10922
slab class   2: chunk size       120 perslab    8738
slab class   3: chunk size       152 perslab    6898
slab class   4: chunk size       192 perslab    5461
slab class   5: chunk size       240 perslab    4369
slab class   6: chunk size       304 perslab    3449
slab class   7: chunk size       384 perslab    2730
slab class   8: chunk size       480 perslab    2184
slab class   9: chunk size       600 perslab    1747
slab class  10: chunk size       752 perslab    1394
slab class  11: chunk size       944 perslab    1110
slab class  12: chunk size      1184 perslab     885
slab class  13: chunk size      1480 perslab     708
slab class  14: chunk size      1856 perslab     564
slab class  15: chunk size      2320 perslab     451
slab class  16: chunk size      2904 perslab     361
slab class  17: chunk size      3632 perslab     288
slab class  18: chunk size      4544 perslab     230
slab class  19: chunk size      5680 perslab     184
slab class  20: chunk size      7104 perslab     147
slab class  21: chunk size      8880 perslab     118
slab class  22: chunk size     11104 perslab      94
slab class  23: chunk size     13880 perslab      75
slab class  24: chunk size     17352 perslab      60
slab class  25: chunk size     21696 perslab      48
slab class  26: chunk size     27120 perslab      38
slab class  27: chunk size     33904 perslab      30
slab class  28: chunk size     42384 perslab      24
slab class  29: chunk size     52984 perslab      19
slab class  30: chunk size     66232 perslab      15
slab class  31: chunk size     82792 perslab      12
slab class  32: chunk size    103496 perslab      10
slab class  33: chunk size    129376 perslab       8
slab class  34: chunk size    161720 perslab       6
slab class  35: chunk size    202152 perslab       5
slab class  36: chunk size    252696 perslab       4
slab class  37: chunk size    315872 perslab       3
slab class  38: chunk size    394840 perslab       2
slab class  39: chunk size    524288 perslab       2
<27 server listening (auto-negotiate)
<28 server listening (auto-negotiate)
```

6.6 使用 memcached

6.6.1 memcached 开发库和工具

与memcached通信的不同语言的连接器。libmemcached提供了C库和命令行工具。

范例: 查看memcached相关包

```
[root@centos8 ~]#yum list "*memcached*"
Last metadata expiration check: 0:09:46 ago on Thu 13 Feb 2020 07:14:15 PM CST.
Installed Packages
memcached.x86_64              1.5.9-2.el8          @AppStream
Available Packages
```

```
libmemcached.x86_64           1.0.18-15.el8        AppStream
libmemcached-libs.i686        1.0.18-15.el8        AppStream
libmemcached-libs.x86_64      1.0.18-15.el8        AppStream
perl-Cache-Memcached.noarch   1.30-21.el8          epel
python2-memcached.noarch      1.58-8.el8           epel
python3-memcached.noarch      1.58-8.el8           epel

[root@centos7 ~]#yum list "*memcached*"
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base:
Installed Packages
memcached.x86_64                                        1.4.15-10.el7_3.1
@base
Available Packages
libmemcached.i686                                       1.0.16-5.el7        base
libmemcached.x86_64                                     1.0.16-5.el7        base
libmemcached-devel.i686                                 1.0.16-5.el7        base
libmemcached-devel.x86_64                               1.0.16-5.el7        base
memcached-devel.i686                                    1.4.15-10.el7_3.1   base
memcached-devel.x86_64                                  1.4.15-10.el7_3.1   base
opensips-memcached.x86_64                               1.10.5-4.el7        epel
perl-Cache-Memcached.noarch                             1.30-8.el7          epel
php-ZendFramework-Cache-Backend-Libmemcached.noarch     1.12.20-1.el7       epel
php-ZendFramework-Cache-Backend-Memcached.noarch        1.12.20-1.el7       epel
php-pecl-memcached.x86_64                               2.2.0-1.el7         epel
python-memcached.noarch                                 1.48-4.el7          base
uwsgi-router-memcached.x86_64                           2.0.17.1-2.el7      epel
```

协议

查看/usr/share/doc/memcached-1.4.15/protocol.txt

```
[root@centos8 ~]#dnf  info  libmemcached
Last metadata expiration check: 1:04:38 ago on Wed 15 Jul 2020 03:07:47 PM CST.
Available Packages
Name         : libmemcached
Version      : 1.0.18
Release      : 15.el8
Architecture : x86_64
Size         : 140 k
Source       : libmemcached-1.0.18-15.el8.src.rpm
Repository   : AppStream
Summary      : Client library and command line tools for memcached server
URL          : http://libmemcached.org/
License      : BSD
Description  : libmemcached is a C/C++ client library and tools for the
memcached server
             : (http://memcached.org/). It has been designed to be light on
memory
             : usage, and provide full access to server side methods.
             :
             : It also implements several command line tools:
             :
             : memaslap    Load testing and benchmarking a server
```

```
             : memcapable  Checking a Memcached server capibilities and
compatibility
             : memcat      Copy the value of a key to standard output
             : memcp       Copy data to a server
             : memdump     Dumping your server
             : memerror    Translate an error code to a string
             : memexist    Check for the existance of a key
             : memflush    Flush the contents of your servers
             : memparse    Parse an option string
             : memping     Test to see if a server is available.
             : memrm       Remove a key(s) from the server
             : memslap     Generate testing loads on a memcached cluster
             : memstat     Dump the stats of your servers to standard output
             : memtouch    Touches a key

[root@centos8 ~]#dnf  repoquery -l libmemcached
Last metadata expiration check: 1:05:59 ago on Wed 15 Jul 2020 03:07:47 PM CST.
/usr/bin/memaslap
/usr/bin/memcapable
/usr/bin/memcat
/usr/bin/memcp
/usr/bin/memdump
/usr/bin/memerror
/usr/bin/memexist
/usr/bin/memflush
/usr/bin/memparse
/usr/bin/memping
/usr/bin/memrm
/usr/bin/memslap
/usr/bin/memstat
/usr/bin/memtouch
/usr/lib/.build-id
/usr/lib/.build-id/1b
/usr/lib/.build-id/1b/be22224bfebeca90608feb39b80727ae54628e
/usr/lib/.build-id/38
/usr/lib/.build-id/38/5751e779437bb3a4081e7cfb4ef77088a07d97
/usr/lib/.build-id/42
/usr/lib/.build-id/42/b350284fa158955530a88eb8f96c1f870b4fe2
/usr/lib/.build-id/56
/usr/lib/.build-id/56/42b8e27074c76b9801e7719479084a600db691
/usr/lib/.build-id/59
/usr/lib/.build-id/59/9c9b9482f1c789b5b4446f1b97f5d7e52f30ab
/usr/lib/.build-id/72
/usr/lib/.build-id/72/493b49e4933a11db401650eb5de14a38e22c5b
/usr/lib/.build-id/78
/usr/lib/.build-id/78/c5db64ca4e70a9fd1ef5da96ad28db07946bfa
/usr/lib/.build-id/98
/usr/lib/.build-id/98/6fde24b289939e1bb1621f74c46e68329b140a
/usr/lib/.build-id/a9
/usr/lib/.build-id/a9/028d7389fc0be4422efaf38782416365d67969
/usr/lib/.build-id/dc
/usr/lib/.build-id/dc/d8bd345f28d55f95c5a510ac88e440c97778d3
/usr/lib/.build-id/de
/usr/lib/.build-id/de/0354325ffca355fc6682f3ac4207f111d42ec1
/usr/lib/.build-id/e0
/usr/lib/.build-id/e0/e81e353fdc4b4cfe6e44c20d65e1eb4740249c
/usr/lib/.build-id/e1
/usr/lib/.build-id/e1/91e088f007b253329756088bedb68948191166
```

```
/usr/lib/.build-id/e8
/usr/lib/.build-id/e8/b2f6d6a35c022c66e400c6f6db3d507be54dcd
/usr/share/man/man1/memaslap.1.gz
/usr/share/man/man1/memcapable.1.gz
/usr/share/man/man1/memcat.1.gz
/usr/share/man/man1/memcp.1.gz
/usr/share/man/man1/memdump.1.gz
/usr/share/man/man1/memerror.1.gz
/usr/share/man/man1/memexist.1.gz
/usr/share/man/man1/memflush.1.gz
/usr/share/man/man1/memparse.1.gz
/usr/share/man/man1/memping.1.gz
/usr/share/man/man1/memrm.1.gz
/usr/share/man/man1/memslap.1.gz
/usr/share/man/man1/memstat.1.gz
/usr/share/man/man1/memtouch.1.gz

#测试memcached是否可访问
[root@centos8 ~]#memping  --servers=10.0.0.7
[root@centos8 ~]#echo $?
0
[root@centos8 ~]#memping  --servers=10.0.0.77
Failed to ping 10.0.0.77:11211 SYSTEM ERROR

#查看memcached状态
[root@centos8 ~]#memstat --servers=10.0.0.101
Server: 10.0.0.101 (11211)
     pid: 25582
     uptime: 648
     time: 1594801017
     version: 1.5.9
     libevent: 2.1.8-stable
     pointer_size: 64
     rusage_user: 0.051100
     rusage_system: 0.079158
     max_connections: 1024
     curr_connections: 3
     total_connections: 4
     rejected_connections: 0
     connection_structures: 4
     reserved_fds: 20
     cmd_get: 0
     cmd_set: 2
     cmd_flush: 0
     cmd_touch: 0
     get_hits: 0
     get_misses: 0
     get_expired: 0
     get_flushed: 0
     delete_misses: 0
     delete_hits: 0
     incr_misses: 0
     incr_hits: 0
     decr_misses: 0
     decr_hits: 0
     cas_misses: 0
     cas_hits: 0
     cas_badval: 0
```

```
      touch_hits: 0
      touch_misses: 0
      auth_cmds: 0
      auth_errors: 0
      bytes_read: 72
      bytes_written: 1911
      limit_maxbytes: 67108864
      accepting_conns: 1
      listen_disabled_num: 0
      time_in_listen_disabled_us: 0
      threads: 4
      conn_yields: 0
      hash_power_level: 16
      hash_bytes: 524288
      hash_is_expanding: 0
      slab_reassign_rescues: 0
      slab_reassign_chunk_rescues: 0
      slab_reassign_evictions_nomem: 0
      slab_reassign_inline_reclaim: 0
      slab_reassign_busy_items: 0
      slab_reassign_busy_deletes: 0
      slab_reassign_running: 0
      slabs_moved: 0
      lru_crawler_running: 0
      lru_crawler_starts: 1275
      lru_maintainer_juggles: 1141
      malloc_fails: 0
      log_worker_dropped: 0
      log_worker_written: 0
      log_watcher_skipped: 0
      log_watcher_sent: 0
      bytes: 0
      curr_items: 0
      total_items: 2
      slab_global_page_pool: 0
      expired_unfetched: 2
      evicted_unfetched: 0
      evicted_active: 0
      evictions: 0
      reclaimed: 2
      crawler_reclaimed: 0
      crawler_items_checked: 0
      lrutail_reflocked: 0
      moves_to_cold: 1
      moves_to_warm: 0
      moves_within_lru: 0
      direct_reclaims: 0
      lru_bumps_dropped: 0
[root@centos8 ~]#
```

6.6.2 memcached 操作命令

帮助文档:

```
[root@centos8 ~]#cat /usr/share/doc/memcached/protocol.txt
```

五种基本 memcached 命令执行最简单的操作。这些命令和操作包括：

- set
- add
- replace
- get
- delete

```
#前三个命令是用于操作存储在 memcached 中的键值对的标准修改命令,都使用如下所示的语法：
command <key> <flags> <expiration time> <bytes>
<value>

#参数说明如下：
command set/add/replace
key          key 用于查找缓存值
flags        可以包括键值对的整型参数，客户机使用它存储关于键值对的额外信息
expiration time      在缓存中保存键值对的时间长度（以秒为单位，0 表示永远）
bytes      在缓存中存储的字节数
value      存储的值（始终位于第二行）

#增加key，过期时间为秒，bytes为存储数据的字节数
add key flags exptime bytes
```

范例：

```
[root@centos8 ~]#systemctl start memcached
[root@centos8 ~]#telnet localhost 11211
Trying ::1...
Connected to localhost.
Escape character is '^]'.
stats
STAT pid 27208
STAT uptime 242
STAT time 1603035824
STAT version 1.5.9
STAT libevent 2.1.8-stable
STAT pointer_size 64
STAT rusage_user 0.004214
STAT rusage_system 0.040611
STAT max_connections 1024
STAT curr_connections 2
STAT total_connections 4
STAT rejected_connections 0
STAT connection_structures 3
STAT reserved_fds 20
STAT cmd_get 8
STAT cmd_set 4
STAT cmd_flush 1
STAT cmd_touch 0
STAT get_hits 5
STAT get_misses 3
STAT get_expired 0
STAT get_flushed 0
STAT delete_misses 0
STAT delete_hits 1
STAT incr_misses 0
```

```
STAT incr_hits 0
STAT decr_misses 0
STAT decr_hits 0
STAT cas_misses 0
STAT cas_hits 0
STAT cas_badval 0
STAT touch_hits 0
STAT touch_misses 0
STAT auth_cmds 0
STAT auth_errors 0
STAT bytes_read 224
STAT bytes_written 3963
STAT limit_maxbytes 67108864
STAT accepting_conns 1
STAT listen_disabled_num 0
STAT time_in_listen_disabled_us 0
STAT threads 4
STAT conn_yields 0
STAT hash_power_level 16
STAT hash_bytes 524288
STAT hash_is_expanding 0
STAT slab_reassign_rescues 0
STAT slab_reassign_chunk_rescues 0
STAT slab_reassign_evictions_nomem 0
STAT slab_reassign_inline_reclaim 0
STAT slab_reassign_busy_items 0
STAT slab_reassign_busy_deletes 0
STAT slab_reassign_running 0
STAT slabs_moved 0
STAT lru_crawler_running 0
STAT lru_crawler_starts 765
STAT lru_maintainer_juggles 523
STAT malloc_fails 0
STAT log_worker_dropped 0
STAT log_worker_written 0
STAT log_watcher_skipped 0
STAT log_watcher_sent 0
STAT bytes 0
STAT curr_items 0
STAT total_items 4
STAT slab_global_page_pool 0
STAT expired_unfetched 0
STAT evicted_unfetched 0
STAT evicted_active 0
STAT evictions 0
STAT reclaimed 3
STAT crawler_reclaimed 0
STAT crawler_items_checked 1
STAT lrutail_reflocked 4
STAT moves_to_cold 5
STAT moves_to_warm 1
STAT moves_within_lru 0
STAT direct_reclaims 0
STAT lru_bumps_dropped 0
END

stats items #显示各个 slab 中 item 的数目和存储时长(最后一次访问距离现在的秒数)。
STAT items:1:number 2
```

```
STAT items:1:number_hot 0
STAT items:1:number_warm 0
STAT items:1:number_cold 2
STAT items:1:age_hot 0
STAT items:1:age_warm 0
STAT items:1:age 17
STAT items:1:evicted 0
STAT items:1:evicted_nonzero 0
STAT items:1:evicted_time 0
STAT items:1:outofmemory 0
STAT items:1:tailrepairs 0
STAT items:1:reclaimed 0
STAT items:1:expired_unfetched 0
STAT items:1:evicted_unfetched 0
STAT items:1:evicted_active 0
STAT items:1:crawler_reclaimed 0
STAT items:1:crawler_items_checked 0
STAT items:1:lrutail_reflocked 0
STAT items:1:moves_to_cold 2
STAT items:1:moves_to_warm 0
STAT items:1:moves_within_lru 0
STAT items:1:direct_reclaims 0
STAT items:1:hits_to_hot 0
STAT items:1:hits_to_warm 0
STAT items:1:hits_to_cold 1
STAT items:1:hits_to_temp 0
END
stats slabs #用于显示各个slab的信息，包括chunk的大小、数目、使用情况等
STAT 1:chunk_size 96
STAT 1:chunks_per_page 10922
STAT 1:total_pages 1
STAT 1:total_chunks 10922
STAT 1:used_chunks 1
STAT 1:free_chunks 10921
STAT 1:free_chunks_end 0
STAT 1:mem_requested 67
STAT 1:get_hits 1
STAT 1:cmd_set 3
STAT 1:delete_hits 0
STAT 1:incr_hits 0
STAT 1:decr_hits 0
STAT 1:cas_hits 0
STAT 1:cas_badval 0
STAT 1:touch_hits 0
STAT active_slabs 1
STAT total_malloced 1048576
END

#加
add mykey 1 60 4
test
STORED
#查
get mykey
VALUE mykey 1 4
test
END
```

```
#改
set mykey 1 60 5
test1
STORED
get mykey
VALUE mykey 1 5
test1
END

#删除
delete  mykey
DELETED
get mykey
END

#清空
flush_all
OK
get mykey
END
quit
```

6.6.3 python 语言连接 memcached

6.6.3.1 范例: python3 测试代码

```
[root@centos8 ~]#yum -y install python3 python3-memcached
[root@centos8 ~]#cat m3.py
#!/usr/bin/python3
#coding:utf-8
import memcache
m = memcache.Client(['127.0.0.1:11211'], debug=True)
for i in range(10):
    m.set("key%d" % i,"v%d" % i)
    ret = m.get('key%d' % i)
    print("%s" % ret)

[root@centos8 ~]#chmod +x m3.py
[root@centos8 ~]#./m3.py
v0
v1
v2
v3
v4
v5
v6
v7
v8
v9
```

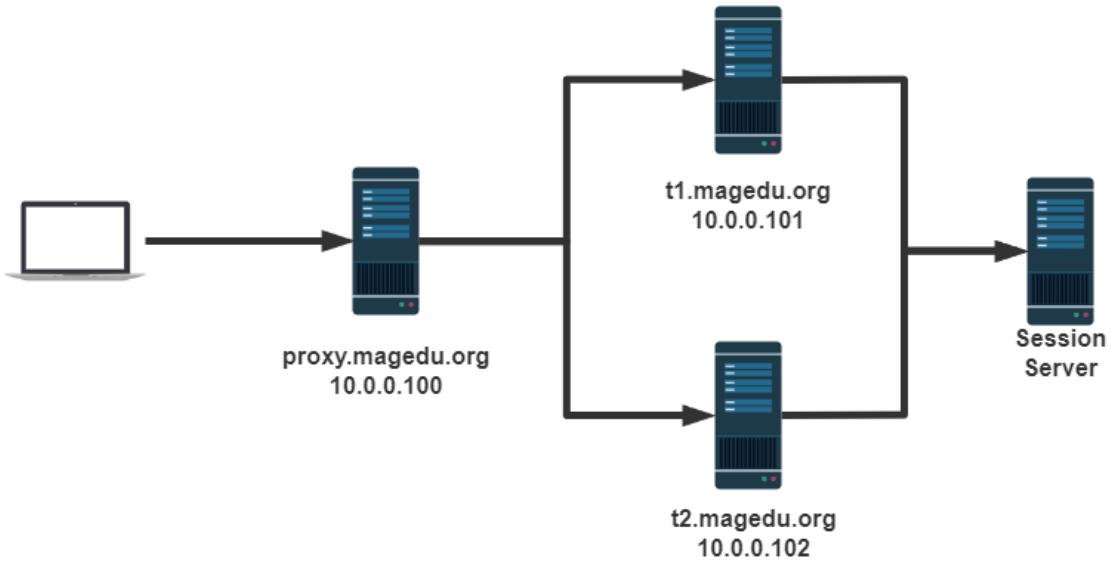
6.6.3.2 范例: python2 测试代码

```
[root@centos7 ~]#yum -y install python-memcached

[root@centos7 ~]#cat m2.py
#!/usr/bin/env python
#coding:utf-8
import memcache
m = memcache.Client(['127.0.0.1:11211'], debug=True)
for i in range(10):
    m.set("key%d" % i,"v%d" % i)
    ret = m.get('key%d' % i)
    print ret

[root@centos7 ~]#python m2.py
v0
v1
v2
v3
v4
v5
v6
v7
v8
v9

[root@centos7 ~]#telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get key1
VALUE key1 0 2
v1
END
get key9
VALUE key9 0 2
v9
END
get key10
END
quit
Connection closed by foreign host.
[root@centos7 ~]#
```

7 session 共享服务器

7.1 msm 介绍

注意：当前MSM不支持 tomcat 10版本



msm（memcached session manager）提供将Tomcat的session保持到memcached或redis的程序，可以实现高可用。

项目早期托管在google code,目前在Github

github网站链接: https://github.com/magro/memcached-session-manager

https://github.com/magro/memcached-session-manager

memcached session manager

chat on gitter | build passing | maven central 2.3.2

memcached-session-manager is a tomcat session manager that keeps sessions in memcached or Redis, for highly available, scalable and fault tolerant web applications. It supports both sticky and non-sticky configurations, and is currently working with tomcat 6.x, 7.x, 8.x and 9.x. For sticky sessions session failover (tomcat crash) is supported, for non-sticky sessions this is the default (a session is served by default by different tomcats for different requests). Also memcached failover (memcached crash) is supported via migration of sessions. There shall also be no single point of failure, so when a memcached fails the session will not be lost (but either be available in tomcat or in another memcached).

支持Tomcat的 6.x、7.x、8.x、9.x

- Tomcat的Session管理类，Tomcat版本不同
  - memcached-session-manager-2.3.2.jar
  - memcached-session-manager-tc8-2.3.2.jar
- Session数据的序列化、反序列化类
  - 官方推荐kyro
  - 在webapp中WEB-INF/lib/下
- 驱动类
  - memcached(spymemcached.jar)
  - Redis(jedis.jar)

7.2 安装

参考链接: https://github.com/magro/memcached-session-manager/wiki/SetupAndConfiguration

将spymemcached.jar、memcached-session-manage、kyro相关的jar文件都放到Tomcat的lib目录中去，这个目录是`$CATALINA_HOME/lib/`，对应本次安装就是/usr/local/tomcat/lib。

```
kryo-3.0.3.jar
asm-5.2.jar
objenesis-2.6.jar
reflectasm-1.11.9.jar
minlog-1.3.1.jar
kryo-serializers-0.45.jar
msm-kryo-serializer-2.3.2.jar
memcached-session-manager-tc8-2.3.2.jar
spymemcached-2.12.3.jar
memcached-session-manager-2.3.2.jar
```

7.3 sticky 模式

7.3.1 sticky 模式工作原理

sticky 模式即前端tomcat和后端memcached有关联(粘性)关系

参考文档:https://github.com/magro/memcached-session-manager/wiki/SetupAndConfiguration

```
Tomcat-1 (t1) will primarily store it's sessions in memcached-2 (m2) which is
running on another machine (m2 is a regular node for t1). Only if m2 is not
available, t1 will store it's sessions in memcached-1 (m1, m1 is the failoverNode
for t1). With this configuration, sessions won't be lost when machine 1 (serving
t1 and m1) crashes. The following really nice ASCII art shows this setup.
Tomcat-1（t1）主要将其会话存储在另一台计算机上运行的memcached-2（m2）中（m2是t1的常规节
点）。 仅当m2不可用时，t1才会将其会话存储在memcached-1中（m1，m1是t1的failoverNode）。 使
用此配置，当计算机1（服务于t1和m1）崩溃时，会话不会丢失。 以下非常好的ASCII艺术显示了此设置。

<t1>   <t2>
  . \ / .
  .  X  .
  . / \ .
<m1>   <m2>
```

t1和m1部署可以在一台主机上，t2和m2部署也可以在同一台。

当新用户发请求到Tomcat1时, Tomcat1生成session返回给用户的同时,也会同时发给memcached2备份。即Tomcat1 session为**主session**，memcached2 session为**备用session**，使用memcached相当于备份了一份Session

如果Tomcat1发现memcached2 失败,无法备份Session到memcached2,则将Sessoin备份存放在memcached1中

7.3.2 配置过程

7.3.2.1 下载相关jar包

下载相关jar包,参考下面官方说明的下载链接

https://github.com/magro/memcached-session-manager/wiki/SetupAndConfiguration

tomcat和memcached相关包

Configure tomcat

The configuration of tomcat requires two things: you need to drop some jars in your `$CATALINA_HOME/lib/` and `WEB-INF/lib/` directories and you have to configure the memcached session manager in the related `<Context>` element (e.g. in `META-INF/context.xml` inside the application files).

Add memcached-session-manager jars to tomcat

Independent of the chosen serialization strategy you always need the memcached-session-manager-${version}.jar and either memcached-session-manager-tc6-${version}.jar for tomcat6, memcached-session-manager-tc7-${version}.jar for tomcat7 (attention: tomcat 7.0.23+), memcached-session-manager-tc8-${version}.jar for tomcat8 or memcached-session-manager-tc9-${version}.jar for tomcat9.

If you're using memcached, you also need the spymemcached-${version}.jar. Tested up to v2.12.3.

If you're using couchbase, you need additionally these jars: couchbase-client-1.4.0.jar jettison-1.3.jar, commons-codec-1.5.jar, httpcore-4.3.jar, httpcore-nio-4.3.jar, netty-3.5.5.Final.jar.

If you're using Redis, you need the jedis-3.0.0.jar.

Please download the appropriate jars and put them in `$CATALINA_HOME/lib/`.

序列化相关下载

Add custom serializers to your webapp (optional)

If you want to use java's built in serialization nothing more has to be done. If you want to use a custom serialization strategy (e.g. because of better performance) this has to be deployed with your webapp so that they're available in `WEB-INF/lib/`.

As msm is available in maven central (under groupId `de.javakaffee.msm`) you can just pull it in using the dependency management of your build system. With maven you can use this dependency definition for the kryo-serializer:

```
<dependency>
    <groupId>de.javakaffee.msm</groupId>
    <artifactId>msm-kryo-serializer</artifactId>
    <version>1.9.7</version>
    <scope>runtime</scope>
</dependency>
```

For javolution the artifactId is `msm-javolution-serializer`, for xstream `msm-xstream-serializer` and for flexjson it's `msm-flexjson-serializer`.

If you're not using a dependency management based on maven repositories these are the jars you need for the different serializers:

- kryo-serializer: msm-kryo-serializer, kryo-serializers-0.34+, kryo-3.x, minlog, reflectasm, asm-5.x, objenesis-2.x
- javolution-serializer: msm-javolution-serializer, javolution-5.4.3.1
- xstream-serializer: msm-xstream-serializer, xstream, xmlpull, xpp3_min
- flexjson-serializer: msm-flexjson-serializer, flexjson

7.3.2.2 修改tomcat配置

修改 `$CATALINA_HOME/conf/context.xml`

特别注意，t1配置中为failoverNodes="n1"， t2配置为failoverNodes="n2"

```
#以下是sticky的配置
<Context>
  ...
  <Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
    memcachedNodes="n1:10.0.0.101:11211,n2:10.0.0.102:11211"
    failoverNodes="n1"
    requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"

 transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFac
tory"
    />
</Context>
```

配置说明

```
memcachedNodes="n1:host1.yourdomain.com:11211,n2:host2.yourdomain.com:11211"

memcached的节点：n1、n2只是别名，可以重新命名。
failoverNodes 为故障转移节点，n1是备用节点，n2是主存储节点。另一台Tomcat将此处的n1改为n2，
其主节点是n1，备用节点是n2。
```

如果配置成功，可以在logs/catalina.out中看到下面的内容

```
12-APR-2020 16:24:08.975 INFO [t1.magedu.com-startStop-1]
de.javakaffee.web.msm.MemcachedSessionService.startInternal --------
-  finished initialization:
- sticky: true
- operation timeout: 1000
- node ids: [n2]
- failover node ids: [n1]
- storage key prefix: null
- locking mode: null (expiration: 5s)
```

配置成功后，网页访问以下，页面中看到了Session。然后运行下面的Python程序，就可以看到是否存储到了memcached中了。

7.3.2.3 准备测试msm的python脚本

范例：安装python环境准备python程序查看memcached中的SessionID

```
[root@centos8 ~]#yum -y install python3 python3-memcached
#或者执行下面两步
[root@centos8 ~]#dnf install python3 -y
[root@centos8 ~]#pip3 install  python-memcached

#脚本内容
[root@centos8 ~]#cat showmemcached.py
#!/usr/bin/python3
import memcache

mc = memcache.Client(['10.0.0.101:11211'], debug=True)

stats = mc.get_stats()[0]
print(stats)
for k,v in stats[1].items():
```

```
    print(k, v)

print('-' * 30)
# 查看全部key
print(mc.get_stats('items')) # stats items 返回 items:5:number 1
print('-' * 30)
print(mc.get_stats('cachedump 5 0')) # stats cachedump 5 0 # 5和上面的items返回的值有关；0表示全部
```

t1、t2、n1、n2依次启动成功，分别使用http://t1.magedu.org:8080/ 和http://t2.magedu.org:8080/观察。

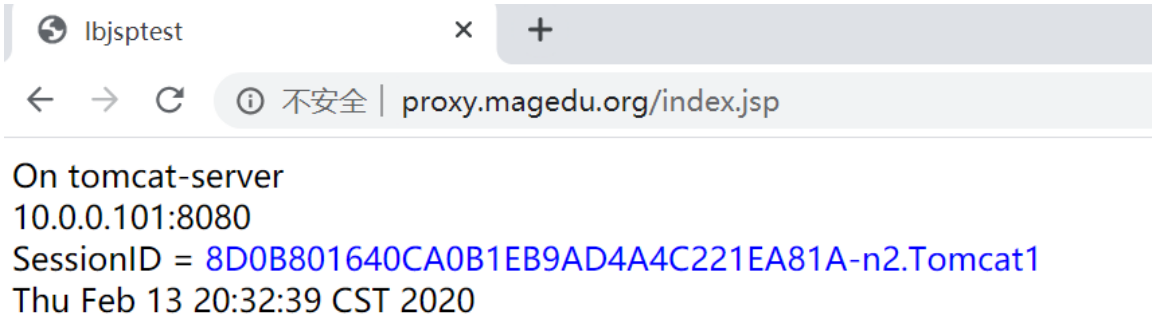
开启负载均衡调度器，通过http://proxy.magedu.com来访问看看效果

```
On tomcats
10.0.0.102:8080
SessionID = 2A19B1EB6D9649C9FED3E7277FDFD470-n2.Tomcat1
Wed Jun 26 16:32:11 CST 2019

On tomcats
10.0.0.101:8080
SessionID = 2A19B1EB6D9649C9FED3E7277FDFD470-n1.Tomcat2
Wed Jun 26 16:32:36 CST 2019
```

可以看到浏览器端被调度到不同Tomcat上，但是都获得了同样的SessionID。

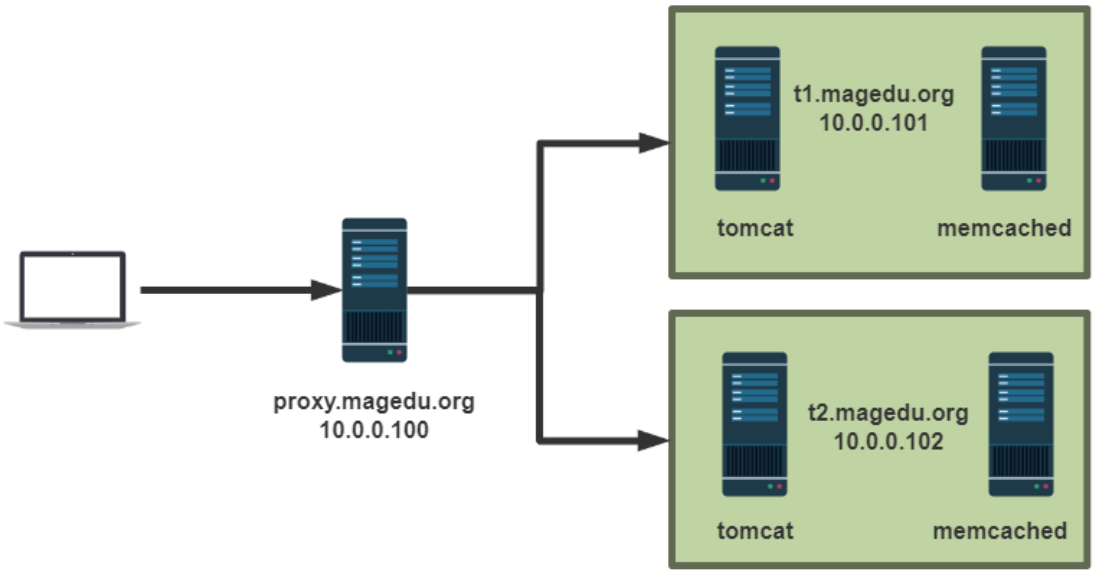
停止t2、n2看看效果，恢复看看效果。

范例：访问tomcat，查看memcached中SessionID信息




```
[root@centos8 ~]#python3 showmemcached.py
.....
[('10.0.0.48:11211 (1)', {'8D0B801640CA0B1EB9AD4A4C221EA81A-n2.Tomcat1': '[97 b; 1581598952 s]'})]
#以上结果表示SessionID来自于memcached的n2节点，但是最终是由tomcat1返回的SessionID
```

7.3.3 实战案例 1：tomcat和memcached集成在一台主机



环境准备：

- 时间同步，确保NTP或Chrony服务正常运行。
- 防火墙规则
- 禁用SELinux
- 三台主机

IP	主机名	服务	软件
10.0.0.100	proxy	调度器	CentOS8、Nginx、HTTPD
10.0.0.101	t1	tomcat1	CentOS8、JDK8、Tomcat8、memcached
10.0.0.102	t2	tomcat2	CentOS8、JDK8、Tomcat8、memcached

7.3.3.1 配置nginx充当proxy

```
[root@proxy ~]#cat /etc/nginx/nginx.conf
http {
......
    upstream tomcat-server {
        #ip_hash;
        server t1.magedu.org:8080;
        server t2.magedu.org:8080;
    }
    server {
        ......
        location / {
        }
        location ~* \.(jsp|do)$ {
            proxy_pass http://tomcat-server;
            #proxy_set_header Host  $http_host; #转发主机头至后端服务器
        }

[root@proxy ~]#cat /etc/hosts
10.0.0.100 proxy.magedu.org proxy
10.0.0.101 t1.magedu.org t1
10.0.0.102 t2.magedu.org t2
```

7.3.3.2 配置memcached

7.3.3.2.1 在 tomcat1 上配置 memcached

```
[root@t1 ~]#dnf -y install memcached
[root@t1 ~]#vim /etc/sysconfig/memcached
[root@t1 ~]#cat  /etc/sysconfig/memcached
PORT="11211"
USER="memcached"
MAXCONN="1024"
CACHESIZE="64"
#注释下面行
#OPTIONS="-l 127.0.0.1,::1"

[root@t1 ~]#systemctl enable --now  memcached.service
```

7.3.3.2.2 在 tomcat2 上配置 memcached

配置和t1相同

```
[root@t2 ~]#dnf -y install memcached
[root@t2 ~]#vim /etc/sysconfig/memcached
[root@t2 ~]#cat  /etc/sysconfig/memcached
PORT="11211"
USER="memcached"
MAXCONN="1024"
CACHESIZE="64"
#注释下面行
#OPTIONS="-l 127.0.0.1,::1"

[root@t2 ~]#systemctl enable --now  memcached.service
```

7.3.3.3 配置 tomcat

7.3.3.3.1 配置 tomcat1

```
[root@t1 tomcat]#vim conf/server.xml
  <Engine name="Catalina" defaultHost="t1.magedu.org" jvmRoute="Tomcat1">
  ......
        <Host name="t1.magedu.org" appBase="/data/webapps" autoDeploy="true" >
     </Host>
    </Engine>
  </Service>
</Server>

[root@t1 tomcat]#vim conf/context.xml
<Context>
......
    <Manager pathname="" />
    -->
###倒数第一行前,即</Context>行的前面,加下面内容
  <Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
    memcachedNodes="n1:10.0.0.101:11211,n2:10.0.0.102:11211"

    failoverNodes="n1"
```

```
    requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"

transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFact
ory"
    />
</Context>  #最后一行

#将相关包传到lib/目录下,共10个文件
asm-5.2.jar
kryo-3.0.3.jar
kryo-serializers-0.45.jar
memcached-session-manager-2.3.2.jar
memcached-session-manager-tc8-2.3.2.jar
minlog-1.3.1.jar
msm-kryo-serializer-2.3.2.jar
objenesis-2.6.jar
reflectasm-1.11.9.jar
spymemcached-2.12.3.jar

[root@t1 tomcat]#ls lib/ -t |tail
kryo-3.0.3.jar
asm-5.2.jar
objenesis-2.6.jar
reflectasm-1.11.9.jar
minlog-1.3.1.jar
kryo-serializers-0.45.jar
msm-kryo-serializer-2.3.2.jar
memcached-session-manager-tc8-2.3.2.jar
spymemcached-2.12.3.jar
memcached-session-manager-2.3.2.jar

[root@t1 tomcat]#systemctl restart tomcat

[root@t1 tomcat]#cat /data/webapps/ROOT/index.jsp
<%@ page import="java.util.*" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<h1> tomcat website </h1>
<div>On  <%=request.getServerName() %></div>
<div><%=request.getLocalAddr() + ":" + request.getLocalPort() %></div>
<div>SessionID = <span style="color:blue"><%=session.getId() %></span></div>
<%=new Date()%>
</body>
</html>
```

7.3.3.3.2 配置 tomcat2

```
#t2参考上面t1做配置
[root@t2 tomcat]#vim conf/server.xml
  <Engine name="Catalina" defaultHost="t2.magedu.org" jvmRoute="Tomcat2">
  ......
      <Host name="t2.magedu.org" appBase="/data/webapps" autoDeploy="true" >
```

```
     </Host>
    </Engine>
  </Service>
</Server>

[root@t2 tomcat]#vim conf/context.xml
<Context>
......
    <Manager pathname="" />
    -->
####################加下面内容################
  <Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
    memcachedNodes="n1:10.0.0.101:11211,n2:10.0.0.102:11211"

    failoverNodes="n2"  #只修改此行,和t1不同,其它都一样
    requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"

transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFact
ory"
    />
####################加上面内容################
</Context>

#将相关包传到lib/目录下
asm-5.2.jar
kryo-3.0.3.jar
kryo-serializers-0.45.jar
memcached-session-manager-2.3.2.jar
memcached-session-manager-tc8-2.3.2.jar
minlog-1.3.1.jar
msm-kryo-serializer-2.3.2.jar
objenesis-2.6.jar
reflectasm-1.11.9.jar
spymemcached-2.12.3.jar

[root@t2 tomcat]#ls lib/ -t |tail
kryo-3.0.3.jar
asm-5.2.jar
objenesis-2.6.jar
reflectasm-1.11.9.jar
minlog-1.3.1.jar
kryo-serializers-0.45.jar
msm-kryo-serializer-2.3.2.jar
memcached-session-manager-tc8-2.3.2.jar
spymemcached-2.12.3.jar
memcached-session-manager-2.3.2.jar

[root@t2 tomcat]#systemctl restart tomcat

[root@t2 tomcat]#cat /data/webapps/ROOT/index.jsp
<%@ page import="java.util.*" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
```

```
<h1> tomcat website </h1>
<div>On  <%=request.getServerName() %></div>
<div><%=request.getLocalAddr() + ":" + request.getLocalPort() %></div>
<div>SessionID = <span style="color:blue"><%=session.getId() %></span></div>
<%=new Date()%>
</body>
</html>
```

7.3.3.3.3 查看tomcat日志

```
[root@t1 tomcat]#tail -n 20 logs/catalina.out
2020-07-13 09:00:28.580 INFO net.spy.memcached.MemcachedConnection:  Setting
retryQueueSize to -1
2020-07-13 09:00:28.581 INFO net.spy.memcached.MemcachedConnection:  Added {QA
sa=/10.0.0.101:11211, #Rops=0, #Wops=0, #iq=0, topRop=null, topWop=null,
toWrite=0, interested=0} to connect queue
2020-07-13 09:00:28.581 INFO net.spy.memcached.MemcachedConnection:  Added {QA
sa=/10.0.0.102:11211, #Rops=0, #Wops=0, #iq=0, topRop=null, topWop=null,
toWrite=0, interested=0} to connect queue
13-Jul-2020 09:00:28.582 INFO [t1.magedu.org-startStop-1]
de.javakaffee.web.msm.RequestTrackingHostValve.<init> Setting ignorePattern to
.*\.(ico|png|gif|jpg|css|js)$
13-Jul-2020 09:00:28.582 INFO [t1.magedu.org-startStop-1]
de.javakaffee.web.msm.MemcachedSessionService.setLockingMode Setting lockingMode
to null
13-Jul-2020 09:00:28.582 INFO [t1.magedu.org-startStop-1]
de.javakaffee.web.msm.MemcachedSessionService.createTranscoderFactory Creating
transcoder factory de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory
13-Jul-2020 09:00:28.583 INFO [t1.magedu.org-startStop-1]
de.javakaffee.web.msm.serializer.kryo.KryoTranscoder.<init> Starting with
initialBufferSize 102400, maxBufferSize 2048000 and defaultSerializerFactory
de.javakaffee.web.msm.serializer.kryo.DefaultFieldSerializerFactory
13-Jul-2020 09:00:28.583 INFO [t1.magedu.org-startStop-1]
de.javakaffee.web.msm.MemcachedSessionService.startInternal --------
-  finished initialization:
- sticky: true
- operation timeout: 1000
- node ids: [n2]
- failover node ids: [n1]
- storage key prefix: null
- locking mode: null (expiration: 5s)
--------
13-Jul-2020 09:00:28.587 INFO [t1.magedu.org-startStop-1]
org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web
application directory [/data/webapps/ROOT] has finished in [160] ms
13-Jul-2020 09:00:28.596 INFO [main] org.apache.coyote.AbstractProtocol.start
Starting ProtocolHandler ["http-nio-8080"]
13-Jul-2020 09:00:28.619 INFO [main] org.apache.coyote.AbstractProtocol.start
Starting ProtocolHandler ["ajp-nio-0.0.0.0-8009"]
13-Jul-2020 09:00:28.621 INFO [main] org.apache.catalina.startup.Catalina.start
Server startup in 2097 ms
[root@t1 tomcat]#
```

7.3.3.4 python测试脚本

在t1 上安装部署python3环境,访问memcached

```
[root@t1 ~]#dnf -y install python3  python3-memcached
#或者下面方式也可以安装
[root@t1 ~]#dnf -y install python3
[root@t1 ~]#pip3 install python-memcached

#准备python测试脚本
[root@t1 ~]#cat showmemcached.py
#!/usr/bin/python3
import memcache
mc = memcache.Client(['10.0.0.101:11211','10.0.0.102:11211'], debug=True)
print('-' * 30)
#查看全部 key
#for x in mc.get_stats('items'):  # stats items 返回 items:5:number 1
#    print(x)
#print('-' * 30)

for x in mc.get_stats('cachedump 5 0'):  # stats cachedump 5 0 # 5和上面的items返回的值有关；0表示全部
    print(x)


[root@t1 ~]#chmod +x showmemcached.py

#运行python脚本
[root@t1 ~]#./showmemcached.py
('10.0.0.101:11211 (1)', {})
('10.0.0.102:11211 (1)', {})
```

7.3.3.5 浏览器访问

第一次刷新页面,可以看到下面显示

第一次运行脚本查看结果

```
[root@t1 ~]#./showmemcached.py
------------------------------
('10.0.0.101:11211 (1)', {})
('10.0.0.102:11211 (1)', {'3B4E6FB0934692150DD9944845830FCC-n2.Tomcat1': '[97 b;
1603970381 s]'})

#只有10.0.0.102上有信息,说明SessionID是由Tomcat1生成,并备份到n2,即t2上面的memcached
```

第二次刷新页面后,运行脚本可以查看到下面显示

第二次执行脚本

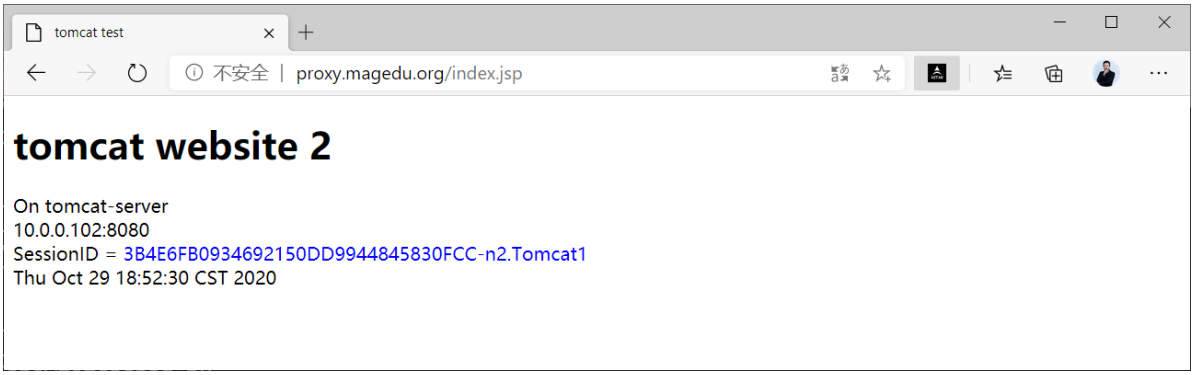
```
[root@t1 ~]#./showmemcached.py
------------------------------
('10.0.0.101:11211 (1)', {'3B4E6FB0934692150DD9944845830FCC-n1.Tomcat2': '[97 b;
1603970442 s]'})
('10.0.0.102:11211 (1)', {})
```

第三次刷新页面

第三次执行脚本

```
[root@t1 ~]#./showmemcached.py
------------------------------
('10.0.0.101:11211 (1)', {'3B4E6FB0934692150DD9944845830FCC-n1.Tomcat2': '[97 b;
1603970442 s]'})
('10.0.0.102:11211 (1)', {'3B4E6FB0934692150DD9944845830FCC-n2.Tomcat1': '[97 b;
1603970381 s]'})
```

第四次刷新页面



第四次执行脚本

```
[root@t1 ~]#./showmemcached.py
------------------------------
('10.0.0.101:11211 (1)', {'3B4E6FB0934692150DD9944845830FCC-n1.Tomcat2': '[97 b;
1603970442 s]'})
('10.0.0.102:11211 (1)', {'3B4E6FB0934692150DD9944845830FCC-n2.Tomcat1': '[97 b;
1603970381 s]'})
```

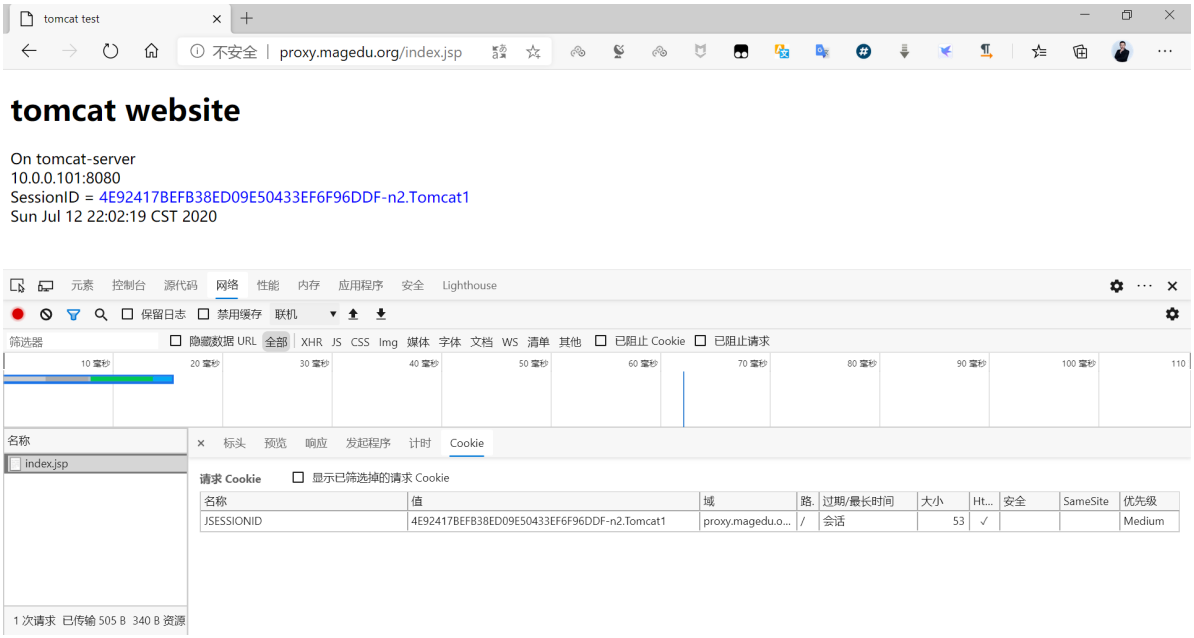
之后多次刷新页面,执行脚本后,session信息不在变化

7.3.3.6 故障访问

7.3.3.6.1 模拟tomcat故障

```
[root@t2 ~]#systemctl stop tomcat
```

刷新几次页面,看到下面SessionID显示不变



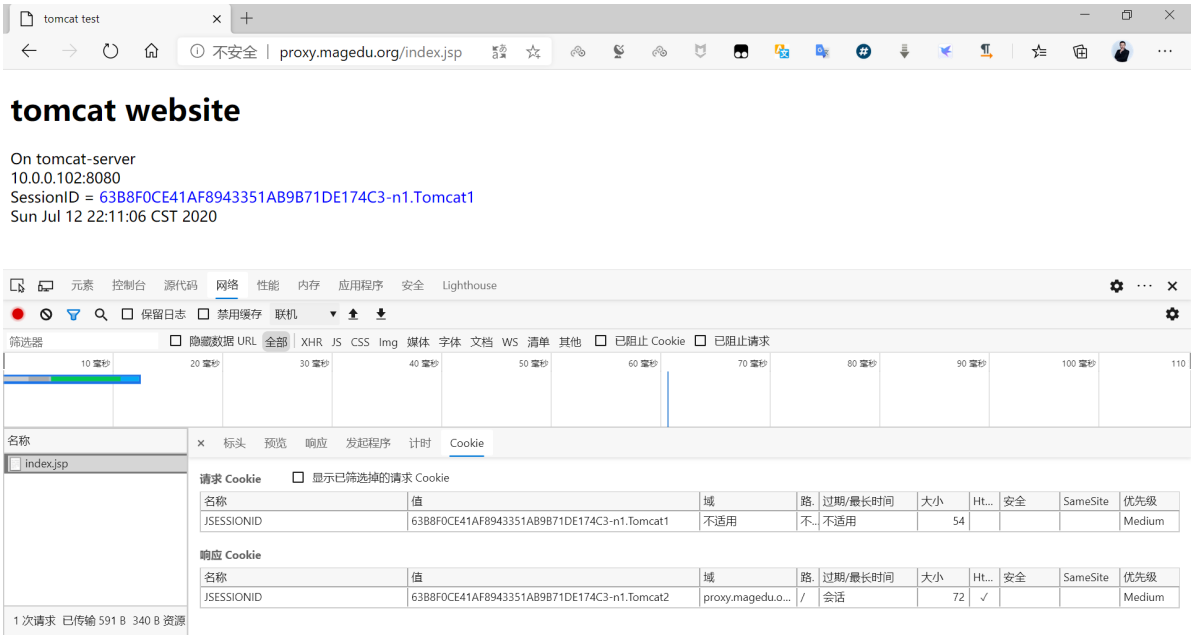
运行脚本看到下面结果

```
[root@t1 ~]#./showmemcached.py
--------------------------------
('10.0.0.101:11211 (1)', {'items:5:number': '1', 'items:5:number_hot': '0',
'items:5:number_warm': '0', 'items:5:number_cold': '1', 'items:5:age_hot': '0',
'items:5:age_warm': '0', 'items:5:age': '1531', 'items:5:evicted': '0',
'items:5:evicted_nonzero': '0', 'items:5:evicted_time': '0',
'items:5:outofmemory': '0', 'items:5:tailrepairs': '0', 'items:5:reclaimed':
'0', 'items:5:expired_unfetched': '0', 'items:5:evicted_unfetched': '0',
'items:5:evicted_active': '0', 'items:5:crawler_reclaimed': '0',
'items:5:crawler_items_checked': '9', 'items:5:lrutail_reflocked': '0',
'items:5:moves_to_cold': '2', 'items:5:moves_to_warm': '0',
'items:5:moves_within_lru': '0', 'items:5:direct_reclaims': '0',
'items:5:hits_to_hot': '0', 'items:5:hits_to_warm': '0', 'items:5:hits_to_cold':
'1', 'items:5:hits_to_temp': '0'})
('10.0.0.102:11211 (1)', {})
--------------------------------
('10.0.0.101:11211 (1)', {'4E92417BEFB38ED09E50433EF6F96DDF-n1.Tomcat2': '[97 b;
1594562924 s]'})
('10.0.0.102:11211 (1)', {})
```

7.3.3.6.2 模拟memcached故障

```
[root@t2 ~]#systemctl start tomcat
[root@t2 ~]#systemctl stop memcached.service
```

刷新几次页面可以看到下面显示



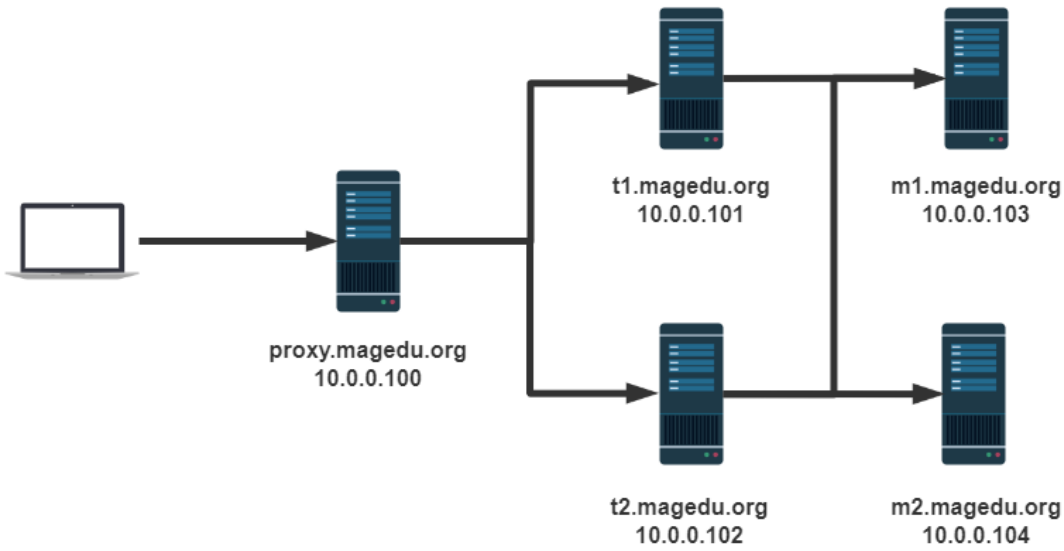
再次运行脚本

```
[root@t1 ~]#./showmemcached.py
------------------------------
MemCached: MemCache: inet:10.0.0.102:11211: connect: [Errno 111] Connection
refused.  Marking dead.
('10.0.0.101:11211 (1)', {'items:5:number': '2', 'items:5:number_hot': '0',
'items:5:number_warm': '0', 'items:5:number_cold': '2', 'items:5:age_hot': '0',
'items:5:age_warm': '0', 'items:5:age': '60', 'items:5:evicted': '0',
'items:5:evicted_nonzero': '0', 'items:5:evicted_time': '0',
'items:5:outofmemory': '0', 'items:5:tailrepairs': '0', 'items:5:reclaimed':
'1', 'items:5:expired_unfetched': '1', 'items:5:evicted_unfetched': '0',
'items:5:evicted_active': '0', 'items:5:crawler_reclaimed': '0',
'items:5:crawler_items_checked': '11', 'items:5:lrutail_reflocked': '0',
'items:5:moves_to_cold': '13', 'items:5:moves_to_warm': '0',
'items:5:moves_within_lru': '0', 'items:5:direct_reclaims': '0',
'items:5:hits_to_hot': '0', 'items:5:hits_to_warm': '0', 'items:5:hits_to_cold':
'5', 'items:5:hits_to_temp': '0'})
------------------------------
('10.0.0.101:11211 (1)', {'63B8F0CE41AF8943351AB9B71DE174C3-n1.Tomcat1': '[97 b;
1594564819 s]', '63B8F0CE41AF8943351AB9B71DE174C3-n1.Tomcat2': '[97 b;
1594564785 s]'})

#看到只有10.0.0.101有信息,并且生成两个sessionID,说明当memcached2挂掉后,tomcat1只能将
Session信息写入至memcached1做为备份
```

恢复memcached

```
[root@t2 ~]#systemctl start memcached.service
[root@t1 ~]#./showmemcached.py
------------------------------
('10.0.0.101:11211 (1)', {'items:5:number': '2', 'items:5:number_hot': '0',
'items:5:number_warm': '0', 'items:5:number_cold': '2', 'items:5:age_hot': '0',
'items:5:age_warm': '0', 'items:5:age': '23', 'items:5:evicted': '0',
'items:5:evicted_nonzero': '0', 'items:5:evicted_time': '0',
'items:5:outofmemory': '0', 'items:5:tailrepairs': '0', 'items:5:reclaimed':
'1', 'items:5:expired_unfetched': '1', 'items:5:evicted_unfetched': '0',
'items:5:evicted_active': '0', 'items:5:crawler_reclaimed': '0',
'items:5:crawler_items_checked': '13', 'items:5:lrutail_reflocked': '0',
'items:5:moves_to_cold': '20', 'items:5:moves_to_warm': '0',
'items:5:moves_within_lru': '0', 'items:5:direct_reclaims': '0',
'items:5:hits_to_hot': '0', 'items:5:hits_to_warm': '0', 'items:5:hits_to_cold':
'5', 'items:5:hits_to_temp': '0'})
('10.0.0.102:11211 (1)', {'items:5:number': '1', 'items:5:number_hot': '0',
'items:5:number_warm': '0', 'items:5:number_cold': '1', 'items:5:age_hot': '0',
'items:5:age_warm': '0', 'items:5:age': '20', 'items:5:evicted': '0',
'items:5:evicted_nonzero': '0', 'items:5:evicted_time': '0',
'items:5:outofmemory': '0', 'items:5:tailrepairs': '0', 'items:5:reclaimed':
'0', 'items:5:expired_unfetched': '0', 'items:5:evicted_unfetched': '0',
'items:5:evicted_active': '0', 'items:5:crawler_reclaimed': '0',
'items:5:crawler_items_checked': '0', 'items:5:lrutail_reflocked': '0',
'items:5:moves_to_cold': '2', 'items:5:moves_to_warm': '0',
'items:5:moves_within_lru': '0', 'items:5:direct_reclaims': '0',
'items:5:hits_to_hot': '0', 'items:5:hits_to_warm': '0', 'items:5:hits_to_cold':
'0', 'items:5:hits_to_temp': '0'})
------------------------------
('10.0.0.101:11211 (1)', {'63B8F0CE41AF8943351AB9B71DE174C3-n1.Tomcat2': '[97 b;
1594564785 s]', '63B8F0CE41AF8943351AB9B71DE174C3-n1.Tomcat1': '[97 b;
1594564819 s]'})
('10.0.0.102:11211 (1)', {'63B8F0CE41AF8943351AB9B71DE174C3-n2.Tomcat1': '[97 b;
1594564796 s]'})
```

7.3.4 实战案例 2 : tomcat和memcached 分别处于不同主机



环境准备：

- 时间同步，确保NTP或Chrony服务正常运行。
- 防火墙规则
- 禁用SELinux
- 五台主机

IP	主机名	服务	软件
10.0.0.100	proxy	调度器	CentOS8、Nginx、HTTPD
10.0.0.101	t1	tomcat1	CentOS8、JDK8、Tomcat8
10.0.0.102	t2	tomcat2	CentOS8、JDK8、Tomcat8
10.0.0.103	m1	memcached1	CentOS8、memcached
10.0.0.104	m2	memcached2	CentOS8、memcached

7.3.4.1 准备proxy主机的配置，利用nginx作为反向代理

```
[root@proxy ~]#cat /etc/nginx/nginx.conf
http {
        ......
        upstream tomcat-server {
        server t1.magedu.org:8080;
        server t2.magedu.org:8080;
}
    server {
        ......
        location / {
        }
        location ~* \.(jsp|do)$ {
            proxy_pass http://tomcat-server;
            #proxy_set_header Host  $http_host; #转发主机头至后端服务器
        }

[root@proxy ~]#cat /etc/hosts
10.0.0.100 proxy.magedu.org proxy
10.0.0.101 t1.magedu.org t1
10.0.0.102 t2.magedu.org t2

#准备一台测试机(可选)
[root@centos7 ~]#cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
centos7.wangxiaochun.com
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.0.0.100 proxy.magedu.org
[root@centos7 ~]#hostname -I
10.0.0.7
```

7.3.4.2 在m1和m2上分别配置memcached

```
#在m1和m2上做相同的配置
[root@m1 ~]#dnf -y install memcached
[root@m1 ~]#vim /etc/sysconfig/memcached
[root@m1 ~]#cat  /etc/sysconfig/memcached
PORT="11211"
USER="memcached"
MAXCONN="1024"
CACHESIZE="64"
#OPTIONS="-l 127.0.0.1,::1"
OPTIONS=""
[root@m1 ~]#systemctl start memcached.service

#m2的配置和m1相同
```

7.3.4.3 在t1和t2上准备tomcat

t1 的配置

```
[root@t1 tomcat]#vim conf/server.xml
  <Engine name="Catalina" defaultHost="t1.magedu.org" jvmRoute="Tomcat1">
  ......
        <Host name="t1.magedu.org" appBase="/data/webapps" autoDeploy="true" >
     </Host>
    </Engine>
  </Service>
</Server>

[root@t1 tomcat]#vim conf/context.xml
.....
<Context>
......
    <Manager pathname="" />
    -->
#在最后一行前加下面内容
  <Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
    memcachedNodes="n1:10.0.0.103:11211,n2:10.0.0.104:11211"
    failoverNodes="n1"
    requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"

transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFact
ory"
    />
</Context> #此行是最后一行

#将相关包传到lib/目录下
asm-5.2.jar
kryo-3.0.3.jar
kryo-serializers-0.45.jar
memcached-session-manager-2.3.2.jar
memcached-session-manager-tc8-2.3.2.jar
minlog-1.3.1.jar
msm-kryo-serializer-2.3.2.jar
objenesis-2.6.jar
reflectasm-1.11.9.jar
```

```
spymemcached-2.12.3.jar

[root@t1 tomcat]#ls lib/ -t |tail
kryo-3.0.3.jar
asm-5.2.jar
objenesis-2.6.jar
reflectasm-1.11.9.jar
minlog-1.3.1.jar
kryo-serializers-0.45.jar
msm-kryo-serializer-2.3.2.jar
memcached-session-manager-tc8-2.3.2.jar
spymemcached-2.12.3.jar
memcached-session-manager-2.3.2.jar

[root@t1 tomcat]#cat /data/webapps/ROOT/index.jsp
<%@ page import="java.util.*" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<div>On  <%=request.getServerName() %></div>
<div><%=request.getLocalAddr() + ":" + request.getLocalPort() %></div>
<div>SessionID = <span style="color:blue"><%=session.getId() %></span></div>
<%=new Date()%>
</body>
</html>

[root@t1 ~]#systemctl restart tomcat
```

t2参考上面t1做类似的配置

```
[root@t2 tomcat]#vim conf/server.xml
  <Engine name="Catalina" defaultHost="t2.magedu.org" jvmRoute="Tomcat2">
  ......
       <Host name="t2.magedu.org" appBase="/data/webapps" autoDeploy="true" >
     </Host>
    </Engine>
  </Service>
</Server>

[root@t2 tomcat]#vim conf/context.xml
.....
<Context>
......
    <Manager pathname="" />
    -->
#在最后一行前加下面内容
  <Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
    memcachedNodes="n1:10.0.0.103:11211,n2:10.0.0.104:11211"
    failoverNodes="n2" #只修改此行,和t1不同,其它都一样
    requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"

transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFact
ory"
```

```
    />
</Context> #此行是最后一行

#将相关包传到lib/目录下
asm-5.2.jar
kryo-3.0.3.jar
kryo-serializers-0.45.jar
memcached-session-manager-2.3.2.jar
memcached-session-manager-tc8-2.3.2.jar
minlog-1.3.1.jar
msm-kryo-serializer-2.3.2.jar
objenesis-2.6.jar
reflectasm-1.11.9.jar
spymemcached-2.12.3.jar

[root@t2 tomcat]#cat /data/webapps/ROOT/index.jsp
<%@ page import="java.util.*" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<div>On  <%=request.getServerName() %></div>
<div><%=request.getLocalAddr() + ":" + request.getLocalPort() %></div>
<div>SessionID = <span style="color:blue"><%=session.getId() %></span></div>
<%=new Date()%>
</body>
</html>

[root@t2 ~]#systemctl restart tomcat
```

查看日志

```
[root@t1 ~]#tail  /usr/local/tomcat/logs/catalina.out
.......
...... INFO [t1.magedu.org-startStop-1]
de.javakaffee.web.msm.MemcachedSessionService.startInternal --------
-  finished initialization:
- sticky: true
- operation timeout: 1000
- node ids: [n2]
- failover node ids: [n1]
- storage key prefix: null
- locking mode: null (expiration: 5s)
```

7.3.4.4 测试的python脚本

在proxy 上安装部署python3环境,访问memcached

```
[root@proxy ~]#dnf -y install python3 python3-memcached

#准备python测试脚本
[root@proxy ~]#cat showmemcached.py
```

```
#!/usr/bin/python3
import memcache
mc = memcache.Client(['10.0.0.103:11211','10.0.0.104:11211'], debug=True)
print('-' * 30)
for x in mc.get_stats('cachedump 5 0'):  # stats cachedump 5 0 # 5和上面的items返回的值有关；0表示全部
    print(x)

[root@proxy ~]#chmod +x showmemcached.py

#运行python脚本
[root@proxy ~]#./showmemcached.py
------------------------------
('10.0.0.103:11211 (1)', {})
('10.0.0.104:11211 (1)', {})
```

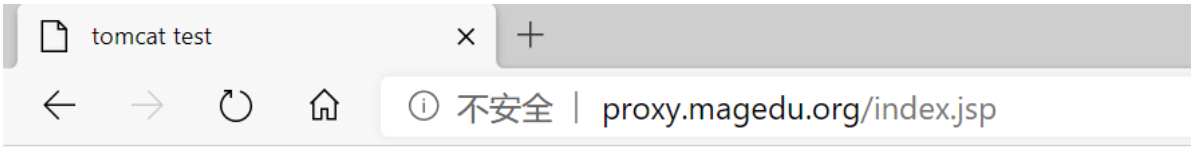
7.3.4.5 浏览器访问测试

第一次刷新浏览器可以看到以下显示

运行脚本,可以看到只有m2上有SessionID信息

```
[root@proxy ~]#./showmemcached.py
------------------------------
('10.0.0.103:11211 (1)', {})
('10.0.0.104:11211 (1)', {'8F9B4782FFFE6F1840728C344A179EE8-n2.Tomcat1': '[97 b; 1594812395 s]'})
```

第二次刷新页面,可以看到主机轮询,SessionID不变,但SessionID来自别一个tomcat



tomcat website

On tomcat-server
10.0.0.102:8080
SessionID = 8F9B4782FFFE6F1840728C344A179EE8-n1.Tomcat2
Wed Jul 15 19:01:01 CST 2020

多次刷新页面,可以看到主机轮询,SessionID不变,发现以下规律

当tomcat为t1,发现SessionID为n1.Tomcat2

当tomcat为t2,发现SessionID为n2.Tomcat1

tomcat website

On tomcat-server
10.0.0.102:8080
SessionID = 8F9B4782FFFE6F1840728C344A179EE8-n2.Tomcat1
Wed Jul 15 19:15:18 CST 2020

tomcat website

On tomcat-server
10.0.0.101:8080
SessionID = 8F9B4782FFFE6F1840728C344A179EE8-n1.Tomcat2
Wed Jul 15 19:18:18 CST 2020

运行脚本,可以看到m1和m2上都有了SessionID信息

```
[root@proxy ~]#./showmemcached.py
------------------------------
('10.0.0.103:11211 (1)', {'8F9B4782FFFE6F1840728C344A179EE8-n1.Tomcat2': '[97 b;
1594812654 s]'})
('10.0.0.104:11211 (1)', {'8F9B4782FFFE6F1840728C344A179EE8-n2.Tomcat1': '[97 b;
1594812396 s]'})
[root@proxy ~]#
```

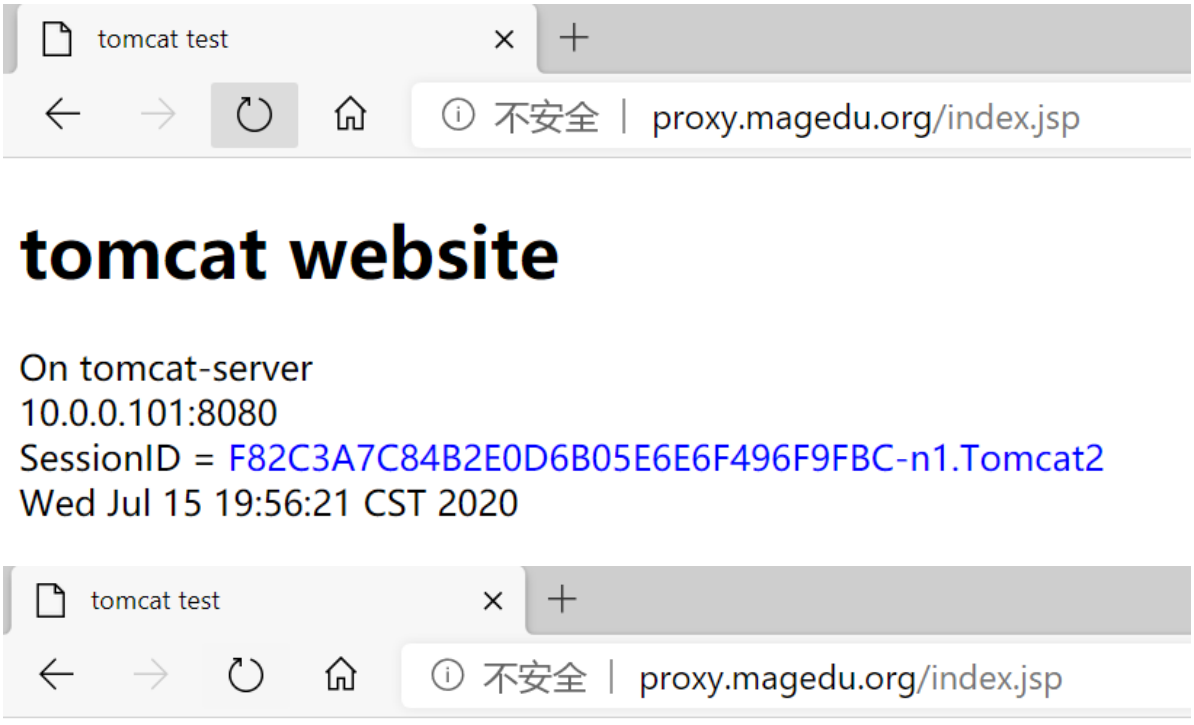
7.3.4.6 模拟故障

```
[root@t2 ~]#systemctl stop tomcat
```

多次刷新页面,SessionID不变

```
[root@m2 ~]#systemctl stop memcached.service
```

多次刷新页面,SessionID不变



运行脚本

```
[root@proxy ~]#./showmemcached.py
------------------------------
MemCached: MemCache: inet:10.0.0.104:11211: connect: [Errno 111] Connection
refused.  Marking dead.
('10.0.0.103:11211 (1)', {'F82C3A7C84B2E0D6B05E6E6F496F9FBC-n1.Tomcat2': '[97 b;
1594815951 s]', 'F82C3A7C84B2E0D6B05E6E6F496F9FBC-n1.Tomcat1': '[97 b;
1594815949 s]'})
```

7.4 non-sticky 模式

7.4.1 non-sticky 模式工作原理

non-sticky 模式即前端tomcat和后端memcached**无关联(无粘性)**关系

从msm 1.4.0之后版本开始支持non-sticky模式。

Tomcat session为中转Session，对每一个SessionID随机选中后端的memcached节点n1(或者n2)为主session，而另一个memcached节点n2(或者是n1)为备session。产生的新的Session会发送给主、备memcached，并清除本地Session。

后端两个memcached服务器对一个session来说是一个是主,一个是备,但对所有session信息来说每个memcached即是主同时也是备

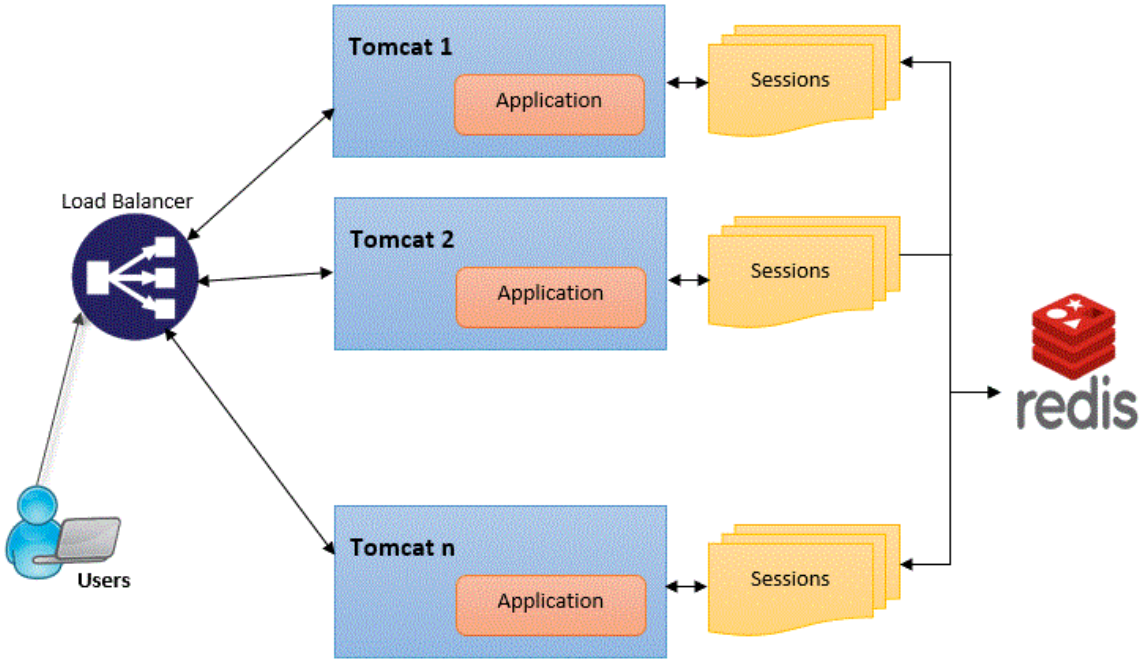
如果n1下线，n2则转正。n1再次上线，n2依然是主Session存储节点。

7.4.2 memcached配置

放到`$CATALINA_HOME/conf/context.xml`中

```
<Context>
  ...
  <Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
    memcachedNodes="n1:10.0.0.101:11211,n2:10.0.0.102:11211"
    sticky="false"
    sessionBackupAsync="false"
    lockingMode="uriPattern:/path1|/path2"
    requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"

 transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFac
tory"
    />
</Context>
```

7.4.3 redis 配置



支持将session存放在Redis中,但当前对Redis的支持不允许连接到多个Redis节点,可以通过Redis的集群服务实现防止redis的单点失败

参考文档：

https://github.com/ran-jit/tomcat-cluster-redis-session-manager/wiki

https://github.com/magro/memcached-session-manager/wiki/SetupAndConfiguration#example-for-non-sticky-sessions--kryo--redis

> **Example for non-sticky sessions + kryo + Redis**
>
> The following example shows a configuration which uses a Redis server at the URL "redis.example.com" for session storage for non-sticky sessions. Here the configuration (for both/all tomcats) would look like this:
>
> ```
> <Context>
>   ...
>   <Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
>     memcachedNodes="redis://redis.example.com"
>     sticky="false"
>     sessionBackupAsync="false"
>     lockingMode="uriPattern:/path1|/path2"
>     requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"
>     transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory"
>     />
> </Context>
> ```
>
> The configuration supports userinfo as part of the URI for password protected Redis nodes. The format is "redis://:password@redis.example.com:portnumber". If portnumber is not specified, the default port 6379 is used. The built-in support for Redis does currently not allow connections to multiple Redis nodes, nor does it support the failoverNodes property. However, with Redis, automatic failover could be implemented directly in Redis by building a Redis cluster. For example, Amazon ElastiCache for Redis implements a failover mode which constantly replicates all data from the primary node to one or more slaves, then, if the master goes down, promotes another node to be the master and in the process changes the DNS entry of the primary node to point to the new master. MSM supports this mode by automatically reconnecting to the Redis server when the connection goes down, potentially picking up the new DNS settings and therefore connecting to the new master. In the future, support for Redis Sentinel or Redis Cluster may be added.

下载 jedis.jar，放到`$CATALINA_HOME/lib/`，对应本次安装就是/usr/local/tomcat/lib。

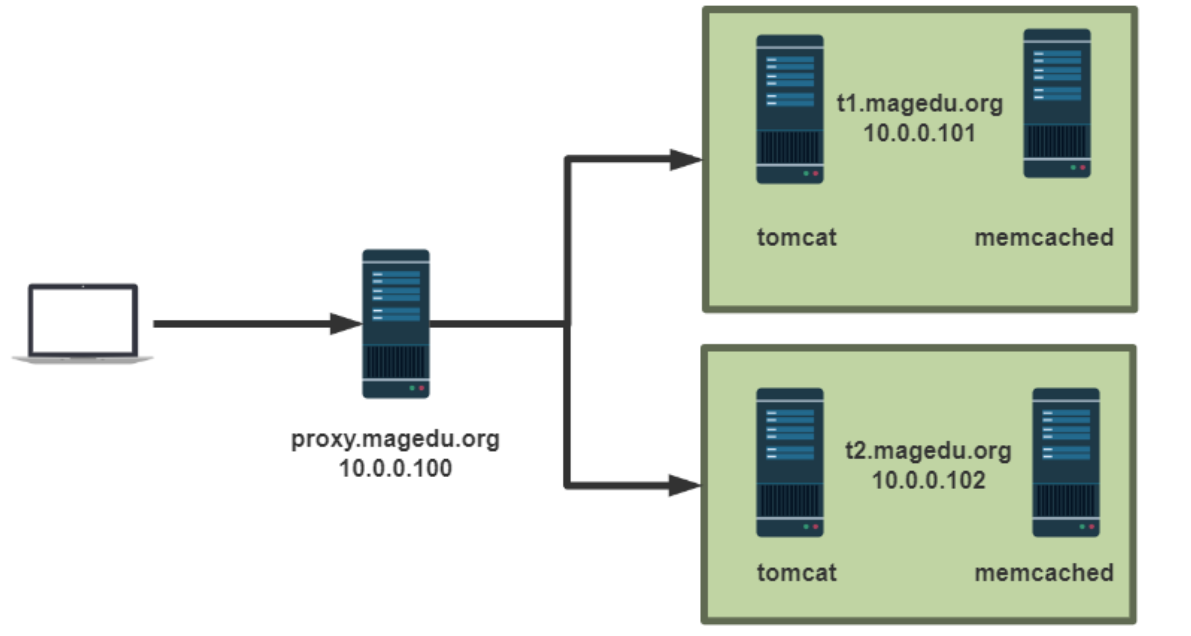
```
# yum install redis
# vim /etc/redis.conf
bind 0.0.0.0

# systemctl start redis
```

放到 `$CATALINA_HOME/conf/context.xml` 中

```
<Context>
  ...
  <Manager  className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
    memcachedNodes="redis://:password@redis.example.com:portnumber"
    sticky="false"
    sessionBackupAsync="false"
    lockingMode="uriPattern:/path1|/path2"
    requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"

 transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFac
tory"
    />
</Context>
```

浏览器访问，使用redis相关工具可以观察到redis中的信息

7.4.4 实战案例: memcached 实现non-sticky模式



环境准备：

- 时间同步，确保NTP或Chrony服务正常运行。
- 防火墙规则
- 禁用SELinux
- 三台主机

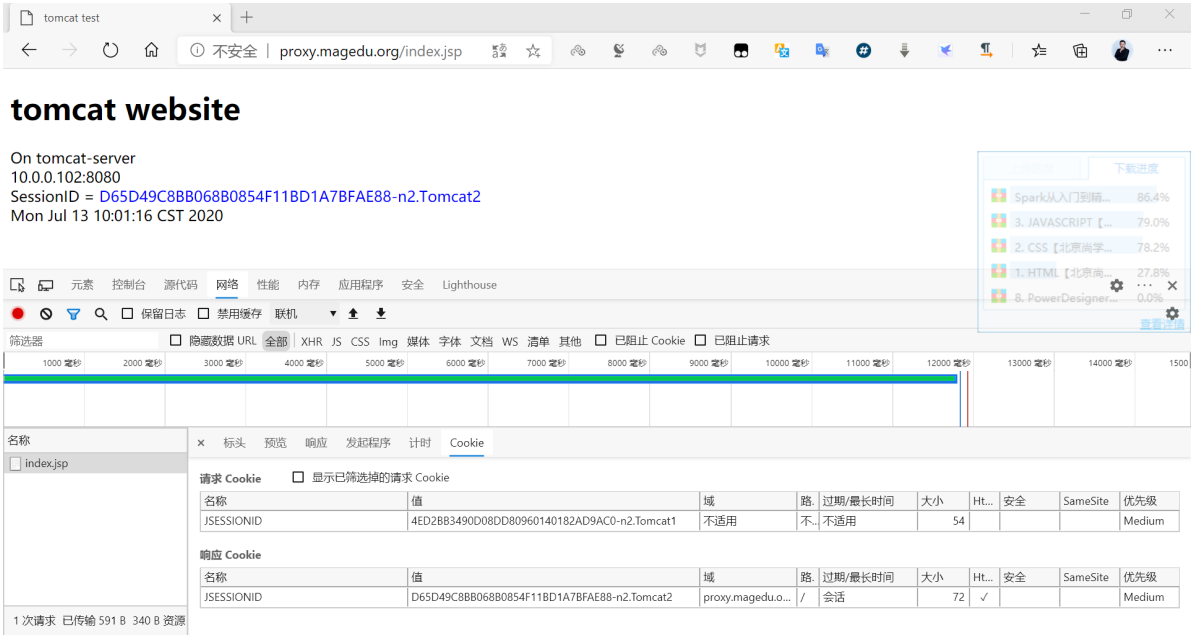
IP	主机名	服务	
10.0.0.100	proxy	调度器	Nginx、HTTPD
10.0.0.101	t1	tomcat1	JDK8、Tomcat8,memcached
10.0.0.102	t2	tomcat2	JDK8、Tomcat8,memcached

7.4.4.1 修改tomcat配置

```
#在前面实验基础上修改,memcached配置不变,只需要修改tomcat配置
[root@t1 tomcat]#vim conf/context.xml
<Context>
  ...
  <Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
    memcachedNodes="n1:10.0.0.101:11211,n2:10.0.0.102:11211"
    sticky="false"                    #下面三行和sticky模式不同
    sessionBackupAsync="false"
    lockingMode="uriPattern:/path1|/path2"
    requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"

transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFact
ory"
    />
</Context>

[root@t1 tomcat]#systemctl restart tomcat

#查看日志
[root@t1 ~]#tail -f /usr/local/tomcat/logs/catalina.out
..... INFO [t1.magedu.org-startStop-1]
de.javakaffee.web.msm.MemcachedSessionService.startInternal --------
-  finished initialization:
- sticky: false
- operation timeout: 1000
- node ids: [n1, n2]
- failover node ids: []
- storage key prefix: null
- locking mode: uriPattern:/path1|/path2 (expiration: 5s)
.......

#t2和t1相同操作
[root@t2 tomcat]#vim conf/context.xml
[root@t2 tomcat]#systemctl restart tomcat

#运行脚本查看key
[root@t1 ~]#cat ./showmemcached.py
#!/usr/bin/python3
import memcache
mc = memcache.Client(['10.0.0.101:11211','10.0.0.102:11211'], debug=True)
print('-' * 30)
```

```
for x in mc.get_stats('cachedump 5 0'):  # stats cachedump 5 0 # 5和上面的items返回的值有关；0表示全部
    print(x)

[root@t1 ~]#./showmemcached.py
------------------------------
('10.0.0.101:11211 (1)', {})
('10.0.0.102:11211 (1)', {})
[root@t1 ~]#
```

7.4.4.2 通过浏览器访问




```
#再次运行脚本后可以看到,t2为memcached的主节点,t1为备份节点
[root@t1 ~]#./showmemcached.py
------------------------------
('10.0.0.101:11211 (1)', {'bak:D65D49C8BB068B0854F11BD1A7BFAE88-n2.Tomcat2':
'[97 b; 1594609275 s]'})
('10.0.0.102:11211 (1)', {'D65D49C8BB068B0854F11BD1A7BFAE88-n2.Tomcat2': '[97 b;
1594609274 s]'})
```

7.4.4.3 模拟故障

模拟t1的memcached 故障

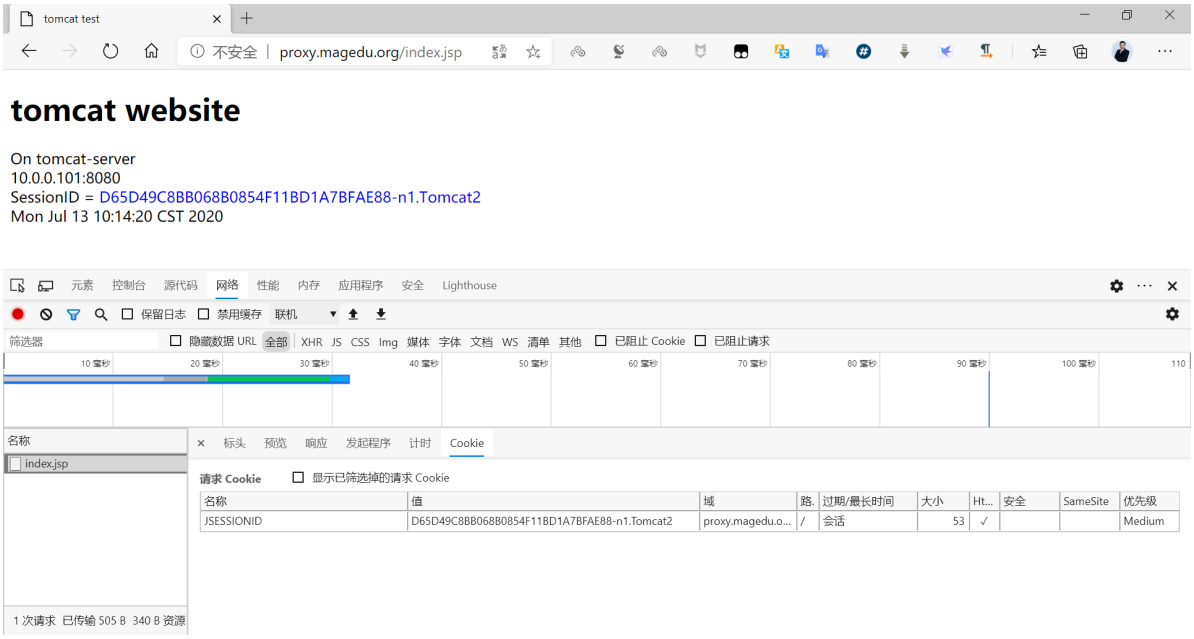
```
[root@t1 ~]#systemctl stop memcached.service
[root@t1 ~]#./showmemcached.py
------------------------------
MemCached: MemCache: inet:10.0.0.101:11211: connect: [Errno 111] Connection
refused.  Marking dead.
('10.0.0.102:11211 (1)', {'D65D49C8BB068B0854F11BD1A7BFAE88-n2.Tomcat2': '[97 b;
1594609274 s]'})

[root@t1 ~]#systemctl start memcached.service
[root@t1 ~]#./showmemcached.py
------------------------------
```

```
('10.0.0.101:11211 (1)', {})
('10.0.0.102:11211 (1)', {'D65D49C8BB068B0854F11BD1A7BFAE88-n2.Tomcat2': '[97 b;
1594609274 s]'})

#刷新几次页面后,再运行脚本
[root@t1 ~]#./showmemcached.py
--------------------------------
('10.0.0.101:11211 (1)', {'bak:D65D49C8BB068B0854F11BD1A7BFAE88-n2.Tomcat2':
'[97 b; 1594609854 s]'})
('10.0.0.102:11211 (1)', {'D65D49C8BB068B0854F11BD1A7BFAE88-n2.Tomcat2': '[97 b;
1594609274 s]'})
```

模拟t2的memcached 故障

```
[root@t2 ~]#systemctl stop memcached.service
[root@t1 ~]#./showmemcached.py
--------------------------------
MemCached: MemCache: inet:10.0.0.102:11211: connect: [Errno 111] Connection
refused.  Marking dead.
('10.0.0.101:11211 (1)', {'bak:D65D49C8BB068B0854F11BD1A7BFAE88-n2.Tomcat2':
'[97 b; 1594609854 s]'})
```

再刷新页面

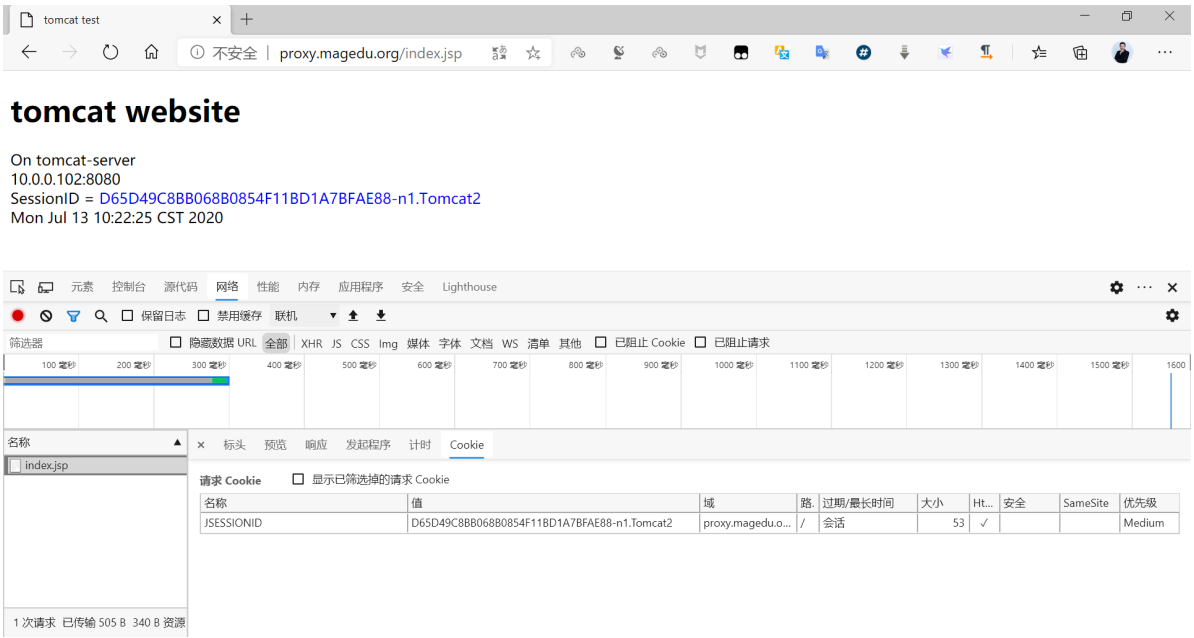



```
#运行脚本看到在t1上生成相同的SessionID,并成为memcached新主
[root@t1 ~]#./showmemcached.py
--------------------------------
MemCached: MemCache: inet:10.0.0.102:11211: connect: [Errno 111] Connection
refused.  Marking dead.
('10.0.0.101:11211 (1)', {'D65D49C8BB068B0854F11BD1A7BFAE88-n1.Tomcat2': '[97 b;
1594610030 s]', 'bak:D65D49C8BB068B0854F11BD1A7BFAE88-n2.Tomcat2': '[97 b;
1594609854 s]'})


[root@t2 ~]#systemctl start memcached.service
[root@t1 ~]#./showmemcached.py
--------------------------------
```

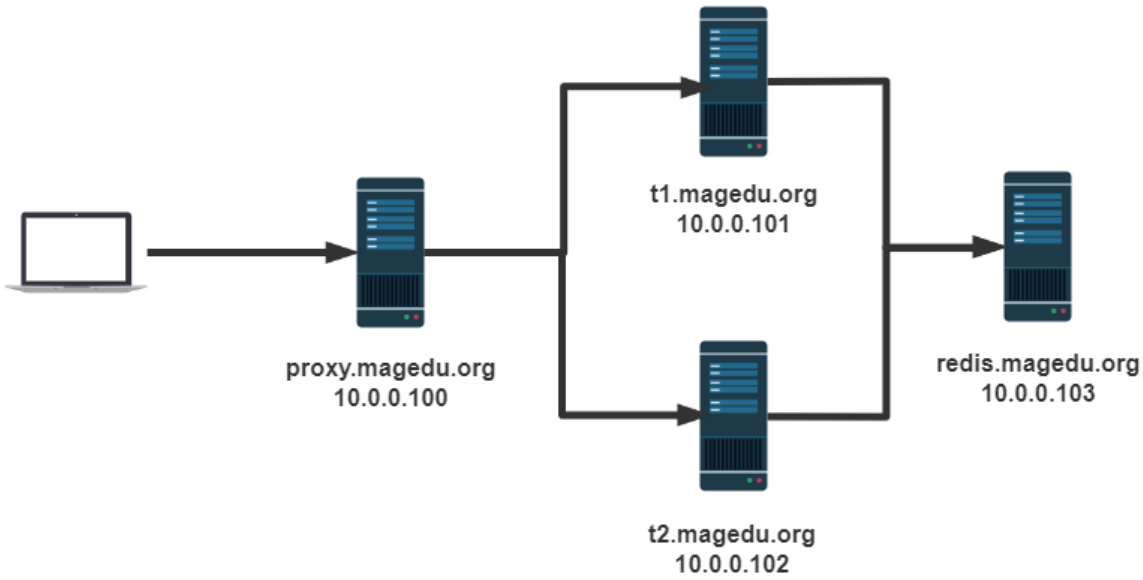
```
('10.0.0.101:11211 (1)', {'D65D49C8BB068B0854F11BD1A7BFAE88-n1.Tomcat2': '[97 b;
1594610030 s]', 'bak:D65D49C8BB068B0854F11BD1A7BFAE88-n2.Tomcat2': '[97 b;
1594609854 s]'})
('10.0.0.102:11211 (1)', {})

#再刷新页,运行脚本看到t2成为备用节点
[root@t1 ~]#./showmemcached.py
--------------------------------
('10.0.0.101:11211 (1)', {'D65D49C8BB068B0854F11BD1A7BFAE88-n1.Tomcat2': '[97 b;
1594610030 s]', 'bak:D65D49C8BB068B0854F11BD1A7BFAE88-n2.Tomcat2': '[97 b;
1594609854 s]'})
('10.0.0.102:11211 (1)', {'bak:D65D49C8BB068B0854F11BD1A7BFAE88-n1.Tomcat2':
'[97 b; 1594610367 s]'})
```

模拟t1的tomcat故障,刷新页面,可以看到SessionID不变

```
[root@t1 ~]#systemctl stop tomcat
```



7.4.5 实战案例: redis 实现 non-sticky 模式的msm



环境准备：

- 时间同步，确保NTP或Chrony服务正常运行
- 防火墙规则
- 禁用SELinux
- 四台主机

IP	主机名	服务	软件包
10.0.0.100	proxy	调度器	Nginx、HTTPD
10.0.0.101	t1.magedu.org	tomcat1	JDK8、Tomcat8
10.0.0.102	t2.magedu.org	tomcat2	JDK8、Tomcat8
10.0.0.103	redis.magedu.org	redis	Redis

7.4.5.1 上传redis库到tomcat服务器

```
[root@t1 ~]#ll /usr/local/tomcat/lib/jedis-3.0.0.jar
-rw-r--r-- 1 root root 586620 Jun 26  2019 /usr/local/tomcat/lib/jedis-3.0.0.jar
[root@t2 ~]#ll /usr/local/tomcat/lib/jedis-3.0.0.jar
-rw-r--r-- 1 root root 586620 Jun 26  2019 /usr/local/tomcat/lib/jedis-3.0.0.jar
```

7.4.5.2 安装并配置 Redis 服务

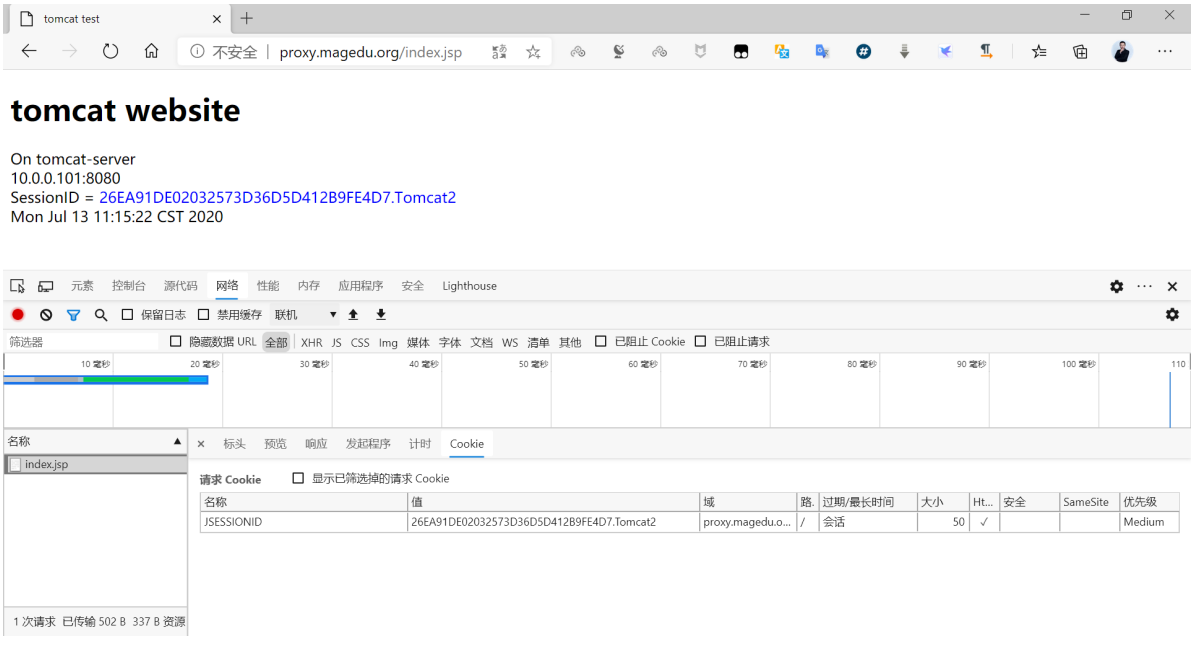
```
[root@redis ~]#dnf -y install redis
[root@redis ~]#sed -i.bak 's/^bind.*/bind 0.0.0.0/' /etc/redis.conf
[root@redis ~]#systemctl enable --now redis
[root@redis ~]#ss -ntl
State             Recv-Q   Send-Q Local Address:Port Peer Address:Port

LISTEN            0        128          0.0.0.0:22    0.0.0.0:*
LISTEN            0        100        127.0.0.1:25    0.0.0.0:*
LISTEN            0        128          0.0.0.0:6379  0.0.0.0:*
LISTEN            0        128             [::]:22       [::]:*
LISTEN            0        100             [::1]:25      [::]:*
```

7.4.5.3 修改tomcat 配置指定redis服务器地址

```
[root@t1 ~]#vim /usr/local/tomcat/conf/context.xml
<Context>
  ...
  <Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
    memcachedNodes="redis://10.0.0.103:6379"      #和non-sticky的memcached相比,只修改此行
    sticky="false"
    sessionBackupAsync="false"
    lockingMode="uriPattern:/path1|/path2"
    requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"

transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory"
    />
</Context>
[root@t1 ~]#systemctl restart tomcat

#查看日志
[root@t1 ~]#tail -f /usr/local/tomcat/logs/catalina.out
....., INFO [t1.magedu.org-startStop-1]
de.javakaffee.web.msm.MemcachedSessionService.startInternal --------
-  finished initialization:
- sticky: false
- operation timeout: 1000
- node ids: []
- failover node ids: []
- storage key prefix: null
- locking mode: uriPattern:/path1|/path2 (expiration: 5s)
.......

#t2和t1配置相同
[root@t2 ~]#vim /usr/local/tomcat/conf/context.xml
[root@t2 ~]#systemctl restart tomcat
```

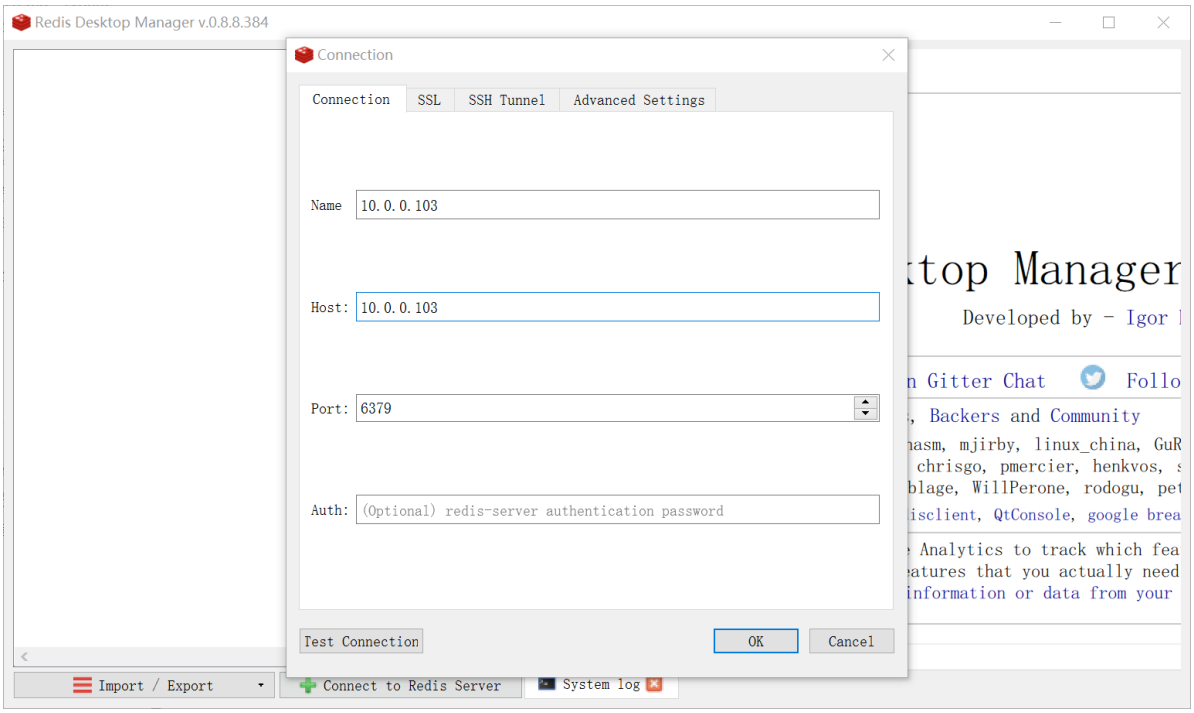
7.4.5.4 测试访问

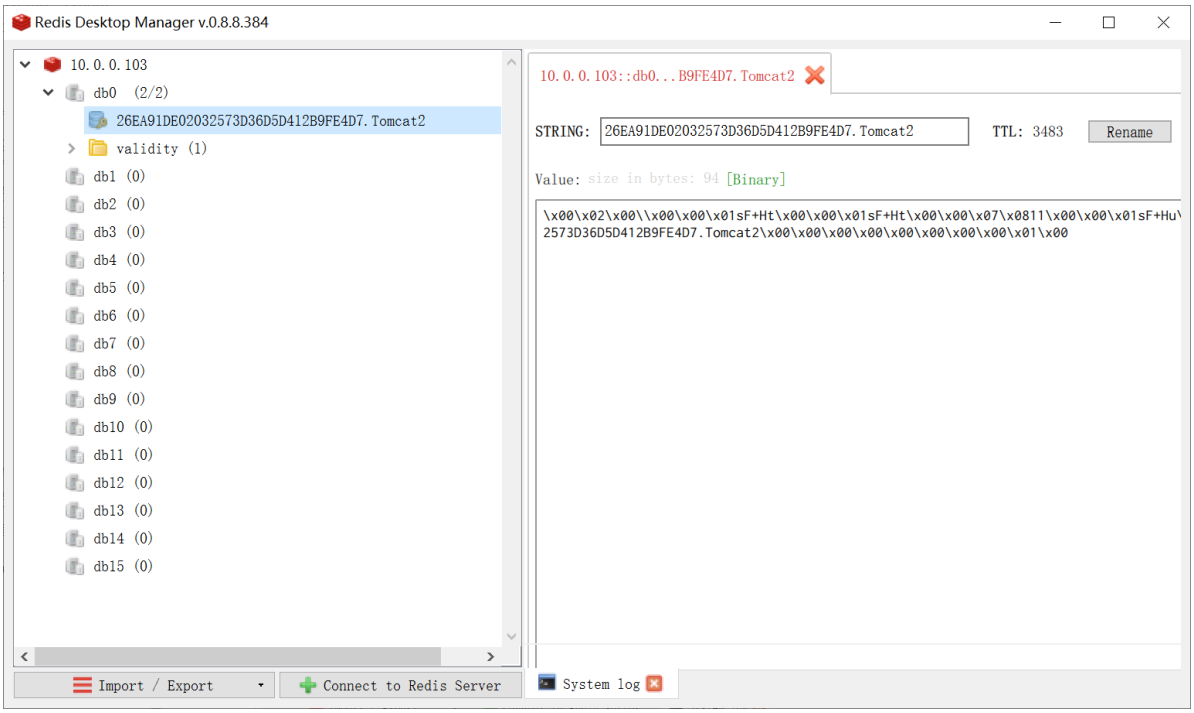
浏览器刷新访问多次,主机轮询,但SessionID不变

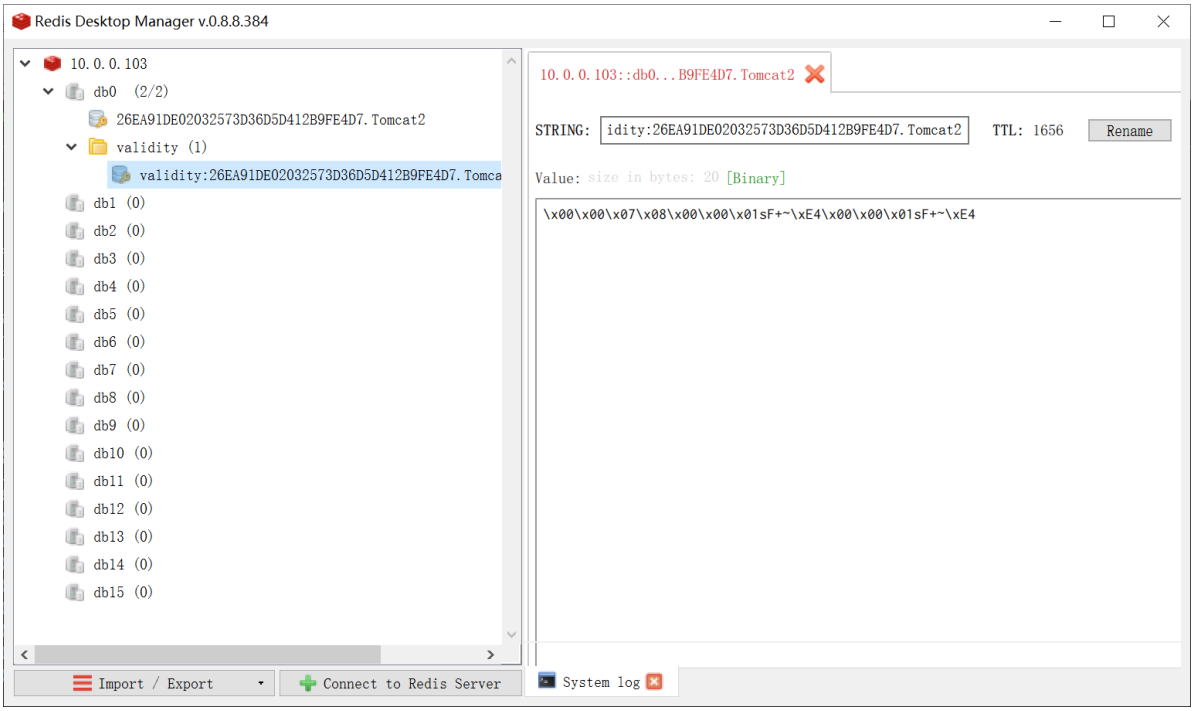


使用redis工具连接redis 查看SessionID

```
[root@redis ~]#redis-cli
127.0.0.1:6379> KEYS *
1) "4245D5D28B2CDAB292623A01DD5D2C83.Tomcat2"
2) "validity:4245D5D28B2CDAB292623A01DD5D2C83.Tomcat2"
127.0.0.1:6379> GET 4245D5D28B2CDAB292623A01DD5D2C83.Tomcat2
"\x00\x02\x00\\\x00\x00\x01sR\xf6\x8b@\x00\x00\x01sR\xf6\x8b@\x00\x00\a\b11\x00\
x00\x01sR\xf6\x8b@\x00\x00\x01sR\xf6\x8bA\x00(4245D5D28B2CDAB292623A01DD5D2C83.T
omcat2\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00"
127.0.0.1:6379>
```





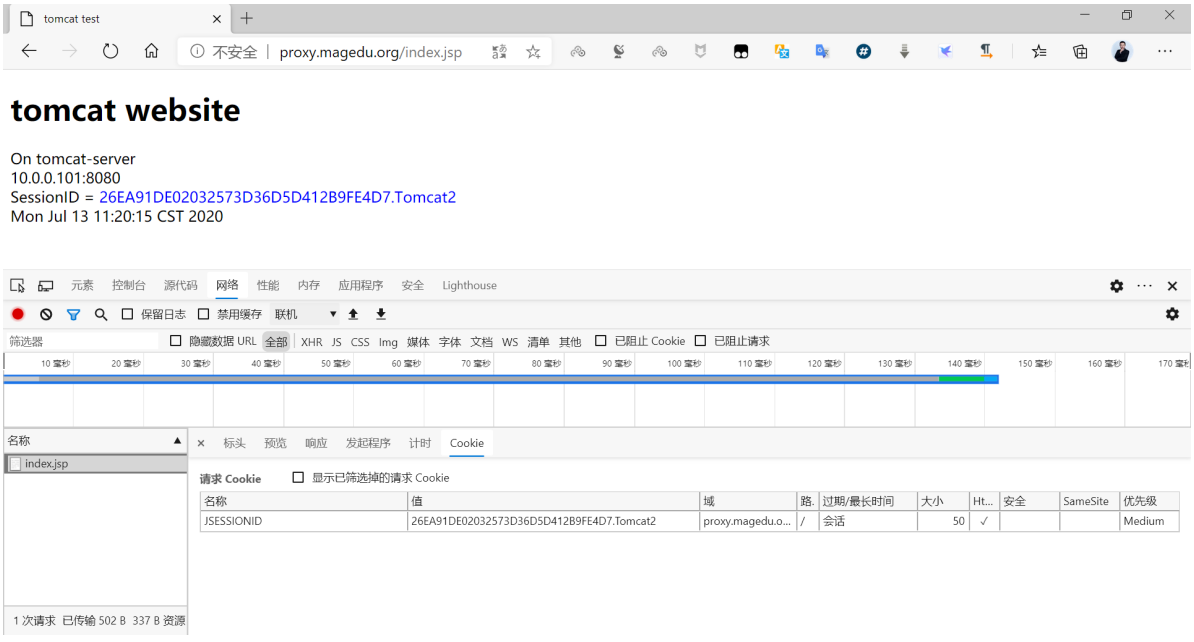


7.4.5.5 模拟故障

7.4.5.5.1 模拟tomcat故障,查看是否SessionID变化

```
[root@t2 ~]#systemctl stop tomcat
```

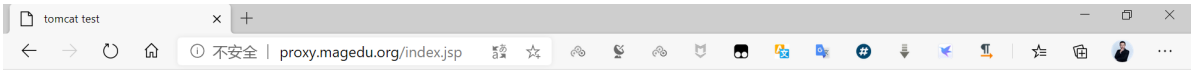
刷新页面,可以看到SessinID不变



7.4.5.5.2 模拟Redis 故障

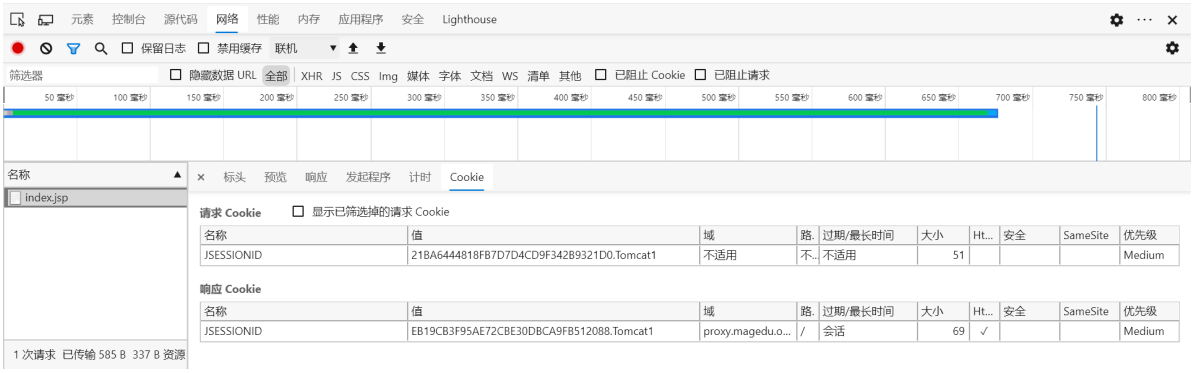
模拟Redis 故障,再刷新页面,可以看到SessionID不断变化,说明redis存在单点故障

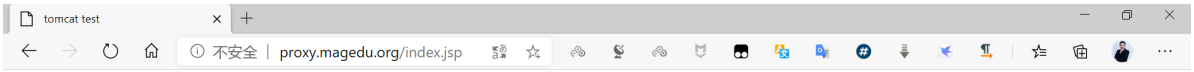
```
[root@t1 ~]#systemctl start tomcat
[root@redis ~]#systemctl stop redis
```



tomcat website

On tomcat-server
10.0.0.101:8080
SessionID = EB19CB3F95AE72CBE30DBCA9FB512088.Tomcat1
Mon Jul 13 11:21:22 CST 2020





tomcat website

On tomcat-server
10.0.0.102:8080
SessionID = 081BAF8E8122B69887DB71FEBF1A4032.Tomcat2
Mon Jul 13 11:23:08 CST 2020

7.5 使用 redisson 利用 redis 实现session共享

Redisson是Redis官方推荐的Java版的Redis客户端。它提供的功能非常多，也非常强大

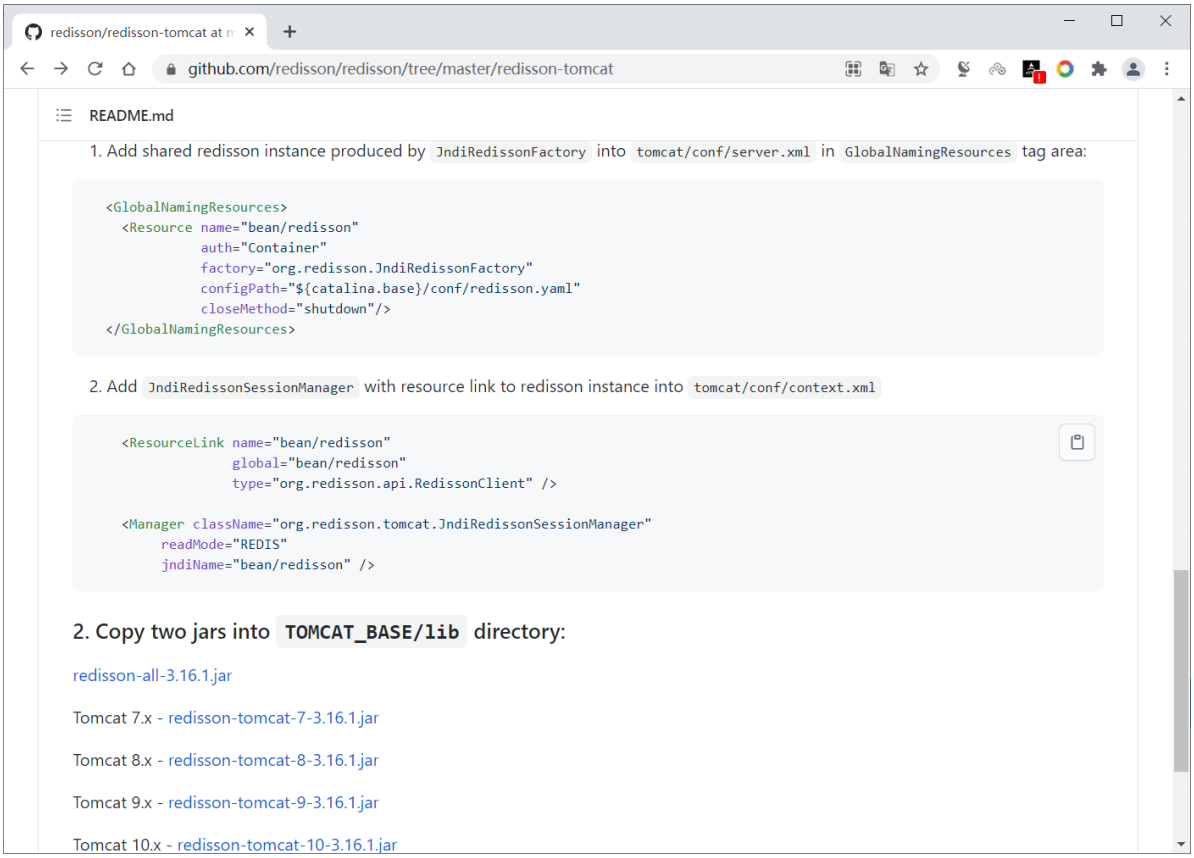
redisson 是基于redis的扩展库，使得redis除了应用于缓存以外，还能做队列等数据结构，直接使用的分布式锁，以及事务调度器等。

官网:

```
https://github.com/redisson/redisson
```

tomcat 配置Redisson的文档链接

```
https://github.com/redisson/redisson/tree/master/redisson-tomcat
```



相关jar包下载链接

```
https://repository.sonatype.org/service/local/artifact/maven/redirect?r=central-proxy&g=org.redisson&a=redisson-all&v=3.16.1&e=jar
https://repository.sonatype.org/service/local/artifact/maven/redirect?r=central-proxy&g=org.redisson&a=redisson-tomcat-8&v=3.16.1&e=jar
https://repository.sonatype.org/service/local/artifact/maven/redirect?r=central-proxy&g=org.redisson&a=redisson-tomcat-9&v=3.16.1&e=jar
```

范例: 使用redisson实现session共享

```
[root@centos8 ~]#vim /usr/local/tomcat/conf/context.xml
#倒数第一行前加下面内容
<Manager className="org.redisson.tomcat.RedissonSessionManager"
configPath="${catalina.base}/conf/redisson.conf" readMode="MEMORY"
updateMode="DEFAULT"/>
</Context>

#创建下面文件内容
[root@centos8 ~]#vim /usr/local/tomcat/conf/redisson.conf
{
   "singleServerConfig":{
      "idleConnectionTimeout":10000,
      "connectTimeout":10000,
      "timeout":3000,
      "retryAttempts":3,
      "retryInterval":1500,
      "password":null,
      "subscriptionsPerConnection":5,
      "clientName":null,
      "address": "redis://10.0.0.100:6379",  #指向redis服务器地址
      "subscriptionConnectionMinimumIdleSize":1,
```

```
      "subscriptionConnectionPoolSize":50,
      "connectionMinimumIdleSize":32,
      "connectionPoolSize":64,
      "database":0,
      "dnsMonitoringInterval":5000
   },
   "threads":0,
   "nettyThreads":0,
   "codec":{
      "class":"org.redisson.codec.JsonJacksonCodec"
   },
   "transportMode":"NIO"
}

#准备两个jar包到lib目录
#下载链接https://github.com/redisson/redisson/tree/master/redisson-tomcat
[root@centos8 ~]#ls /usr/local/tomcat/lib/redisson-*
/usr/local/tomcat/lib/redisson-all-3.16.1.jar
/usr/local/tomcat/lib/redisson-tomcat-9-3.16.1.jar

#重启tomcat生效
systemctl restart tomcat

#准备测试页面
[root@centos8 ~]#cat /usr/local/tomcat/webapps/ROOT/test.jsp
<%@ page import="java.util.*" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<h1> Tomcat Website </h1>
<div>On  <%=request.getServerName() %></div>
<div><%=request.getLocalAddr() + ":" + request.getLocalPort() %></div>
<div>SessionID = <span style="color:blue"><%=session.getId() %></span></div>
<%=new Date()%>
</body>
</html>

#访问此页面
[root@centos8 ~]#curl 127.0.0.1:8080/test.jsp

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<h1> Tomcat Website </h1>
<div>On  127.0.0.1</div>
<div>127.0.0.1:8080</div>
<div>SessionID = <span
style="color:blue">F14D14DF86AD76D55789269190950393</span></div>
Fri Aug 06 14:44:18 CST 2021
</body>
```

```
</html>


#访问redis,查看生成数据
[root@ubuntu1804 ~]#redis-cli
127.0.0.1:6379> keys *
1) "redisson:tomcat_session:4B5FC68977FC980E3C32564C357A2785"
s127.0.0.1:6379> type "redisson:tomcat_session:4B5FC68977FC980E3C32564C357A2785"
hash
127.0.0.1:6379> hgetall redisson:tomcat_session:4B5FC68977FC980E3C32564C357A2785
 1) "session:thisAccessedTime"
 2) "[\"java.lang.Long\",1628231049422]"
 3) "session:isNew"
 4) "false"
 5) "session:lastAccessedTime"
 6) "[\"java.lang.Long\",1628231049422]"
 7) "session:maxInactiveInterval"
 8) "1800"
 9) "session:isValid"
10) "true"
11) "session:creationTime"
12) "[\"java.lang.Long\",1628231049358]"
```

7.6 利用 Tomcat Clustering Redis Session Manager 利用 redis 实现session共享

Redis session manager 是一个插件。 它将会话存储到 Redis 中，以便在 Tomcat 服务器集群中轻松分发 HTTP 请求。

在这里，会话被实现为非粘性（意味着，每个请求都可以转到集群中的任何服务器，这与 Apache 提供的 Tomcat 集群设置不同。）

请求Sessions会立即存入Redis（Session属性必须是Serializable），供其他服务器使用。 当 tomcat 收到客户端的请求时，Sessions 直接从 Redis 加载。从而无需在负载均衡器中启用粘性会话 (JSESSIONID)。

支持Redis默认、哨兵和集群模式，基于配置。

参考文档

```
https://github.com/ran-jit/tomcat-cluster-redis-session-manager
```

范例：

```
#下载相关文件并解压缩
wget https://github.com/ran-jit/tomcat-cluster-redis-session-
manager/releases/download/3.0.1.1/tomcat-cluster-redis-session-manager.zip
unzip /opt/tomcat-cluster-redis-session-manager.zip -d /opt

#复制jar包到tomcat/lib目录中
cp /opt/tomcat-cluster-redis-session-manager/lib/* /usr/local/tomcat/lib/
chown -R tomcat.tomcat /usr/local/tomcat/lib

#复制redis配置文件到tomcat/conf目录中
cp /opt/tomcat-cluster-redis-session-manager/conf/redis-data-cache.properties
/usr/local/tomcat/conf/
```

```
#修改redis配置信息
vim  /usr/local/tomcat/conf/redis-data-cache.properties
#修改下面两行
redis.hosts=10.0.0.100:6379 #指向redis服务器地址
redis.password=123456

#添加两行配置文件在 tomcat/conf/context.xml
vim /usr/local/tomcat/conf/context.xml
#在最后一行前加下两行
<Valve className="tomcat.request.session.redis.SessionHandlerValve" />
<Manager className="tomcat.request.session.redis.SessionManager" />
</Context> #此是最后一行

#修改session过期时间为60m，默认30m,此步可选
vim /usr/local/tomcat/conf/web.xml
<session-config>
<session-timeout>60</session-timeout>
</session-config>

#重启服务
systemctl restart  tomcat

#nginx配置反向代理和均衡负载
vim /etc/nginx/conf.d/session.conf
upstream tomcat-server {
        server t1.magedu.org:8080;
        server t2.magedu.org:8080;
}
server {
        listen 80;
        server_name www.magedu.org;
        location /  {
            proxy_pass http://tomcat-server;
            proxy_set_header Host  $http_host;
        }
}

#tomcat测试页面
cat /data/webapps/ROOT/test.jsp
<%@ page import="java.util.*" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tomcat test</title>
</head>
<body>
<h1> Tomcat1 Website </h1>
<div>On  <%=request.getServerName() %></div>
<div><%=request.getLocalAddr() + ":" + request.getLocalPort() %></div>
<div>SessionID = <span style="color:blue"><%=session.getId() %></span></div>
<%=new Date()%>
</body>
</html>
```

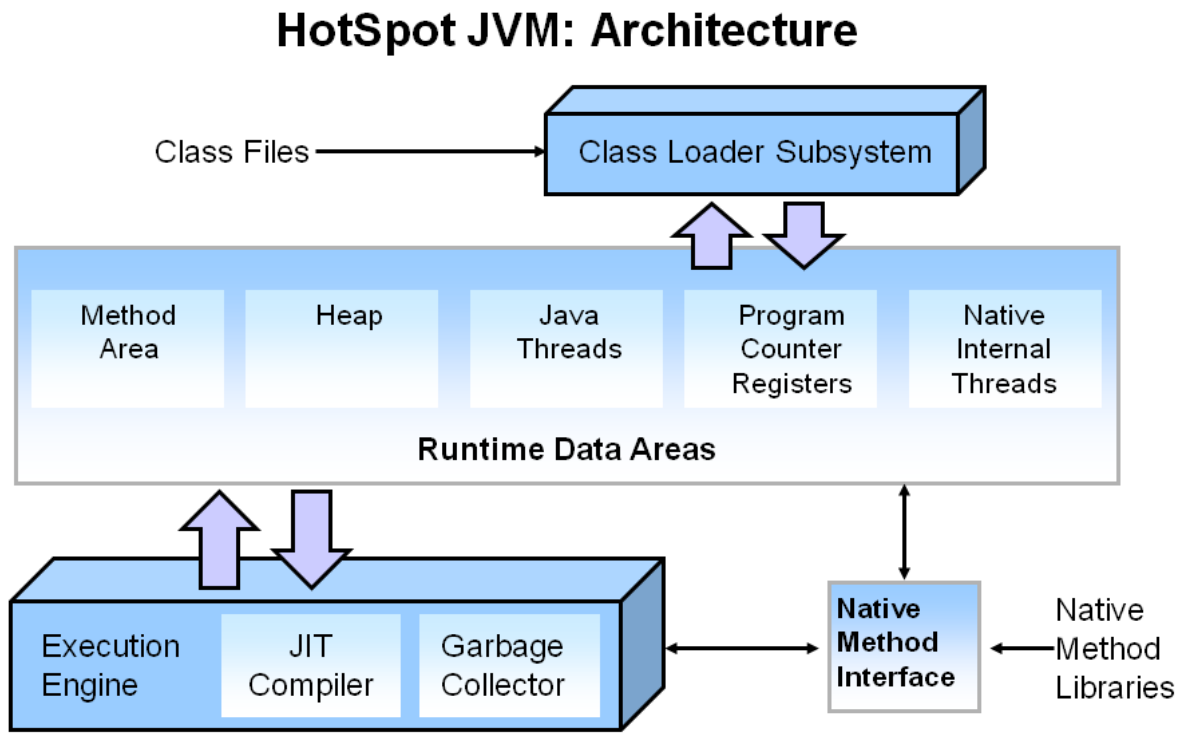
8 Tomcat 性能优化

在目前流行的互联网架构中，Tomcat在目前的网络编程中是举足轻重的，由于Tomcat的运行依赖于JVM，从虚拟机的角度把Tomcat的调整分为外部环境调优 JVM 和 Tomcat 自身调优两部分

8.1 JVM组成

```
[root@t1 ~]#java -version
java version "1.8.0_271"
Java(TM) SE Runtime Environment (build 1.8.0_271-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.271-b09, mixed mode)
```

8.1.1 JVM组成



JVM 组成部分

- 类加载子系统: 使用Java语言编写.java Source Code文件，通过javac编译成.class Byte Code文件。class loader类加载器将所需所有类加载到内存，必要时将类实例化成实例
- 运行时数据区: 最消耗内存的空间,需要优化
- 执行引擎: 包括JIT (JustInTimeCompiler)即时编译器, GC垃圾回收器
- 本地方法接口: 将本地方法栈通过JNI(Java Native Interface)调用Native Method Libraries, 比如:C,C++库等,扩展Java功能,融合不同的编程语言为Java所用

JVM运行时数据区域由下面部分构成:

- **Method Area (线程共享):** 方法区是所有线程共享的内存空间，存放已加载的类信息(构造方法,接口定义),常量(final),静态变量(static), 运行时常量池等。但实例变量存放在堆内存中. 从JDK8开始此空间由永久代改名为元空间
- **heap (线程共享):** 堆在虚拟机启动时创建,存放创建的所有对象信息。如果对象无法申请到可用内存将抛出OOM异常.堆是靠**GC**垃圾回收器管理的,通过-Xmx -Xms 指定最大堆和最小堆空间大小
- **Java stack (线程私有):** Java栈是每个线程会分配一个栈，存放java中8大基本数据类型,对象引用,实例的本地变量,方法参数和返回值等,基于FILO()（First In Last Out）,每个方法为一个栈帧

- **Program Counter Register (线程私有)**：PC寄存器就是一个指针,指向方法区中的方法字节码,每一个线程用于记录当前线程正在执行的字节码指令地址。由执行引擎读取下一条指令.因为线程需要切换，当一个线程被切换回来需要执行的时候，知道执行到哪里了
- **Native Method stack (线程私有)**：本地方法栈为本地方法执行构建的内存空间，存放本地方法执行时的局部变量、操作数等。

  所谓本地方法，使用native 关健字修饰的方法,比如:Thread.sleep方法. 简单的说是非Java实现的方法，例如操作系统的C编写的库提供的本地方法，Java调用这些本地方法接口执行。但是要注意，本地方法应该避免直接编程使用，因为Java可能跨平台使用，如果用了Windows API，换到了Linux平台部署就有了问题

8.1.2 虚拟机

目前Oracle官方使用的是HotSpot， 它最早由一家名为"Longview Technologies"公司设计，使用了很多优秀的设计理念和出色的性能，1997年该公司被SUN公司收购。后来随着JDK一起发布了HotSpot VM。目前HotSpot是最主要的 JVM。

安卓程序需要运行在JVM上，而安卓平台使用了Google自研的Java虚拟机——Dalvid，适合于内存、处理器能力有限系统。

8.2 GC (Garbage Collection) 垃圾收集器

在堆内存中如果创建的对象不再使用,仍占用着内存,此时即为垃圾.需要即使进行垃圾回收,从而释放内存空间给其它对象使用

其实不同的开发语言都有垃圾回收问题,C,C++需要程序员人为回收,造成开发难度大,容易出错等问题,但执行效率高,而JAVA和Python中不需要程序员进行人为的回收垃圾,而由JVM或相关程序自动回收垃圾,减轻程序员的开发难度,但可能会造成执行效率低下

堆内存里面经常创建、销毁对象，内存也是被使用、被释放。如果不妥善处理，一个使用频繁的进程，可能会出现虽然有足够的内存容量，但是无法分配出可用内存空间，因为没有连续成片的内存了，内存全是碎片化的空间。

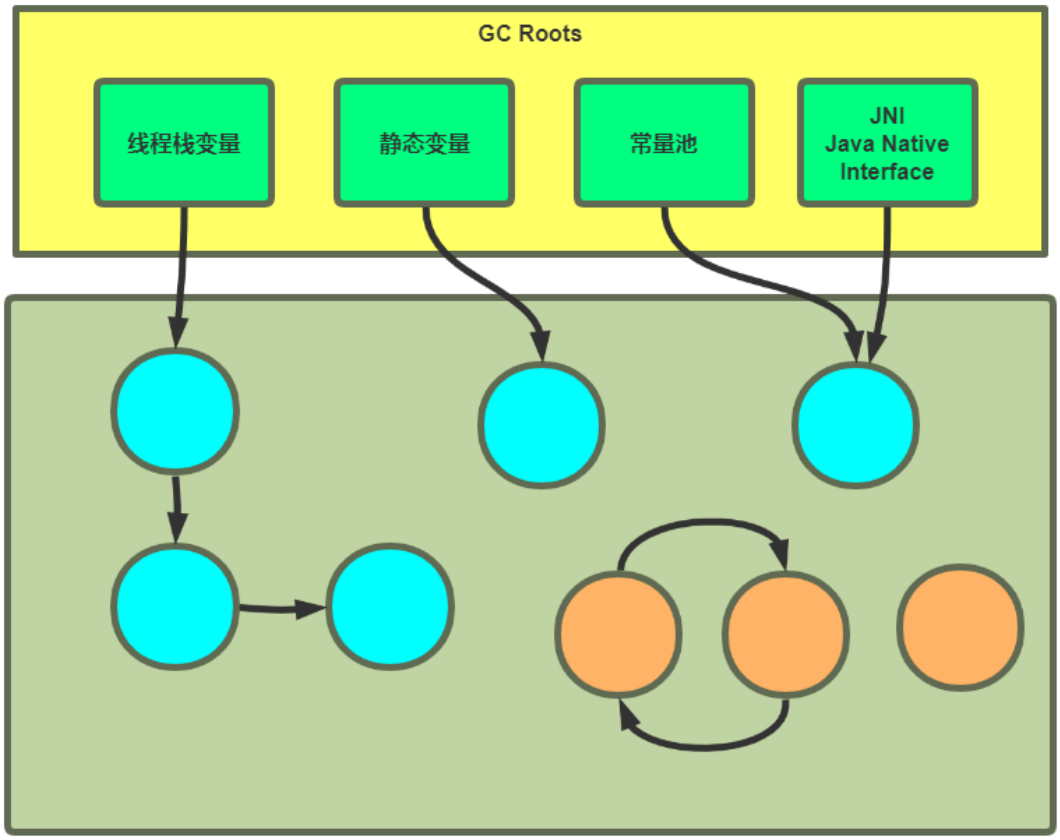
所以需要有合适的垃圾回收机制,确保正常释放不再使用的内存空间,还需要保证内存空间尽可能的保持一定的连续

对于垃圾回收,需要解决三个问题

- 哪些是垃圾要回收
- 怎么回收垃圾
- 什么时候回收垃圾

8.2.1 Garbage 垃圾确定方法

- 引用计数: 每一个堆内对象上都与一个私有引用计数器，记录着被引用的次数，引用计数清零，该对象所占用堆内存就可以被回收。循环引用的对象都无法将引用计数归零，就无法清除。Python中即使用此种方式
- 根搜索(可达)算法 Root Searching

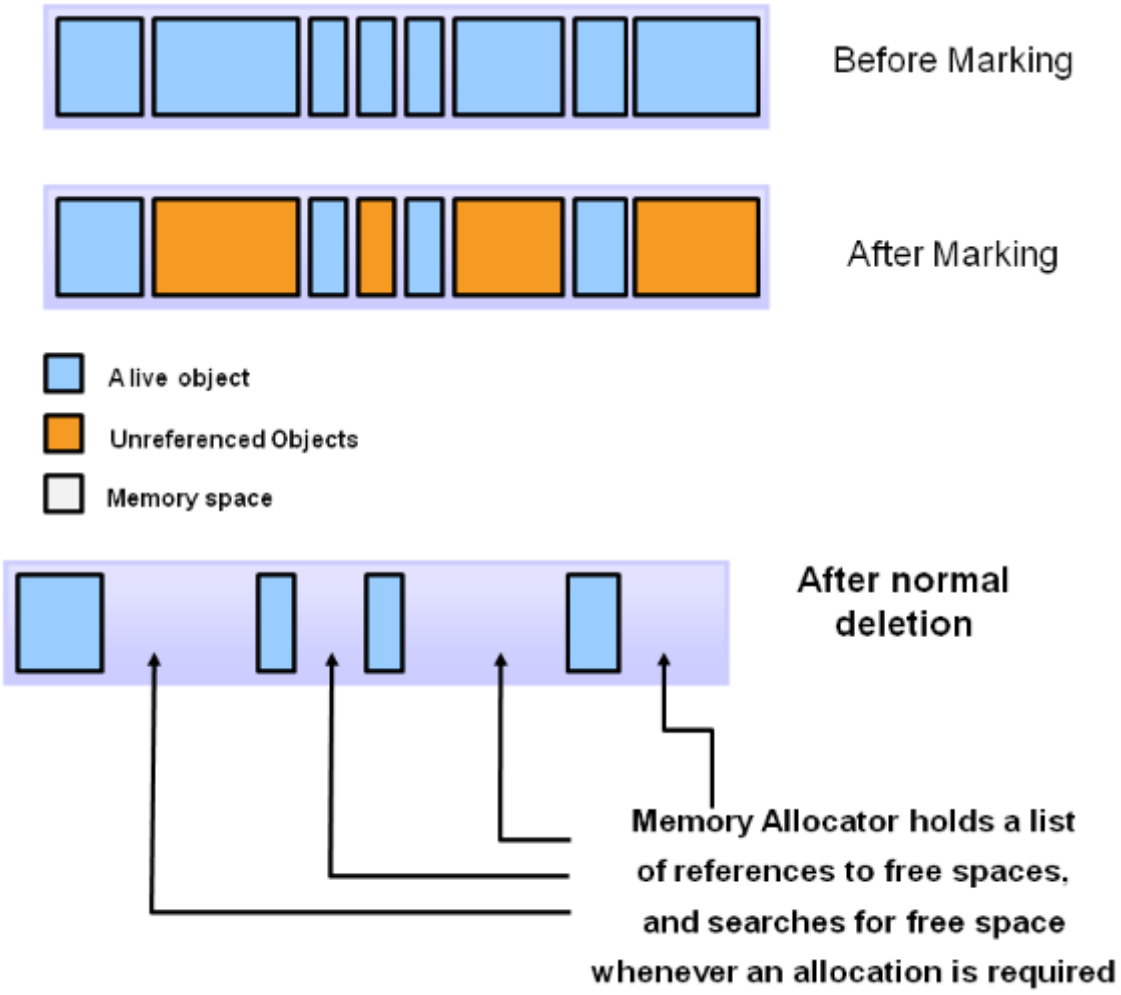


8.2.2 垃圾回收基本算法

8.2.2.1 标记-清除 Mark-Sweep

分垃圾标记阶段和内存释放两个阶段。

- 标记阶段，找到所有可访问对象打个标记。清理阶段，遍历整个堆
- 对未标记对象(即不再使用的对象)逐一进行清理。



特点:

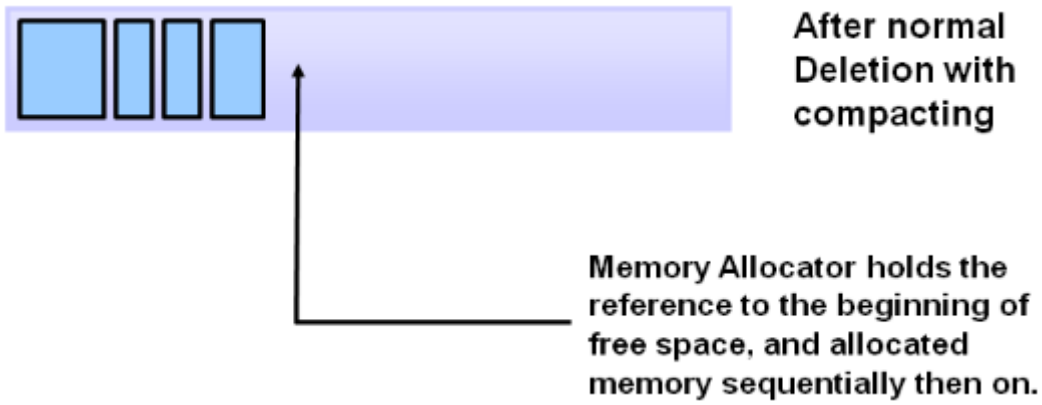
优点：算法简单

缺点：标记-清除最大的问题会造成**内存碎片**,但是不浪费空间,效率较高(如果对象较多时,逐一删除效率也会受到影响)

8.2.2.2 标记-压缩 (压实)Mark-Compact

分垃圾标记阶段和内存整理两个阶段。

- 标记阶段，找到所有可访问对象打个标记。
- 内存清理阶段时，整理时将对象向内存一端移动，整理后存活对象连续的集中在内存一端。

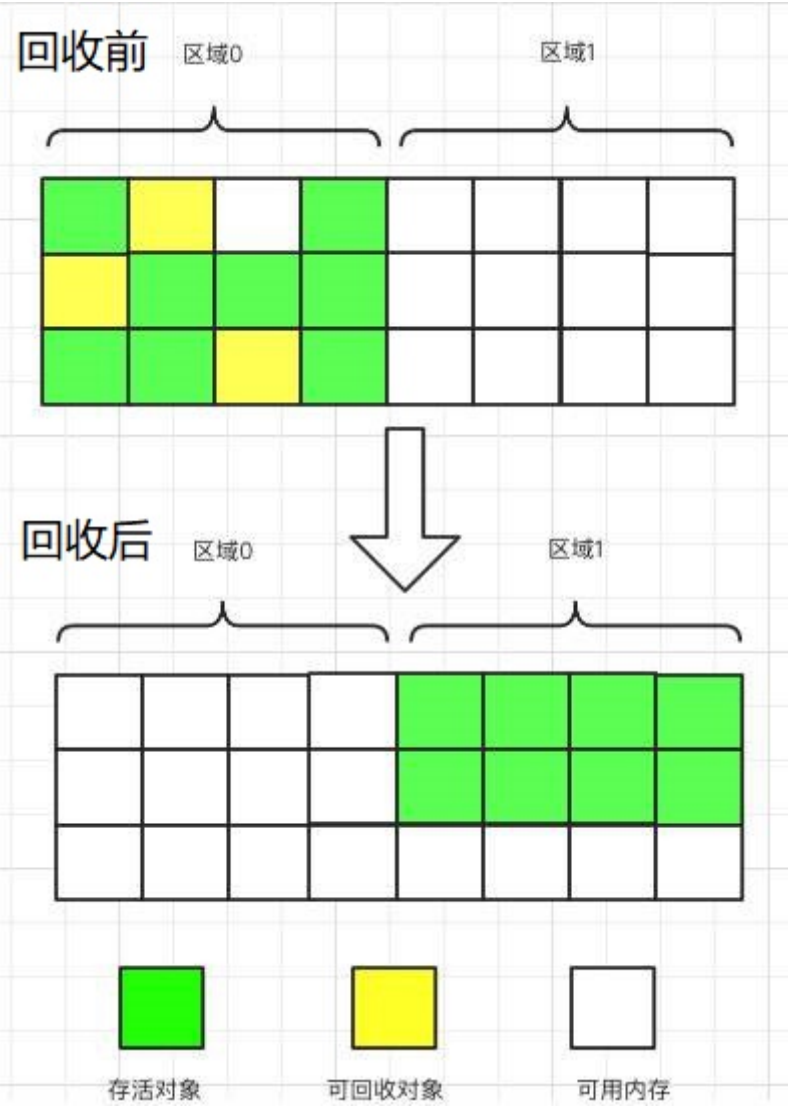


特点:

标记-压缩算法好处是整理后内存空间连续分配，有大段的连续内存可分配，没有内存碎片。

缺点是内存整理过程有消耗,**效率相对低下**

8.2.2.3 复制 Copying



先将可用内存分为大小相同两块区域A和B，每次只用其中一块，比如A。当A用完后，则将A中存活的对象复制到B。复制到B的时候连续的使用内存，最后将A一次性清除干净。

特点

好处是没有碎片，复制过程中保证对象使用连续空间,且一次性清除所有垃圾,所以即使对象很多，收回效率也很高

缺点是比较**浪费内存**，只能使用原来一半内存，因为内存对半划分了，复制过程毕竟也是有代价。

8.2.2.4 多种算法总结

没有最好的算法,在不同场景选择最合适的算法

- 效率: 复制算法>标记清除算法> 标记压缩算法
- 内存整齐度: 复制算法=标记压缩算法> 标记清除算法
- 内存利用率: 标记压缩算法=标记清除算法>复制算法

8.2.2.5 STW

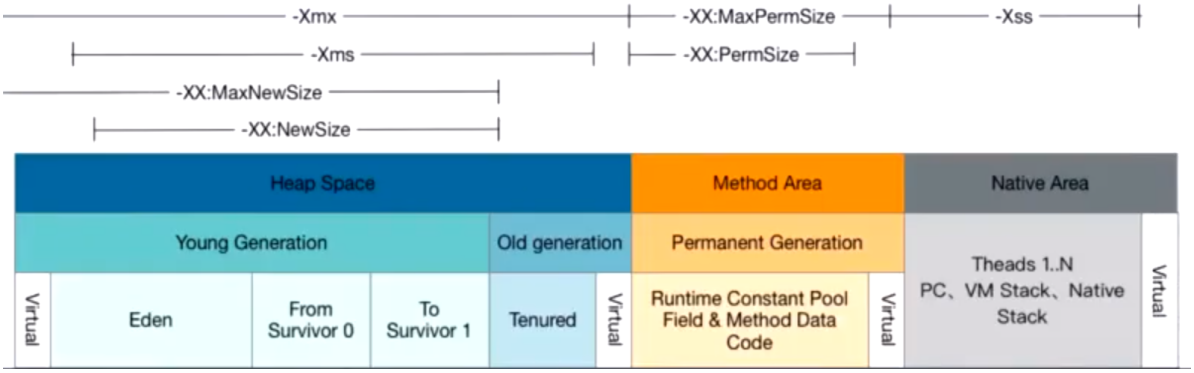
对于大多数垃圾回收算法而言，GC线程工作时，停止所有工作的线程，称为**Stop The World**。GC 完成时，恢复其他工作线程运行。这也是JVM运行中最头疼的问题。

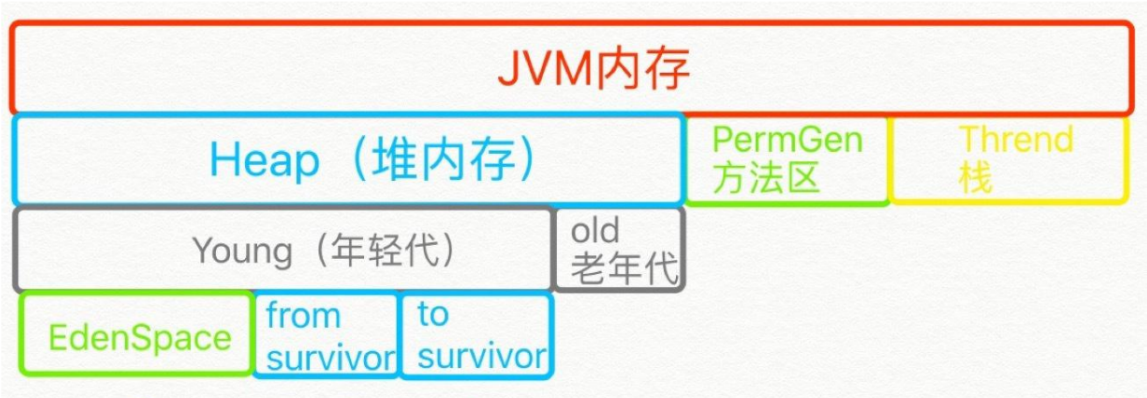
8.2.3 分代堆内存GC策略

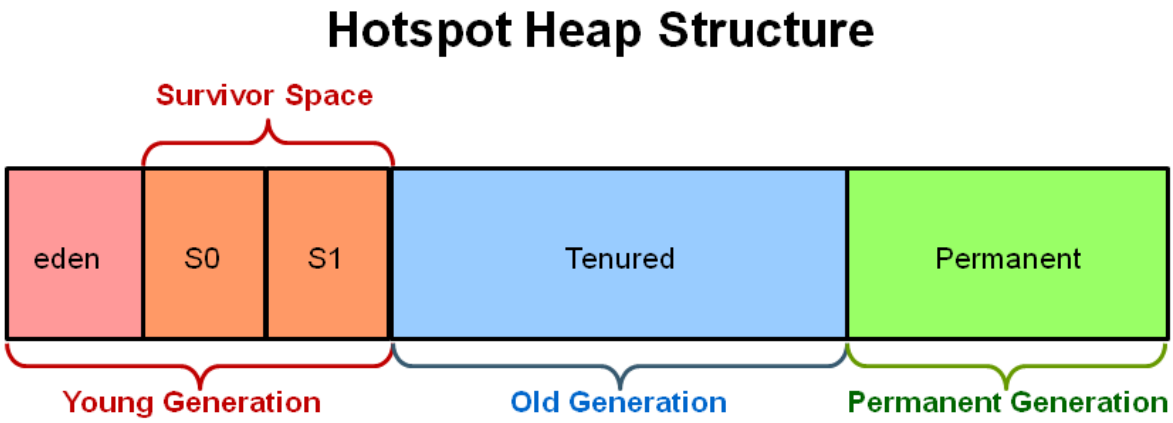
上述垃圾回收算法都有优缺点，能不能对不同数据进行区分管理，不同分区对数据实施不同回收策略，分而治之。

8.2.3.1 堆内存分代

将heap内存空间分为三个不同类别: 年轻代、老年代、持久代





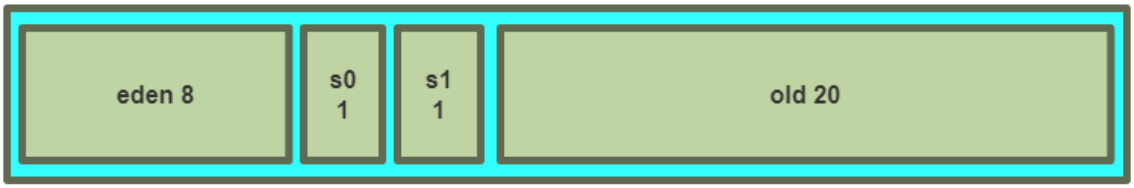


Heap堆内存分为

- **年轻代Young：** Young Generation
  - **伊甸园区eden**: 只有一个,刚刚创建的对象
  - **幸存(存活)区Servivor Space**：有2个幸存区，一个是from区，一个是to区。大小相等、地位相同、可互换。
    - from 指的是本次复制数据的源区
    - to 指的是本次复制数据的目标区
- **老年代Tenured：** Old Generation, 长时间存活的对象

默认空间大小比例:

默认JVM试图分配最大内存的总内存的1/4,初始化默认总内存为总内存的1/64,年青代中heap的1/3，老年代占2/3



永久代： JDK1.7之前使用, 即Method Area方法区,保存JVM自身的类和方法,存储JAVA运行时的环境信息, JDK1.8后 改名为 MetaSpace,此空间不存在垃圾回收,关闭JVM会释放此区域内存,此空间物理上不属于heap内存,但逻辑上存在于heap内存

- 永久代必须指定大小限制,字符串常量JDK1.7存放在永久代,1.8后存放在heap中
- MetaSpace 可以设置,也可不设置,无上限

规律: 一般情况99%的对象都是临时对象

范例: 在tomcat 状态页可以看到以下的内存分代

JVM

剩余内存： 7.35 MB 总内存： 25.30 MB 最大内存 232.00 MB

内存池	类型	初始化	总共	最大值	已用
Eden Space	Heap memory	4.31 MB	7.00 MB	64.00 MB	6.16 MB (9%)
Survivor Space	Heap memory	0.50 MB	0.87 MB	8.00 MB	0.87 MB (10%)
Tenured Gen	Heap memory	10.68 MB	17.42 MB	160.00 MB	10.90 MB (6%)
Code Cache	Non-heap memory	2.43 MB	7.62 MB	240.00 MB	7.57 MB (3%)
Compressed Class Space	Non-heap memory	0.00 MB	2.62 MB	1024.00 MB	2.49 MB (0%)
Metaspace	Non-heap memory	0.00 MB	24.37 MB	-0.00 MB	23.88 MB

范例: 查看JVM内存分配情况

```
[root@centos8 ~]#cat Heap.java
public class Heap {
    public static void main(String[] args){
        //返回JVM试图使用的最大内存,字节单位
        long max = Runtime.getRuntime().maxMemory();
        //返回JVM初始化总内存
        long total = Runtime.getRuntime().totalMemory();

        System.out.println("max="+max+"字节\t"+(max/(double)1024/1024)+"MB");
        System.out.println("total="+total+"字节\t"+
(total/(double)1024/1024)+"MB");
    }
}
[root@centos8 ~]#javac Heap.java
[root@centos8 ~]#java -classpath . Heap
max=243269632字节 232.0MB
total=16252928字节    15.5MB

[root@centos8 ~]#java  -XX:+PrintGCDetails Heap
max=243269632字节 232.0MB
total=16252928字节 15.5MB
Heap
 def new generation   total 4928K, used 530K [0x00000000f1000000,
0x00000000f1550000, 0x00000000f6000000)
  eden space 4416K,  12% used [0x00000000f1000000, 0x00000000f1084a60,
0x00000000f1450000)
  from space 512K,   0% used [0x00000000f1450000, 0x00000000f1450000,
0x00000000f14d0000)
```

```
  to   space 512K,   0% used [0x00000000f14d0000, 0x00000000f14d0000,
0x00000000f1550000)
 tenured generation   total 10944K, used 0K [0x00000000f6000000,
0x00000000f6ab0000, 0x0000000100000000)
   the space 10944K,   0% used [0x00000000f6000000, 0x00000000f6000000,
0x00000000f6000200, 0x00000000f6ab0000)
 Metaspace       used 2525K, capacity 4486K, committed 4864K, reserved 1056768K
  class space    used 269K, capacity 386K, committed 512K, reserved 1048576K

[root@centos8 ~]#echo  "scale=2;(4928+10944)/1024" |bc
15.50
#说明年轻代+老年代占用了所有heap空间, Metaspace实际不占heap空间,逻辑上存在于Heap
```

8.2.3.2 年轻代回收 Minor GC

1. 起始时，所有新建对象(特大对象直接进入老年代)都出生在eden，当eden满了，**启动GC。**这个称为**Young GC** 或者 **Minor GC。**
2. 先**标记**eden存活对象，然后将存活对象**复制**到s0（假设本次是s0，也可以是s1，它们可以调换），eden剩余所有空间都清空。**GC完成。**
3. 继续新建对象，当eden再次满了，**启动GC。**
4. 先同时**标记**eden和s0中存活对象，然后将存活对象**复制**到s1。将eden和s0清空,此次**GC完成**
5. 继续新建对象，当eden满了，**启动GC。**
6. 先**标记**eden和s1中存活对象，然后将存活对象**复制**到s0。将eden和s1清空,此次**GC完成**

以后就重复上面的步骤。

通常场景下,大多数对象都不会存活很久，而且创建活动非常多，新生代就需要频繁垃圾回收。

但是，如果一个对象一直存活，它最后就在from、to来回复制，如果from区中对象复制次数达到阈值(默认15次,CMS为6次,可通过java的选项 **-XX:MaxTenuringThreshold=N** 指定)，就直接复制到老年代。

8.2.3.3 老年代回收 Major GC

进入老年代的数据较少，所以老年代区被占满的速度较慢，所以垃圾回收也不频繁。

如果老年代也满了,会触发老年代GC,称为**Old GC**或者 **Major GC。**

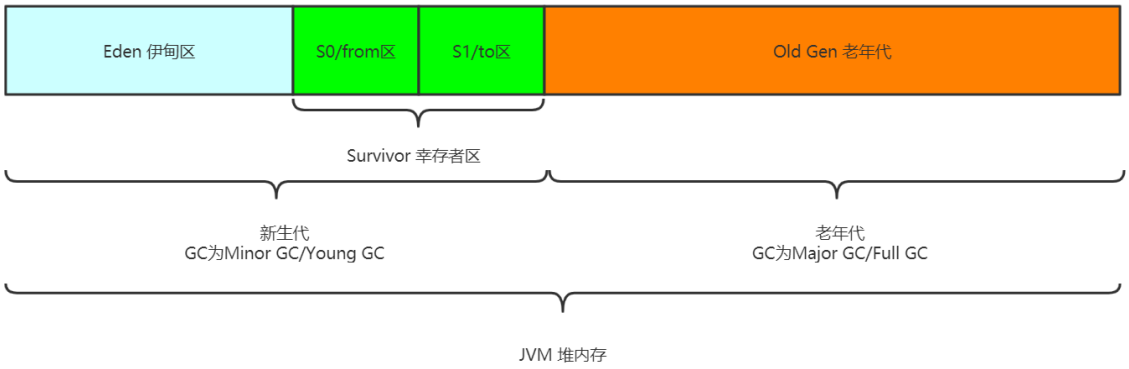
由于老年代对象一般来说存活次数较长，所以较常采用**标记-压缩**算法。

当老年代满时,会触发 **Full GC**,即对所有"代"的内存进行垃圾回收

Minor GC比较频繁，Major GC较少。但一般Major GC时，由于老年代对象也可以引用新生代对象，所以先进行一次Minor GC，然后在Major GC会提高效率。可以认为回收老年代的时候完成了一次Full GC。

所以可以认为 **MajorGC = FullGC**

8.2.3.4 GC 触发条件



Minor GC 触发条件：当eden区满了触发

Full GC 触发条件：

- 老年代满了
- System.gc()手动调用。不推荐

年轻代：

- 存活时长低
- 适合复制算法

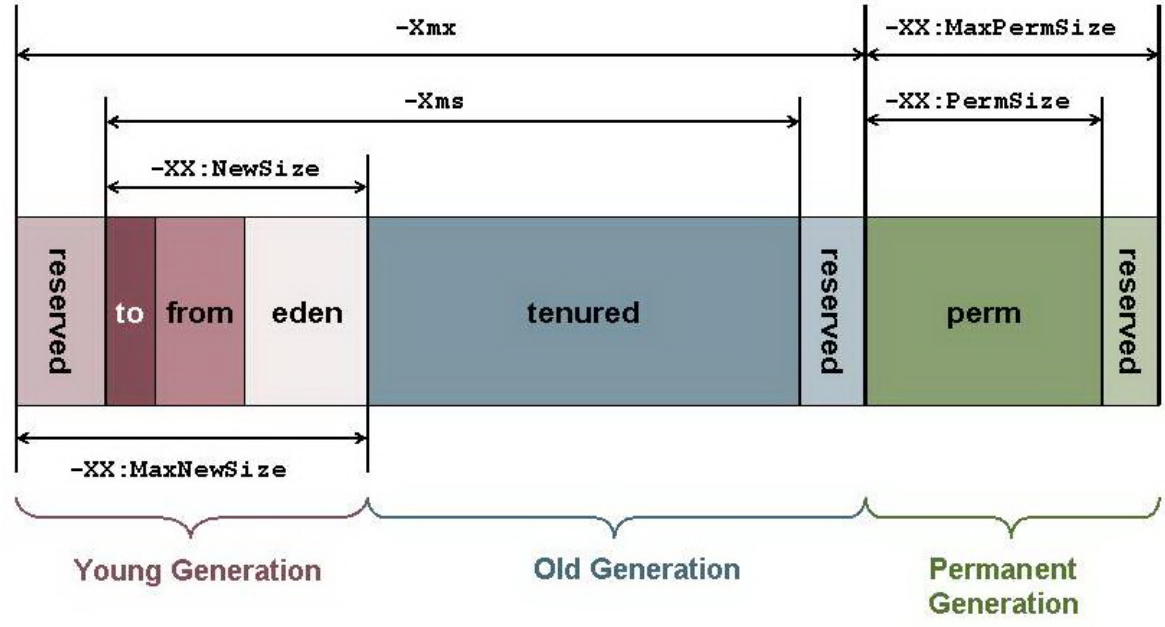
老年代：

- 区域大,存活时长高
- 适合标记压缩算法

8.2.4 java 内存调整相关参数

8.2.4.1 JVM 内存常用相关参数

Java 命令行参考文档: https://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html



帮助：man java

选项分类

- **-选项名称** 此为标准选项,所有HotSpot都支持
- **-X选项名称** 此为稳定的非标准选项
- **-XX:选项名称** 非标准的不稳定选项，下一个版本可能会取消

参数	说明	举例
-Xms	设置应用程序**初始**使用的堆内存大小（年轻代+老年代）	-Xms2g
-Xmx	设置应用程序能获得的**最大**堆内存 早期JVM不建议超过32G，内存管理效率下降	-Xmx4g
-XX:NewSize	设置初始新生代大小	-XX:NewSize=128m
-XX:MaxNewSize	设置最大新生代内存空间	-XX:MaxNewSize=256m
-Xmnsize	同时设置-XX:NewSize 和 -XX:MaxNewSize，代替两者	-Xmn1g
-XX:NewRatio	以比例方式设置新生代和老年代	-XX:NewRatio=2 即：new:old=1:2
-XX:SurvivorRatio	以比例方式设置eden和survivor(S0或S1)	-XX:SurvivorRatio=6 即：Eden:S0:S1=6:1:1
-Xss	设置每个线程私有的栈空间大小,依据具体线程大小和数量	-Xss256k

范例: 查看java的选项帮助

```
#查看java命令标准选项
[root@centos ~]#java
Usage: java [-options] class [args...]
           (to execute a class)
   or  java [-options] -jar jarfile [args...]
           (to execute a jar file)
where options include:
    -d32      use a 32-bit data model if available
    -d64      use a 64-bit data model if available
    -server   to select the "server" VM
                  The default VM is server.

    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                  A : separated list of directories, JAR archives,
                  and ZIP archives to search for class files.
    -D<name>=<value>
                  set a system property
    -verbose:[class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -version:<value>
                  Warning: this feature is deprecated and will be removed
                  in a future release.
                  require the specified version to run
    -showversion  print product version and continue
    -jre-restrict-search | -no-jre-restrict-search
                  Warning: this feature is deprecated and will be removed
                  in a future release.
                  include/exclude user private JREs in the version search
```

```
    -? -help      print this help message
    -X            print help on non-standard options  #非标准选项
    -ea[:<packagename>...|:<classname>]
    -enableassertions[:<packagename>...|:<classname>]
                  enable assertions with specified granularity
    -da[:<packagename>...|:<classname>]
    -disableassertions[:<packagename>...|:<classname>]
                  disable assertions with specified granularity
    -esa | -enablesystemassertions
                  enable system assertions
    -dsa | -disablesystemassertions
                  disable system assertions
    -agentlib:<libname>[=<options>]
                  load native agent library <libname>, e.g. -agentlib:hprof
                  see also, -agentlib:jdwp=help and -agentlib:hprof=help
    -agentpath:<pathname>[=<options>]
                  load native agent library by full pathname
    -javaagent:<jarpath>[=<options>]
                  load Java programming language agent, see java.lang.instrument
    -splash:<imagepath>
                  show splash screen with specified image
See http://www.oracle.com/technetwork/java/javase/documentation/index.html for
more details.
[root@t1 ~]#


#查看java的非标准选项
[root@centos8 ~]#java -X
    -Xmixed           mixed mode execution (default)
    -Xint             interpreted mode execution only
    -Xbootclasspath:<directories and zip/jar files separated by :>
                      set search path for bootstrap classes and resources
    -Xbootclasspath/a:<directories and zip/jar files separated by :>
                      append to end of bootstrap class path
    -Xbootclasspath/p:<directories and zip/jar files separated by :>
                      prepend in front of bootstrap class path
    -Xdiag            show additional diagnostic messages
    -Xnoclassgc       disable class garbage collection
    -Xincgc           enable incremental garbage collection
    -Xloggc:<file>    log GC status to a file with time stamps
    -Xbatch           disable background compilation
    -Xms<size>        set initial Java heap size
    -Xmx<size>        set maximum Java heap size
    -Xss<size>        set java thread stack size
    -Xprof            output cpu profiling data
    -Xfuture          enable strictest checks, anticipating future default
    -Xrs              reduce use of OS signals by Java/VM (see documentation)
    -Xcheck:jni       perform additional checks for JNI functions
    -Xshare:off       do not attempt to use shared class data
    -Xshare:auto      use shared class data if possible (default)
    -Xshare:on        require using shared class data, otherwise fail.
    -XshowSettings    show all settings and continue
    -XshowSettings:all
                      show all settings and continue
    -XshowSettings:vm show all vm related settings and continue
    -XshowSettings:properties
                      show all property settings and continue
    -XshowSettings:locale
```

```
                    show all locale related settings and continue

The -X options are non-standard and subject to change without notice.


#查看所有不稳定选项的当前生效值
[root@centos8 ~]#java -XX:+PrintFlagsFinal
[Global flags]
     intx ActiveProcessorCount                      = -1
         {product}
    uintx AdaptiveSizeDecrementScaleFactor          = 4
          {product}
    uintx AdaptiveSizeMajorGCDecayTimeScale         = 10
         {product}
    uintx AdaptiveSizePausePolicy                   = 0
          {product}
    uintx AdaptiveSizePolicyCollectionCostMargin    = 50
         {product}
    uintx AdaptiveSizePolicyInitializingSteps       = 20
         {product}
    uintx AdaptiveSizePolicyOutputInterval          = 0
          {product}
    uintx AdaptiveSizePolicyWeight                  = 10
         {product}
    uintx AdaptiveSizeThroughPutPolicy              = 0
          {product}
 .......

#查看所有不稳定选项的默认值
[root@centos8 ~]#java -XX:+PrintFlagsInitial
[Global flags]
     intx ActiveProcessorCount                      = -1
         {product}
    uintx AdaptiveSizeDecrementScaleFactor          = 4
          {product}
    uintx AdaptiveSizeMajorGCDecayTimeScale         = 10
         {product}
    uintx AdaptiveSizePausePolicy                   = 0
          {product}
    uintx AdaptiveSizePolicyCollectionCostMargin    = 50
         {product}
    uintx AdaptiveSizePolicyInitializingSteps       = 20
         {product}
.......

#查看当前命令行的使用的选项设置
[root@centos8 ~]#java -XX:+PrintCommandLineFlags
-XX:InitialHeapSize=15598528 -XX:MaxHeapSize=249576448 -
XX:+PrintCommandLineFlags -XX:+UseCompressedClassPointers -XX:+UseCompressedOops
-XX:+UseParallelGC

#上面的-XX:+UseParallelGC 说明当前使用Parallel Scavenge + Parallel Old
```

范例: 查看和指定JVM内存分配

```
#默认JVM试图分配最大内存的总内存的1/4,初始化默认总内存为总内存的1/64
[root@centos8 ~]#cat Heap.java
```

```
public class Heap {
    public static void main(String[] args){
        //返回JVM试图使用的最大内存,字节单位
        long max = Runtime.getRuntime().maxMemory();
        //返回JVM初始化总内存
        long total = Runtime.getRuntime().totalMemory();

        System.out.println("max="+max+"字节\t"+(max/(double)1024/1024)+"MB");
        System.out.println("total="+total+"字节\t"+
(total/(double)1024/1024)+"MB");
    }
}

#编译生成class文件
[root@centos8 ~]#javac Heap.java

#通过$CLASSPATH指定类文件路径,否则无法找到类,也可以通过 java -cp /path指定类路径
[root@centos8 ~]#echo $CLASSPATH
/usr/local/jdk/lib/:/usr/local/jdk/jre/lib/

[root@centos8 ~]#cp Heap.class  /usr/local/jdk/lib

#查看当前内存默认值
[root@centos8 ~]#java  -XX:+PrintGCDetails Heap
max=243269632字节 232.0MB
total=16252928字节 15.5MB
Heap
 def new generation   total 4928K, used 530K [0x00000000f1000000,
0x00000000f1550000, 0x00000000f6000000)
  eden space 4416K,  12% used [0x00000000f1000000, 0x00000000f1084a60,
0x00000000f1450000)
  from space 512K,   0% used [0x00000000f1450000, 0x00000000f1450000,
0x00000000f14d0000)
  to   space 512K,   0% used [0x00000000f14d0000, 0x00000000f14d0000,
0x00000000f1550000)
 tenured generation   total 10944K, used 0K [0x00000000f6000000,
0x00000000f6ab0000, 0x0000000100000000)
   the space 10944K,   0% used [0x00000000f6000000, 0x00000000f6000000,
0x00000000f6000200, 0x00000000f6ab0000)
 Metaspace       used 2525K, capacity 4486K, committed 4864K, reserved 1056768K
  class space    used 269K, capacity 386K, committed 512K, reserved 1048576K

#指定内存空间
[root@centos8 ~]#java -Xms1024m -Xmx1024m -XX:+PrintGCDetails Heap
max=1037959168字节    989.875MB
total=1037959168字节  989.875MB
Heap
 def new generation   total 314560K, used 11185K [0x00000000c0000000,
0x00000000d5550000, 0x00000000d5550000)
  eden space 279616K,   4% used [0x00000000c0000000, 0x00000000c0aec408,
0x00000000d1110000)
  from space 34944K,   0% used [0x00000000d1110000, 0x00000000d1110000,
0x00000000d3330000)
  to   space 34944K,   0% used [0x00000000d3330000, 0x00000000d3330000,
0x00000000d5550000)
 tenured generation   total 699072K, used 0K [0x00000000d5550000,
0x0000000100000000, 0x0000000100000000)
```

```
   the space 699072K,   0% used [0x00000000d5550000, 0x00000000d5550000,
0x00000000d5550200, 0x0000000100000000)
 Metaspace       used 2525K, capacity 4486K, committed 4864K, reserved 1056768K
  class space    used 269K, capacity 386K, committed 512K, reserved 1048576K

#以下计算结果和max一样,说明Metaspace逻辑存在,但物理上并不属于heap空间
[root@centos8 ~]#echo '(314560+699072)*1024'|bc
1037959168
```

范例: 查看OOM

```
[root@centos8 ~]#cat  HeapOom2.java
import java. util. Random;
public class HeapOom2 {
    public static void main(String[] args) {
        String str = "I am lao wang";
        while (true){
            str += str + new Random().nextInt(88888888); //生成0到88888888之间的随
机数字
        }
    }
}

[root@centos8 ~]#javac -cp . HeapOom2.java

[root@centos8 ~]#java -Xms100m -Xmx100m -XX:+PrintGCDetails -cp . HeapOom2
[GC (Allocation Failure) [DefNew: 27213K->2940K(30720K), 0.0026366 secs] 27213K-
>5624K(99008K), 0.0027124 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [DefNew: 24949K->0K(30720K), 0.0044922 secs] 27633K-
>8307K(99008K), 0.0045959 secs] [Times: user=0.00 sys=0.00, real=0.01 secs]
[GC (Allocation Failure) [DefNew: 16550K->0K(30720K), 0.0037276 secs] 24857K-
>19041K(99008K), 0.0037898 secs] [Times: user=0.00 sys=0.01, real=0.00 secs]
[GC (Allocation Failure) [DefNew: 21468K->0K(30720K), 0.0111687 secs] 40509K-
>40509K(99008K), 0.0112437 secs] [Times: user=0.00 sys=0.01, real=0.01 secs]
[GC (Allocation Failure) [DefNew: 21951K->0K(30720K), 0.0084048 secs] 62460K-
>61977K(99008K), 0.0084641 secs] [Times: user=0.00 sys=0.01, real=0.00 secs]
[GC (Allocation Failure) [DefNew: 21468K->21468K(30720K), 0.0000177 secs]
[Tenured: 61977K->35141K(68288K), 0.0068683 secs] 83445K->35141K(99008K),
[Metaspace: 2479K->2479K(1056768K)], 0.0069358 secs] [Times: user=0.01 sys=0.00,
real=0.01 secs]
[Full GC (Allocation Failure) [Tenured: 35141K->32444K(68288K), 0.0091498 secs]
35141K->32444K(99008K), [Metaspace: 2479K->2479K(1056768K)], 0.0091975 secs]
[Times: user=0.01 sys=0.00, real=0.01 secs]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.util.Arrays.copyOf(Arrays.java:3332)
    at
java.lang.AbstractStringBuilder.ensureCapacityInternal(AbstractStringBuilder.jav
a:124)
    at java.lang.AbstractStringBuilder.append(AbstractStringBuilder.java:674)
    at java.lang.StringBuilder.append(StringBuilder.java:208)
    at HeapOom2.main(HeapOom2.java:6)
Heap
 def new generation   total 30720K, used 1048K [0x00000000f9c00000,
0x00000000fbd50000, 0x00000000fbd50000)
  eden space 27328K,   3% used [0x00000000f9c00000, 0x00000000f9d06158,
0x00000000fb6b0000)
```

```
  from space 3392K,   0% used [0x00000000fba00000, 0x00000000fba00000,
0x00000000fbd50000)
  to   space 3392K,   0% used [0x00000000fb6b0000, 0x00000000fb6b0000,
0x00000000fba00000)
 tenured generation   total 68288K, used 32444K [0x00000000fbd50000,
0x0000000100000000, 0x0000000100000000)
   the space 68288K,  47% used [0x00000000fbd50000, 0x00000000fdcff248,
0x00000000fdcff400, 0x0000000100000000)
 Metaspace       used 2510K, capacity 4486K, committed 4864K, reserved 1056768K
  class space    used 268K, capacity 386K, committed 512K, reserved 1048576K
```

8.2.4.2 JDK 工具监控使用情况

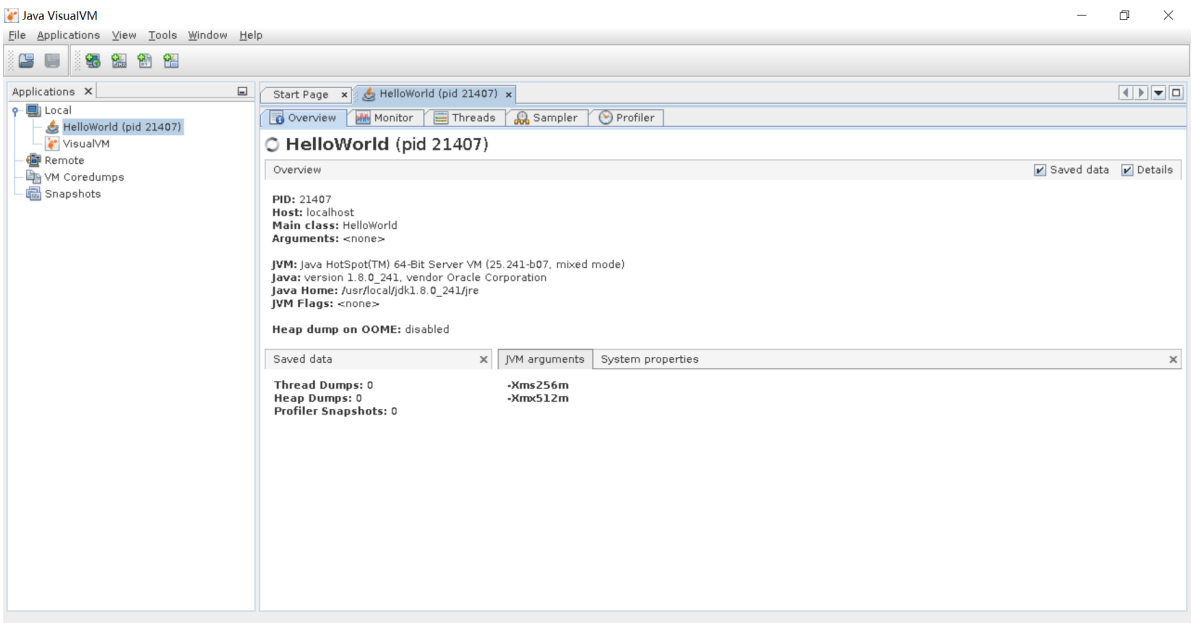
8.2.4.2.1 案例1: jvisualvm工具

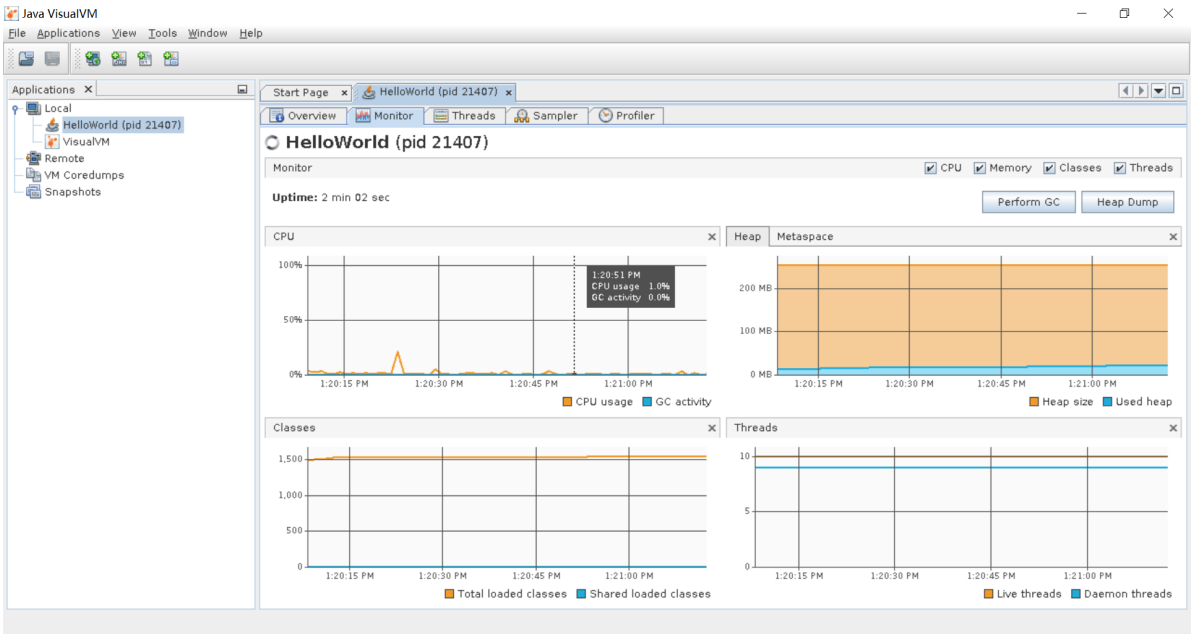
范例: 指定参数运行Java程序

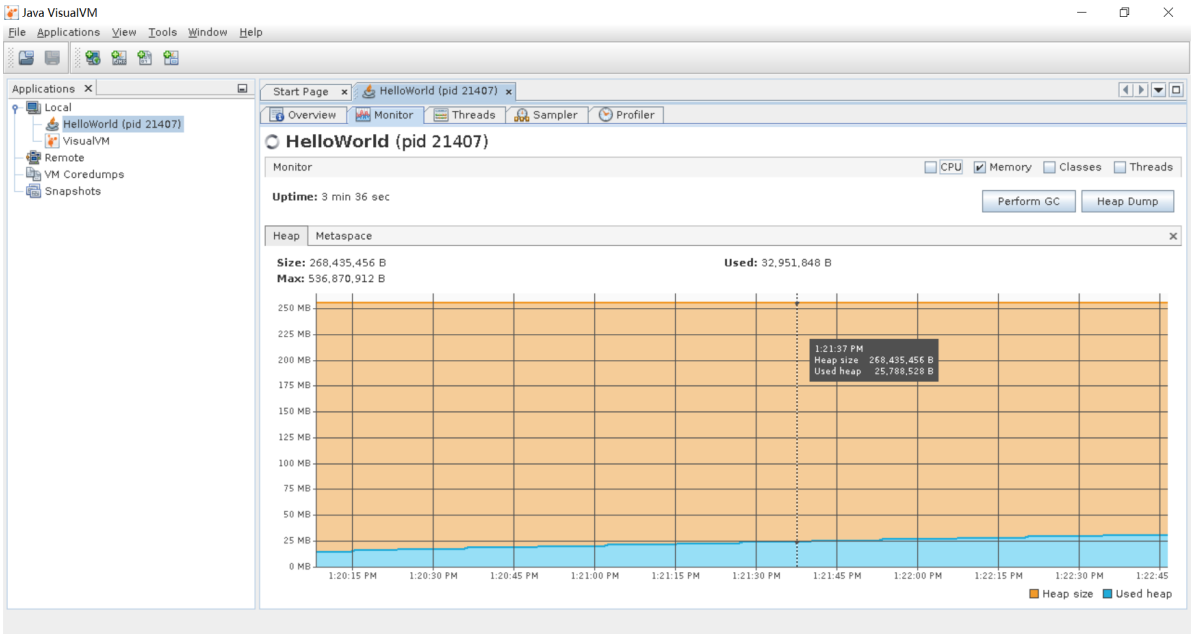
```
#java -cp .  -Xms512m -Xmx1g HelloWorld
```

```
#测试用java程序
javac HelloWorld.java
java -classpath . -Xms512m -Xmx1g HelloWorld

[root@tomcat ~]#cat HelloWorld.java
public class HelloWorld extends Thread {
    public static void main(String[] args) {
        try {
            while (true) {
                Thread.sleep(2000);
                System.out.println("hello magedu");
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

#编译为字节码.class文件
[root@tomcat ~]#javac HelloWorld.java
[root@tomcat ~]#ll HelloWorld.class
-rw-r--r-- 1 root root 590 Jul 13 16:31 HelloWorld.class
[root@tomcat ~]#file HelloWorld.class
HelloWorld.class: compiled Java class data, version 52.0 (Java 1.8)

#指定路径运行class文件
[root@tomcat ~]#java -cp .  -Xms512m -Xmx1g HelloWorld
hello magedu
hello magedu
hello magedu

#或者用下面方法指定CLASS文件的搜索路径
[root@tomcat ~]#echo $CLASSPATH
/usr/local/jdk/lib/:/usr/local/jdk/jre/lib/
[root@tomcat ~]#mv HelloWorld.class  /usr/local/jdk/lib/

#分别执行多次观察
[root@tomcat ~]#java HelloWorld
```

```
[root@tomcat ~]#java -Xms256m -Xmx512m  HelloWorld
[root@tomcat ~]#java -Xms128m -Xmx512m  -XX:NewSize=64m -XX:MaxNewSize=200m
HelloWorld
hello magedu

[root@tomcat ~]#jps
21299 Main
21418 Jps
21407 HelloWorld


#将Linux的图形工具显示到windows桌面
#方法1
#注意:先在windows上开启Xwindows Server，如Xmanager
[root@tomcat ~]#export DISPLAY=10.0.0.1:0.0

#方法2:使用 MobaXterm 连接
[root@tomcat ~]#yum -y install xorg-x11-xauth xorg-x11-fonts-* xorg-x11-font-
utils xorg-x11-fonts-Type1
[root@tomcat ~]#exit

#运行图形工具
[roottomcat ~]#which jvisualvm
/usr/local/jdk/bin/jvisualvm

[root@tomcat ~]#jvisualvm
```







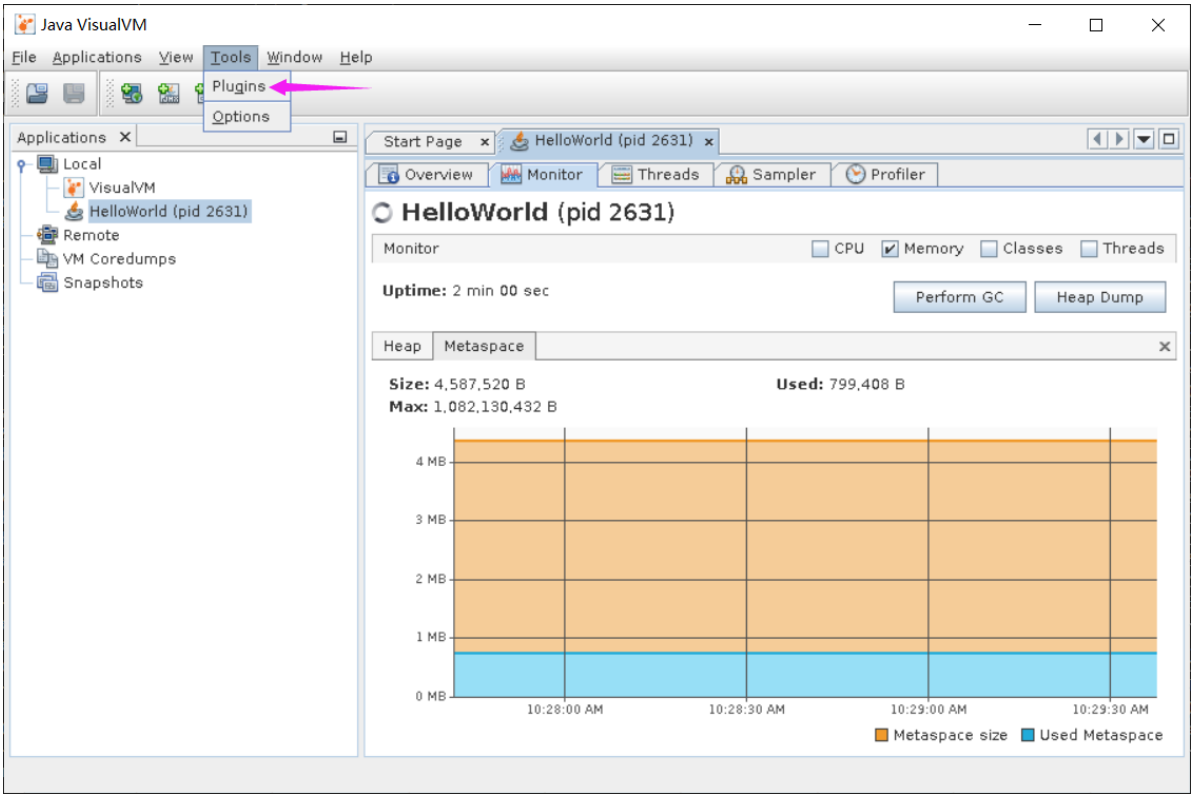
8.2.4.2.2 案例2: 使用 jvisualvm的 Visual GC 插件

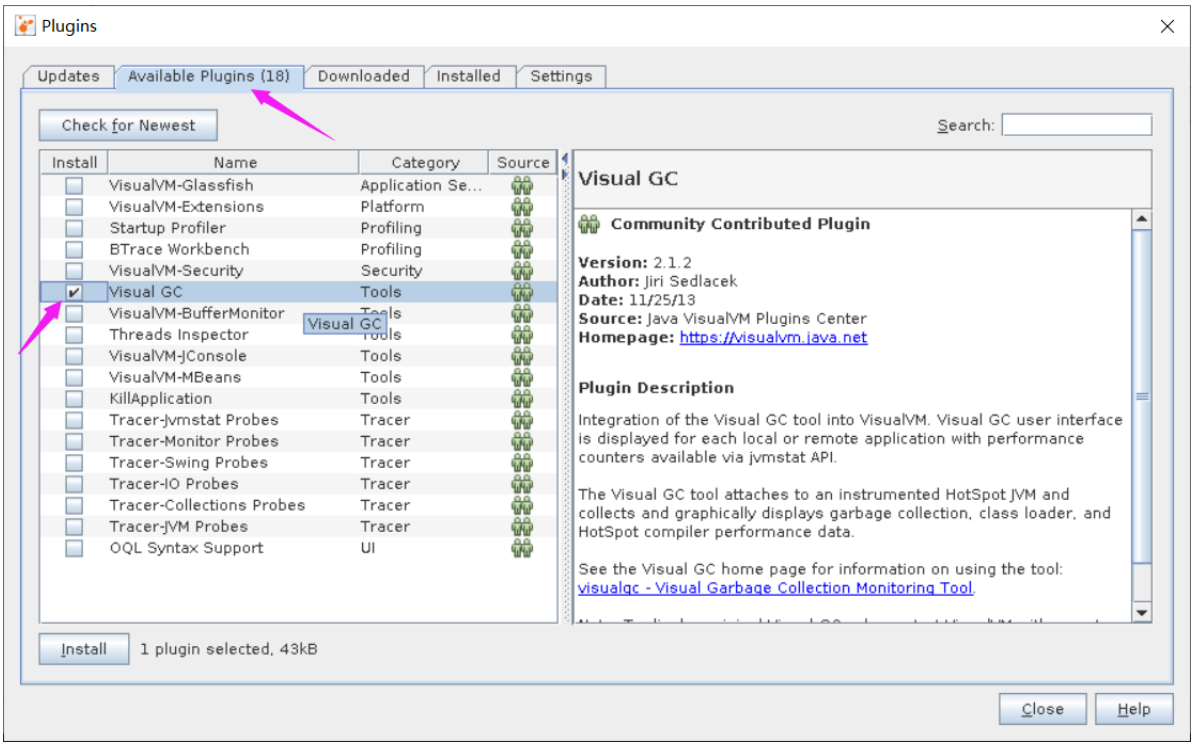
范例: 使用 jvisualvm的 Visual GC 插件 观察 java程序的OOM

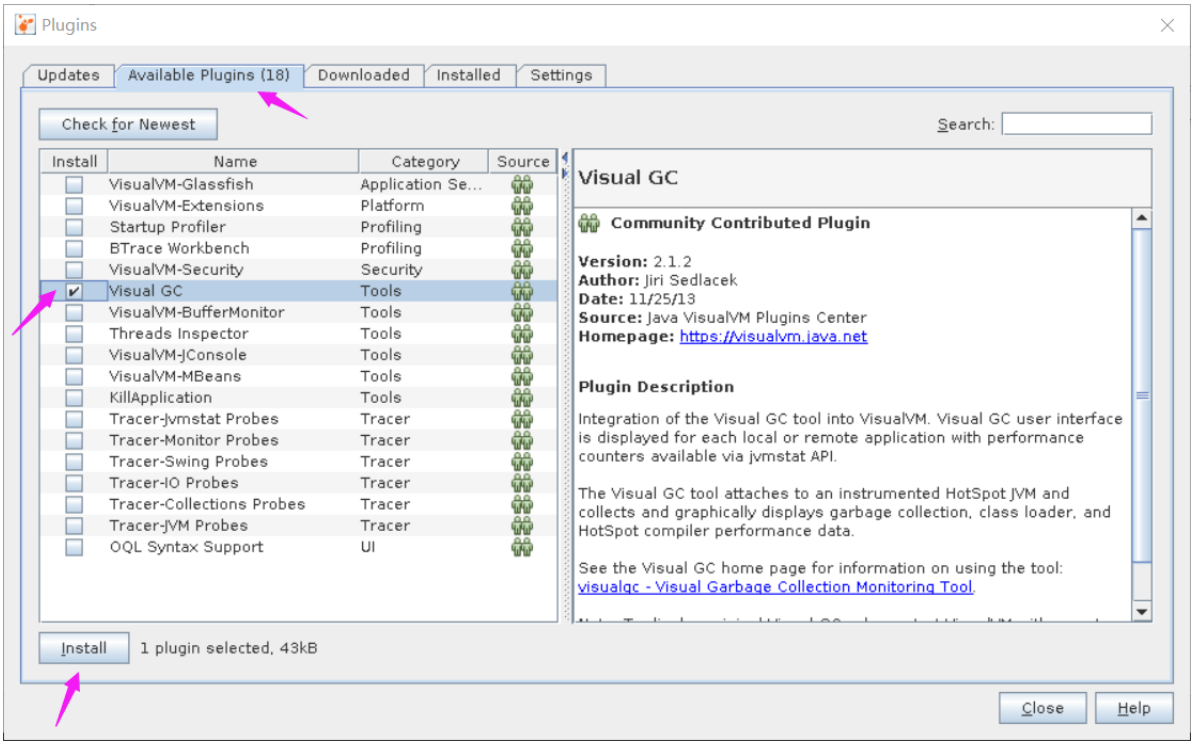
```
[root@centos8 ~]#cat HeapOom.java
import java.util.ArrayList;
import java.util.List;

public class HeapOom {
    public static void main(String[] args) {
        List<byte[]> list =new ArrayList<byte[]>();
        int i = 0;
        boolean flag = true;
        while(flag){
            try{
                i++;
                list.add(new byte[1024* 1024]);//每次增加一个1M大小的数组对象
                Thread.sleep(1000);
            }catch(Throwable e){
                e.printStackTrace();
                flag =false;
```

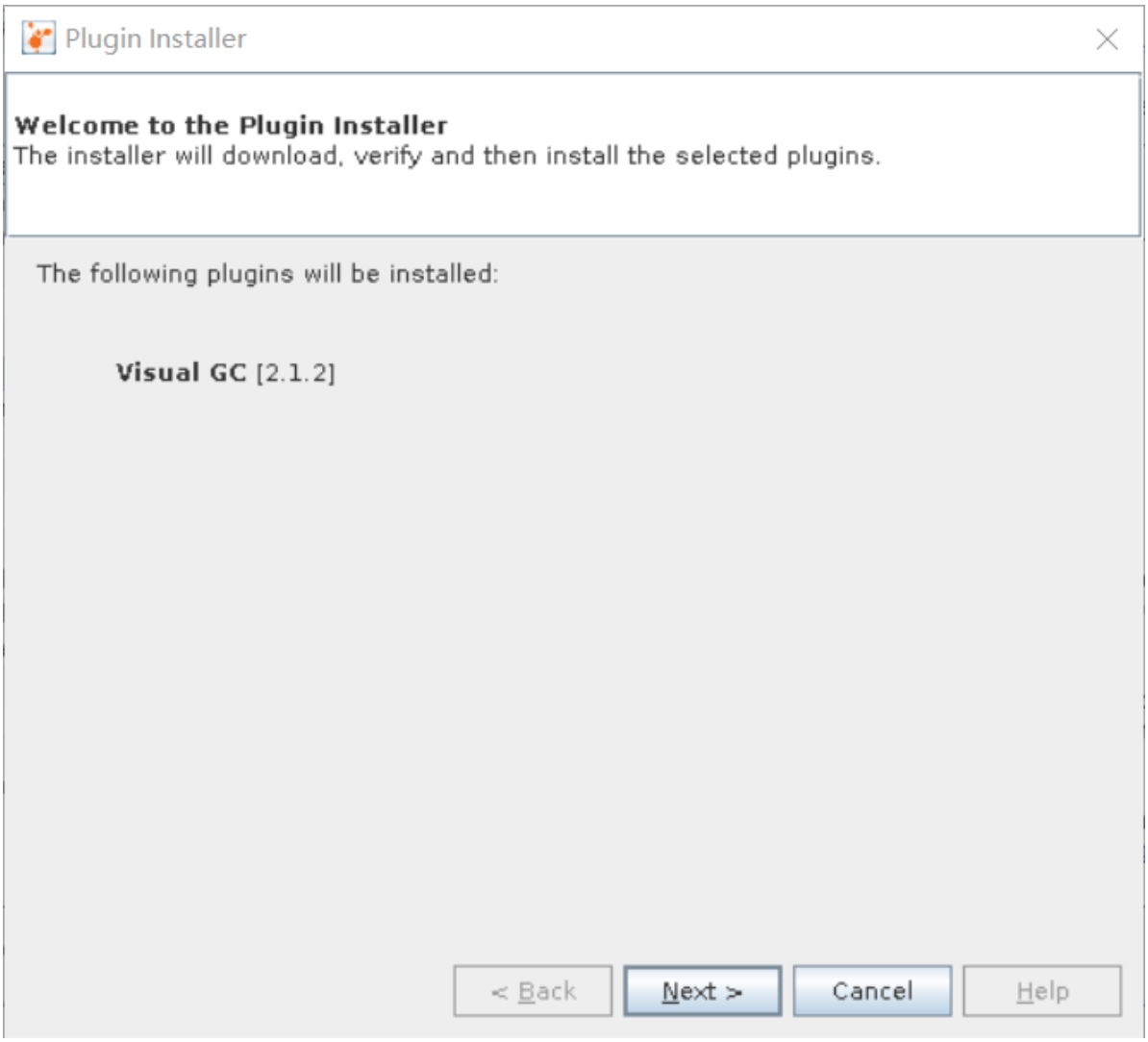
```
                System.out.println("count="+i);//记录运行的次数
            }
        }
    }
}
```

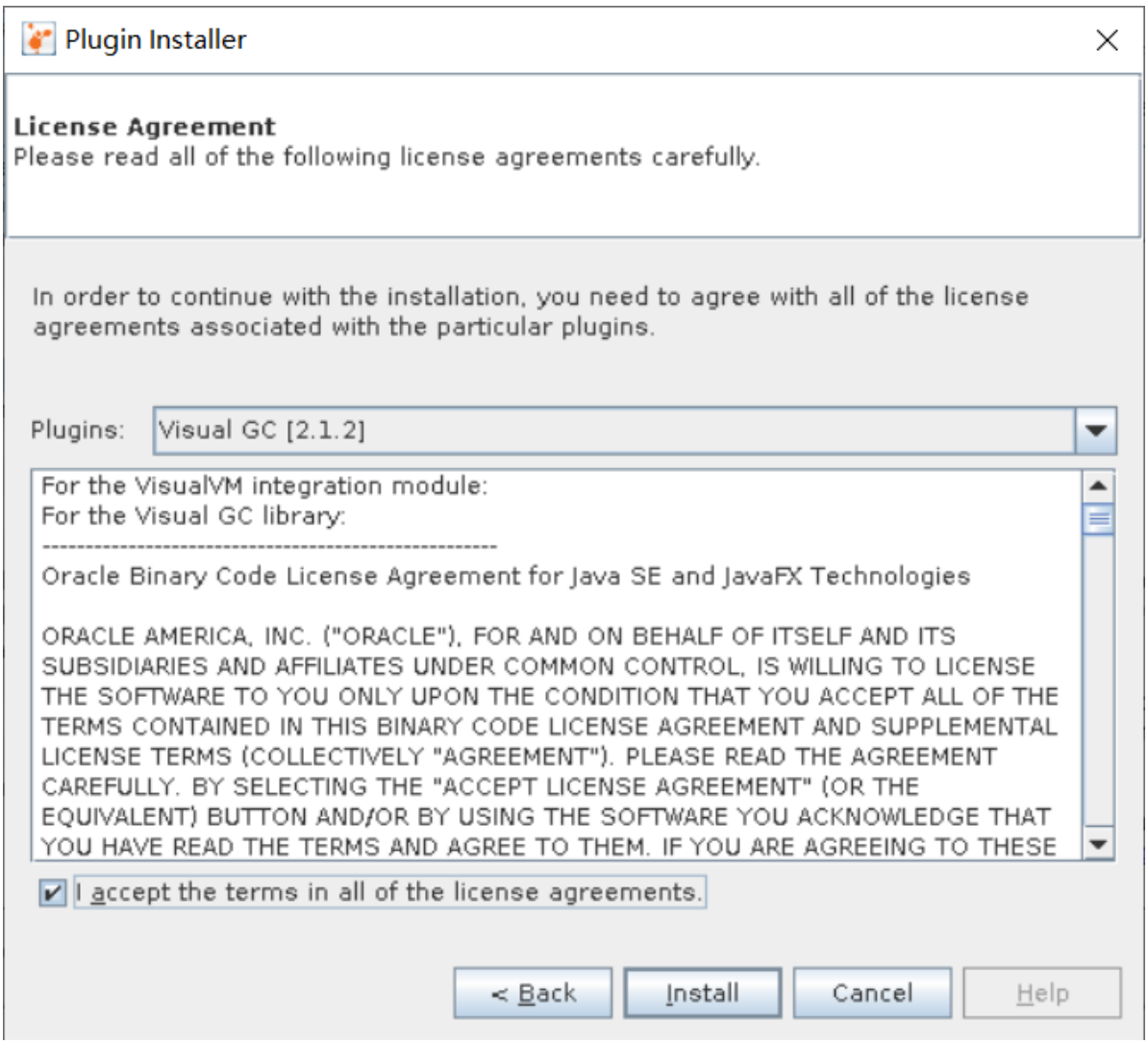
安装VirtualGC插件

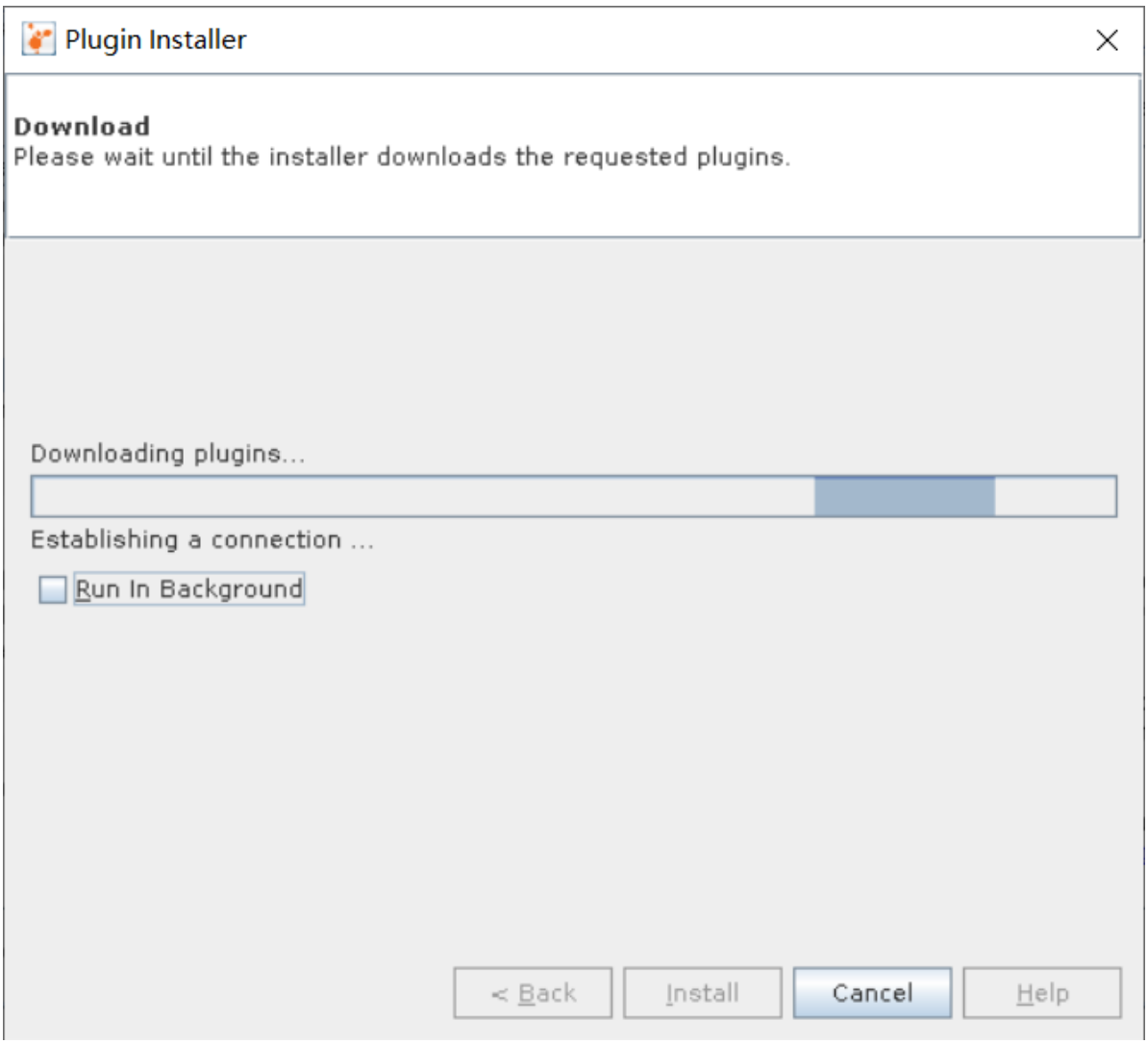


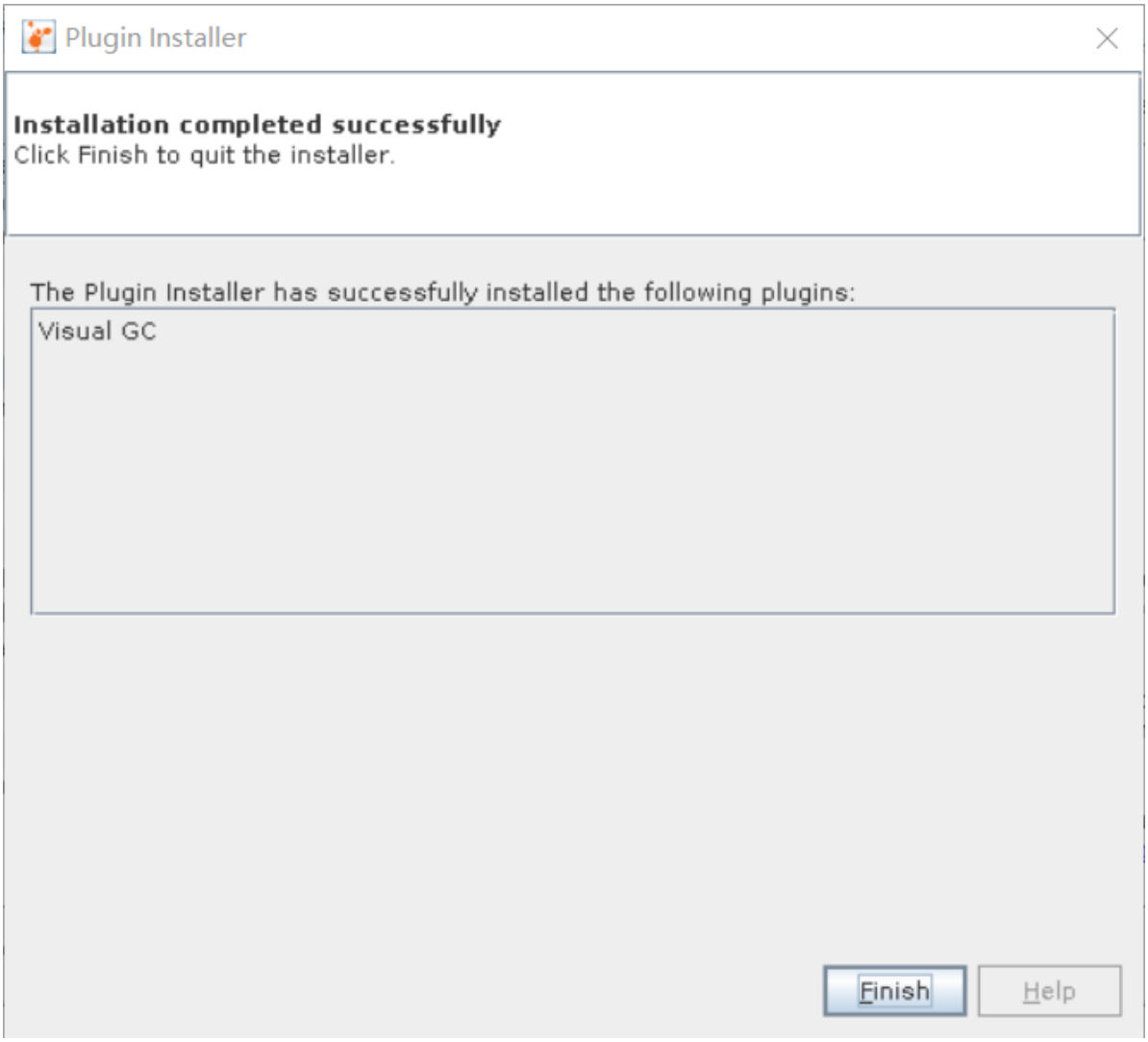


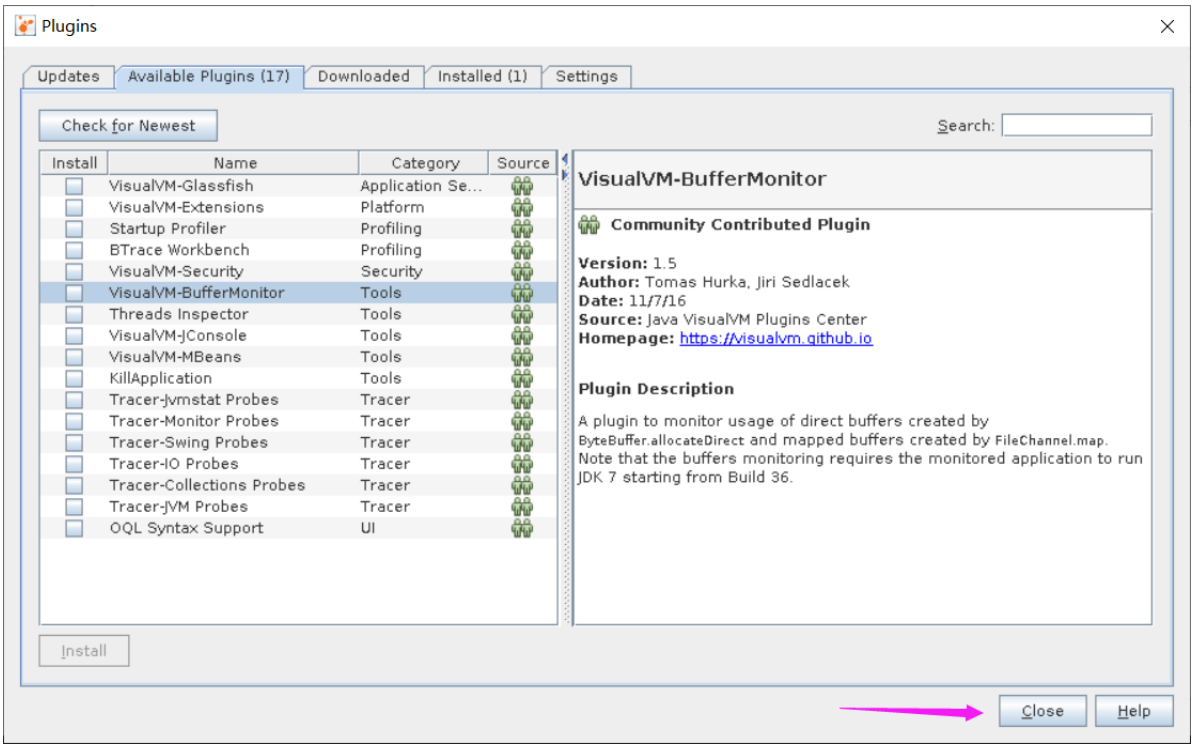

Plugins
Updates
Available Plugins (18)
Downloaded
Installed
Settings
Check for Newest
Search:
Install
Name
Category
Source
VisualVM-Glassfish
Application Se...
VisualVM-Extensions
Platform
Startup Profiler
Profiling
BTrace Workbench
Profiling
VisualVM-Security
Security
Visual GC
Tools
VisualVM-BufferMonitor
Tools
Threads Inspector
Tools
VisualVM-JConsole
Tools
VisualVM-MBeans
Tools
KillApplication
Tools
Tracer-Jvmstat Probes
Tracer
Tracer-Monitor Probes
Tracer
Tracer-Swing Probes
Tracer
Tracer-IO Probes
Tracer
Tracer-Collections Probes
Tracer
Tracer-JVM Probes
Tracer
OQL Syntax Support
UI
Visual GC
Community Contributed Plugin
Version: 2.1.2
Author: Jiri Sedlacek
Date: 11/25/13
Source: Java VisualVM Plugins Center
Homepage: https://visualvm.java.net
Plugin Description
Integration of the Visual GC tool into VisualVM. Visual GC user interface is displayed for each local or remote application with performance counters available via jvmstat API.
The Visual GC tool attaches to an instrumented HotSpot JVM and collects and graphically displays garbage collection, class loader, and HotSpot compiler performance data.
See the Visual GC home page for information on using the tool:
visualgc - Visual Garbage Collection Monitoring Tool.
Install
1 plugin selected, 43kB
Close
Help


Plugin Installer
Welcome to the Plugin Installer
The installer will download, verify and then install the selected plugins.
The following plugins will be installed:
Visual GC [2.1.2]
< Back
Next >
Cancel
Help


Plugin Installer
License Agreement
Please read all of the following license agreements carefully.
In order to continue with the installation, you need to agree with all of the license agreements associated with the particular plugins.
Plugins:
Visual GC [2.1.2]
For the VisualVM integration module:
For the Visual GC library:
Oracle Binary Code License Agreement for Java SE and JavaFX Technologies
ORACLE AMERICA, INC. ("ORACLE"), FOR AND ON BEHALF OF ITSELF AND ITS SUBSIDIARIES AND AFFILIATES UNDER COMMON CONTROL, IS WILLING TO LICENSE THE SOFTWARE TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS BINARY CODE LICENSE AGREEMENT AND SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT"). PLEASE READ THE AGREEMENT CAREFULLY. BY SELECTING THE "ACCEPT LICENSE AGREEMENT" (OR THE EQUIVALENT) BUTTON AND/OR BY USING THE SOFTWARE YOU ACKNOWLEDGE THAT YOU HAVE READ THE TERMS AND AGREE TO THEM. IF YOU ARE AGREEING TO THESE
I accept the terms in all of the license agreements.
< Back
Install
Cancel
Help

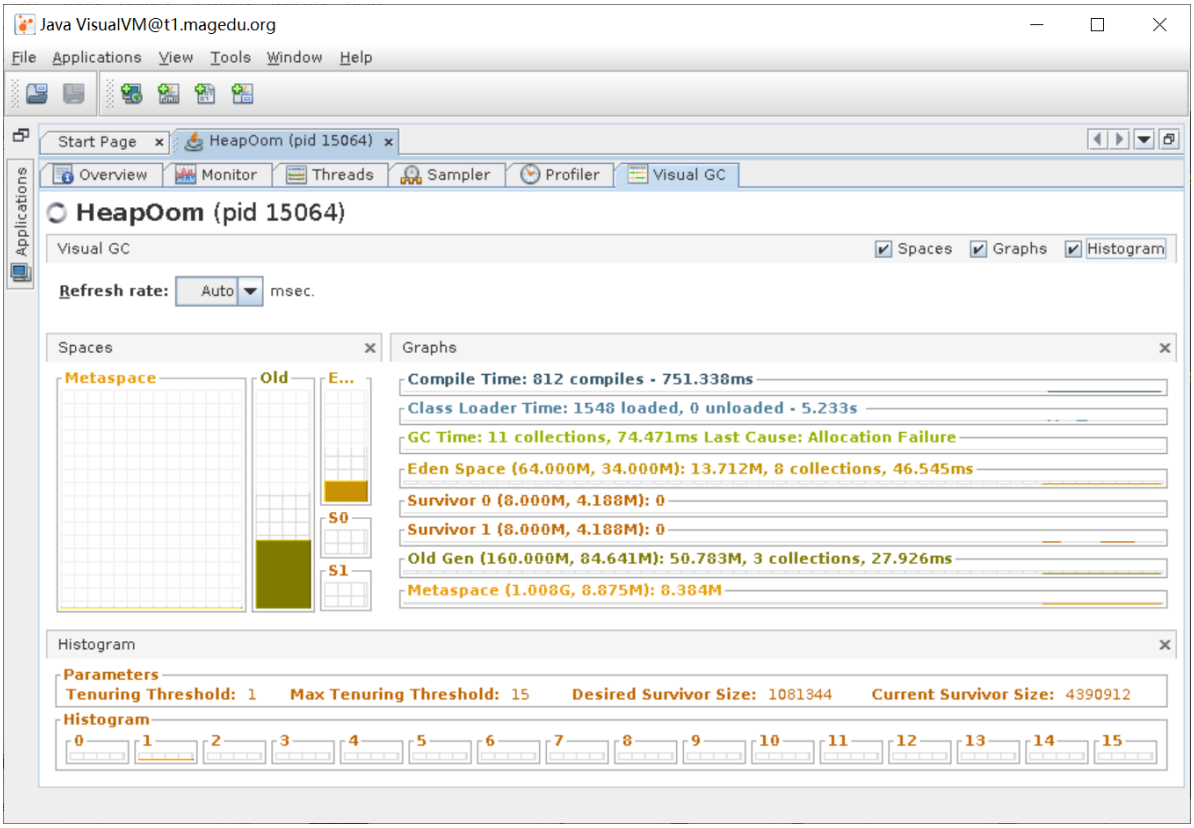

Plugin Installer
Download
Please wait until the installer downloads the requested plugins.
Downloading plugins...
Establishing a connection ...
Run In Background
< Back
Install
Cancel
Help





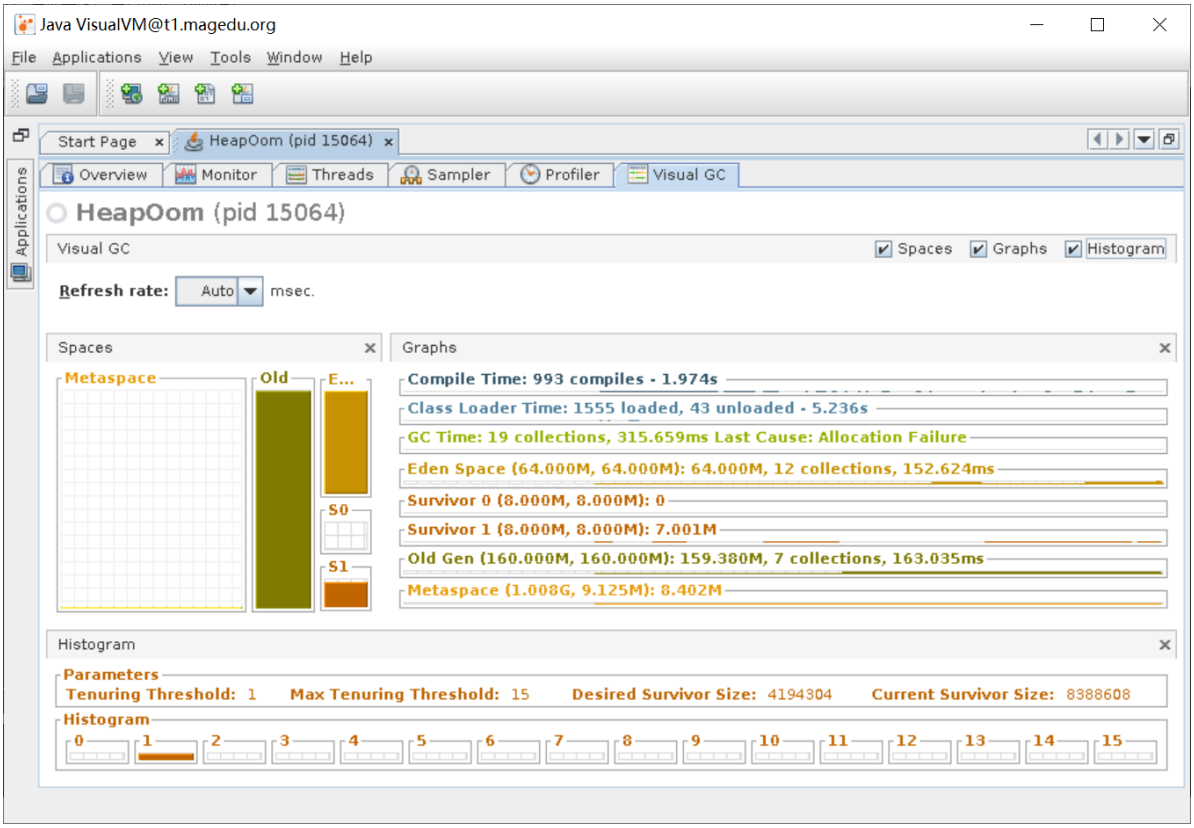
运行上面的OOM程序,重新运行jvisualm, 可以看到多了一下VisualGC页标,下面的图示

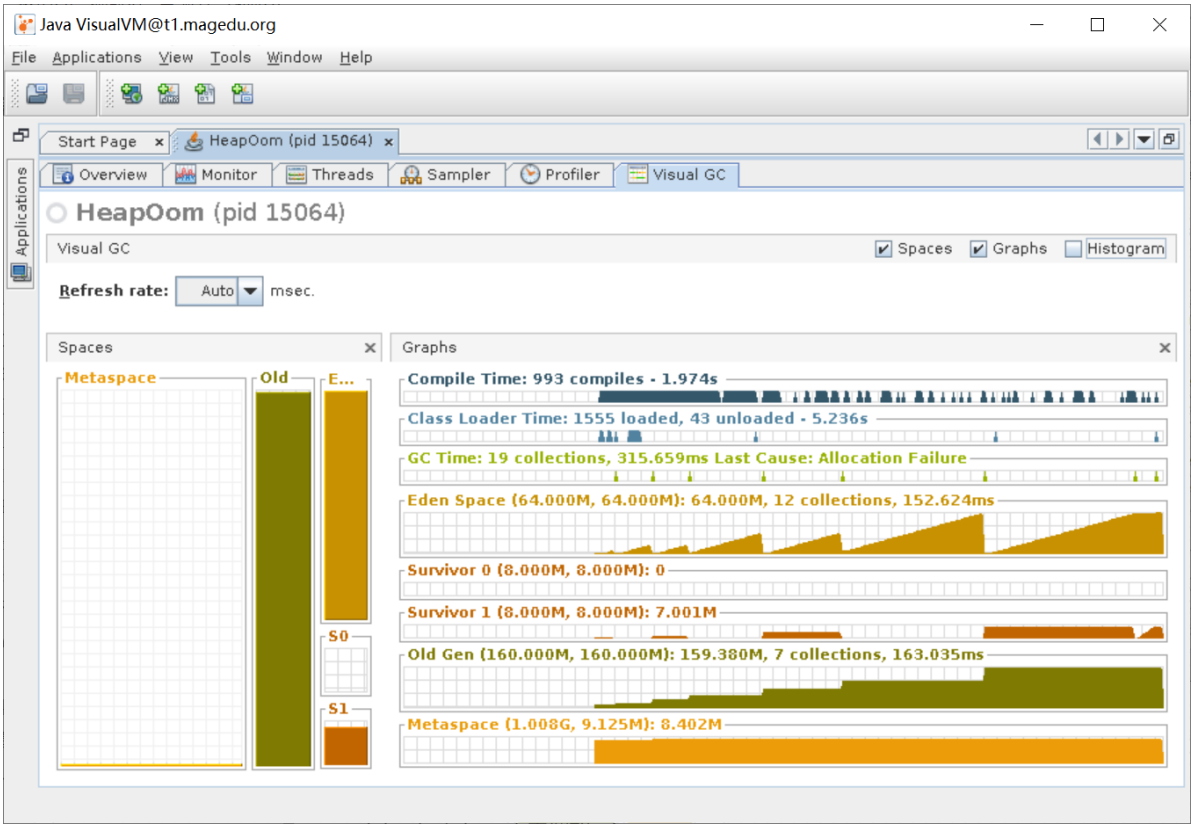
```
[root@centos8 ~]#java -Xms100m -Xmx200m HeapOom
```



当OOM时退出JAVA程序时,会显示下面图示

```
[root@centos8 ~]#java HeapOom
java.lang.OutOfMemoryError: Java heap space
    at HeapOom.main(HeapOom.java:12)
count=229
```





8.2.4.2.3 Jprofiler定位OOM的问题原因

JProfiler是一款功能强大的Java开发分析工具，它可以快速的帮助用户分析出存在的错误，软件还可对需要的显示类进行标记，包括了内存的分配情况和信息的视图等

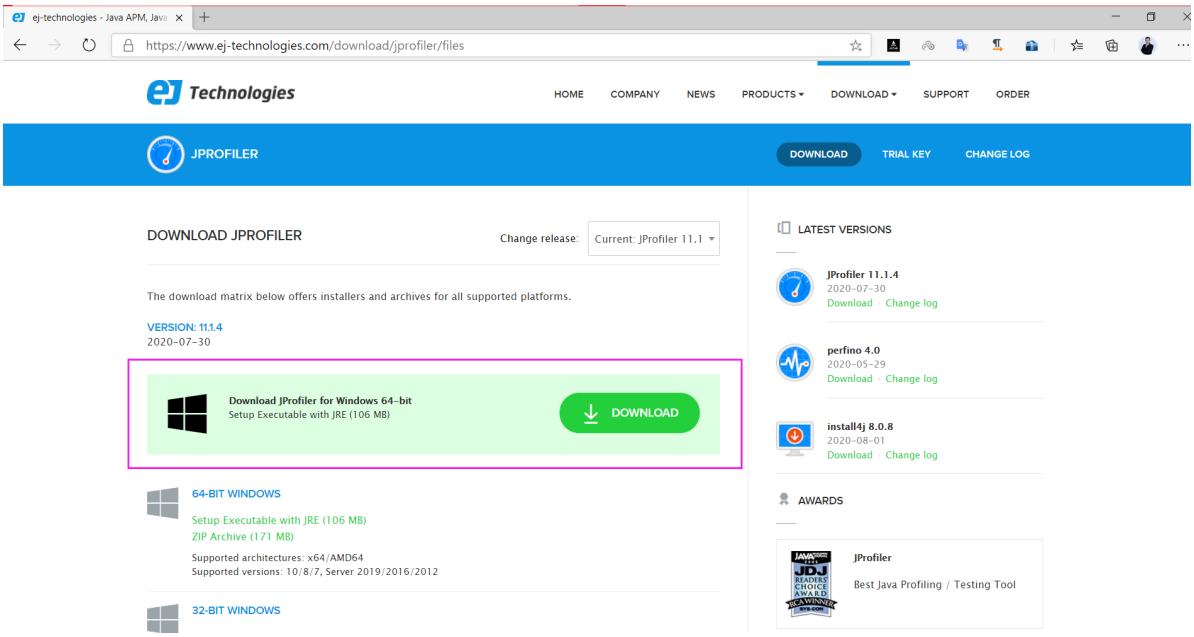
JProfiler官网：http://www.ej-technologies.com/products/jprofiler/overview.html

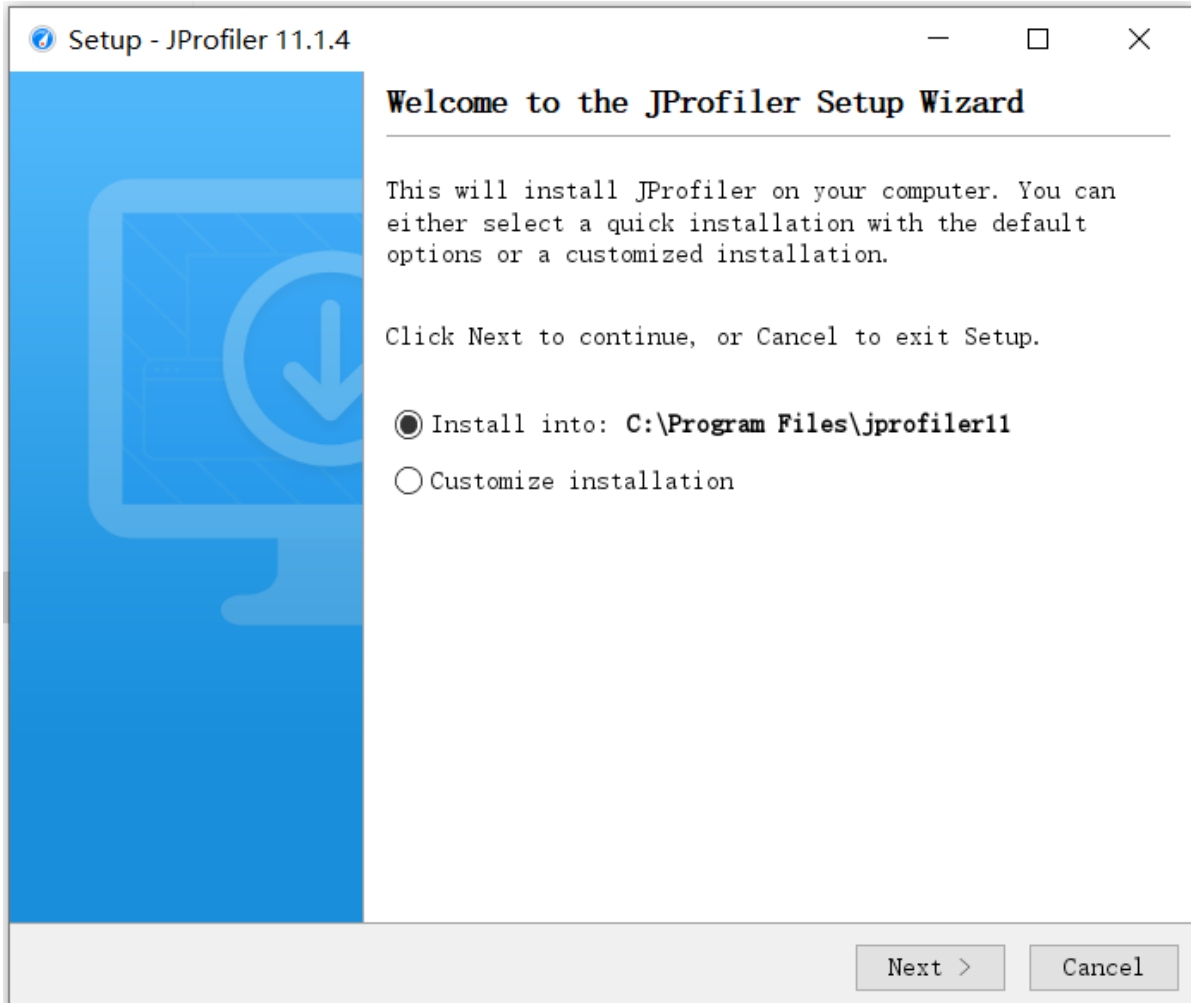
范例: 安装jprofiler工具定位OOM原因和源码问题的位置

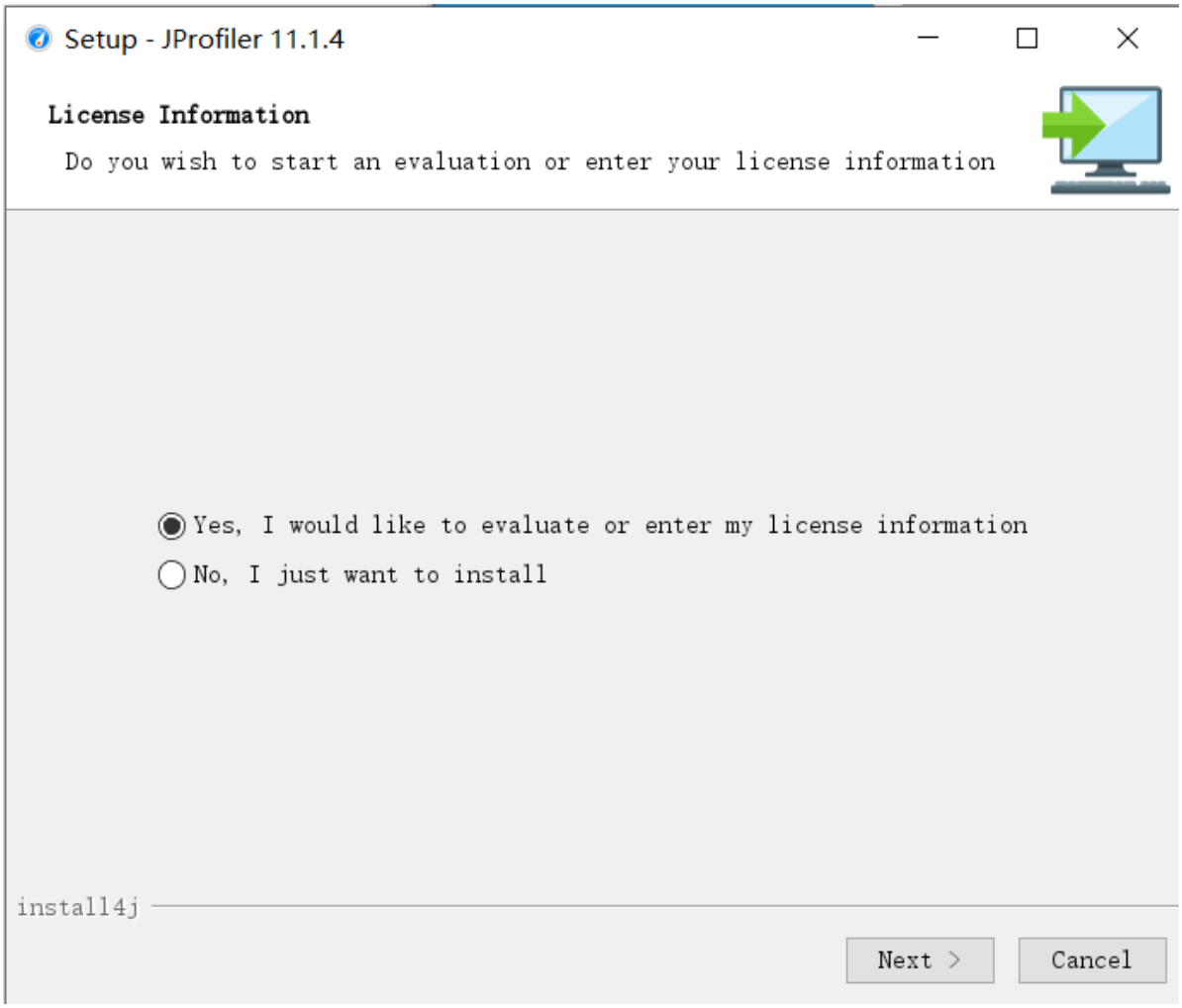
```
#安装OpenJDK
[root@centos8 ~]#dnf -y install java-1.8.0-openjdk-devel

[root@centos8 ~]#cat HeapOom.java
import java.util.ArrayList;
import java.util.List;

public class HeapOom {
    public static void main(String[] args) {
        List<byte[]> list =new ArrayList<byte[]>();
        int i = 0;
        boolean flag = true;
        while(flag){
            try{
                i++;
                list.add(new byte[1024* 1024]);//每次增加一个1M大小的数组对象
                Thread.sleep(1000);
            }catch(Throwable e){
                e.printStackTrace();
                flag = false;
                System.out.println("count="+i);//记录运行的次数
            }
        }
    }
}
```

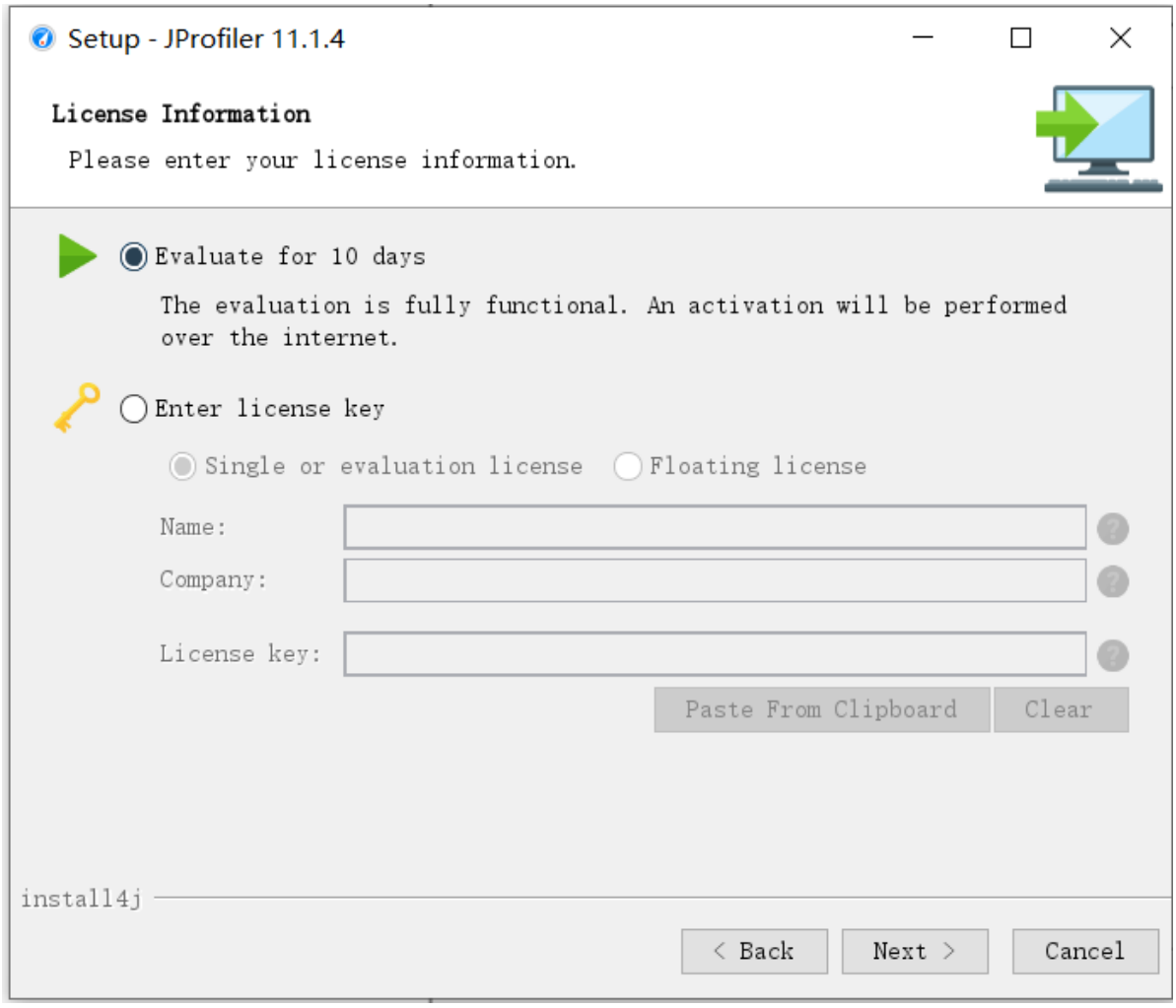
```
[root@centos8 ~]#javac HeapOom.java
[root@centos8 ~]#java -cp .  -Xms5m -Xmx10m -XX:+HeapDumpOnOutOfMemoryError
HeapOom
java.lang.OutOfMemoryError: Java heap space
Dumping heap to java_pid96271.hprof ...
Heap dump file created [8134070 bytes in 0.031 secs]
java.lang.OutOfMemoryError: Java heap space
    at HeapOom.main(HeapOom.java:12)
count=8

[root@centos8 ~]#cat HeapOom2.java
import java. util. Random;
public class HeapOom2 {
    public static void main(String[] args) {
        String str = "I am lao wang";
        while (true){
            str += str + new Random().nextInt(88888888);
        }
    }
}

[root@centos8 ~]#javac HeapOom2.java
[root@centos8 ~]#java -cp . -Xms5m -Xmx10m -XX:+HeapDumpOnOutOfMemoryError
HeapOom2
java.lang.OutOfMemoryError: Java heap space
Dumping heap to java_pid96339.hprof ...
Heap dump file created [4925877 bytes in 0.016 secs]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.util.Arrays.copyOf(Arrays.java:3332)
    at
java.lang.AbstractStringBuilder.ensureCapacityInternal(AbstractStringBuilder.jav
a:124)
    at java.lang.AbstractStringBuilder.append(AbstractStringBuilder.java:674)
    at java.lang.StringBuilder.append(StringBuilder.java:208)
    at HeapOom2.main(HeapOom2.java:6)
```

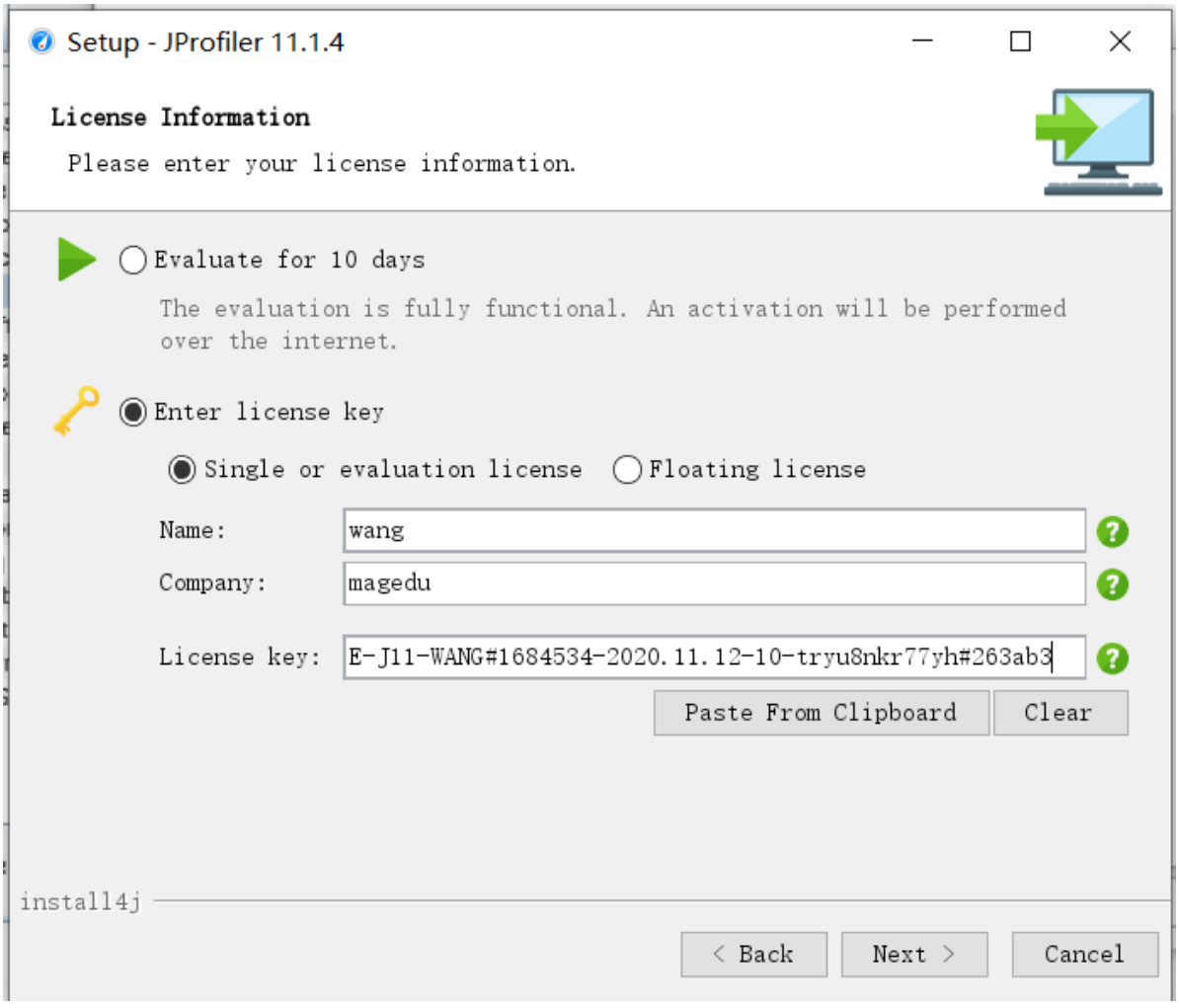
下载并安装Jprofiler



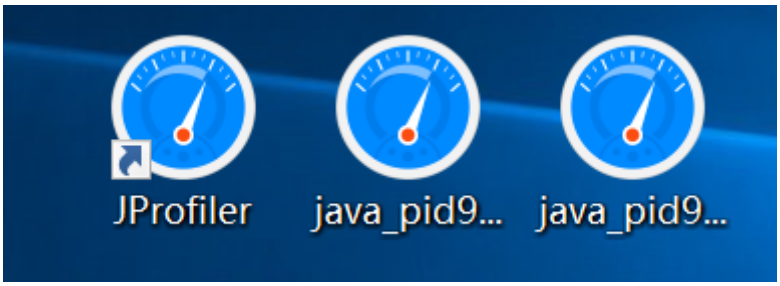

Setup - JProfiler 11.1.4
Welcome to the JProfiler Setup Wizard
This will install JProfiler on your computer. You can either select a quick installation with the default options or a customized installation.
Click Next to continue, or Cancel to exit Setup.
Install into: C:\Program Files\jprofiler11
Customize installation
Next >
Cancel

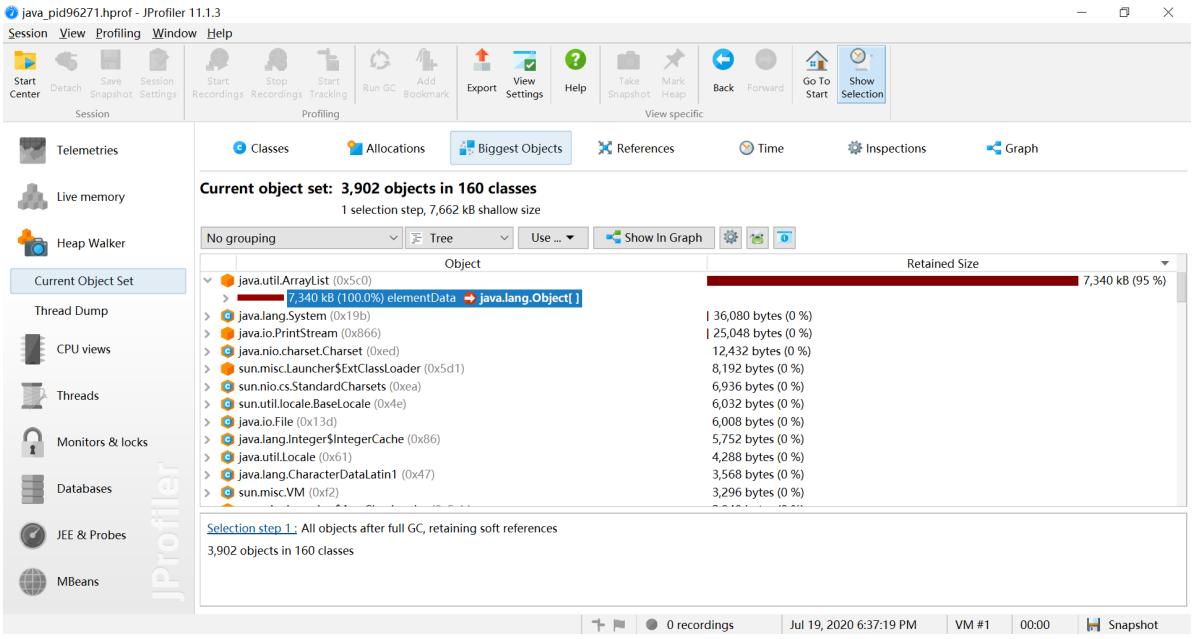

Setup - JProfiler 11.1.4
License Information
Do you wish to start an evaluation or enter your license information
Yes, I would like to evaluate or enter my license information
No, I just want to install
install4j
Next >
Cancel

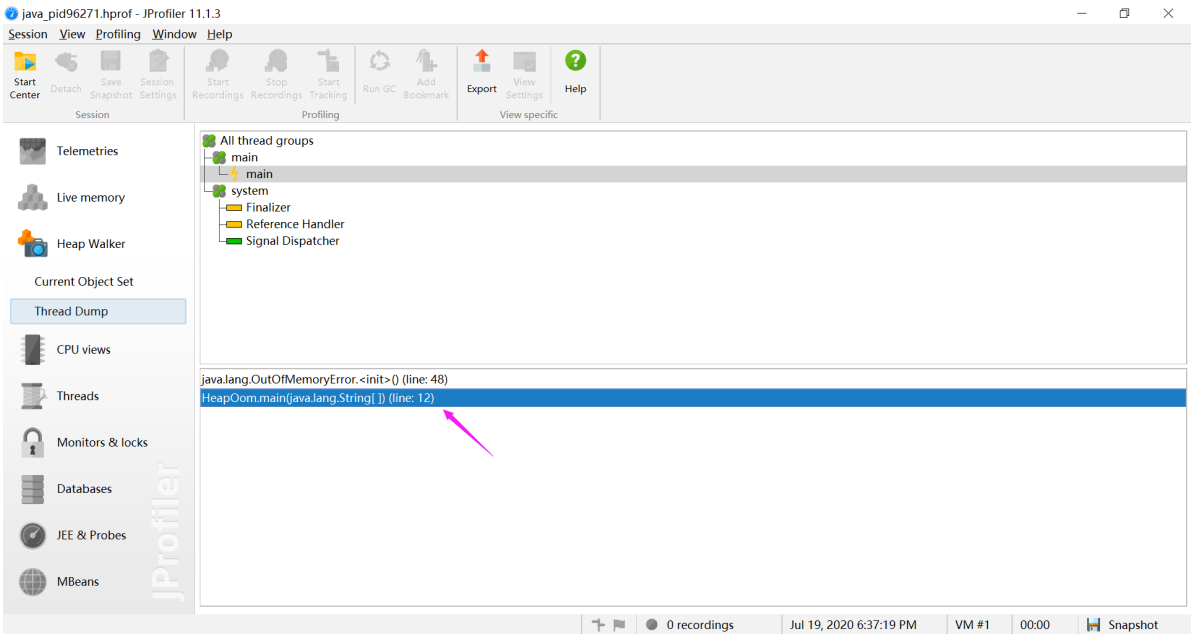

Setup - JProfiler 11.1.4
License Information
Please enter your license information.
Evaluate for 10 days
The evaluation is fully functional. An activation will be performed over the internet.
Enter license key
Single or evaluation license
Floating license
Name:
Company:
License key:
Paste From Clipboard
Clear
install4j
< Back
Next >
Cancel

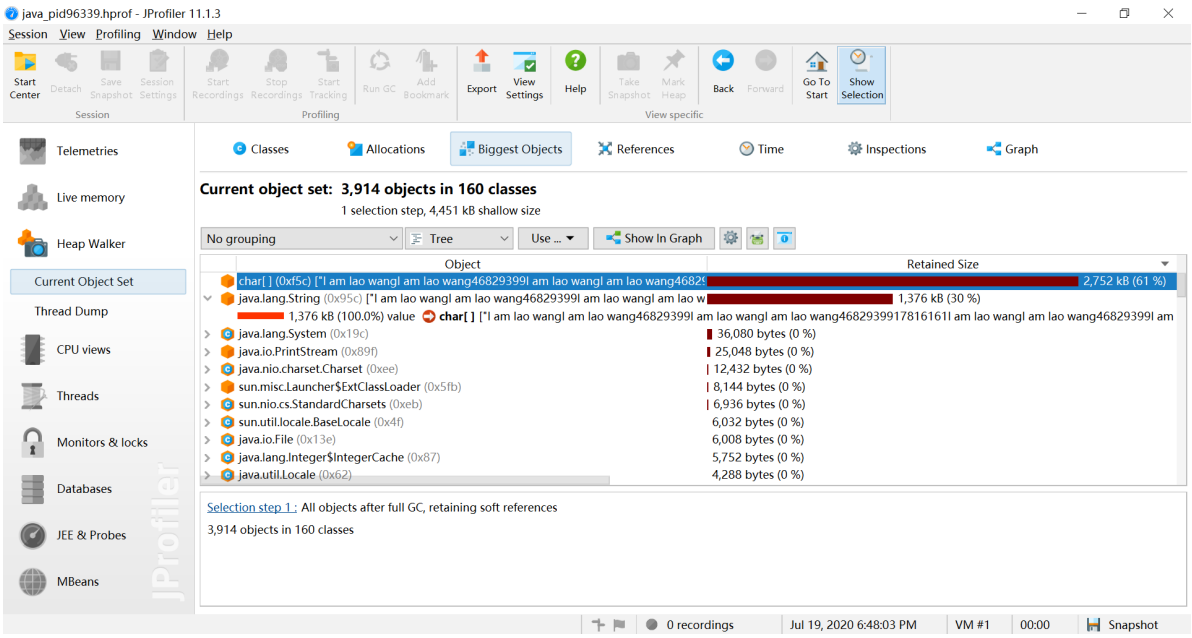

Setup - JProfiler 11.1.4
License Information
Please enter your license information.
Evaluate for 10 days
The evaluation is fully functional. An activation will be performed over the internet.
Enter license key
Single or evaluation license
Floating license
Name:
wang
Company:
magedu
License key:
E-J11-WANG#1684534-2020.11.12-10-tryu8nkr77yh#263ab3
Paste From Clipboard
Clear
install4j
< Back
Next >
Cancel

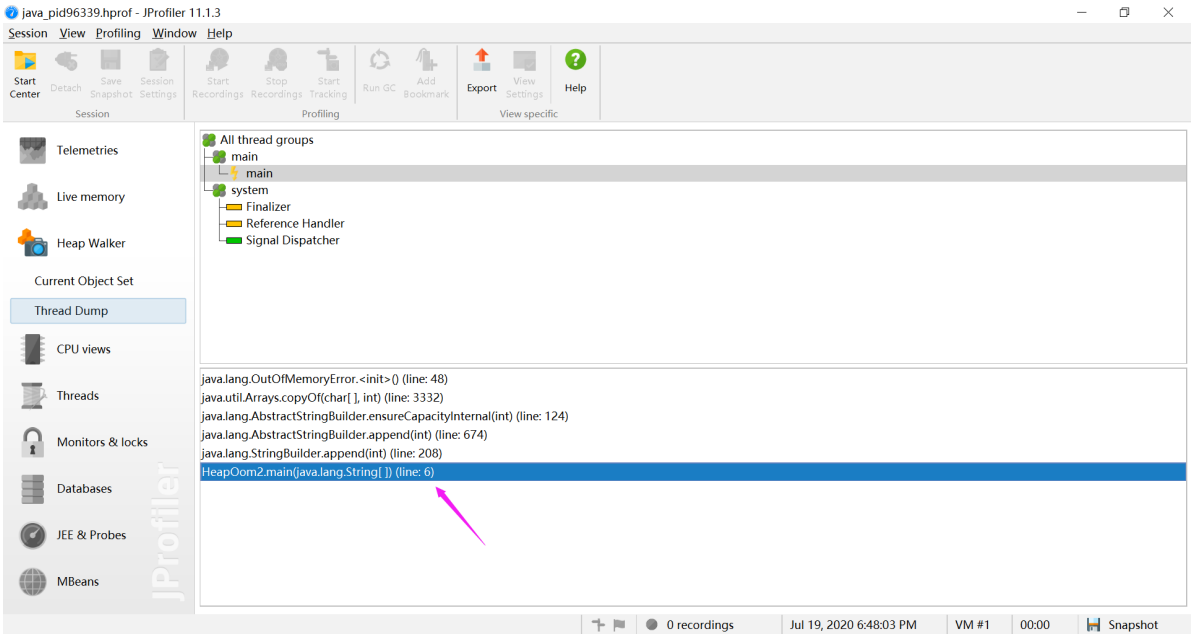
下载并安装JProfiler(傻瓜式安装略)后,双击打开前面生成的两个文件java_pid96271.hprof和java_pid96339,可以看到下面显示,从中分析OOM原因











8.2.4.3 Tomcat的JVM参数设置

默认不指定，-Xmx大约使用了1/4的内存，当前本机内存指定约为1G。

在bin/catalina.sh中增加一行

```
......
# OS specific support.  $var _must_ be set to either true or false.
#添加下面一行
JAVA_OPTS="-server  -Xms128m -Xmx512m -XX:NewSize=48m -XX:MaxNewSize=200m"

cygwin=false
darwin=false
........
```

```
-server：VM运行在server模式，为在服务器端最大化程序运行速度而优化
-client：VM运行在Client模式，为客户端环境减少启动时间而优化
```

重启 Tomcat 观察

```
[root@tomcat ~]#ps aux|grep tomcat
tomcat    22194  5.1 15.1 2497008 123480 ?      Sl   13:31   0:03
/usr/local/jdk/jre/bin/java -
Djava.util.logging.config.file=/usr/local/tomcat/conf/logging.properties -
Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -server -
Xmx512m -Xms128m -XX:NewSize=48m -XX:MaxNewSize=200m -
Djdk.tls.ephemeralDHKeySize=2048 -
Djava.protocol.handler.pkgs=org.apache.catalina.webresources -
Dorg.apache.catalina.security.SecurityListener.UMASK=0027 -
Dignore.endorsed.dirs= -classpath
/usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar -
Dcatalina.base=/usr/local/tomcat -Dcatalina.home=/usr/local/tomcat -
Djava.io.tmpdir=/usr/local/tomcat/temp org.apache.catalina.startup.Bootstrap
start
```

浏览器访问server status页面，可以看到以下页面

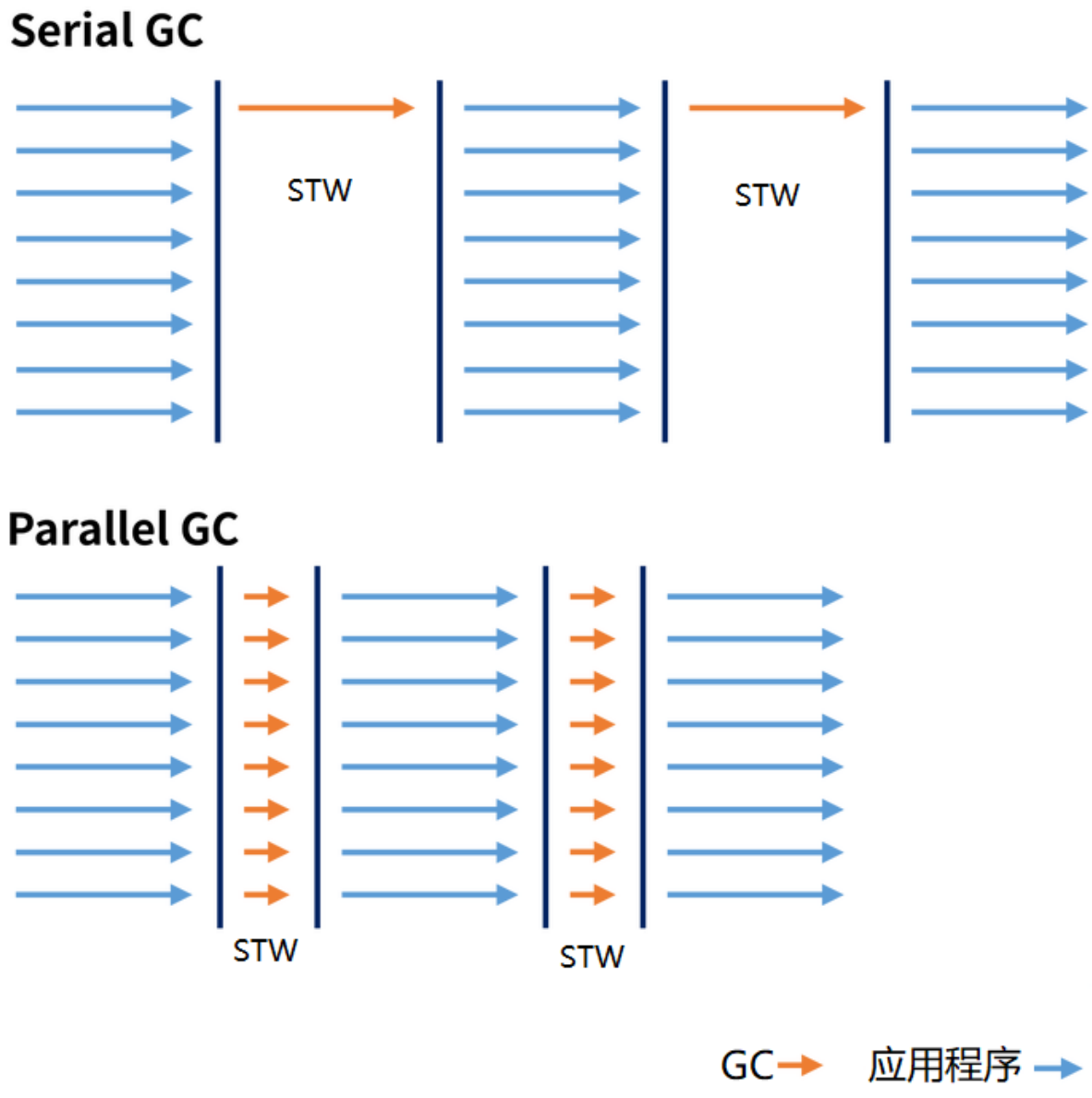
8.2.5 垃圾收集方式

按**工作模式**不同：指的是GC线程和工作线程是否一起运行

- 独占垃圾回收器：只有GC在工作，**STW** 一直进行到回收完毕，工作线程才能继续执行
- 并发垃圾回收器：让**GC线程**垃圾回收某些阶段可以和**工作线程**一起进行,如:标记阶段并行,回收阶段仍然串行

按**回收线程**数：指的是GC线程是否串行或并行执行

- 串行垃圾回收器：一个GC线程完成回收工作
- 并行垃圾回收器：多个GC线程同时一起完成回收工作，充分利用CPU资源



8.2.6 调整策略

对JVM调整策略应用极广

- 在WEB领域中Tomcat等
- 在大数据领域Hadoop生态各组件
- 在消息中间件领域的Kafka等
- 在搜索引擎领域的ElasticSearch、Solr等

注意: 在不同领域和场景对JVM需要不同的调整策略

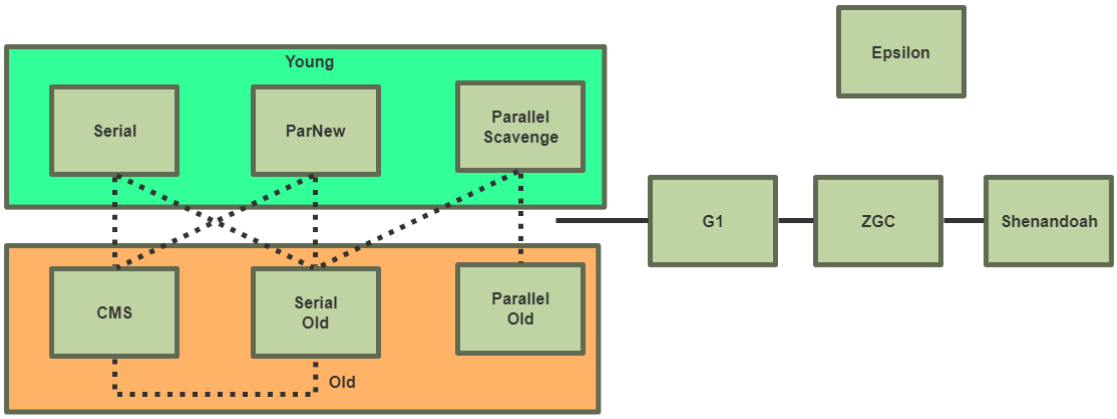
- 减少 STW 时长，串行变并行
- 减少 GC 次数，要分配合适的内存大小

一般情况下，大概可以使用以下原则：

- 客户端或较小程序，内存使用量不大，可以使用串行回收
- 对于服务端大型计算，可以使用并行回收
- 大型WEB应用，用户端不愿意等待，尽量少的STW，可以使用并发回收

8.2.7 垃圾回收器

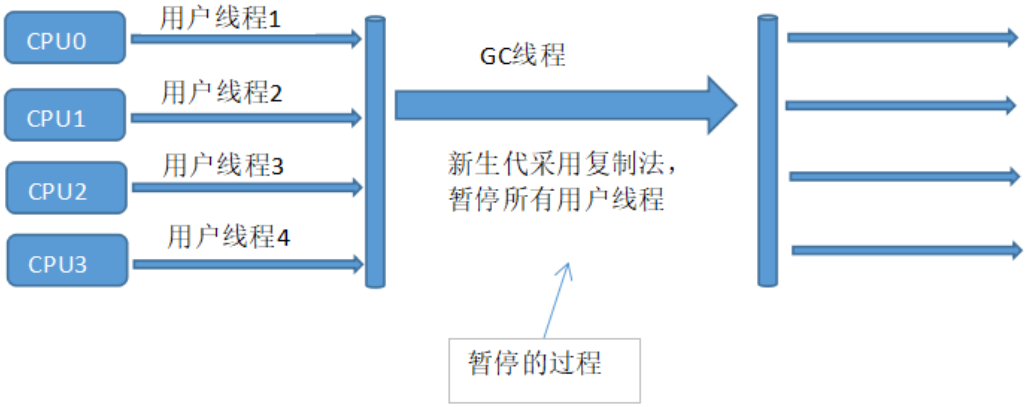
8.2.7.1 常用垃圾回收器



8.2.7.1.1 按分代设置不同垃圾回收器

新生代

- 新生代串行收集器**Serial**：单线程、独占式串行，采用**复制算法**,简单高效但会造成STW



- 新生代并行回收收集器**PS(Parallel Scavenge)**：多线程并行、独占式，会产生STW, 使用**复制算法**

  关注调整**吞吐量**,此收集器关注点是达到一个可控制的吞吐量

  吞吐量 = 运行用户代码时间/（运行用户代码时间+垃圾收集时间），比如虚拟机总共运行100分钟，其中垃圾回收花掉1分钟，那吞吐量就是99%。

  高吞吐量可以高效率利用CPU时间，尽快完成运算任务，主要适合在**后台运算**而不需要太多交互的任务。

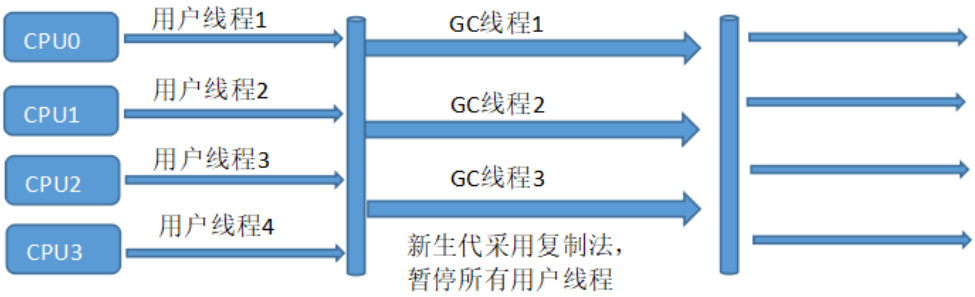
  除此之外，Parallel Scavenge 收集器具有自适应调节策略，它可以将内存管理的调优任务交给虚拟机去完成。自适应调节策略也是Parallel Scavenge与 ParNew 收集器的一个重要区别。

  此为默认的新生代的垃圾回收器

  和ParNew不同,PS不可以和老年代的CMS组合

- 新生代并行收集器**ParNew**：就是Serial 收集器的多线程版,将单线程的串行收集器变成了多线程并行、独占式,使用**复制算法**,相当于PS的改进版

  经常和CMS配合使用,关注点是尽可能地缩短垃圾收集时用户线程的停顿时间，适合需要与**用户交互**的程序，良好的响应速度能提升用户体验

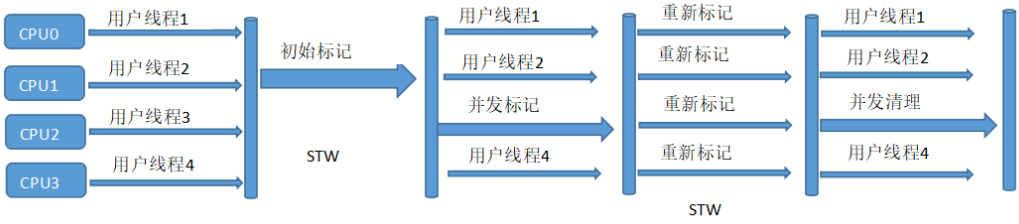


老年代：

- 老年代串行收集器**Serial Old**：Serial Old是Serial收集器的老年代版本,单线程、独占式串行，回收算法使用**标记压缩**
- 老年代并行回收收集器**Parallel Old**：多线程、独占式并行，回收算法使用**标记压缩**，关注调整**吞吐量**

  Parallel Old收集器是Parallel Scavenge 收集器的老年代版本，这个收集器是JDK1.6之后才开始提供，从HotSpot虚拟机的垃圾收集器的图中也可以看出，Parallel Scavenge 收集器无法与CMS收集器配合工作,因为一个是为了吞吐量，一个是为了客户体验（也就是暂停时间的缩短）

  此为默认的新老年代的垃圾回收器
- **CMS (Concurrent Mark Sweep并发标记清除算法) 收集器**
  - 在某些阶段尽量使用和工作线程一起运行，减少STW时长(**200ms**以内), 提升**响应**速度,是互联网服务端BS系统上较佳的回收算法
  - 分为4个阶段：初始标记、并发标记、重新标记、并发清除，在初始标记、重新标记时需要STW。



  - 初始标记：此过程需要STW（Stop The Word），只标记一下GC Roots能直接关联到的对象，速度很快。
  - 并发标记：就是GC Roots进行扫描可达链的过程，为了找出哪些对象需要收集。这个过程远远慢于初始标记，但它是和用户线程一起运行的，不会出现STW，所有用户并不会感受到。
  - 重新标记：为了修正在并发标记期间，用户线程产生的垃圾，这个过程会比初始标记时间稍微长一点，但是也很快，和初始标记一样会产生STW。
  - 并发清理：在重新标记之后，对现有的垃圾进行清理，和并发标记一样也是和用户线程一起运行的，耗时较长（和初始标记比的话），不会出现STW。
  - 由于整个过程中，耗时最长的并发标记和并发清理都是与用户线程一起执行的，所以总体上来说，CMS收集器的内存回收过程是与用户线程一起并发执行的。

8.2.7.1.2 以下收集器不再按明确的分代单独设置

- **G1(Garbage First)收集器**
  - 是最新垃圾回收器，从JDK1.6实验性提供，JDK1.7发布，其设计目标是在多处理器、大内存服务器端提供优于CMS收集器的吞吐量和停顿控制的回收器。JDK9将G1设为默认的收集器,建议 JDK9版本以后使用。
  - 基于**标记压缩**算法,不会产生大量的空间碎片，有利于程序的长期执行。
  - 分为4个阶段：初始标记、并发标记、最终标记、筛选回收。并发标记并发执行，其它阶段STW只有GC线程并行执行。
  - G1收集器是面向服务端的收集器，它的思想就是首先回收尽可能多的垃圾（这也是Garbage-First名字的由来）
  - G1能充分的利用多CPU，多核环境下的硬件优势，使用多个CPU来缩短STW停顿的时间**(10ms以内)。**
  - 可预测的停顿：这是G1相对于CMS的另一大优势，G1和CMS一样都是关注于降低停顿时间，但是G1能够让使用者明确的指定在一个M毫秒的时间片段内，消耗在垃圾收集的时间不得超过N毫秒。
  - 通过此选项指定: +UseG1GC
- ZGC收集器: 减少STW时长(**1ms**以内)，可以PK C++的效率,目前实验阶段
- Shenandoah收集器：和ZGC竞争关系,目前实验阶段
- Epsilon收集器: 调试 JDK 使用,内部使用,不用于生产环境

JVM 1.8 默认的垃圾回收器：PS + ParallelOld,所以大多数都是针对此进行调优

8.2.7.2 垃圾收集器设置

优化调整Java 相关参数的目标: 尽量减少FullGC和STW

通过以下选项可以单独指定新生代、老年代的垃圾收集器

- -server 指定为Server模式,也是默认值,一般使用此工作模式
- -XX:+UseSerialGC
  - 运行在Client模式下，新生代是Serial, 老年代使用SerialOld
- -XX:+UseParNewGC
  - 新生代使用ParNew，老年代使用SerialOld
- -XX:+UseParallelGC
  - 运行于server模式下，新生代使用Parallel Scavenge, 老年代使用 Parallel Old
- -XX:+UseParallelOldGC
  - 新生代使用Paralell Scavenge, 老年代使用Paralell Old，和上面-XX:+UseParallelGC 相同
  - -XX:ParallelGCThreads=N，在关注吞吐量的场景使用此选项增加并行线程数
- **-XX:+UseConcMarkSweepGC**
  - 新生代使用ParNew, 老年代优先使用CMS，备选方式为Serial Old
  - 响应时间要短，停顿短使用这个垃圾收集器
  - -XX:CMSInitiatingOccupancyFraction=N，N为0-100整数表示达到老年代的大小的百分比多少触发回收
    - 默认68
  - -XX:+UseCMSCompactAtFullCollection 开启此值,在CMS收集后，进行内存碎片整理
  - -XX:CMSFullGCsBeforeCompaction=N 设定多少次CMS后，进行一次内存碎片整理
  - -XX:+CMSParallelRemarkEnabled 降低标记停顿

    范例:

```
-XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=80 -
XX:+UseCMSCompactAtFullCollection -XX:CMSFullGCsBeforeCompaction=5
```

范例：查看默认垃圾回收器

```
#Linux
[root@rocky8 ~]#java   -XX:+PrintCommandLineFlags  -version
-XX:InitialHeapSize=31964352 -XX:MaxHeapSize=511429632 -
XX:+PrintCommandLineFlags -XX:+UseCompressedClassPointers -XX:+UseCompressedOops
-XX:+UseParallelGC
java version "1.8.0_301"
Java(TM) SE Runtime Environment (build 1.8.0_301-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.301-b09, mixed mode)

#Windows
C:\Users\wang>java   -XX:+PrintCommandLineFlags  -version
-XX:InitialHeapSize=1071317952 -XX:MaxHeapSize=17141087232 -
XX:+PrintCommandLineFlags -XX:+UseCompressedClassPointers -XX:+UseCompressedOops
-XX:-UseLargePagesIndividualAllocation -XX:+UseParallelGC
java version "1.8.0_281"
Java(TM) SE Runtime Environment (build 1.8.0_281-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.281-b09, mixed mode)
```

范例：查看垃圾回收器

```
#-XX:+UseSerialGC表示Serial+SerialOld
[root@rocky8 ~]#java -Xms10m -Xmx10m -XX:+PrintGCDetails -XX:+UseSerialGC -cp .
HeapOom2
[GC (Allocation Failure) [DefNew: 2376K->320K(3072K), 0.0004583 secs] 2376K-
>657K(9920K), 0.0004984 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [DefNew: 2374K->0K(3072K), 0.0003771 secs] 2711K-
>1257K(9920K), 0.0003866 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [DefNew: 2447K->0K(3072K), 0.0004045 secs] 3704K-
>2857K(9920K), 0.0004243 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [DefNew: 1643K->0K(3072K), 0.0001604 secs] 4500K-
>4457K(9920K), 0.0001758 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [DefNew: 1600K->0K(3072K), 0.0002896 secs][Tenured:
6057K->2856K(6848K), 0.0009171 secs] 6057K->2856K(9920K), [Metaspace: 2480K-
>2480K(1056768K)], 0.0012276 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [DefNew: 1647K->1647K(3072K), 0.0000053 secs][Tenured:
6056K->2056K(6848K), 0.0006369 secs] 7703K->2056K(9920K), [Metaspace: 2480K-
>2480K(1056768K)], 0.0006801 secs] [Times: user=0.00 sys=0.00, real=0.01 secs]
[GC (Allocation Failure) [DefNew: 1600K->1600K(3072K), 0.0000057 secs][Tenured:
5256K->5256K(6848K), 0.0005282 secs] 6856K->5256K(9920K), [Metaspace: 2480K-
>2480K(1056768K)], 0.0005636 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[Full GC (Allocation Failure) [Tenured: 5256K->5042K(6848K), 0.0008440 secs]
5256K->5042K(9920K), [Metaspace: 2480K->2480K(1056768K)], 0.0008589 secs]
[Times: user=0.00 sys=0.00, real=0.00 secs]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.util.Arrays.copyOf(Arrays.java:3332)
    at
java.lang.AbstractStringBuilder.ensureCapacityInternal(AbstractStringBuilder.jav
a:124)
    at java.lang.AbstractStringBuilder.append(AbstractStringBuilder.java:674)
    at java.lang.StringBuilder.append(StringBuilder.java:208)
    at HeapOom2.main(HeapOom2.java:6)
```

```
Heap
 def new generation   total 3072K, used 105K [0x00000000ff600000,
0x00000000ff950000, 0x00000000ff950000)
  eden space 2752K,   3% used [0x00000000ff600000, 0x00000000ff61a5c8,
0x00000000ff8b0000)
  from space 320K,   0% used [0x00000000ff900000, 0x00000000ff900000,
0x00000000ff950000)
  to   space 320K,   0% used [0x00000000ff8b0000, 0x00000000ff8b0000,
0x00000000ff900000)
 tenured generation   total 6848K, used 5042K [0x00000000ff950000,
0x0000000100000000, 0x0000000100000000)
   the space 6848K,  73% used [0x00000000ff950000, 0x00000000ffe3cac8,
0x00000000ffe3cc00, 0x0000000100000000)
 Metaspace       used 2514K, capacity 4486K, committed 4864K, reserved 1056768K
  class space    used 268K, capacity 386K, committed 512K, reserved 1048576K

#-XX:+UseParNewGC表示ParNew+SerialOld
[root@rocky8 ~]#java -Xms10m -Xmx10m -XX:+PrintGCDetails -XX:+UseParNewGC -cp .
HeapOom2
Java HotSpot(TM) 64-Bit Server VM warning: Using the ParNew young collector with
the Serial old collector is deprecated and will likely be removed in a future
release
[GC (Allocation Failure) [ParNew: 2373K->265K(3072K), 0.0008588 secs] 2373K-
>664K(9920K), 0.0008885 secs] [Times: user=0.00 sys=0.00, real=0.01 secs]
[GC (Allocation Failure) [ParNew: 2316K->99K(3072K), 0.0007617 secs] 2716K-
>1559K(9920K), 0.0007802 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [ParNew: 2543K->24K(3072K), 0.0006585 secs] 4002K-
>3082K(9920K), 0.0006758 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [ParNew: 1665K->6K(3072K), 0.0004363 secs] 4722K-
>4660K(9920K), 0.0004525 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [ParNew: 1603K->2K(3072K), 0.0005977 secs][Tenured:
6252K->2652K(6848K), 0.0009806 secs] 6258K->2652K(9920K), [Metaspace: 2480K-
>2480K(1056768K)], 0.0015996 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [ParNew: 1645K->1645K(3072K), 0.0000045 secs][Tenured:
5847K->1853K(6848K), 0.0006388 secs] 7492K->1853K(9920K), [Metaspace: 2480K-
>2480K(1056768K)], 0.0006629 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [ParNew: 1597K->0K(3072K), 0.0004854 secs][Tenured:
5048K->5048K(6848K), 0.0006832 secs] 6646K->5048K(9920K), [Metaspace: 2480K-
>2480K(1056768K)], 0.0011950 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[Full GC (Allocation Failure) [Tenured: 5048K->5035K(6848K), 0.0006889 secs]
5048K->5035K(9920K), [Metaspace: 2480K->2480K(1056768K)], 0.0007030 secs]
[Times: user=0.00 sys=0.00, real=0.00 secs]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.util.Arrays.copyOf(Arrays.java:3332)
    at
java.lang.AbstractStringBuilder.ensureCapacityInternal(AbstractStringBuilder.jav
a:124)
    at java.lang.AbstractStringBuilder.append(AbstractStringBuilder.java:674)
    at java.lang.StringBuilder.append(StringBuilder.java:208)
    at HeapOom2.main(HeapOom2.java:6)
Heap
 par new generation   total 3072K, used 105K [0x00000000ff600000,
0x00000000ff950000, 0x00000000ff950000)
  eden space 2752K,   3% used [0x00000000ff600000, 0x00000000ff61a5c8,
0x00000000ff8b0000)
  from space 320K,   0% used [0x00000000ff8b0000, 0x00000000ff8b0000,
0x00000000ff900000)
```

```
  to   space 320K,   0% used [0x00000000ff900000, 0x00000000ff900000,
0x00000000ff950000)
 tenured generation   total 6848K, used 5035K [0x00000000ff950000,
0x0000000100000000, 0x0000000100000000)
   the space 6848K,  73% used [0x00000000ff950000, 0x00000000ffe3af00,
0x00000000ffe3b000, 0x0000000100000000)
 Metaspace       used 2514K, capacity 4486K, committed 4864K, reserved 1056768K
  class space    used 268K, capacity 386K, committed 512K, reserved 1048576K

#-XX:+UseParallelGC表示PS+ParOld 此为默认值
[root@rocky8 ~]#java -Xms10m -Xmx10m -XX:+PrintGCDetails -XX:+UseParallelGC  -cp
. HeapOom2
[GC (Allocation Failure) [PSYoungGen: 1953K->491K(2560K)] 1953K->755K(9728K),
0.0009754 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 2512K->492K(2560K)] 2775K->755K(9728K),
0.0005462 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 2512K->96K(2560K)] 4359K->2735K(9728K),
0.0006484 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 888K->96K(2560K)] 5111K->4319K(9728K),
0.0004700 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 96K->96K(2560K)] 4319K->4319K(9728K),
0.0004402 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[Full GC (Allocation Failure) [PSYoungGen: 96K->0K(2560K)] [ParOldGen: 4223K-
>2632K(7168K)] 4319K->2632K(9728K), [Metaspace: 2480K->2480K(1056768K)],
0.0020238 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) --[PSYoungGen: 1624K->1624K(2560K)] 7424K-
>7424K(9728K), 0.0004027 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[Full GC (Ergonomics) [PSYoungGen: 1624K->0K(2560K)] [ParOldGen: 5800K-
>1839K(7168K)] 7424K->1839K(9728K), [Metaspace: 2480K->2480K(1056768K)],
0.0018894 secs] [Times: user=0.00 sys=0.00, real=0.01 secs]
[GC (Allocation Failure) [PSYoungGen: 1584K->0K(1536K)] 6591K->5007K(8704K),
0.0004222 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 0K->0K(2048K)] 5007K->5007K(9216K),
0.0003842 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[Full GC (Allocation Failure) [PSYoungGen: 0K->0K(2048K)] [ParOldGen: 5007K-
>5007K(7168K)] 5007K->5007K(9216K), [Metaspace: 2480K->2480K(1056768K)],
0.0016387 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 0K->0K(2048K)] 5007K->5007K(9216K),
0.0003614 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[Full GC (Allocation Failure) [PSYoungGen: 0K->0K(2048K)] [ParOldGen: 5007K-
>4994K(7168K)] 5007K->4994K(9216K), [Metaspace: 2480K->2480K(1056768K)],
0.0017426 secs] [Times: user=0.01 sys=0.00, real=0.00 secs]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.util.Arrays.copyOf(Arrays.java:3332)
    at
java.lang.AbstractStringBuilder.ensureCapacityInternal(AbstractStringBuilder.jav
a:124)
    at java.lang.AbstractStringBuilder.append(AbstractStringBuilder.java:674)
    at java.lang.StringBuilder.append(StringBuilder.java:208)
    at HeapOom2.main(HeapOom2.java:6)
Heap
 PSYoungGen      total 2048K, used 41K [0x00000000ffd00000, 0x0000000100000000,
0x0000000100000000)
  eden space 1024K, 4% used
[0x00000000ffd00000,0x00000000ffd0a518,0x00000000ffe00000)
  from space 1024K, 0% used
[0x00000000fff00000,0x00000000fff00000,0x0000000100000000)
```

```
  to   space 1024K, 0% used
[0x00000000ffe00000,0x00000000ffe00000,0x00000000fff00000)
 ParOldGen       total 7168K, used 4994K [0x00000000ff600000, 0x00000000ffd00000,
0x00000000ffd00000)
  object space 7168K, 69% used
[0x00000000ff600000,0x00000000ffae0a98,0x00000000ffd00000)
 Metaspace       used 2514K, capacity 4486K, committed 4864K, reserved 1056768K
  class space    used 268K, capacity 386K, committed 512K, reserved 1048576K

 #-XX:+UseParallelOldGC表示PS+ParOld
[root@rocky8 ~]#java -Xms10m -Xmx10m -XX:+PrintGCDetails -XX:+UseParallelOldGC
-cp . HeapOom2
[GC (Allocation Failure) [PSYoungGen: 1933K->486K(2560K)] 1933K->748K(9728K),
0.0009954 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 2482K->486K(2560K)] 2744K->748K(9728K),
0.0005175 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 2482K->96K(2560K)] 4308K->2703K(9728K),
0.0006469 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 878K->96K(2560K)] 5049K->4267K(9728K),
0.0005057 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 96K->96K(2560K)] 4267K->4267K(9728K),
0.0004297 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[Full GC (Allocation Failure) [PSYoungGen: 96K->0K(2560K)] [ParOldGen: 4171K-
>2602K(7168K)] 4267K->2602K(9728K), [Metaspace: 2480K->2480K(1056768K)],
0.0020167 secs] [Times: user=0.00 sys=0.01, real=0.00 secs]
[GC (Allocation Failure) --[PSYoungGen: 1604K->1604K(2560K)] 7334K-
>7334K(9728K), 0.0004226 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[Full GC (Ergonomics) [PSYoungGen: 1604K->0K(2560K)] [ParOldGen: 5729K-
>1819K(7168K)] 7334K->1819K(9728K), [Metaspace: 2480K->2480K(1056768K)],
0.0018280 secs] [Times: user=0.00 sys=0.00, real=0.01 secs]
[GC (Allocation Failure) [PSYoungGen: 1563K->0K(1536K)] 6511K->4947K(8704K),
0.0004604 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 0K->0K(2048K)] 4947K->4947K(9216K),
0.0003834 secs] [Times: user=0.01 sys=0.00, real=0.00 secs]
[Full GC (Allocation Failure) [PSYoungGen: 0K->0K(2048K)] [ParOldGen: 4947K-
>4947K(7168K)] 4947K->4947K(9216K), [Metaspace: 2480K->2480K(1056768K)],
0.0012999 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 0K->0K(2048K)] 4947K->4947K(9216K),
0.0003402 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[Full GC (Allocation Failure) [PSYoungGen: 0K->0K(2048K)] [ParOldGen: 4947K-
>4934K(7168K)] 4947K->4934K(9216K), [Metaspace: 2480K->2480K(1056768K)],
0.0018222 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.util.Arrays.copyOf(Arrays.java:3332)
    at
java.lang.AbstractStringBuilder.ensureCapacityInternal(AbstractStringBuilder.jav
a:124)
    at java.lang.AbstractStringBuilder.append(AbstractStringBuilder.java:674)
    at java.lang.StringBuilder.append(StringBuilder.java:208)
    at HeapOom2.main(HeapOom2.java:6)
Heap
 PSYoungGen      total 2048K, used 41K [0x00000000ffd00000, 0x0000000100000000,
0x0000000100000000)
  eden space 1024K, 4% used
[0x00000000ffd00000,0x00000000ffd0a518,0x00000000ffe00000)
  from space 1024K, 0% used
[0x00000000fff00000,0x00000000fff00000,0x0000000100000000)
```

```
  to   space 1024K, 0% used
[0x00000000ffe00000,0x00000000ffe00000,0x00000000fff00000)
 ParOldGen       total 7168K, used 4934K [0x00000000ff600000, 0x00000000ffd00000,
0x00000000ffd00000)
  object space 7168K, 68% used
[0x00000000ff600000,0x00000000ffad1a08,0x00000000ffd00000)
 Metaspace       used 2514K, capacity 4486K, committed 4864K, reserved 1056768K
  class space    used 268K, capacity 386K, committed 512K, reserved 1048576K


#-XX:+UseConcMarkSweepGC表示ParNew+CMS
[root@rocky8 ~]#java -Xms10m -Xmx10m -XX:+PrintGCDetails -XX:+UseConcMarkSweepGC
-cp . HeapOom2
[GC (Allocation Failure) [ParNew: 2365K->273K(3072K), 0.0004709 secs] 2365K-
>674K(9920K), 0.0005774 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [ParNew: 2324K->100K(3072K), 0.0008058 secs] 2726K-
>1563K(9920K), 0.0008263 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [ParNew: 2543K->25K(3072K), 0.0004260 secs] 4006K-
>3085K(9920K), 0.0004505 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [ParNew: 1665K->6K(3072K), 0.0004862 secs] 4726K-
>4664K(9920K), 0.0005126 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (CMS Initial Mark) [1 CMS-initial-mark: 4657K(6848K)] 6261K(9920K),
0.0001252 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[CMS-concurrent-mark-start]
[CMS-concurrent-mark: 0.000/0.000 secs] [Times: user=0.00 sys=0.00, real=0.00
secs]
[CMS-concurrent-preclean-start]
[CMS-concurrent-preclean: 0.000/0.000 secs] [Times: user=0.00 sys=0.00,
real=0.00 secs]
[GC (Allocation Failure) [ParNew: 1603K->2K(3072K), 0.0005197 secs][CMS
(concurrent mode failure): 6255K->2655K(6848K), 0.0010552 secs] 6261K-
>2655K(9920K), [Metaspace: 2480K->2480K(1056768K)], 0.0016009 secs] [Times:
user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [ParNew: 1644K->1644K(3072K), 0.0000147 secs][CMS:
5850K->1857K(6848K), 0.0007565 secs] 7495K->1857K(9920K), [Metaspace: 2480K-
>2480K(1056768K)], 0.0007937 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [ParNew: 1597K->0K(3072K), 0.0001449 secs][CMS: 5052K-
>5052K(6848K), 0.0007624 secs] 6649K->5052K(9920K), [Metaspace: 2480K-
>2480K(1056768K)], 0.0009314 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[Full GC (Allocation Failure) [CMS: 5052K->5039K(6848K), 0.0007938 secs] 5052K-
>5039K(9920K), [Metaspace: 2480K->2480K(1056768K)], 0.0008031 secs] [Times:
user=0.00 sys=0.00, real=0.00 secs]
[GC (CMS Initial Mark) [1 CMS-initial-mark: 5039K(6848K)] 5039K(9920K),
0.0001806 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[CMS-concurrent-mark-start]
[CMS-concurrent-mark: 0.000/0.000 secs] [Times: user=0.00 sys=0.00, real=0.00
secs]
[CMS-concurrent-preclean-start]
[CMS-concurrent-preclean: 0.000/0.000 secs] [Times: user=0.00 sys=0.00,
real=0.00 secs]
[GC (CMS Final Remark) [YG occupancy: 50 K (3072 K)][Rescan (parallel) ,
0.0001153 secs][weak refs processing, 0.0000026 secs][class unloading, 0.0000639
secs][scrub symbol table, 0.0001370 secs][scrub string table, 0.0000589 secs][1
CMS-remark: 5039K(6848K)] 5089K(9920K), 0.0004434 secs] [Times: user=0.00
sys=0.00, real=0.00 secs]
[CMS-concurrent-sweep-start]
[CMS-concurrent-sweep: 0.000/0.000 secs] [Times: user=0.00 sys=0.00, real=0.00
secs]
```

```
[CMS-concurrent-reset-start]
[CMS-concurrent-reset: 0.000/0.000 secs] [Times: user=0.00 sys=0.00, real=0.00
secs]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.util.Arrays.copyOf(Arrays.java:3332)
    at
java.lang.AbstractStringBuilder.ensureCapacityInternal(AbstractStringBuilder.jav
a:124)
    at java.lang.AbstractStringBuilder.append(AbstractStringBuilder.java:674)
    at java.lang.StringBuilder.append(StringBuilder.java:208)
    at HeapOom2.main(HeapOom2.java:6)
Heap
 par new generation   total 3072K, used 105K [0x00000000ff600000,
0x00000000ff950000, 0x00000000ff950000)
  eden space 2752K,   3% used [0x00000000ff600000, 0x00000000ff61a5e0,
0x00000000ff8b0000)
  from space 320K,   0% used [0x00000000ff8b0000, 0x00000000ff8b0000,
0x00000000ff900000)
  to   space 320K,   0% used [0x00000000ff900000, 0x00000000ff900000,
0x00000000ff950000)
 concurrent mark-sweep generation total 6848K, used 245K [0x00000000ff950000,
0x0000000100000000, 0x0000000100000000)
 Metaspace       used 2514K, capacity 4486K, committed 4864K, reserved 1056768K
  class space    used 268K, capacity 386K, committed 512K, reserved 1048576K
```

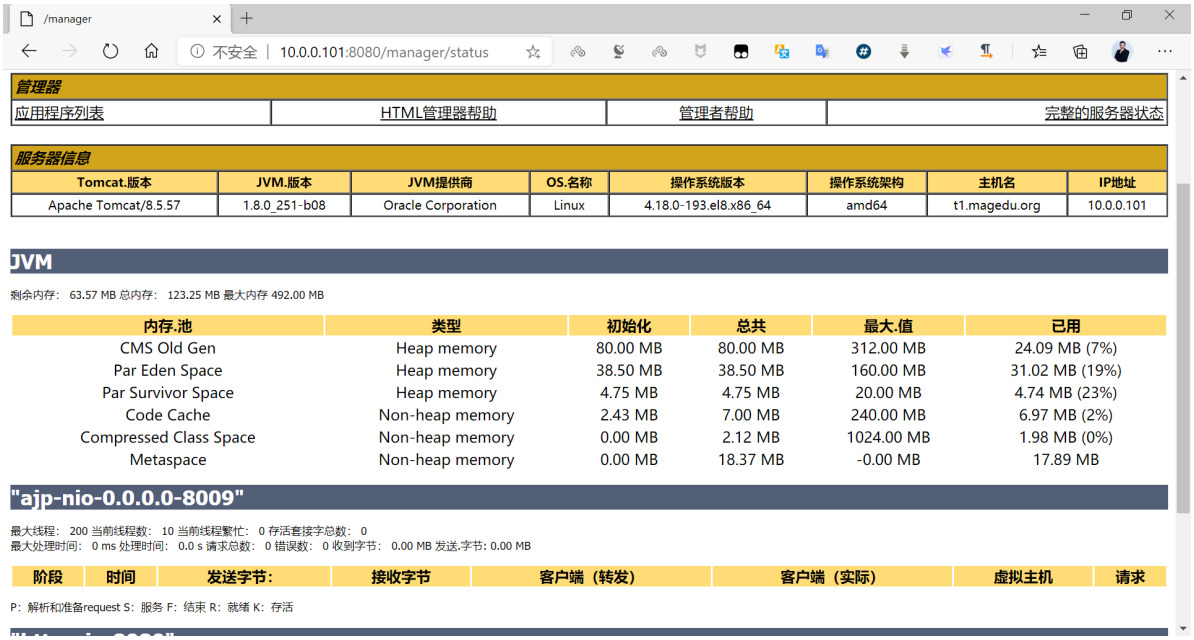
范例: 查看默认模式

```
[root@centos8 ~]#java  |& grep '-server'
    -server   to select the "server" VM
                  The default VM is server.
[root@centos8 ~]#tail -n 2 /usr/local/jdk/jre/lib/amd64/jvm.cfg
-server KNOWN
-client IGNORE
```

范例: 指定垃圾回收设置

```
#将参数加入到bin/catalina.sh中，重启观察Tomcat status。老年代已经使用CMS
[root@tomcat ~]#vim /usr/local/tomcat/bin/catalina.sh
......
# OS specific support.  $var _must_ be set to either true or false.

JAVA_OPTS="-server -Xmx512m -Xms128m -XX:NewSize=48m -XX:MaxNewSize=200m -
XX:+UseConcMarkSweepGC -XX:+UseCMSCompactAtFullCollection -
XX:CMSFullGCsBeforeCompaction=5"

cygwin=false
darwin=false
os400=false
.......
[root@tomcat ~]#systemctl restart tomcat
[root@tomcat ~]#ps aux |grep tomcat
```

```
tomcat     22093  2.2 15.2 3018616 148448 ?      Sl   17:51   0:06
/usr/local/jdk/bin/java -
Djava.util.logging.config.file=/usr/local/tomcat/conf/logging.properties -
Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -server -
Xmx512m -Xms128m -XX:NewSize=48m -XX:MaxNewSize=200m -XX:+UseConcMarkSweepGC -
XX:+UseCMSCompactAtFullCollection -XX:CMSFullGCsBeforeCompaction=5 -
Djdk.tls.ephemeralDHKeySize=2048 -
Djava.protocol.handler.pkgs=org.apache.catalina.webresources -
Dorg.apache.catalina.security.SecurityListener.UMASK=0027 -
Dignore.endorsed.dirs= -classpath
/usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar -
Dcatalina.base=/usr/local/tomcat -Dcatalina.home=/usr/local/tomcat -
Djava.io.tmpdir=/usr/local/tomcat/temp org.apache.catalina.startup.Bootstrap
start
root       22158  0.0  0.1  12108  1084 pts/1    S+   17:56   0:00 grep --
color=auto tomcat
```



开启垃圾回收输出统计信息,适用于调试环境的相关选项

- -XX:+PrintGC 输出GC信息
- -XX:+PrintGCDetails 输出GC详细信息
- -XX:+PrintGCTimeStamps 与前两个组合使用，在信息上加上一个时间戳
- -XX:+PrintHeapAtGC 生成GC前后椎栈的详细信息，日志会更大

注意: 以上适用调试环境，生产环境请**移除**这些参数，否则有非常多的日志输出

8.2.8 JAVA参数总结

参数名称	含义	默认值	
-Xms	初始堆大小	物理内存的1/64(<1GB)	默认(MinHeapFreeRatio参数可以调整)空余堆内存小于40%时，JVM就会增大堆直到-Xmx的最大限制.
-Xmx	最大堆大小	物理内存的1/4(<1GB)	默认(MaxHeapFreeRatio参数可以调整)空余堆内存大于70%时，JVM会减少堆直到 -Xms的最小限制
-Xmn	年轻代大小(1.4or lator)		**注意**：此处的大小是（eden+ 2 survivor space).与jmap -heap中显示的New gen是不同的。 整个堆大小=年轻代大小 + 年老代大小 + 持久代大小. 增大年轻代后,将会减小年老代大小.此值对系统性能影响较大,Sun官方推荐配置为整个堆的3/8
-XX:NewSize	设置年轻代大小(for 1.3/1.4)		
-XX:MaxNewSize	年轻代最大值(for 1.3/1.4)		
-XX:PermSize	设置持久代(perm gen)初始值	物理内存的1/64	
-XX:MaxPermSize	设置持久代最大值	物理内存的1/4	
-Xss	每个线程的堆栈大小		JDK5.0以后每个线程堆栈大小为1M,以前每个线程堆栈大小为256K.更具应用的线程所需内存大小进行 调整.在相同物理内存下,减小这个值能生成更多的线程.但是操作系统对一个进程内的线程数还是有限制的,不能无限生成,经验值在3000~5000左右 一般小的应用， 如果栈不是很深， 应该是128k够用的 大的应用建议使用256k。这个选项对性能影响比较大，需要严格的测试。

参数名称	含义	默认值	
-XX:ThreadStackSize	Thread Stack Size		(0 means use default stack size) [Sparc: 512; Solaris x86: 320 (was 256 prior in 5.0 and earlier); Sparc 64 bit: 1024; Linux amd64: 1024 (was 0 in 5.0 and earlier); all others 0.]
-XX:NewRatio	年轻代(包括Eden和两个Survivor区)与年老代的比值(除去持久代)		-XX:NewRatio=4表示年轻代与年老代所占比值为1:4,年轻代占整个堆栈的1/5 Xms=Xmx并且设置了Xmn的情况下，该参数不需要进行设置。
-XX:SurvivorRatio	Eden区与Survivor区的大小比值		设置为8,则两个Survivor区与一个Eden区的比值为2:8,一个Survivor区占整个年轻代的1/10
-XX:LargePageSizeInBytes	内存页的大小不可设置过大， 会影响Perm的大小		=128m
-XX:+UseFastAccessorMethods	原始类型的快速优化		
-XX:+DisableExplicitGC	关闭System.gc()		这个参数需要严格的测试
-XX:MaxTenuringThreshold	垃圾最大年龄		如果设置为0的话,则年轻代对象不经过Survivor区,直接进入年老代. 对于年老代比较多的应用,可以提高效率.如果将此值设置为一个较大值,则年轻代对象会在Survivor区进行多次复制,这样可以增加对象再年轻代的存活 时间,增加在年轻代即被回收的概率 该参数只有在串行GC时才有效
-XX:+AggressiveOpts	加快编译		
-XX:+UseBiasedLocking	锁机制的性能改善		
-Xnoclassgc	禁用垃圾回收		
-XX:SoftRefLRUPolicyMSPerMB	每兆堆空闲空间中SoftReference的存活时间		可达的对象在上次被引用后将保留一段时间。 缺省值是堆中每个空闲兆字节的生命周期的一秒钟

参数名称	含义	默认值	
-XX:PretenureSizeThreshold	对象超过多大是直接在旧生代分配	0	单位字节 新生代采用Parallel Scavenge GC时无效 另一种直接在旧生代分配的情况是大的数组对象,且数组中无外部引用对象.
-XX:TLABWasteTargetPercent	TLAB占eden区的百分比	1%	
-XX:+*CollectGen0First*	FullGC时是否先YGC	false	

并行收集器相关参数

-XX:+UseParallelGC	**Full GC采用parallel MSC**		**选择垃圾收集器为并行收集器.此配置仅对年轻代有效.即上述配置下,年轻代使用并发收集,而年老代仍旧使用串行收集**
-XX:+UseParNewGC	设置年轻代为并行收集		可与CMS收集同时使用 JDK5.0以上,JVM会根据系统配置自行设置,所以无需再设置此值
-XX:ParallelGCThreads	并行收集器的线程数		此值最好配置与处理器数目相等 同样适用于CMS
-XX:+UseParallelOldGC	年老代垃圾收集方式为并行收集(Parallel Compacting)		这个是JAVA 6出现的参数选项
-XX:MaxGCPauseMillis	每次年轻代垃圾回收的最长时间(最大暂停时间)		如果无法满足此时间,JVM会自动调整年轻代大小,以满足此值.
-XX:+UseAdaptiveSizePolicy	自动选择年轻代区大小和相应的Survivor区比例		设置此选项后,并行收集器会自动选择年轻代区大小和相应的Survivor区比例,以达到目标系统规定的最低相应时间或者收集频率等,此值建议使用并行收集器时,一直打开.
-XX:GCTimeRatio	设置垃圾回收时间占程序运行时间的百分比		公式为1/(1+n)
-XX:+ScavengeBeforeFullGC	Full GC前调用YGC	true	

CMS相关参数

-XX:+UseConcMarkSweepGC	**使用CMS内存收集**		**测试中配置这个以后,-XX:NewRatio=4的配置可能失效,所以,此时年轻代大小最好用-Xmn设置**
-XX:+AggressiveHeap			试图是使用大量的物理内存 长时间大内存使用的优化，能检查计算资源（内存， 处理器数量） 至少需要256MB内存 大量的CPU / 内存， （在1.4.1在4CPU的机器上已经显示有提升）
-XX:CMSFullGCsBeforeCompaction	多少次后进行内存压缩		由于并发收集器不对内存空间进行压缩,整理,所以运行一段时间以后会产生"碎片",使得运行效率降低.此值设置运行多少次GC以后对内存空间进行压缩,整理.
-XX:+CMSParallelRemarkEnabled	降低标记停顿		
-XX+UseCMSCompactAtFullCollection	在FULL GC的时候，对年老代的压缩		CMS是不会移动内存的， 因此， 这个非常容易产生碎片，导致内存不够用， 因此， 内存的压缩这个时候就会被启用。 增加这个参数是个好习惯。 可能会影响性能,但是可以消除碎片
-XX:+UseCMSInitiatingOccupancyOnly	使用手动定义初始化定义开始CMS收集		禁止hostspot自行触发CMS GC
-XX:CMSInitiatingOccupancyFraction=70	使用cms作为垃圾回收使用70%后开始CMS收集	92	

-XX:+UseConcMarkSweepGC	**使用CMS内存收集**		**测试中配置这个以后,-XX:NewRatio=4的配置可能失效,所以,此时年轻代大小最好用-Xmn设置**
-XX:CMSInitiatingPermOccupancyFraction	设置PermGen使用到达多少比率时触发	92	
-XX:+CMSIncrementalMode	设置为增量模式		用于单CPU情况
-XX:+CMSClassUnloadingEnabled			

辅助信息

-XX:+PrintGC			**输出形式:[GC 118250K->113543K(130112K), 0.0094143 secs] [Full GC 121376K->10414K(130112K), 0.0650971 secs]**
-XX:+PrintGCDetails			输出形式:[GC [DefNew: 8614K->781K(9088K), 0.0123035 secs] 118250K->113543K(130112K), 0.0124633 secs] [GC [DefNew: 8614K->8614K(9088K), 0.0000665 secs] 121376K->10414K(130112K), 0.0436268 secs]
-XX:+PrintGCTimeStamps			
-XX:+PrintGC:PrintGCTimeStamps			可与-XX:+PrintGC -XX:+PrintGCDetails混合使用 输出形式:11.851: [GC 98328K->93620K(130112K), 0.0082960 secs]
-XX:+PrintGCApplicationStoppedTime	打印垃圾回收期间程序暂停的时间.可与上面混合使用		输出形式:Total time for which application threads were stopped: 0.0468229 seconds
-XX:+PrintGCApplicationConcurrentTime	打印每次垃圾回收前,程序未中断的执行时间.可与上面混合使用		输出形式:Application time: 0.5291524 seconds
-XX:+PrintHeapAtGC	打印GC前后的详细堆栈信息		
-Xloggc:filename	把相关日志信息记录到文件以便分析. 与上面几个配合使用		

-XX:+PrintGC			**输出形式:[GC 118250K->113543K(130112K), 0.0094143 secs] [Full GC 121376K->10414K(130112K), 0.0650971 secs]**
-XX:+PrintClassHistogram	garbage collects before printing the histogram.		
-XX:+PrintTLAB	查看TLAB空间的使用情况		
XX:+PrintTenuringDistribution	查看每次minor GC后新的存活周期的阈值		

8.3 JVM相关工具

8.3.1 JVM 工具概述

$JAVA_HOME/bin下

命令	说明
jps	查看所有jvm进程
jinfo	查看进程的运行环境参数，主要是jvm命令行参数
jstat	对jvm应用程序的资源和性能进行实时监控
jstack	查看所有线程的运行状态
jmap	查看jvm占用物理内存的状态
jhat	+UseParNew
jconsole	图形工具
jvisualvm	图形工具

8.3.2 jps

JVM 进程状态工具

格式

```
jps：Java virutal machine Process Status tool，
jps [-q] [-mlvV] [<hostid>]
-q：静默模式；
-v：显示传递给jvm的命令行参数；
-m：输出传入main方法的参数；
-l：输出main类或jar完全限定名称；
-v：显示通过flag文件传递给jvm的参数；
[<hostid>]：主机id，默认为localhost；
```

范例:

```
#显示java进程
[root@tomcat ~]#jps
22357 Jps
21560 Main
21407 HelloWorld

#详细列出当前Java进程信息
[root@tomcat ~]#jps -l -v
21560 org.netbeans.Main -Djdk.home=/usr/local/jdk -
Dnetbeans.default_userdir_root=/root/.visualvm -
Dnetbeans.dirs=/usr/local/jdk/lib/visualvm/visualvm:/usr/local/jdk/lib/visualvm/
profiler: -Dnetbeans.home=/usr/local/jdk/lib/visualvm/platform -Xms24m -Xmx256m
-Dsun.jvmstat.perdata.syncWaitMs=10000 -Dsun.java2d.noddraw=true -
Dsun.java2d.d3d=false -Dnetbeans.keyring.no.master=true -
Dplugin.manager.install.global=false --add-exports=java.desktop/sun.awt=ALL-
UNNAMED --add-exports=jdk.jvmstat/sun.jvmstat.monitor.event=ALL-UNNAMED --add-
exports=jdk.jvmstat/sun.jvmstat.monitor=ALL-UNNAMED --add-
exports=java.desktop/sun.swing=ALL-UNNAMED --add-
exports=jdk.attach/sun.tools.attach=ALL-UNNAMED --add-modules=java.activation -
XX:+IgnoreUnrecognizedVMOptions -XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=/root/.visualvm/8u131/var/log/heapdump.hprof
22270 sun.tools.jps.Jps -
Denv.class.path=/usr/local/jdk/lib/:/usr/local/jdk/jre/lib/ -
Dapplication.home=/usr/local/jdk1.8.0_241 -Xms8m
21407 HelloWorld -Xms256m -Xmx512m
```

8.3.3 jinfo

输出给定的java进程的所有配置信息

格式:

```
jinfo [option] <pid>
-flags：打印 VM flags
-sysprops：to print Java system properties
-flag <name>：to print the value of the named VM flag
```

范例:

```
#先获得一个java进程ID，然后jinfo
[root@tomcat ~]#jps
22357 Jps
21560 Main
```

```
21407 HelloWorld

[root@tomcat ~]#jinfo 21407
Attaching to process ID 21407, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.241-b07
Java System Properties:

java.runtime.name = Java(TM) SE Runtime Environment
java.vm.version = 25.241-b07
sun.boot.library.path = /usr/local/jdk1.8.0_241/jre/lib/amd64
java.vendor.url = http://java.oracle.com/
java.vm.vendor = Oracle Corporation
path.separator = :
java.rmi.server.randomIDs = true
file.encoding.pkg = sun.io
java.vm.name = Java HotSpot(TM) 64-Bit Server VM
sun.os.patch.level = unknown
sun.java.launcher = SUN_STANDARD
user.country = US
user.dir = /data
java.vm.specification.name = Java Virtual Machine Specification
java.runtime.version = 1.8.0_241-b07
java.awt.graphicsenv = sun.awt.X11GraphicsEnvironment
os.arch = amd64
java.endorsed.dirs = /usr/local/jdk1.8.0_241/jre/lib/endorsed
java.io.tmpdir = /tmp
line.separator =

java.vm.specification.vendor = Oracle Corporation
os.name = Linux
sun.jnu.encoding = UTF-8
java.library.path = /usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib
java.specification.name = Java Platform API Specification
java.class.version = 52.0
sun.management.compiler = HotSpot 64-Bit Tiered Compilers
os.version = 4.18.0-147.el8.x86_64
user.home = /root
user.timezone = Asia/Shanghai
java.awt.printerjob = sun.print.PSPrinterJob
file.encoding = UTF-8
java.specification.version = 1.8
user.name = root
java.class.path = /usr/local/jdk/lib/:/usr/local/jdk/jre/lib/
java.vm.specification.version = 1.8
sun.arch.data.model = 64
sun.java.command = HelloWorld
java.home = /usr/local/jdk1.8.0_241/jre
user.language = en
java.specification.vendor = Oracle Corporation
awt.toolkit = sun.awt.X11.XToolkit
java.vm.info = mixed mode
java.version = 1.8.0_241
java.ext.dirs = /usr/local/jdk1.8.0_241/jre/lib/ext:/usr/java/packages/lib/ext
```

```
sun.boot.class.path =
/usr/local/jdk1.8.0_241/jre/lib/resources.jar:/usr/local/jdk1.8.0_241/jre/lib/rt
.jar:/usr/local/jdk1.8.0_241/jre/lib/sunrsasign.jar:/usr/local/jdk1.8.0_241/jre/
lib/jsse.jar:/usr/local/jdk1.8.0_241/jre/lib/jce.jar:/usr/local/jdk1.8.0_241/jre
/lib/charsets.jar:/usr/local/jdk1.8.0_241/jre/lib/jfr.jar:/usr/local/jdk1.8.0_24
1/jre/classes
java.vendor = Oracle Corporation
file.separator = /
java.vendor.url.bug = http://bugreport.sun.com/bugreport/
sun.io.unicode.encoding = UnicodeLittle
sun.cpu.endian = little
sun.cpu.isalist =

VM Flags:
Non-default VM flags: -XX:CICompilerCount=2 -XX:InitialHeapSize=268435456 -
XX:MaxHeapSize=536870912 -XX:MaxNewSize=178913280 -XX:MinHeapDeltaBytes=196608 -
XX:NewSize=89456640 -XX:OldSize=178978816 -XX:+UseCompressedClassPointers -
XX:+UseCompressedOops -XX:+UseFastUnorderedTimeStamps
Command line:  -Xms256m -Xmx512m

[root@tomcat ~]#jinfo -flags 21407
Attaching to process ID 21407, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.241-b07
Non-default VM flags: -XX:CICompilerCount=2 -XX:InitialHeapSize=268435456 -
XX:MaxHeapSize=536870912 -XX:MaxNewSize=178913280 -XX:MinHeapDeltaBytes=196608 -
XX:NewSize=89456640 -XX:OldSize=178978816 -XX:+UseCompressedClassPointers -
XX:+UseCompressedOops -XX:+UseFastUnorderedTimeStamps
Command line:  -Xms256m -Xmx512m
```

8.3.4 jstat

输出指定的java进程的统计信息

格式:

```
jstat -help|-options
jstat -<option> [-t] [-h<lines>] <vmid> [<interval> [<count>]]
[<interval> [<count>]]
interval：时间间隔，单位是毫秒；
count：显示的次数；

#返回可用统计项列表
# jstat -options
    -class：class loader
    -compiler：JIT
    -gc：gc
    -gccapacity：统计堆中各代的容量
    -gccause：
    -gcmetacapacity
    -gcnew：新生代
    -gcnewcapacity
    -gcold：老年代
    -gcoldcapacity
    -gcutil
```

```
    -printcompilation


SOC: Current survivor space 0 capacity (kB).
S1C: Current survivor space 1 capacity (kB).
SOU: Survivor space 0 utilization (kB).
S1U: Survivor space 1 utilization (kB).
EC: Current eden space capacity (kB).
EU: Eden space utilization (kB).
OC: Current old space capacity (kB).
OU: Old space utilization (kB).
MC: Metaspace capacity (kB).
MU: Metacspace utilization (kB).
CCSC: Compressed class space capacity (kB).
CCSU: Compressed class space used (kB).
YGC: Number of young generation garbage collection events.
YGCT: Young generation garbage collection time.
FGC: Number of full GC events.
FGCT: Full garbage collection time.
GCT: Total garbage collection time.

TT: Tenuring threshold.
MTT: Maximum tenuring threshold.
DSS: Desired survivor size (kB).
```

范例:

```
[root@tomcat ~]#jstat -gc 21407
 SOC     S1C     SOU     S1U       EC        EU         OC          OU        MC      MU
CCSC   CCSU   YGC     YGCT    FGC    FGCT     GCT
8704.0 8704.0 1563.6  0.0   69952.0  58033.0   174784.0     0.0     9344.0
8830.0 1152.0 1013.0      2    0.050   0      0.000    0.050

SOC:S0区容量
YGC：新生代的垃圾回收次数
YGCT：新生代垃圾回收消耗的时长；
FGC：Full GC的次数
FGCT：Full GC消耗的时长
GCT：GC消耗的总时长

#3次，一秒一次
[root@tomcat ~]#jstat -gcnew 21407 1000 3
 SOC     S1C     SOU     S1U   TT MTT  DSS      EC        EU     YGC     YGCT
8704.0 8704.0 1563.6    0.0 15  15 4352.0  69952.0  62794.3       2    0.050
8704.0 8704.0 1563.6    0.0 15  15 4352.0  69952.0  62794.3       2    0.050
8704.0 8704.0 1563.6    0.0 15  15 4352.0  69952.0  63074.5       2    0.050
```

8.3.5 jstack

程序员常用堆栈情况查看工具

jstack：查看指定的java进程的线程栈的相关信息

格式:

```
jstack [-l] <pid>
jstack -F [-m] [-l] <pid>
-l：long listings，会显示额外的锁信息，因此，发生死锁时常用此选项
-m：混合模式，既输出java堆栈信息，也输出C/C++堆栈信息
-F：当使用"jstack -l PID"无响应，可以使用-F强制输出信息
```

范例:

```
#先获得一个java进程PID，然后jinfo
[root@tomcat ~]#jstack -l 21407
2020-02-15 13:49:56
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.241-b07 mixed mode):

"RMI TCP Connection(4)-10.0.0.101" #16 daemon prio=9 os_prio=0
tid=0x00007f279c29e800 nid=0x5753 runnable [0x00007f279b181000]
   java.lang.Thread.State: RUNNABLE
    at java.net.SocketInputStream.socketRead0(Native Method)
......
```

8.3.6 jmap

Memory Map, 用于查看堆内存的使用状态

```
#查看进程堆内存情况
[root@tomcat ~]#jmap -heap 21407
Attaching to process ID 21407, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.241-b07

using thread-local object allocation.
Mark Sweep Compact GC

Heap Configuration:
   MinHeapFreeRatio         = 40
   MaxHeapFreeRatio         = 70
   MaxHeapSize              = 536870912 (512.0MB)
   NewSize                  = 89456640 (85.3125MB)
   MaxNewSize               = 178913280 (170.625MB)
   OldSize                  = 178978816 (170.6875MB)
   NewRatio                 = 2
   SurvivorRatio            = 8
   MetaspaceSize            = 21807104 (20.796875MB)
   CompressedClassSpaceSize = 1073741824 (1024.0MB)
   MaxMetaspaceSize         = 17592186044415 MB
   G1HeapRegionSize         = 0 (0.0MB)

Heap Usage:
New Generation (Eden + 1 Survivor Space):
   capacity = 80543744 (76.8125MB)
   used     = 4893096 (4.666419982910156MB)
   free     = 75650648 (72.14608001708984MB)
   6.075078903707283% used
Eden Space:
   capacity = 71630848 (68.3125MB)
```

```
   used     = 3301800 (3.1488418579101562MB)
   free     = 68329048 (65.16365814208984MB)
   4.609466580655306% used
From Space:
   capacity = 8912896 (8.5MB)
   used     = 1591296 (1.517578125MB)
   free     = 7321600 (6.982421875MB)
   17.85386029411765% used
To Space:
   capacity = 8912896 (8.5MB)
   used     = 0 (0.0MB)
   free     = 8912896 (8.5MB)
   0.0% used
tenured generation:
   capacity = 178978816 (170.6875MB)
   used     = 0 (0.0MB)
   free     = 178978816 (170.6875MB)
   0.0% used

4926 interned Strings occupying 389832 bytes.
```

8.3.7 jhat

Java Heap Analysis Tool 堆分析工具

格式

```
jmap [option] <pid>
#查看堆空间的详细信息：
jmap -heap <pid>
```

范例:

```
[root@t1 ~]#jmap -heap 96334
Attaching to process ID 96334, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.251-b08

using thread-local object allocation.
Mark Sweep Compact GC

Heap Configuration:
   MinHeapFreeRatio         = 40
   MaxHeapFreeRatio         = 70
   MaxHeapSize              = 251658240 (240.0MB)
   NewSize                  = 5570560 (5.3125MB)
   MaxNewSize               = 83886080 (80.0MB)
   OldSize                  = 11206656 (10.6875MB)


#查看堆内存中的对象的数目：
jmap -histo[:live] <pid>
        live：只统计活动对象；

#保存堆内存数据至文件中，而后使用jvisualvm或jhat进行查看：
```

```
jmap -dump:<dump-options> <pid>
    dump-options:
    live   dump only live objects; if not specified, all objects in the heap are
dumped.
    format=b     binary format
    file=<file>  dump heap to <file>
```

8.3.8 jconsole 和 JMX

图形化工具,可以用来查看Java进程信息

JMX（Java Management Extensions，即Java管理扩展）是一个为JAVA应用程序、设备、系统等植入管理功能的框架。JMX可以跨越一系列异构操作系统平台、系统体系结构和网络传输协议，灵活的开发无缝集成的系统、网络和服务管理应用。

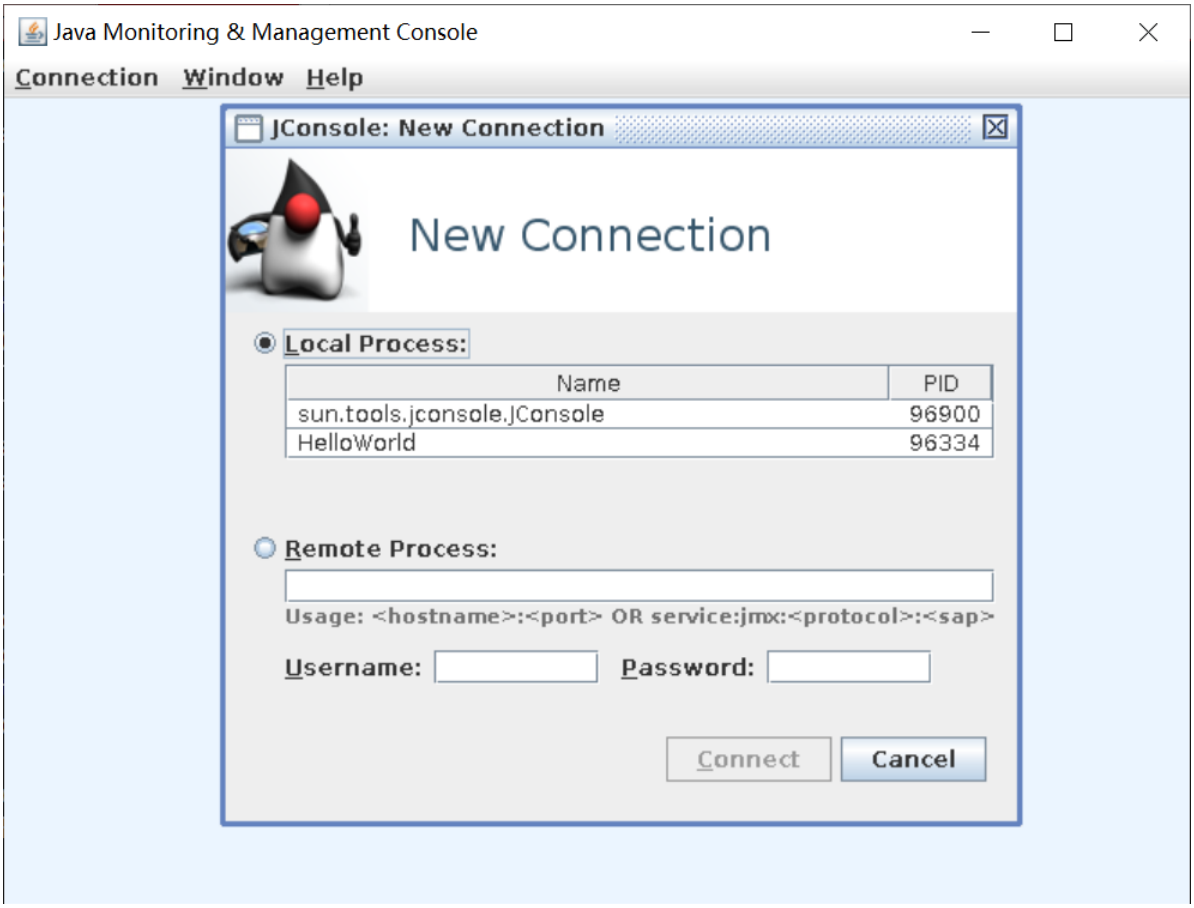
JMX最常见的场景是监控Java程序的基本信息和运行情况，任何Java程序都可以开启JMX，然后使用JConsole或Visual VM进行预览。

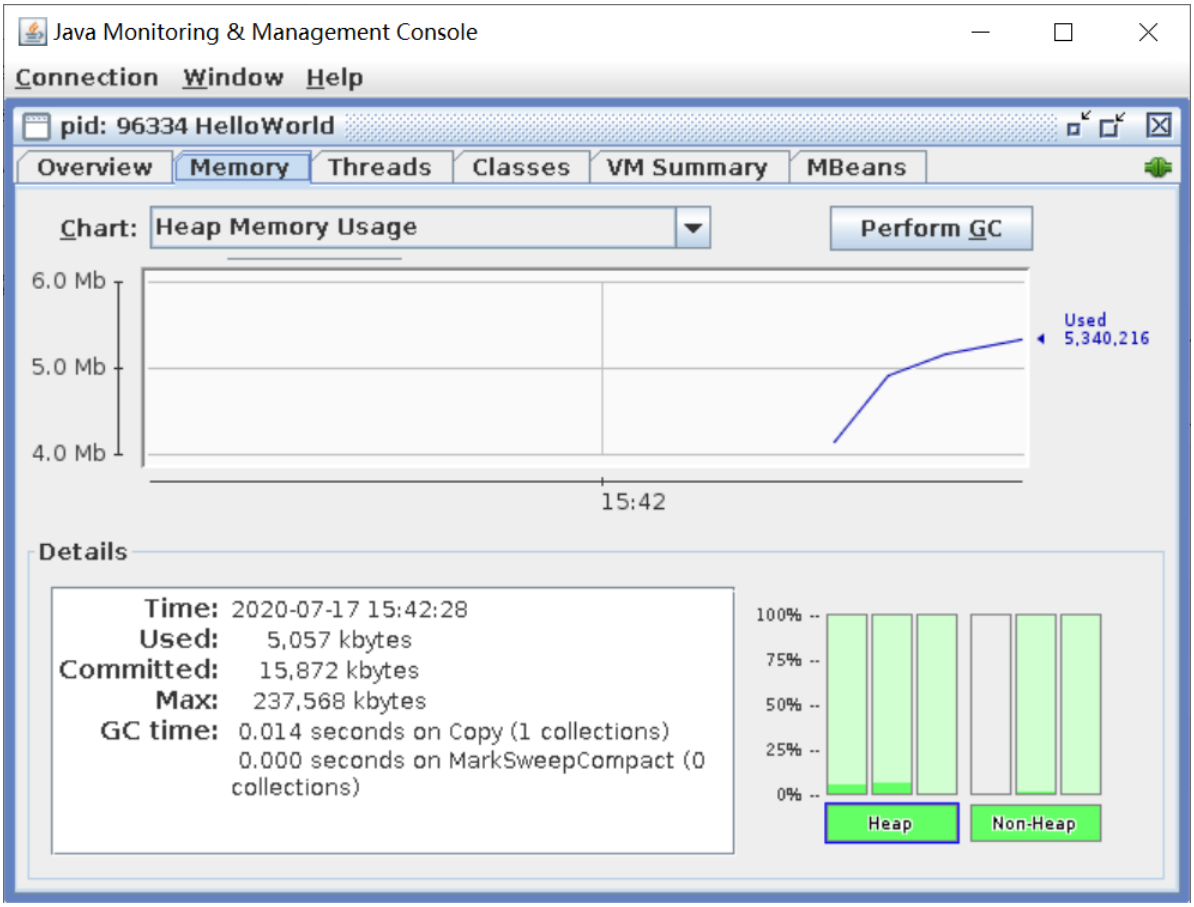
```
#为Java程序开启JMX很简单，只要在运行Java程序的命令后面指定如下命令即可
java \
-Dcom.sun.management.jmxremote           \     #启用远程监控JMX
-Djava.rmi.server.hostname=10.0.0.100    \  #指定自己监听的IP
-Dcom.sun.management.jmxremote.port=12345 \  #指定监听的PORT
-Dcom.sun.management.jmxremote.authenticate=false \
-Dcom.sun.management.jmxremote.ssl=false \
-jar app.jar|app.war
```

在 tomcat 开启远程 JMX 支持 Zabbix 监控，如下配置

```
vim /usr/local/tomcat/bin/catalina.sh
CATALINA_OPTS="$CATALINA_OPTS
-Dcom.sun.management.jmxremote                          #启用远程监控JMX
-Dcom.sun.management.jmxremote.port=XXXXX               #默认启动的JMX端口号，要和
zabbix添加主机时候的端口一致即可
-Dcom.sun.management.jmxremote.authenticate=false      #不使用用户名密码
-Dcom.sun.management.jmxremote.ssl=false               #不使用ssl认证
-Djava.rmi.server.hostname=<JAVA主机IP>"                #tomcat主机自己的IP地址,不要写
zabbix服务器的地址
```

下图是使用Jconsle通过JMX查看Java程序的运行信息

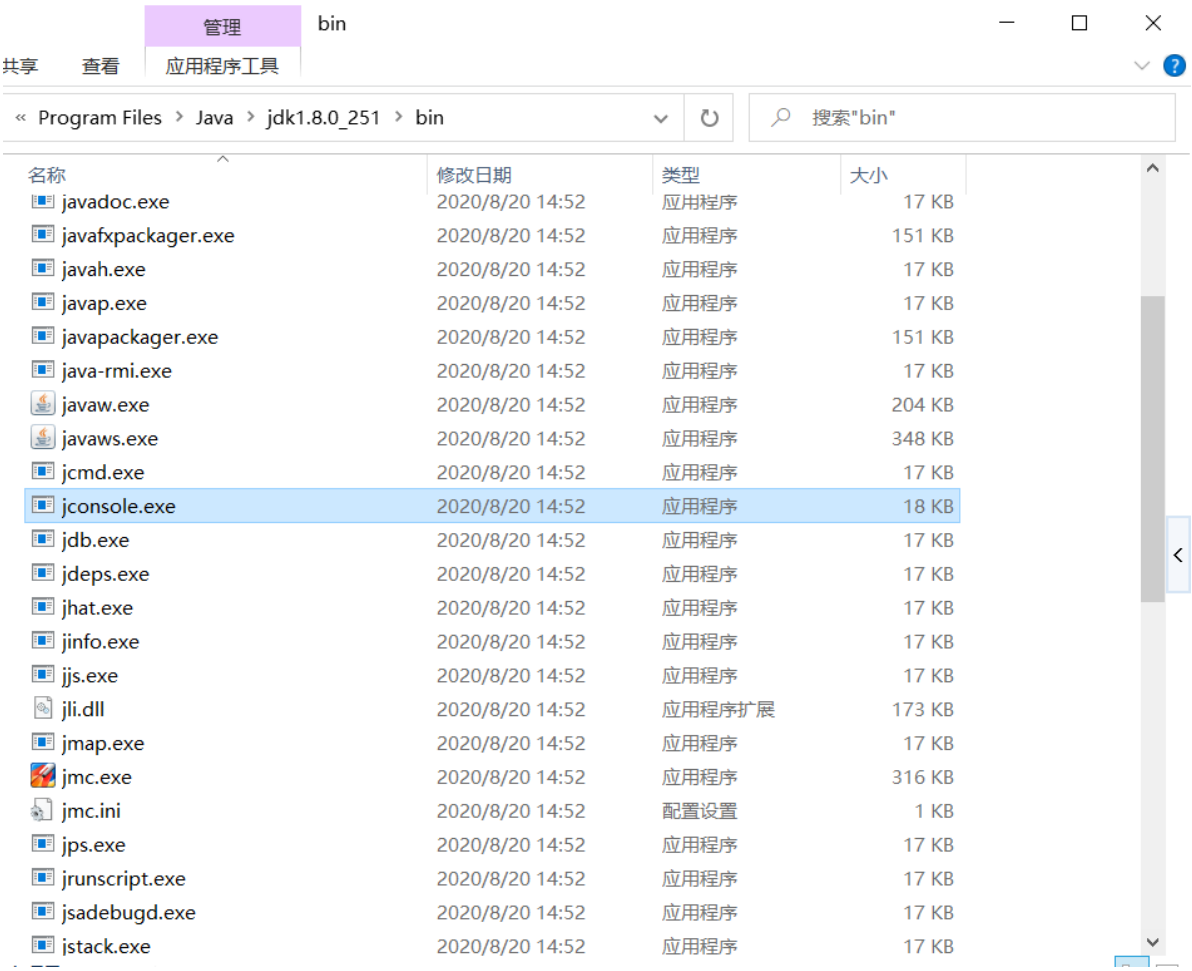


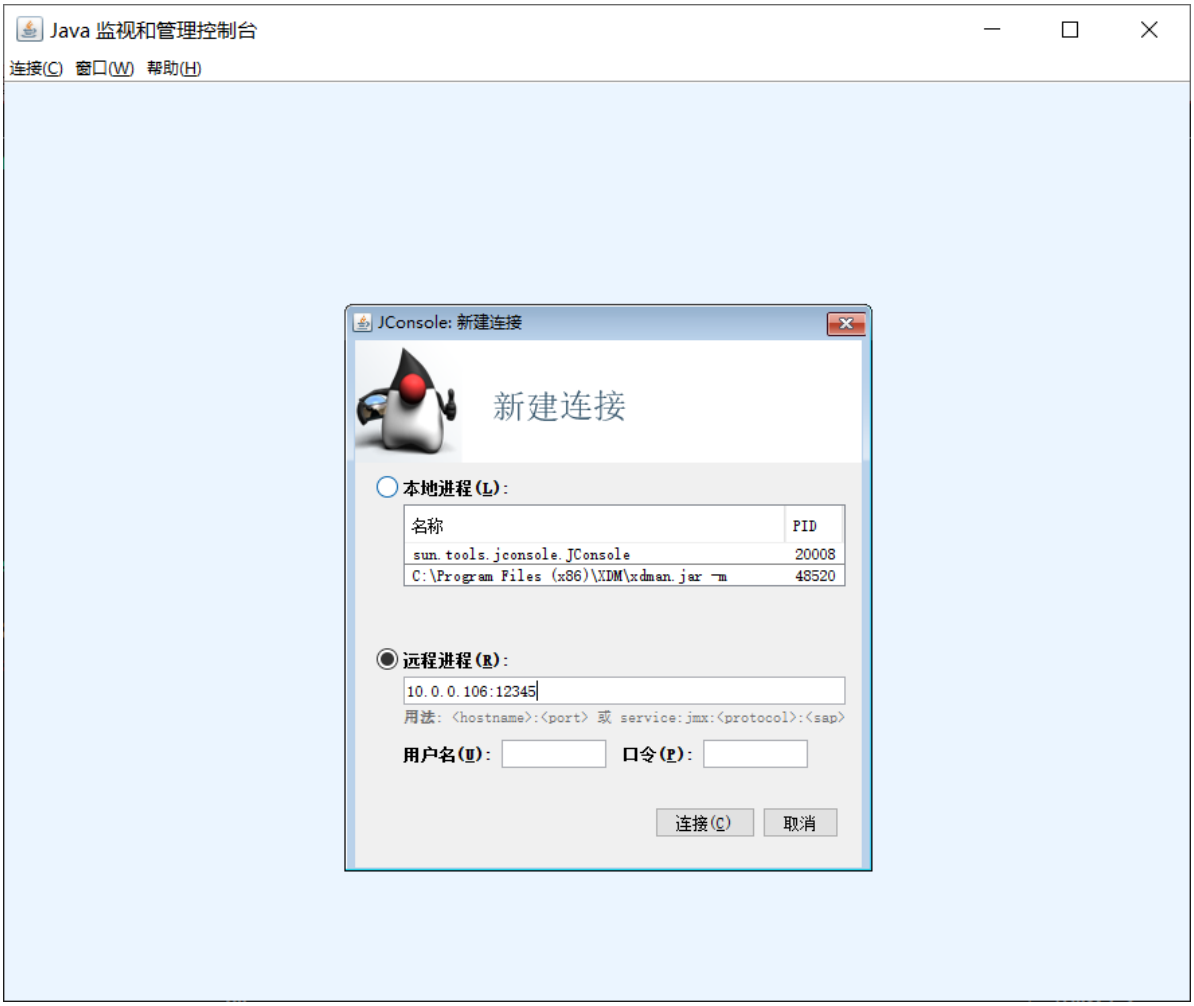


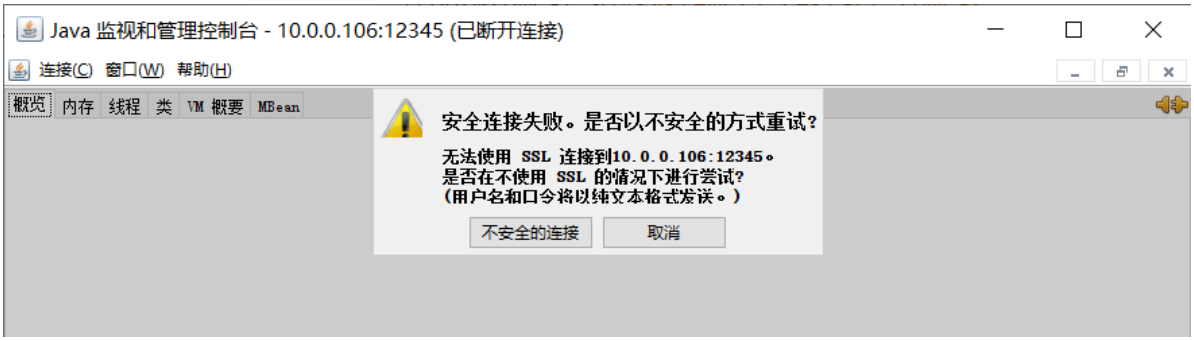
范例: 开启远程JMX功能

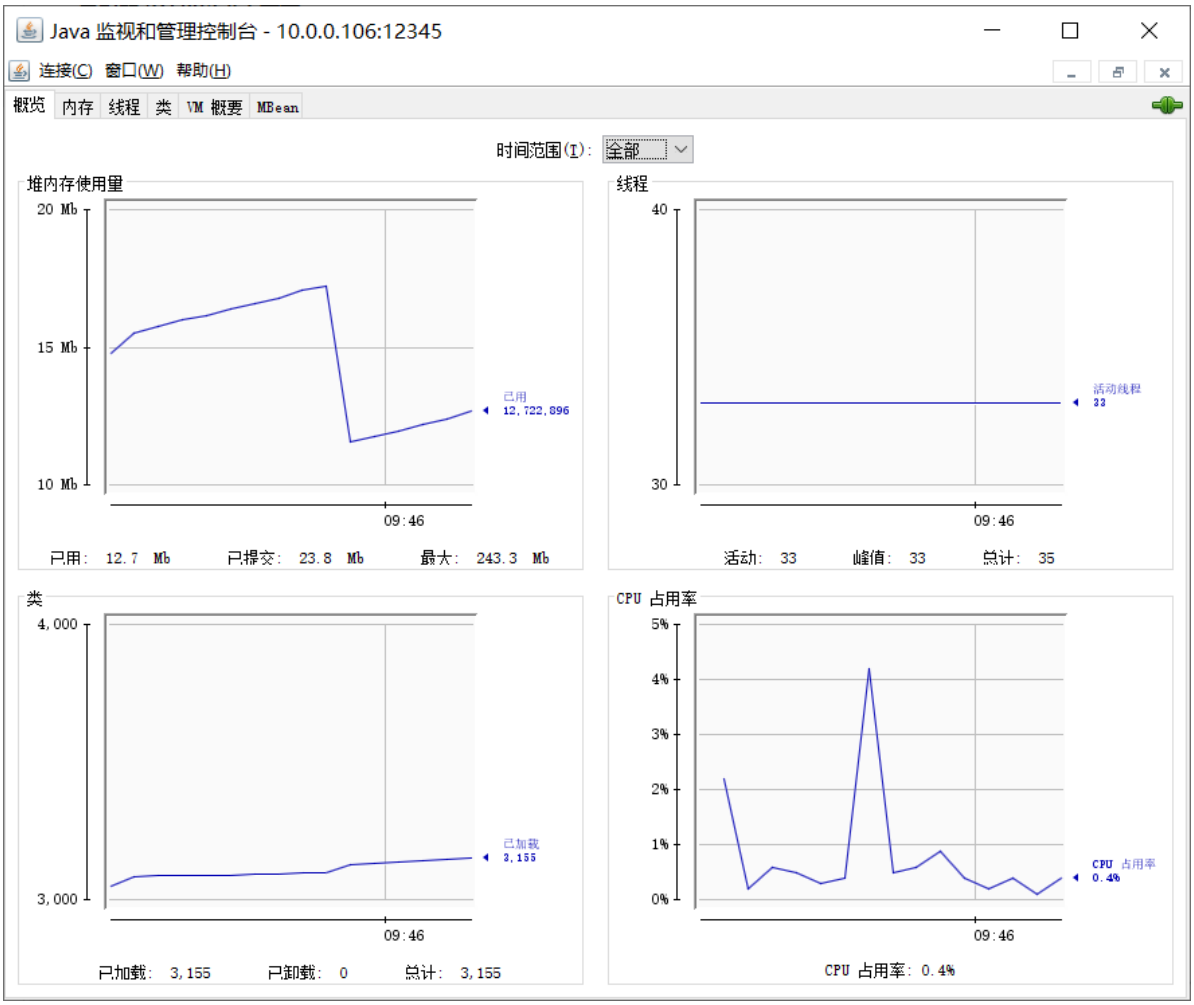
```
[root@tomcat ~]#vim /usr/local/tomcat/bin/catalina.sh
.......
# ----------------------------------------------------------------------------
CATALINA_OPTS="$CATALINA__OPTS \
```

```
-Dcom.sun.management.jmxremote \
-Djava.rmi.server.hostname=10.0.0.101 \
-Dcom.sun.management.jmxremote.port=12345 \
-Dcom.sun.management.jmxremote.authenticate=false \
-Dcom.sun.management.jmxremote.ssl=false"

# OS specific support.  $var _must_ be set to either true or false.
.......

[root@tomcat ~]#systemctl restart tomcat
[root@tomcat ~]#ss -ntl
State               Recv-Q              Send-Q        Local Address:Port    Peer
Address:Port
LISTEN              0                   128                 0.0.0.0:22
 0.0.0.0:*
LISTEN              0                   100               127.0.0.1:25
 0.0.0.0:*
LISTEN              0                   128                    [::]:22
  [::]:*
LISTEN              0                   50                        *:12345
      *:*
LISTEN              0                   100                   [::1]:25
  [::]:*
LISTEN              0                   50                        *:38563
      *:*
LISTEN              0                   1        [::ffff:127.0.0.1]:8005
     *:*
LISTEN              0                   50                        *:40613
      *:*
LISTEN              0                   100                       *:8080
     *:*
```


管理
bin
共享
查看
应用程序工具
Program Files › Java › jdk1.8.0_251 › bin
搜索"bin"
名称 修改日期 类型 大小
javadoc.exe 2020/8/20 14:52 应用程序 17 KB
javafxpackager.exe 2020/8/20 14:52 应用程序 151 KB
javah.exe 2020/8/20 14:52 应用程序 17 KB
javap.exe 2020/8/20 14:52 应用程序 17 KB
javapackager.exe 2020/8/20 14:52 应用程序 151 KB
java-rmi.exe 2020/8/20 14:52 应用程序 17 KB
javaw.exe 2020/8/20 14:52 应用程序 204 KB
javaws.exe 2020/8/20 14:52 应用程序 348 KB
jcmd.exe 2020/8/20 14:52 应用程序 17 KB
jconsole.exe 2020/8/20 14:52 应用程序 18 KB
jdb.exe 2020/8/20 14:52 应用程序 17 KB
jdeps.exe 2020/8/20 14:52 应用程序 17 KB
jhat.exe 2020/8/20 14:52 应用程序 17 KB
jinfo.exe 2020/8/20 14:52 应用程序 17 KB
jjs.exe 2020/8/20 14:52 应用程序 17 KB
jli.dll 2020/8/20 14:52 应用程序扩展 173 KB
jmap.exe 2020/8/20 14:52 应用程序 17 KB
jmc.exe 2020/8/20 14:52 应用程序 316 KB
jmc.ini 2020/8/20 14:52 配置设置 1 KB
jps.exe 2020/8/20 14:52 应用程序 17 KB
jrunscript.exe 2020/8/20 14:52 应用程序 17 KB
jsadebugd.exe 2020/8/20 14:52 应用程序 17 KB
jstack.exe 2020/8/20 14:52 应用程序 17 KB


Java 监视和管理控制台
连接(C) 窗口(W) 帮助(H)
JConsole: 新建连接
新建连接
本地进程(L):
名称 PID
sun.tools.jconsole.JConsole 20008
C:\Program Files (x86)\XDM\xdman.jar -m 48520
远程进程(R):
10.0.0.106:12345
用法: <hostname>:<port> 或 service:jmx:<protocol>:<sap>
用户名(U): 口令(P):
连接(C) 取消





8.4 Tomcat 性能优化常用配置

8.4.1 内存空间优化

```
JAVA_OPTS="-server -Xms4g -Xmx4g -XX:NewSize= -XX:MaxNewSize= "

-server：服务器模式
-Xms：堆内存初始化大小
-Xmx：堆内存空间上限
-XX:NewSize=：新生代空间初始化大小
-XX:MaxNewSize=：新生代空间最大值
```

生产案例：

```
[root@centos8 ~]#vim /usr/local/tomcat/bin/catalina.sh
JAVA_OPTS="-server -Xms4g -Xmx4g -Xss512k -Xmn1g -
XX:CMSInitiatingOccupancyFraction=65 -XX:+AggressiveOpts -XX:+UseBiasedLocking -
XX:+DisableExplicitGC -XX:MaxTenuringThreshold=10 -XX:NewRatio=2 -
XX:PermSize=128m -XX:MaxPermSize=512m -XX:CMSFullGCsBeforeCompaction=5 -
XX:+ExplicitGCInvokesConcurrent -XX:+UseConcMarkSweepGC -XX:+UseParNewGC -
XX:+CMSParallelRemarkEnabled -XX:+UseCMSCompactAtFullCollection -
XX:LargePageSizeInBytes=128m -XX:+UseFastAccessorMethods"

#一台tomcat服务器并发连接数不高,生产建议分配物理内存通常4G到8G较多,如果需要更多连接,一般会利用虚拟化技术实现多台tomcat
```

8.4.2 线程池调整

```
[root@centos8 ~]#vim /usr/local/tomcat/conf/server.xml
......
<Connector port="8080" protocol="HTTP/1.1"  connectionTimeout="20000"
redirectPort="8443" />
......
```

常用属性：

- connectionTimeout： 连接超时时长,单位ms
- maxThreads：最大线程数，默认200
- minSpareThreads：最小空闲线程数
- maxSpareThreads：最大空闲线程数
- acceptCount：当启动线程满了之后，等待队列的最大长度，默认100
- URIEncoding：URI 地址编码格式，建议使用 UTF-8
- enableLookups：是否启用客户端主机名的DNS反向解析，缺省禁用，建议禁用，就使用客户端IP就行
- compression：是否启用传输压缩机制，建议 "on"，CPU和流量的平衡
  - compressionMinSize：启用压缩传输的数据流最小值，单位是字节
  - compressableMimeType：定义启用压缩功能的MIME类型text/html, text/xml, text/css, text/javascript

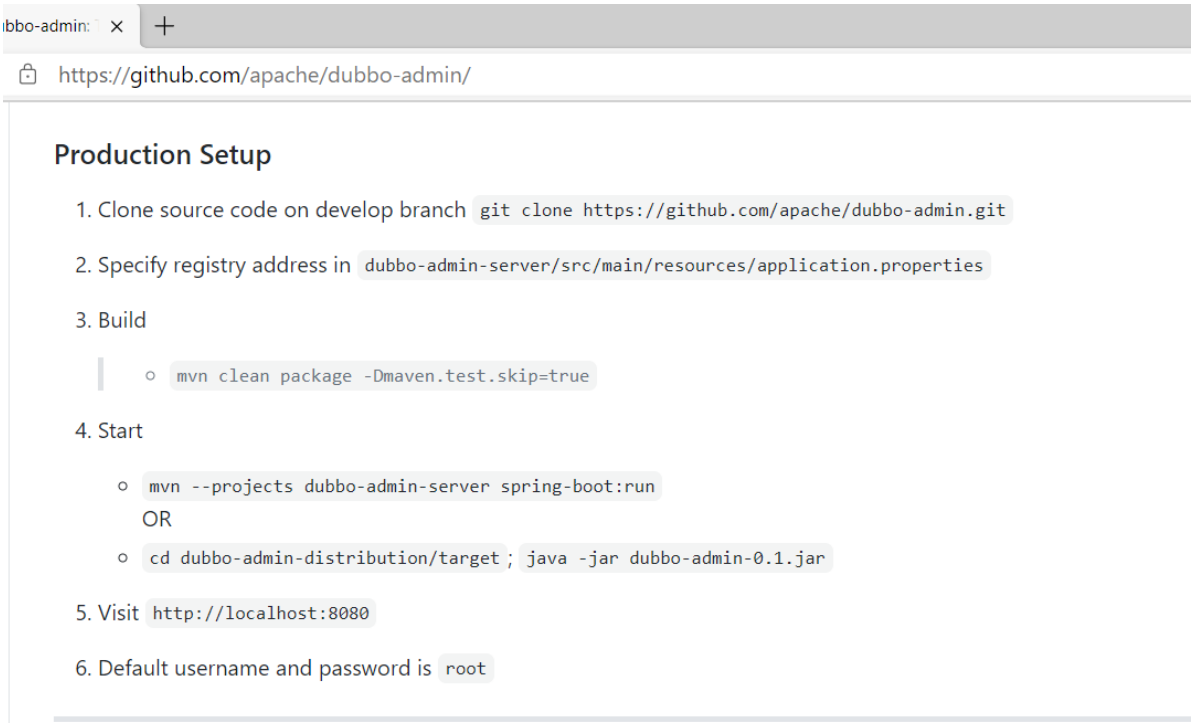
9 Java 程序编译

9.1 源代码克隆与编译过程说明

以github 上 java 开源项目dubbo-admin 为例

github 说明:

```
https://github.com/apache/dubbo-admin/
```



官方部署过程

```
Production Setup
Clone source code on develop branch git clone https://github.com/apache/dubbo-
admin.git

Specify registry address in dubbo-admin-
server/src/main/resources/application.properties

Build

mvn clean package -Dmaven.test.skip=true
Start

mvn --projects dubbo-admin-server spring-boot:run
OR
cd dubbo-admin-distribution/target; java -jar dubbo-admin-0.1.jar
Visit http://localhost:8080

Default username and password is root
```

9.2 maven 部署准备

9.2.1 maven 介绍

Maven 翻译为"专家"、"内行"，是 Apache 基金会旗下的一个纯 Java 开发的开源项目，Maven 是一个项目管理工具，可以对 Java项目进行构建、解决打包依赖等。它为开发者提供了一套完整的构建生命周期框架,开发团队稍微投入一些时间就能够自动完成工程的基础构建配置;在有多个开发团队环境的情况下，Maven 能够在很短的时间内使得每项工作都按照标准进行;那是因为大部分的工程配置操作都非常

简单并且可复用;Maven基干项目对象模型(POM project object model)，它可以通过一小段描述信息（配置)来管理项目的构建;因而,每个maven项目都有一个pom.xml文件

POM(Project Object Model，项目对象模型) 是 Maven 工程的基本工作单元，是一个 XML 文件，包含了项目的基本信息，用于描述项目如何构建，声明项目依赖等，在执行任务或目标时，Maven 会在当前目录中查找 pom 文件，通过读取pom文件获取所需的配置信息，然后执行目标。

pom.xml 文件中可以指定以下配置：

```
项目依赖
插件
执行目标
项目构建 profile
项目版本
项目开发者列表 相关邮件列表信息
用<packaging> 指定项目打包形式，jar或者war
```

maven 官网

```
https://maven.apache.org/
```

maven 官方仓库：

```
http://repo.maven.apache.org
```

maven 下载

```
#官方
https://maven.apache.org/download.cgi

#清华镜像源
http://mirrors.tuna.tsinghua.edu.cn/apache/maven

#官方各版本下载地址，推荐使用次新版本
https://archive.apache.org/dist/maven/maven-3
```

安装maven前必须安装java 环境：

```
Maven 3.3 要求 JDK 1.7 或以上
Maven 3.2 要求 JDK 1.6 或以上
Maven 3.0/3.1 要求 JDK 1.5 或以上
```

9.2.2 maven 安装

9.2.2.1 安装方式1: 包安装

范例：Ubuntu安装maven

```
[root@ubuntu1804 ~]#apt list  maven -a
Listing... Done
maven/bionic-security,bionic-security,bionic-updates,bionic-updates,now 3.6.0-
1~18.04.1 all [installed]
maven/bionic,bionic 3.5.2-2 all
[root@ubuntu1804 ~]#apt -y install maven
```

```
[root@ubuntu1804 ~]#mvn -v
Apache Maven 3.6.0
Maven home: /usr/share/maven
Java version: 11.0.11, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-openjdk-
amd64
Default locale: en_HK, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-112-generic", arch: "amd64", family: "unix"

#镜像加速
[root@ubuntu1804 ~]#vim /etc/maven/settings.xml
  <mirrors>
    <!--阿里云镜像-->
    <mirror>
        <id>nexus-aliyun</id>
        <mirrorOf>*</mirrorOf>
        <name>Nexus aliyun</name>
        <url>http://maven.aliyun.com/nexus/content/groups/public</url>
    </mirror>

  </mirrors>
```

范例: Rocky/CentOS/RHEL安装maven

```
[root@rocky8 ~]#yum -y install maven
[root@rocky8 ~]#mvn -v
Apache Maven 3.5.4 (Red Hat 3.5.4-5)
Maven home: /usr/share/maven
Java version: 1.8.0_292, vendor: Red Hat, Inc., runtime: /usr/lib/jvm/java-
1.8.0-openjdk-1.8.0.292.b10-1.el8_4.x86_64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.18.0-305.3.1.el8_4.x86_64", arch: "amd64", family:
"unix"

#镜像加速
[root@rocky8 ~]#vim /etc/maven/settings.xml
  <mirrors>
    <!--阿里云镜像-->
    <mirror>
        <id>nexus-aliyun</id>
        <mirrorOf>*</mirrorOf>
        <name>Nexus aliyun</name>
        <url>http://maven.aliyun.com/nexus/content/groups/public</url>
    </mirror>

  </mirrors>
```

9.2.2.2 安装方式2: 二进制安装

官方说明:

```
http://maven.apache.org/install.html
```

范例: 二进制安装

```
[root@ubuntu1804 ~]#apt update
[root@ubuntu1804 ~]#apt install openjdk-8-jdk -y
```

```
[root@ubuntu1804 ~]#java -version
openjdk version "1.8.0_282"
OpenJDK Runtime Environment (build 1.8.0_282-8u282-b08-0ubuntu1~18.04-b08)
OpenJDK 64-Bit Server VM (build 25.282-b08, mixed mode)

[root@ubuntu1804 ~]#wget
https://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.6.3/binaries/apache-
maven-3.6.3-bin.tar.gz
[root@ubuntu1804 ~]#tar xf apache-maven-3.6.3-bin.tar.gz  -C /usr/local/
[root@ubuntu1804 ~]#ln -s /usr/local/apache-maven-3.6.3/ /usr/local/maven

[root@ubuntu1804 ~]#echo 'PATH=/usr/local/maven/bin:$PATH' >
/etc/profile.d/maven.sh
[root@ubuntu1804 ~]#echo 'export MAVEN_HOME=/usr/local/maven' >>
/etc/profile.d/maven.sh

[root@ubuntu1804 ~]#. /etc/profile.d/maven.sh
[root@ubuntu1804 ~]#mvn -v
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: /usr/local/maven
Java version: 1.8.0_282, vendor: Private Build, runtime: /usr/lib/jvm/java-8-
openjdk-amd64/jre
Default locale: en_HK, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-112-generic", arch: "amd64", family: "unix"


#镜像加速
[root@ubuntu1804 ~]#vim /usr/local/maven/conf/settings.xml

  <mirrors>
    <!--阿里云镜像-->
    <mirror>
        <id>nexus-aliyun</id>
        <mirrorOf>*</mirrorOf>
        <name>Nexus aliyun</name>
        <url>http://maven.aliyun.com/nexus/content/groups/public</url>
    </mirror>

  </mirrors>
```

9.3 Maven 的打包命令说明

Maven工程构建的各个环节

- clean: 以前编译得到的旧文件class字节码文件删除
- compile:将java源程序编译成class字节码文件
- test:自动测试,例如，自动调用junit程序
- report:报告测试程序执行的结果
- package:应用打包,动态Web工程打成war包,java工程打成jar包
- install: 是指将打包得到的文件复制到仓库中指定的位置
- deploy:将动态Web工程生成的war包复制到Servlet容器下，使其可以运行

范例:

- 进入到包含有“pom.xml”的路径，执行:

```
mvn clean install package
```

- 有的时候受到测试的干扰，导致无法正在进行编译，这时候可以选择跳过测试：

```
mvn clean install package -Dmaven.test.skip=true
选项"-Dmaven.test.skip=true"：跳过测试，并且不编译测试下的源代码
选项"-DskipTests"：不执行测试，但是会进行测试代码的编译
```

- 如果需要编译的代码很庞大，需要考虑对编译环境做一些处理，提成编译效率：

```
启动多线程编译：mvn -T 4 clean install package -Dmaven.test.skip=true
分配编译的CPU 个数：mvn -T 2C clean install package -Dmaven.test.skip=true
启用多线程编译：mvn clean install package -Dmaven.test.skip=true -
Dmaven.compile.fork=true
```

- 所有的 Maven 都是建立在 JVM 上的，所以进行编译的时候还需要考虑JVM 参数优化：

```
如果是windows 找到“maven/bin/mvn.cmd”，如果 linux 找到“maven/bin/mvn”，配置参数
是：“MAVEN_OPTS”
打开属性配置文件：vim /etc/profile
指定内存配置：export MAVEN_OPTS="-Xmx6g -Xms6g" 注意不要超过物理内存一半
使配置立即生效：source /etc/profile
```

9.4 执行 java 代码编译实战案例

9.4.1 编译安装 spring-boot 项目

```
[root@ubuntu1804 ~]#apt -y install git maven

#镜像加速
[root@ubuntu1804 ~]#vim /etc/maven/settings.xml
   <mirror>
        <id>nexus-aliyun</id>
        <mirrorOf>*</mirrorOf>
        <name>Nexus aliyun</name>
        <url>http://maven.aliyun.com/nexus/content/groups/public</url>
    </mirror>
[root@ubuntu1804 ~]#cd /data
[root@ubuntu1804 data]#git clone https://gitee.com/lbtooth/spring-boot-
helloWorld.git
[root@ubuntu1804 data]#cd spring-boot-helloWorld/

#编译
[root@ubuntu1804 spring-boot-helloWorld]#mvn clean package  -
Dmaven.test.skip=true
[root@ubuntu1804 spring-boot-helloWorld]#ls target/
classes                 maven-archiver                          spring-boot-
helloworld-0.9.0-SNAPSHOT.jar.original
generated-sources       maven-status                            surefire-
reports
generated-test-sources  spring-boot-helloworld-0.9.0-SNAPSHOT.jar  test-classes

#默认为8080端口
[root@ubuntu1804 spring-boot-helloWorld]#java -jar target/spring-boot-
helloworld-0.9.0-SNAPSHOT.jar --server.port=8888
```

```
[root@ubuntu1804 spring-boot-helloWorld]#java -jar target/spring-boot-helloworld-0.9.0-SNAPSHOT.jar --server.port=8888

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)
```

```
[root@ubuntu1804 ~]#ss -ntl
State          Recv-Q    Send-Q       Local Address:Port       Peer Address:Port
Process
LISTEN         0         128                0.0.0.0:22              0.0.0.0:*

LISTEN         0         100              127.0.0.1:25              0.0.0.0:*

LISTEN         0         128                   [::]:22                 [::]:*

LISTEN         0         100                      *:8888                  *:*

LISTEN         0         100                  [::1]:25                [::]:*

[root@ubuntu1804 ~]#curl 127.0.0.1:8888
Hello Spring Boot 2.0!

[root@ubuntu1804 ~]#curl 127.0.0.1:8888/hello
Hello World
```

9.4.2 编译 java 程序 jpress

```
[root@rocky8 ~]#wget
'https://gitee.com/JPressProjects/jpress/repository/archive/v4.2.0?
format=tar.gz'

[root@rocky8 ~]#tar xf jpress-v4.2.0.tar_gz

[root@rocky8 ~]#cd jpress-v4.2.0/

[root@rocky8 jpress-v4.2.0]#mvn clean install package -Dmaven.test.skip=true
...
[INFO] module-product-search-db ........................... SUCCESS [  0.095 s]
[INFO] module-product-search-lucene ....................... SUCCESS [  0.101 s]
[INFO] module-product-search-es ........................... SUCCESS [  0.160 s]
[INFO] module-product-search-opensearch ................... SUCCESS [  0.120 s]
[INFO] module-product-service-provider .................... SUCCESS [  0.228 s]
[INFO] module-product-web ................................. SUCCESS [  0.274 s]
[INFO] starter ............................................ SUCCESS [01:20 min]
[INFO] starter-tomcat 4.0 ................................. SUCCESS [  8.736 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 03:00 min
[INFO] Finished at: 2021-12-15T18:18:06+08:00
[INFO] ------------------------------------------------------------------------

[root@rocky8 jpress-v4.2.0]#cp starter-tomcat/target/starter-tomcat-4.0.war
/usr/local/tomcat/webapps/jpress.war
```

10 面试题

- JVM的组成
- JVM的常见垃圾回收器
- JVM的常见启动参数
- JAVA程序出现OOM,如何解决
- Tomcat的优化方法