优医问诊day01

创建项目、代码风格、状态仓库、移动适配、请求工具

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

创建问诊项目

介绍 pnpm 包管理器

一些优势：比同类工具快2倍左右、节省磁盘空间... https://www.pnpm.cn/

安装方式：npm install -g pnpm

创建项目：pnpm create vue

npm	yarn	pnpm
npm install	yarn	pnpm install
npm install axios	yarn add axios	pnpm add axios
npm install axios –D	yarn add axios –D	pnpm add axios –D
npm uninstall axios	yarn remove axios	pnpm remove axios
npm run dev	yarn dev	pnpm dev

创建项目

```
pnpm create vue
```

```
project — zhousg@bogon

Vue.js - The Progressive JavaScript Framework

✔ Project name: … consult-patient-vue3
✔ Add TypeScript? … No / Yes
✔ Add JSX Support? … No / Yes
✔ Add Vue Router for Single Page Application development? … No / Yes
✔ Add Pinia for state management? … No / Yes
✔ Add Vitest for Unit Testing? … No / Yes
✔ Add Cypress for both Unit and End-to-End testing? … No / Yes
✔ Add ESLint for code quality? … No / Yes
✔ Add Prettier for code formatting? … No / Yes

Scaffolding project in /Users/zhousg/project/consult-patient-vue3...

Done. Now run:

  cd consult-patient-vue3
  pnpm install
  pnpm lint
  pnpm dev


# zhousg @ bogon in ~/project [16:33:34]
```

1. 进入项目目录
2. 安装依赖
3. 格式化所有代码
4. 启动项目

项目配置

- Eslint 代码风格
- 开启 TS 托管模式

Eslint 配置代码风格

配置文件 .eslintrc.cjs

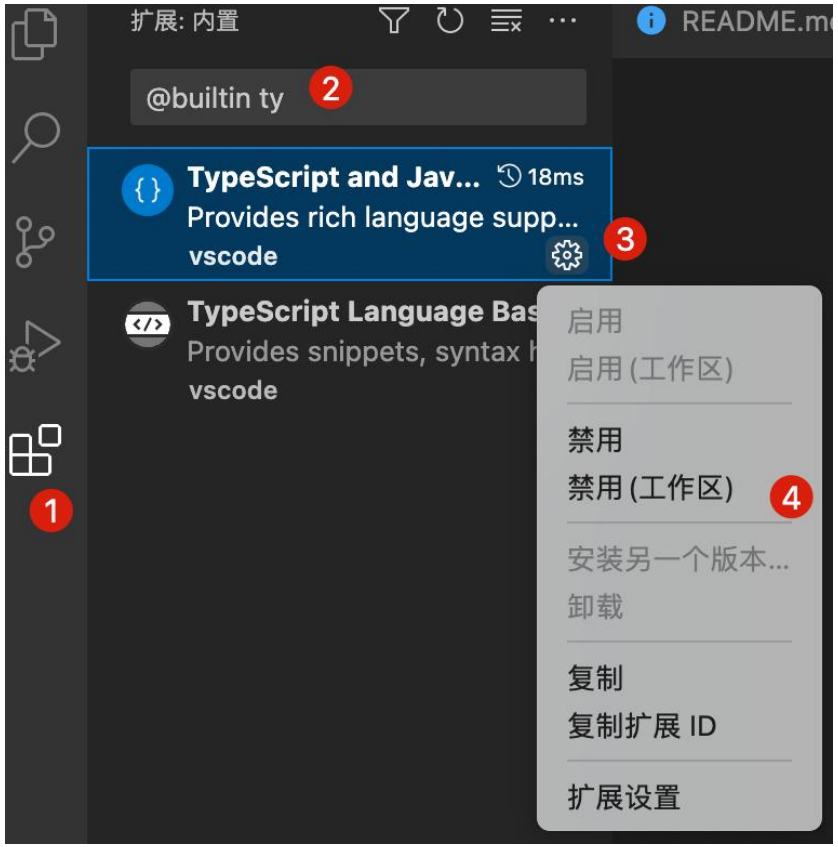
1. prettier 风格配置 https://prettier.io
    1. 单引号
    2. 不使用分号
    3. 宽度80字符
    4. 不加对象|数组最后逗号
    5. 换行符号不限制（win mac 不一致）
2. vue组件名称多单词组成（忽略index.vue）
3. props解构（关闭）

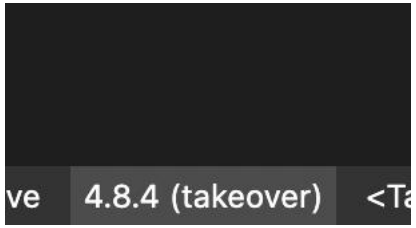
提示：安装Eslint且配置保存修复，不要开启默认的自动保存格式化

```
4  module.exports = {
5    root: true,
6  › extends: […
11   ],
12 › parserOptions: {…
14   },
15   rules: {
16     'prettier/prettier': [  1
17       'warn',
18       {
19         singleQuote: true,
20         semi: false,
21         printWidth: 80,
22         trailingComma: 'none',
23         endOfLine: 'auto'
24       }
25     ],
26     'vue/multi-word-component-names': [ 2
27       'warn',
28       {
29         ignores: ['index']
30       }
31     ],
32     'vue/no-setup-props-destructure': ['off'] 3
33   }
34 }
35
```

开启 TS 托管模式

1. Volar 语法高亮，代码提示，支持Vue3新特性
2. TypeScript Vue Plugin (Volar) 让TS服务知道.vue文件
3. Take Over Mode 托管模式，TS服务性能更好
    1. 关闭 vscode 内置的TS服务
    2. 使用 Volar 提供的TS服务



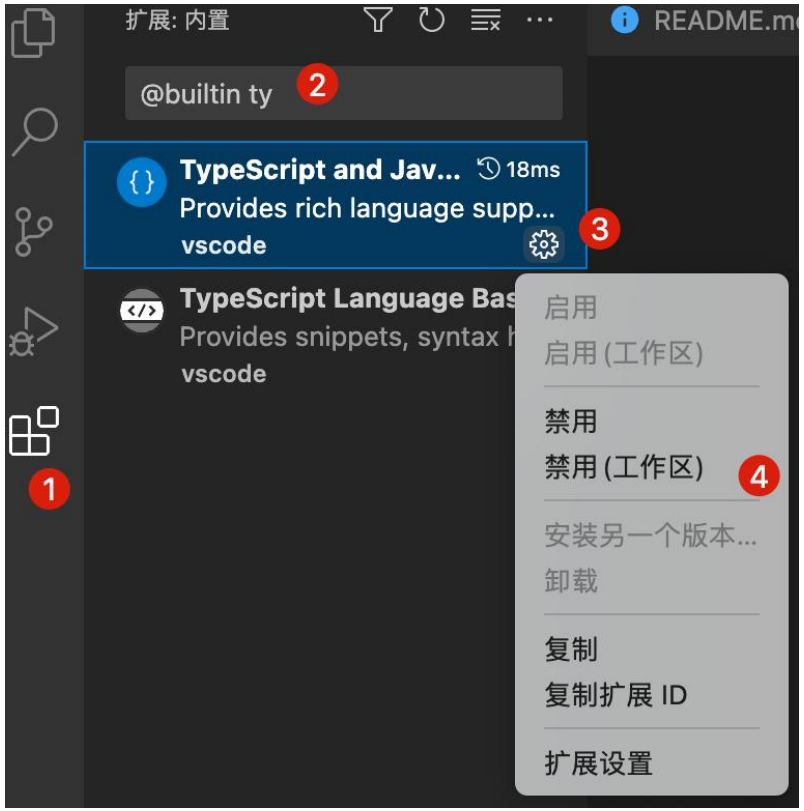


重启 vscode

Eslint 配置

```
4  module.exports = {
5    root: true,
6    extends: [ …
11   ],
12   parserOptions: { …
14   },
15   rules: {
16     'prettier/prettier': [  1
17       'warn',
18       {
19         singleQuote: true,
20         semi: false,
21         printWidth: 80,
22         trailingComma: 'none',
23         endOfLine: 'auto'
24       }
25     ],
26     'vue/multi-word-component-names': [  2
27       'warn',
28       {
29         ignores: ['index']
30       }
31     ],
32     'vue/no-setup-props-destructure': ['off']  3
33   }
34 }
35
```

TS 托管



配置代码检查工作流

提交前做代码检查

1. 初始化 git 仓库，执行 git init 即可
2. 初始化 husky 工具配置，执行 pnpm dlx husky-init && pnpm install 即可

   https://typicode.github.io/husky/
3. 修改 .husky/pre-commit 文件

问题：pnpm lint 是全量检查，耗时问题，历史问题。

暂存区 eslint 校验

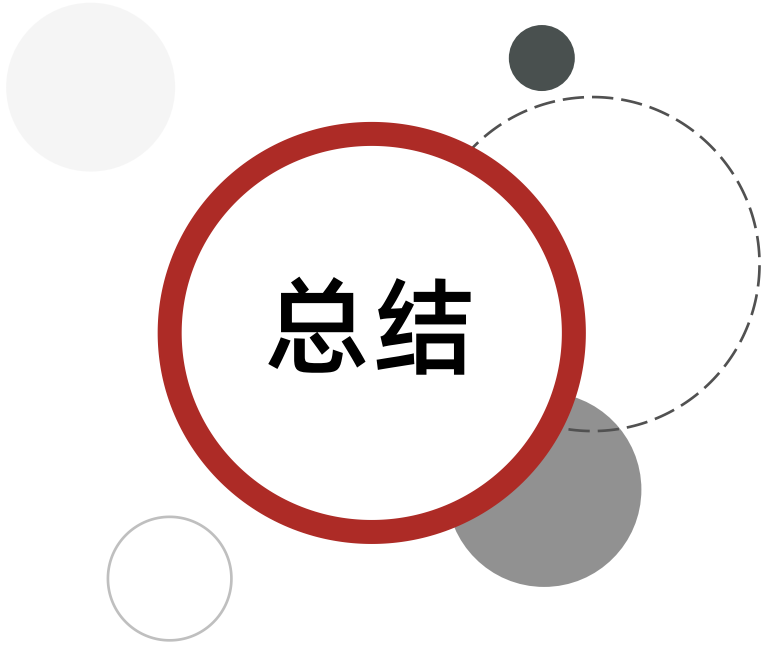
1. 安装 lint-staged 包 pnpm i lint-staged -D

2. package.json 配置 lint-staged 命令

3. .husky/pre-commit 文件修改

```
-pnpm lint
+pnpm lint-staged
```

```
4   "scripts": {
5     "dev": "vite",
6     "build": "run-p type-check build-only",
7     "preview": "vite preview --port 4173",
8     "build-only": "vite build",
9     "type-check": "vue-tsc --noEmit",
10    "lint": "eslint . --ext .vue,.js,.jsx,.
11    "prepare": "husky install",
12  1 "lint-staged": "lint-staged"
13  },
14  "lint-staged": { 2
15    "*.{js,ts,vue}": [
16      "eslint --fix"
17    ]
18  },
```



1. 如何在 git commit 前执行 eslint 检查？

   使用 husky 这个 git hooks 工具

2. 如何只检查暂存区代码？

   使用 lint-staged 工具

04 目录调整

目录调整

```
./src
├── assets          `静态资源，图片...`
├── components      `通用组件`
├── composables     `组合功能通用函数`
├── icons           `svg图标`
├── router          `路由`
│   └── index.ts
├── services        `接口服务API`
├── stores          `状态仓库`
├── styles          `样式`
│   └── main.scss
├── types           `TS类型`
├── utils           `工具函数`
├── views           `页面`
├── main.ts         `入口文件`
└── App.vue         `根组件`
```

安装 sass 预处理器
pnpm i sass -D

导入 main.ts 作为全局样式
import ‘./styles/main.scss’

vue-router初始化

路由初始化

```
import VueRouter from 'vue-router'
// 初始化 vue-router3.x(Vue2)
const router = new VueRouter({
  mode: 'history',
  routes: [],
})

export default router
```

```
import { createRouter, createWebHistory } from 'vue-router'
// 初始化 vue-router4.x(Vue3)
const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: []
})

export default router
```

1. 创建路由实例由 createRouter 实现
2. 路由模式
    1. history 模式使用 createWebHistory()
    2. hash 模式使用 createWebHashHistory()
    3. 参数是基础路径，默认/

扩展：import.meta.env.BASE_URL

import.meta 是 JavaScript 模块暴露的描述模块的信息对象

```
<script type="module" src="./test.js"></script>
```

```
console.log(import.meta) // { url: "xxx/test.js" }
```

env.BASE_URL 是Vite 环境变量：https://cn.vite.dev

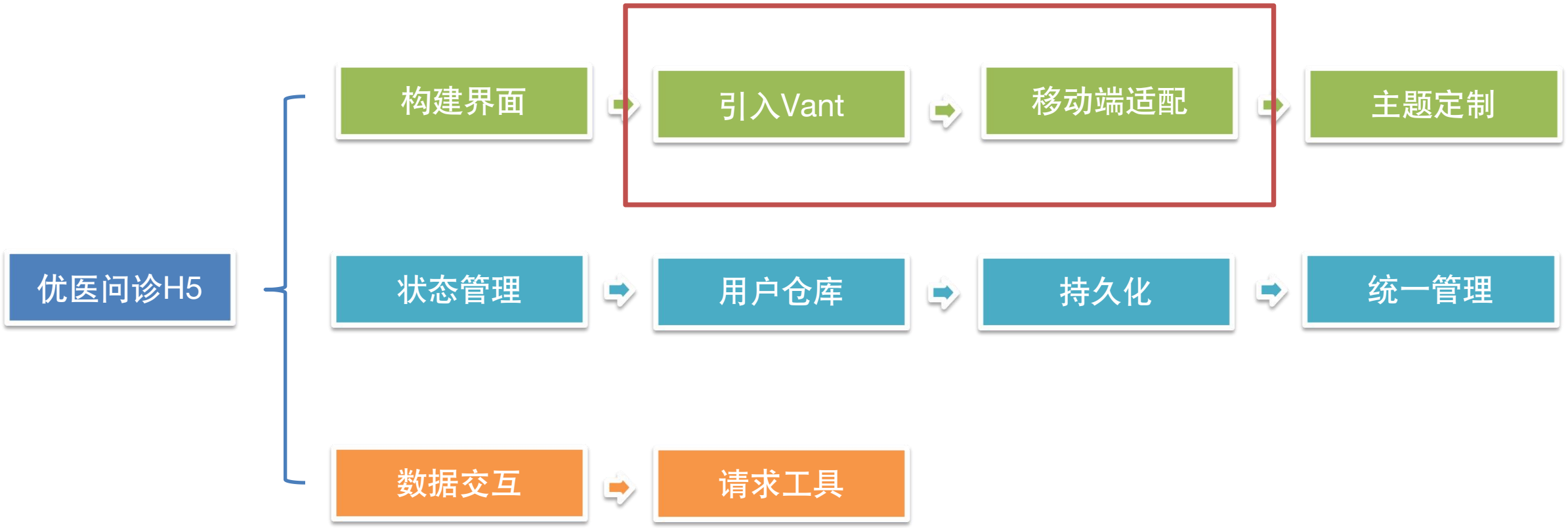
总结

```
import { createRouter, createWebHistory } from 'vue-router'

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: []
})

export default router
```

创建一个路由实例，路由模式是history模式，路由的基础地址是vite.config.ts 中的 base 配置的值，默认是 /

基础架构–引入Vant和移动端适配

基础架构-引入Vant和移动端适配



引入Vant

安装：pnpm add vant

main.ts 引入样式：`import 'vant/lib/index.css'`

使用 vant 组件

```
<script setup lang="ts">
import { Button as VanButton } from 'vant'
</script>

<template>
  <van-button>按钮</van-button>
</template>

<style scoped></style>
```

官方文档：https://vant-contrib.gitee.io/vant/

移动端适配

安装：pnpm add postcss-px-to-viewport -D

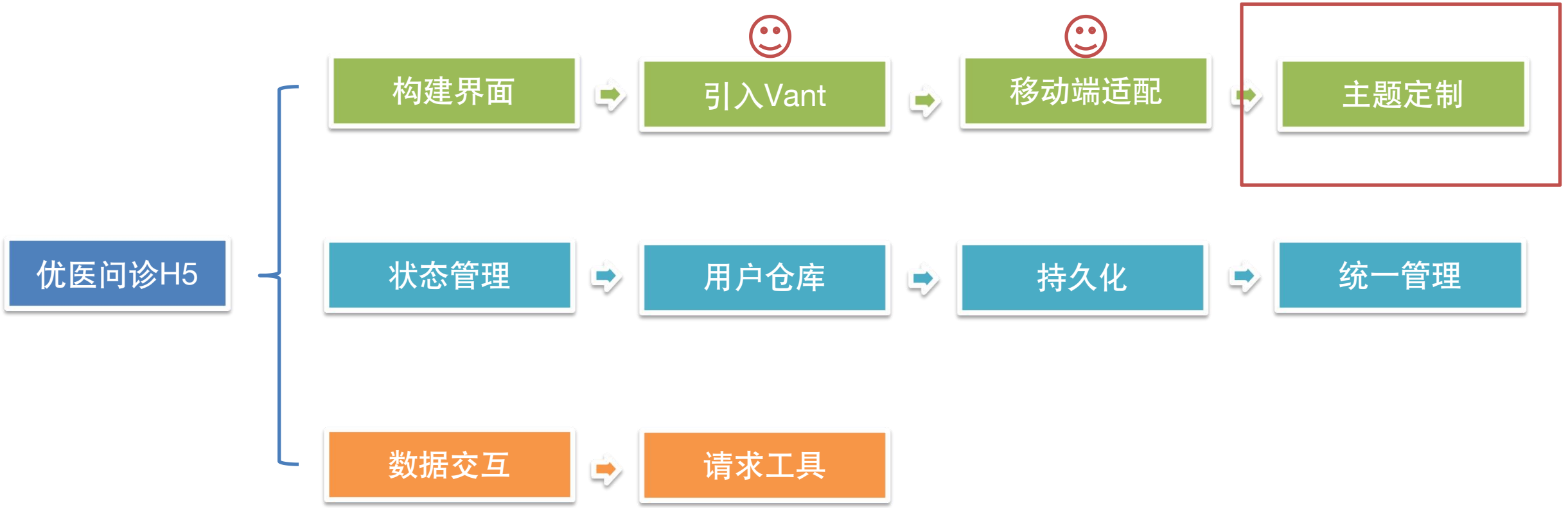
新增配置文件：postcss.config.js

```
// eslint-disable-next-line no-undef
module.exports = {
  plugins: {
    'postcss-px-to-viewport': {
      // 设备宽度375计算vw的值
      viewportWidth: 375,
    },
  },
};
```

适配方案：https://vant-contrib.gitee.io/vant/

基础架构–主题定制

基础架构-主题定制



css 变量

```css
:root {
  // 全局 css 变量
  --main-color: #069;
}

.footer {
  // 局部 css 变量
  --footer-color: #f40;
}

a {
  // 使用变量
  color: var(--main-color);
}
```

全局 css 变量，使用场景：项目主题

局部 css 变量，使用场景：组件变量

提问：主题定制采用 css 变量写在哪里？

项目主题


```scss
:root {
  // 问诊患者：色板
  --cp-primary: #16C2A3;
  --cp-plain: #EAF8F6;
  --cp-orange: #FCA21C;
  --cp-text1: #121826;
  --cp-text2: #3C3E42;
  --cp-text3: #6F6F6F;
  --cp-tag: #848484;
  --cp-dark: #979797;
  --cp-tip: #C3C3C5;
  --cp-disable: #D9DBDE;
  --cp-line: #EDEDED;
  --cp-bg: #F6F7F9;
  --cp-price: #EB5757;


}
```

如何覆盖vant的主题色？

- 找到主题色变量名，覆盖即可。

项目主题

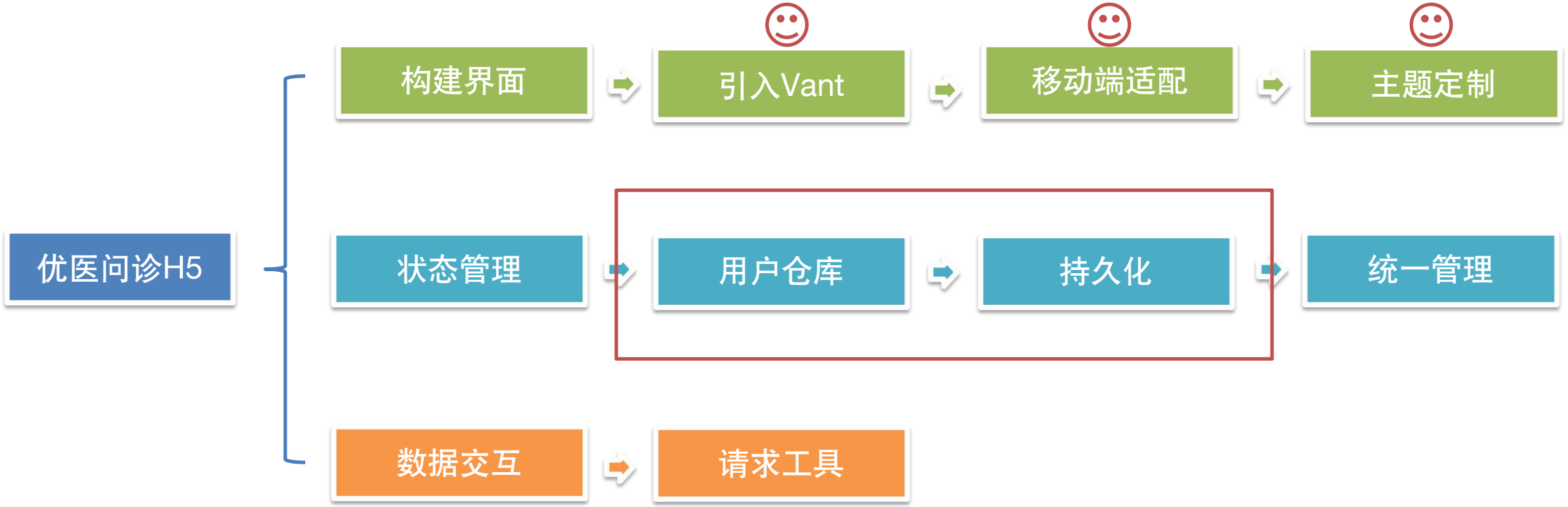

```scss
:root {
  // 问诊患者：色板
  --cp-primary: #16C2A3;
  --cp-plain: #EAF8F6;
  --cp-orange: #FCA21C;
  --cp-text1: #121826;
  --cp-text2: #3C3E42;
  --cp-text3: #6F6F6F;
  --cp-tag: #848484;
  --cp-dark: #979797;
  --cp-tip: #C3C3C5;
  --cp-disable: #D9DBDE;
  --cp-line: #EDEDED;
  --cp-bg: #F6F7F9;
  --cp-price: #EB5757;
  // 覆盖vant主题色
  --van-primary-color: var(--cp-primary);
}
```

如何覆盖vant的主题色？

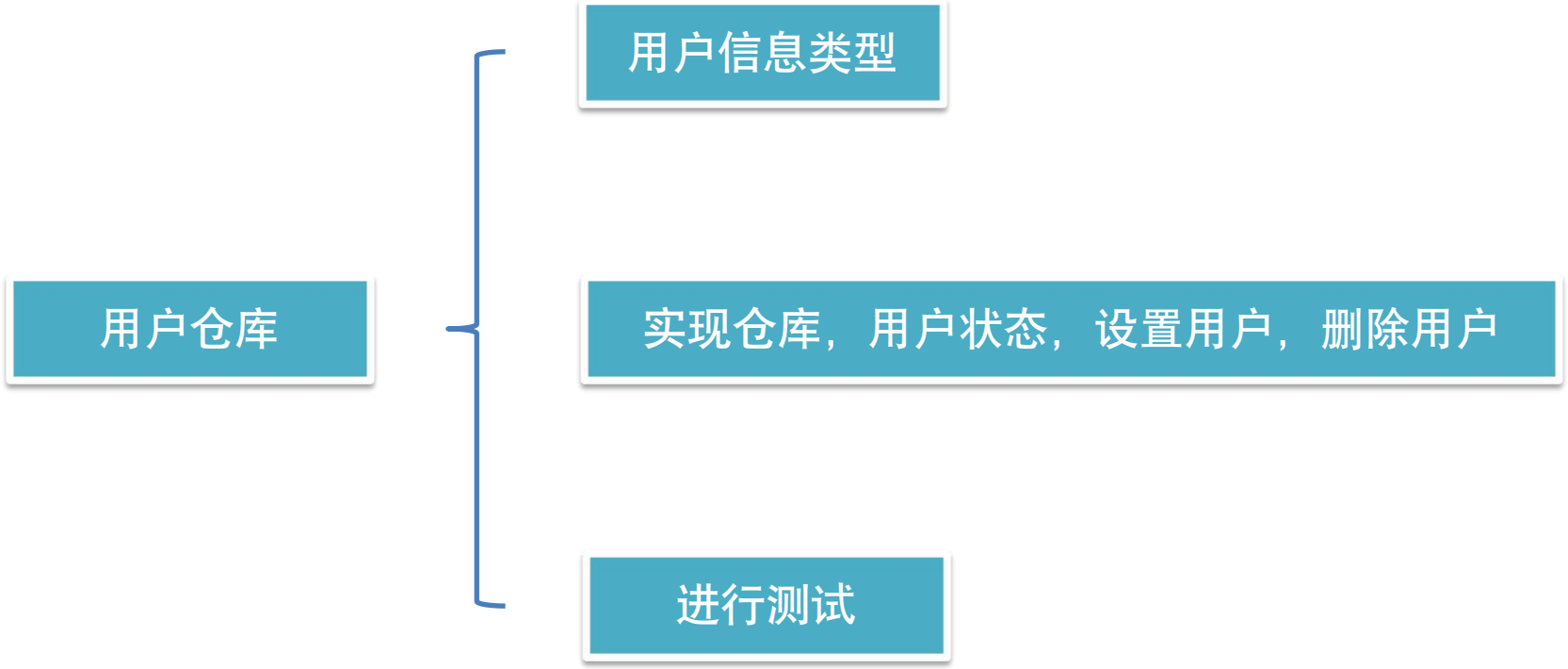
- 找到主题色变量名，覆盖即可。

08 基础架构–用户仓库和持久化

基础架构-用户仓库和持久化



用户仓库



持久化

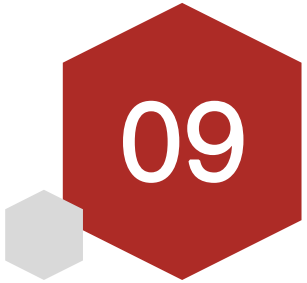
安装：pnpm i pinia-plugin-persistedstate

使用：main.ts

```
import persist from 'pinia-plugin-persistedstate'
// ...省略...
app.use(createPinia().use(persist))
```

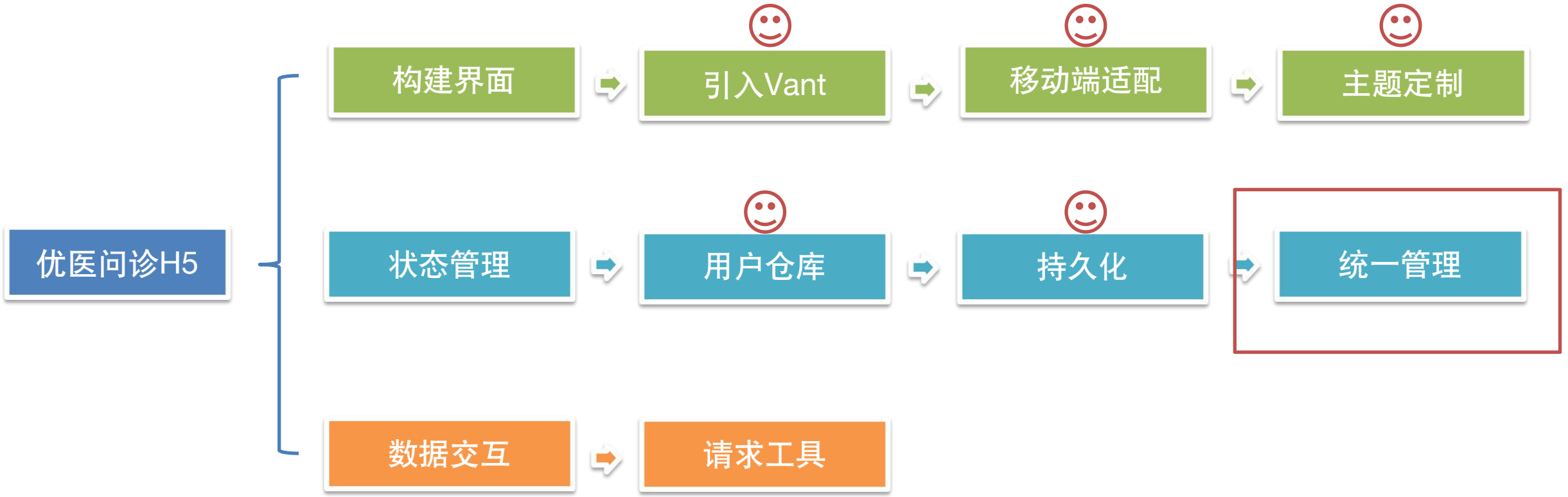
配置：stores/user.ts

```
export const useUserStore = defineStore(
  'cp-user',
  () => {
    // ...省略...
    return {}
  },
  {
    persist: true
  }
)
```



基础架构–仓库统一管理

基础架构-仓库统一管理



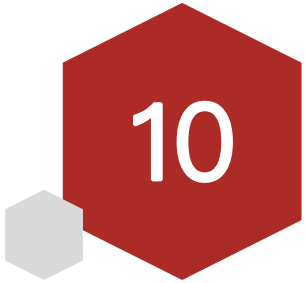
如何统一管理

pinia 独立维护

- 现在：初始化代码在 main.ts 中，仓库代码在 stores 中，代码分散职能不单一
- 优化：由 stores 统一维护，在 stores/index.ts 中完成 pinia 初始化，交付 main.ts 使用

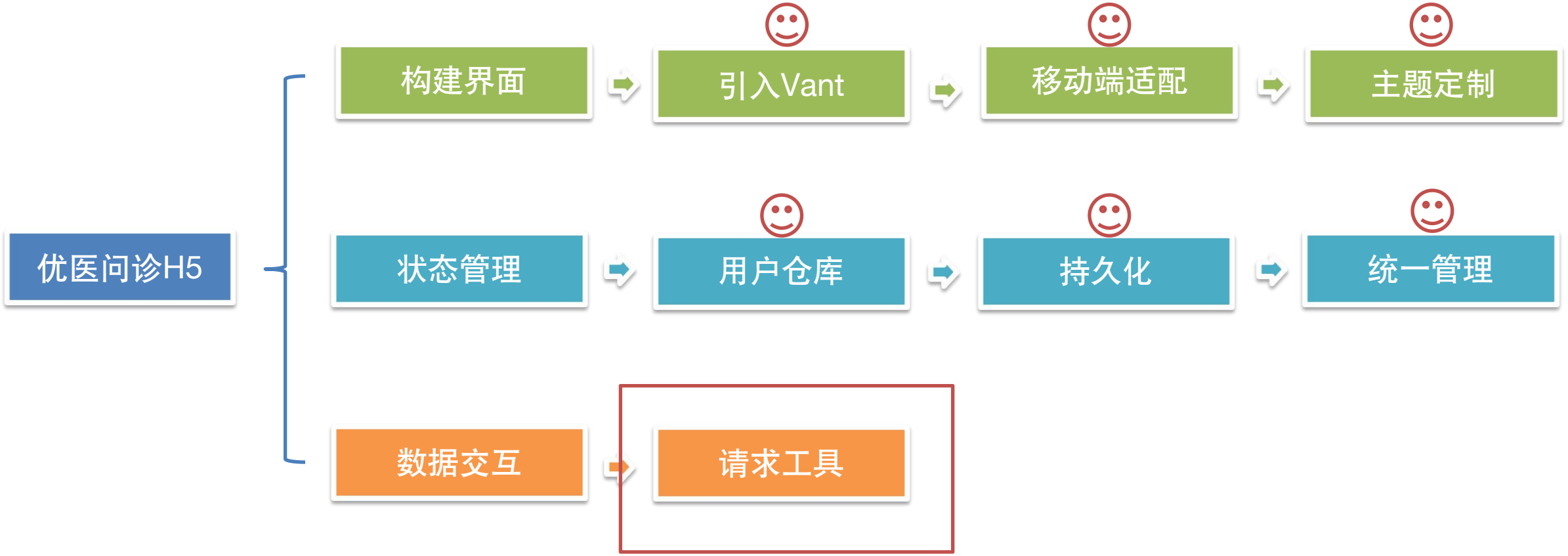
仓库 统一导出

- 现在：使用一个仓库 import { useUserStore } from `./stores/user.ts` 不同仓库路径不一致
- 优化：由 stores/index.ts 统一导出，导入路径统一 `./stores`，而且仓库维护在 stores/modules 中

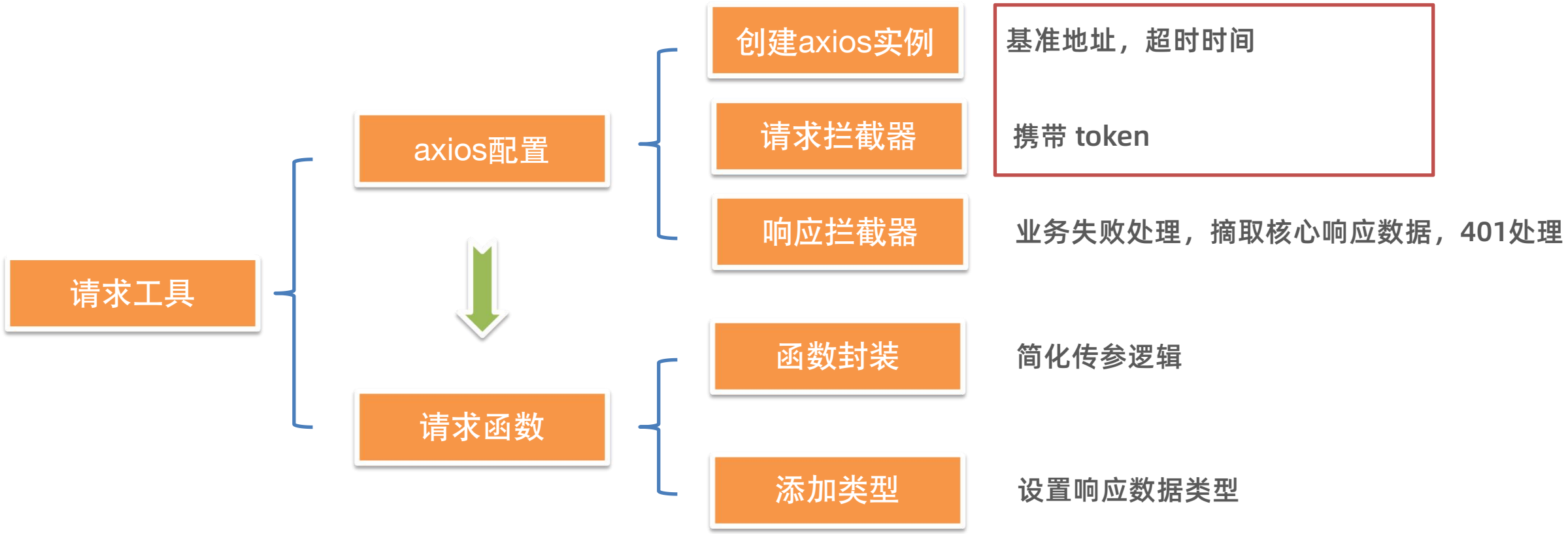


基础架构–请求工具–axios配置

基础架构-请求工具

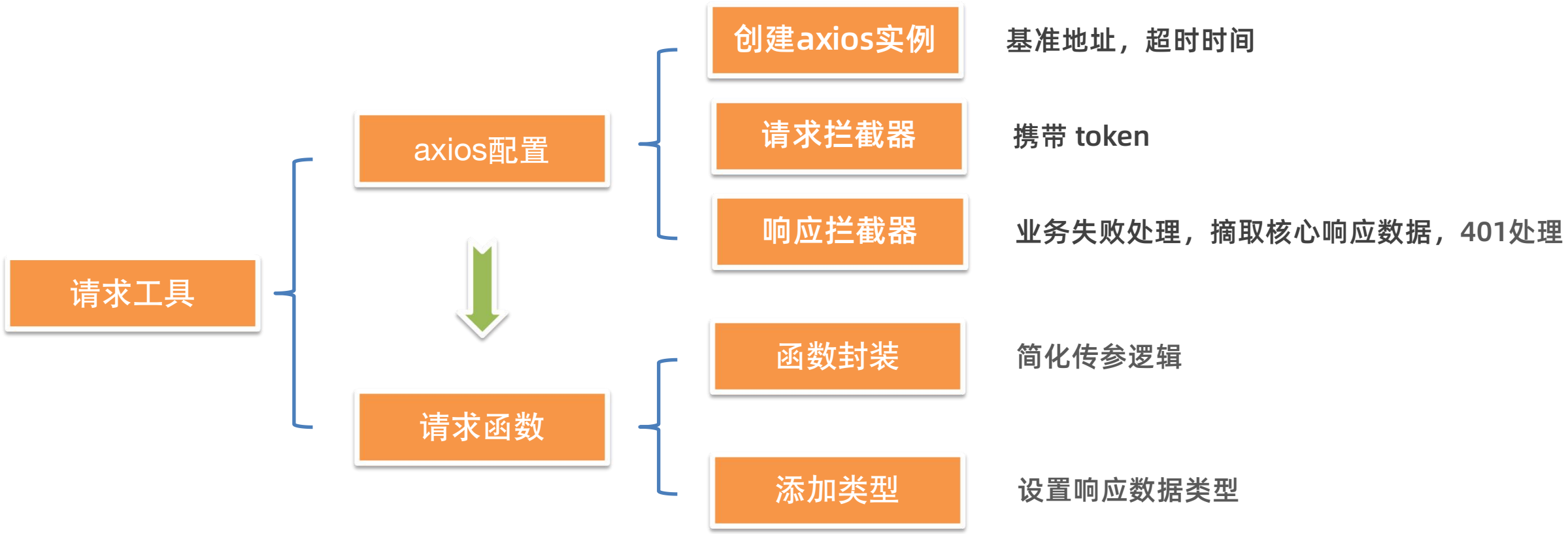


基础架构-请求工具-axios配置

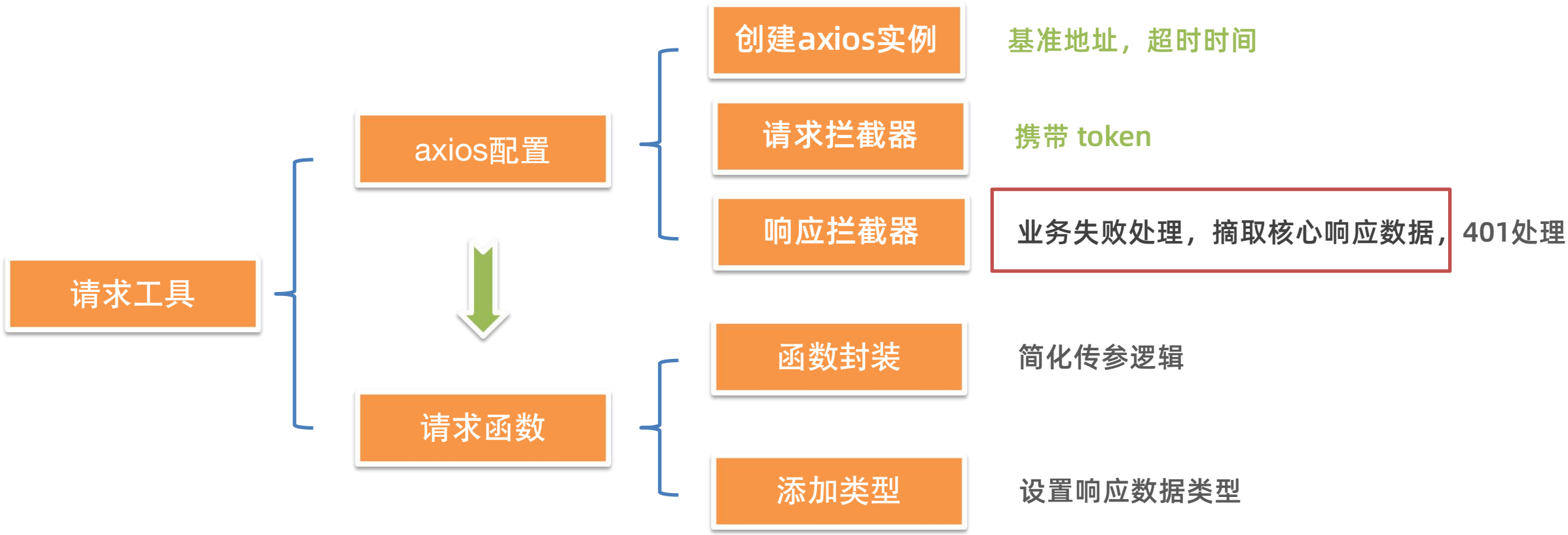


基础架构–请求工具–axios配置

基础架构-请求工具-axios配置



基础架构-请求工具-axios配置



响应成功-业务失败处理&摘取核心响应数据

业务失败处理：

1. 如何判断业务失败？code 不是 10000
2. 失败后需要做什么？弹出轻提示，此时返回一个失败的 promise，传递code给catch

摘取核心响应数据：

1. 现在结果 ===> { data: 响应数据 }
2. 期望结果 ===> 响应数据

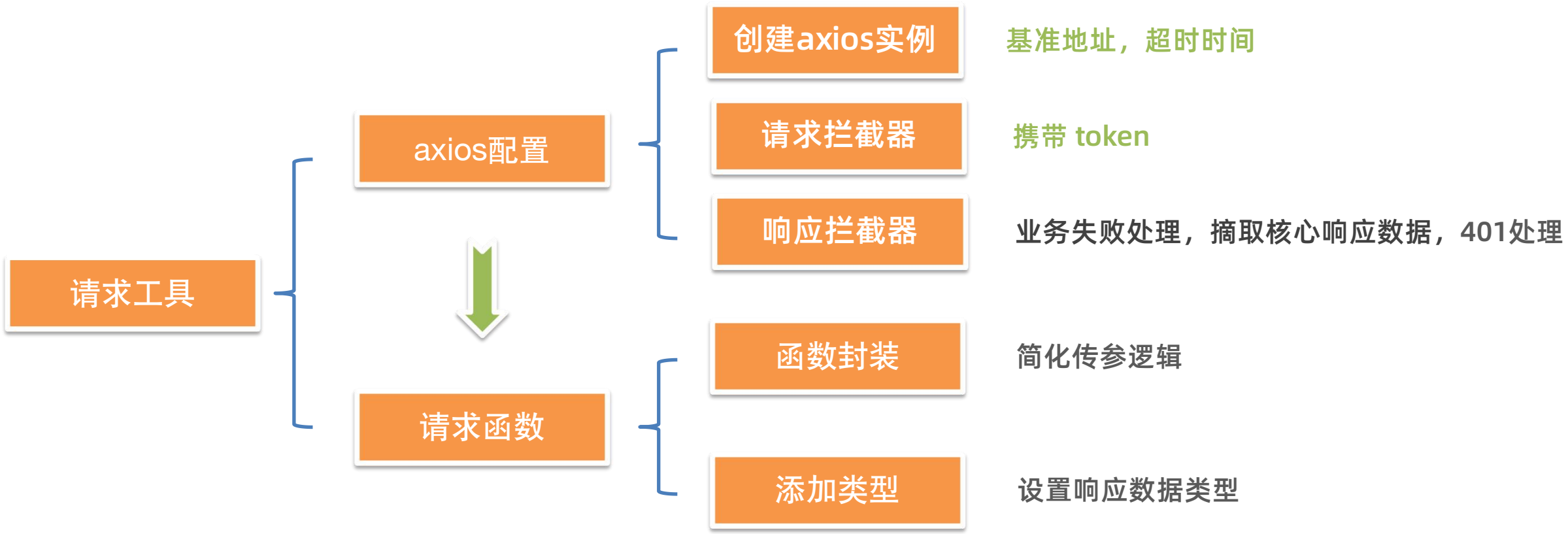
HTTP 状态码: 200　　内容格式: JSON

object {3}　empty object　登录接口响应数据

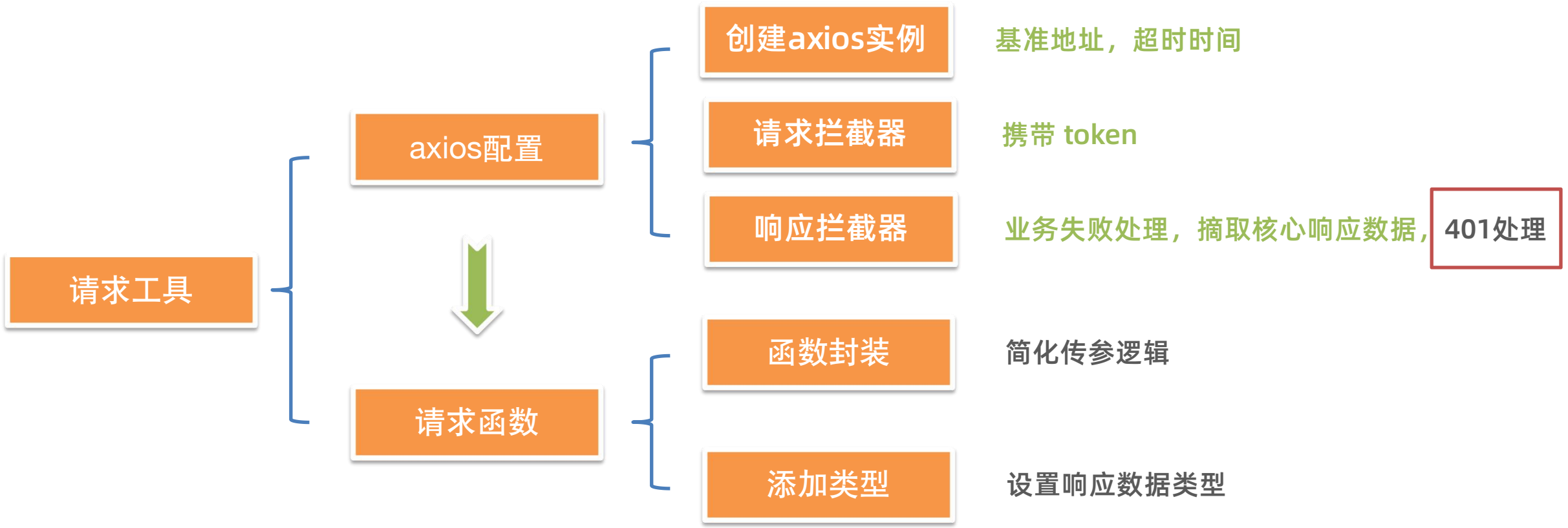
code	number	✓	正常返回10000，其他表示错误
message	string	✓	接口信息
data	object {6}	✓	
token	string	✓	token

基础架构–请求工具–axios配置

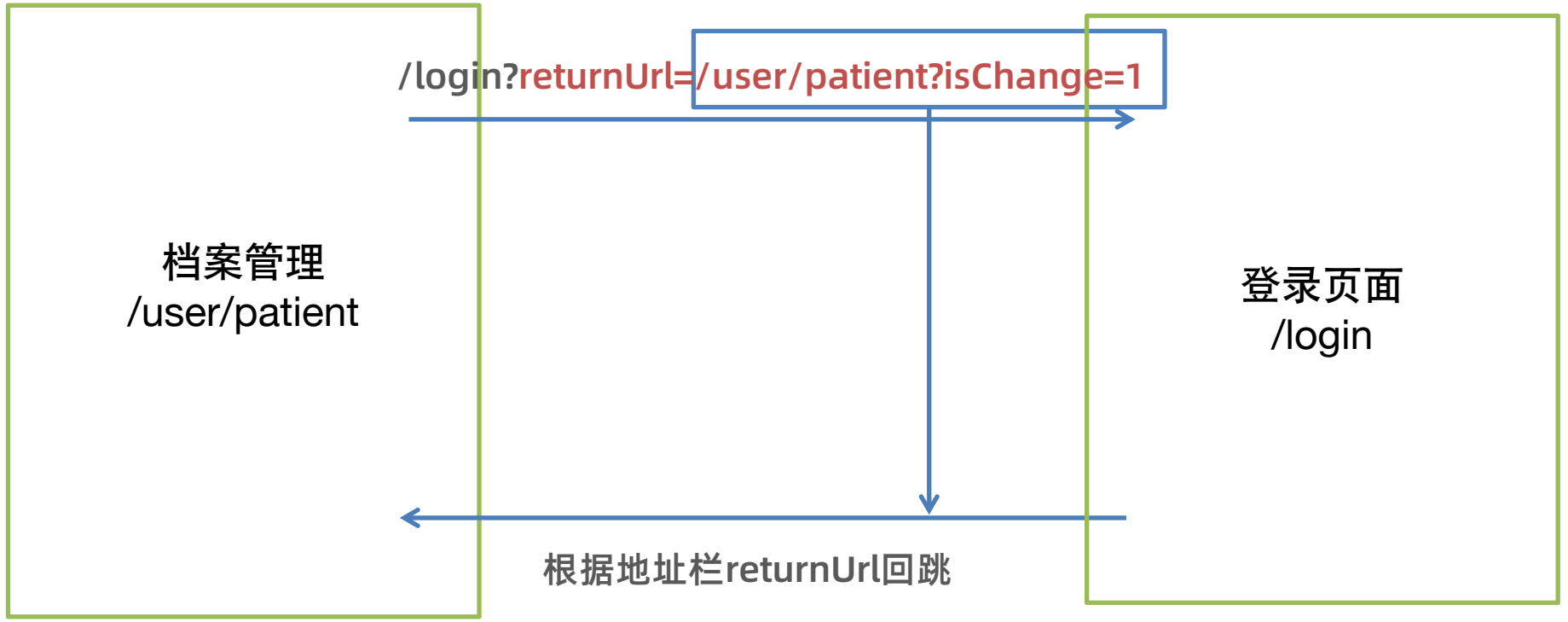
基础架构-请求工具-axios配置



基础架构-请求工具-axios配置

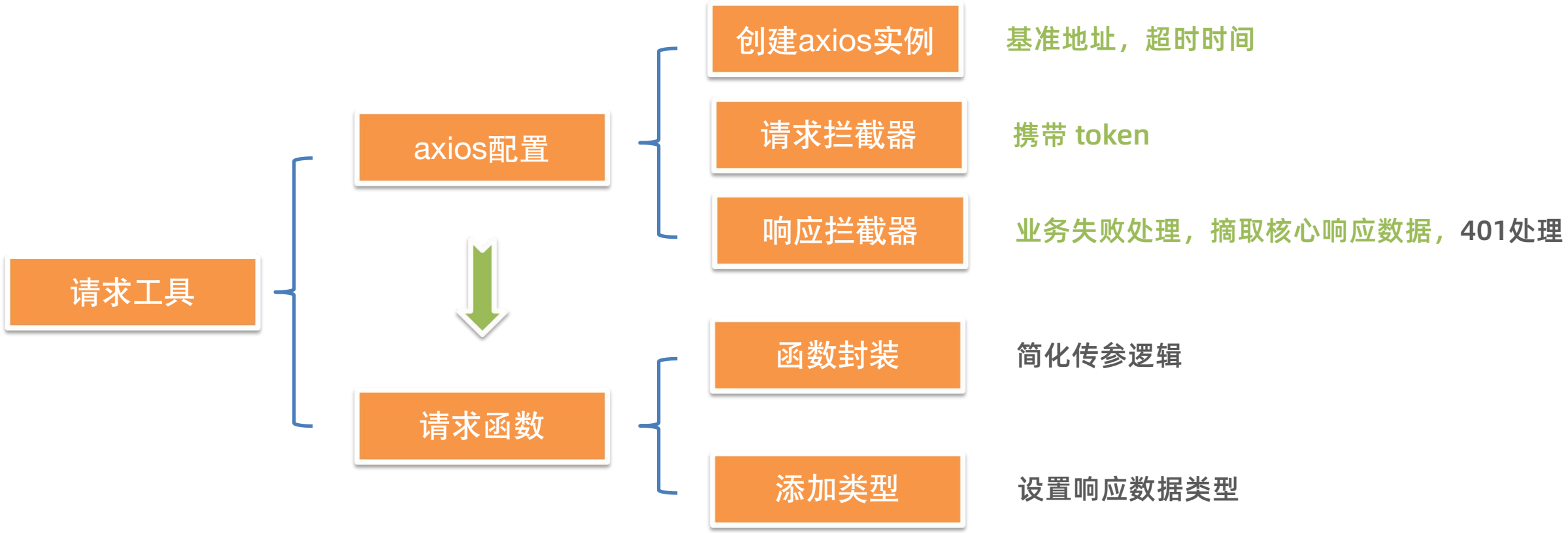


响应失败-token无效401处理

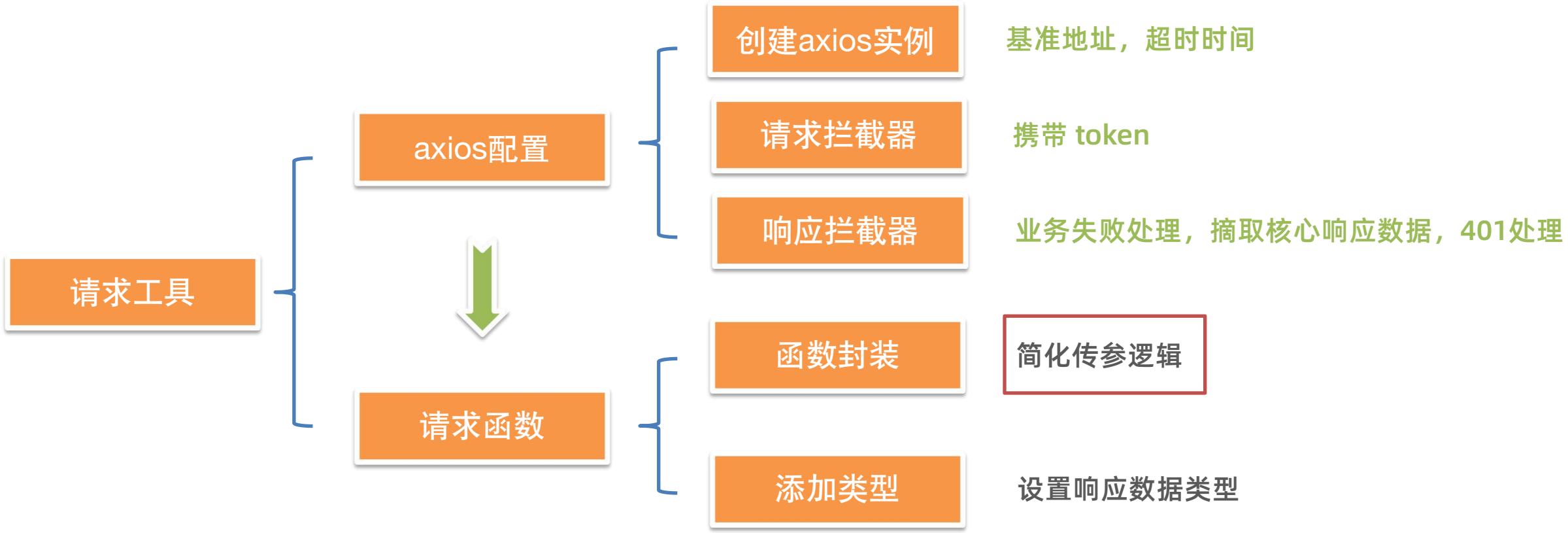


基础架构–请求工具–函数封装

基础架构-请求工具-请求函数



基础架构-请求工具-请求函数



基础架构-请求工具-请求函数-封装函数

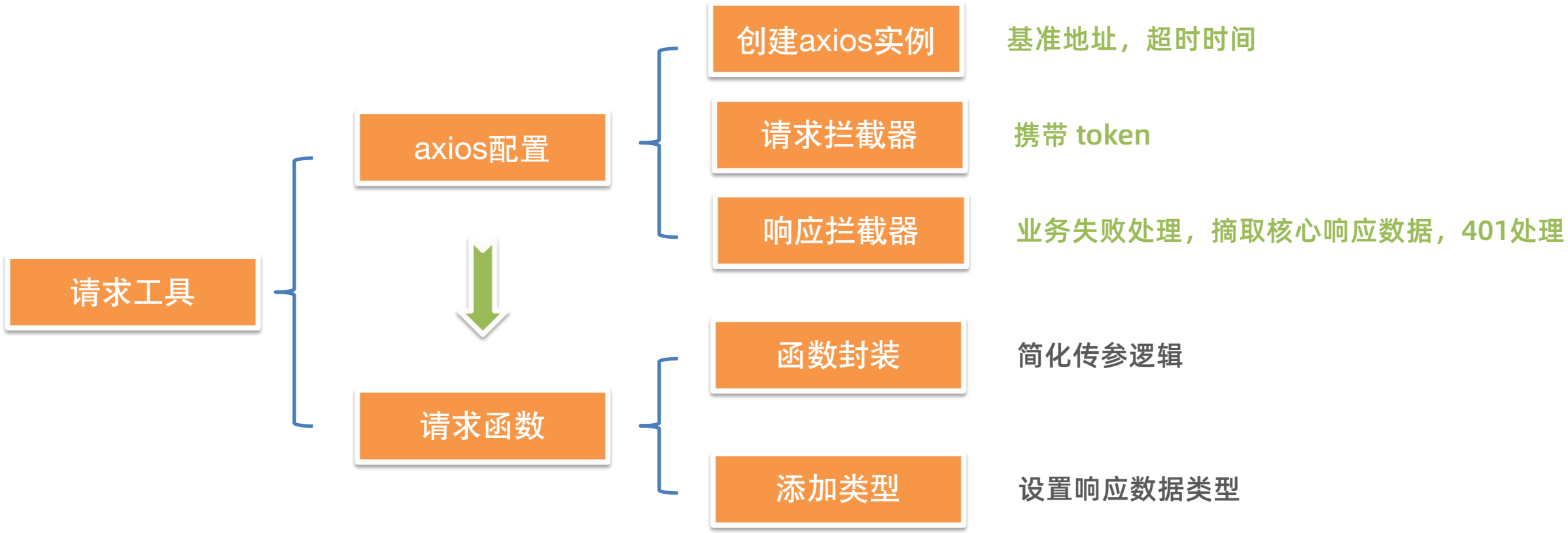
```
axios.request({
  url: '/login',
  method: 'post',
  data: {
    mobile: '13211112222',
    code: '123456'
  }
})

axios.request({
  url: '/articles',
  method: 'get',
  params: {
    type: 'hot'
  }
})
```

request('url', 'method', 'submitData')
返回值 promise

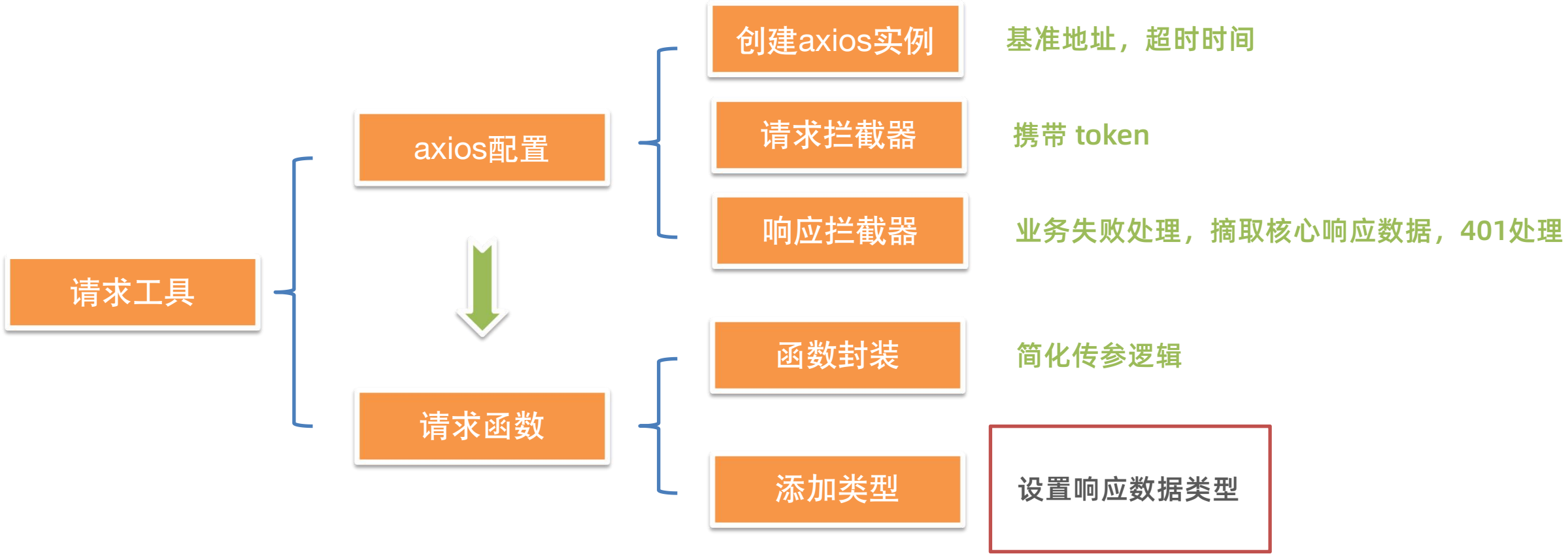
1. 不用传配置对象
2. 不用区分 data 和 params 函数内判断

基础架构–请求工具–函数封装

基础架构-请求工具-函数封装-设置类型



基础架构-请求工具-函数封装-设置类型



设置响应数据类型

```ts
// 用户信息
const { data: user } = await request<User>('/login', 'post', {})
// 文章信息
const { data: article } = await request<Article>('/article', 'get', {})
```

```ts
export const request = <T>(
  url: string,
  method: Method = 'get',
  submitData?: object
) => {
  return instance.request<any, Data<T>>({
    url,
    method,
    [method.toLowerCase() === 'get' ? 'params' : 'data']: submitData
  })
}
```

```ts
type Data<T> = {
  code: number
  message: string
  data: T
}
```

扩展：为什么使用axios.request()泛型第二个参数？

```ts
instance.interceptors.response.use(
  (res) => {
    // 3. 处理业务失败
    if (res.data.code !== 10000) {…
    }
    // 4. 摘取核心响应数据
    return res.data
  },
  (err: AxiosError) => {…
  }
)
```

```ts
type AxiosResponse = {
  data: any
  // 或者axios.request()第一个泛型参数类型
  headers: any
  config: any
  request?: any
  // ...
}
```

```ts
const result = await axios.request<any, { name: string }>({
  url: 'patient/myUser',
  method: 'get'
})
```

自定义响应数据后，采用泛型第二个参数设置即可

传智教育旗下高端IT教育品牌