优医问诊day09

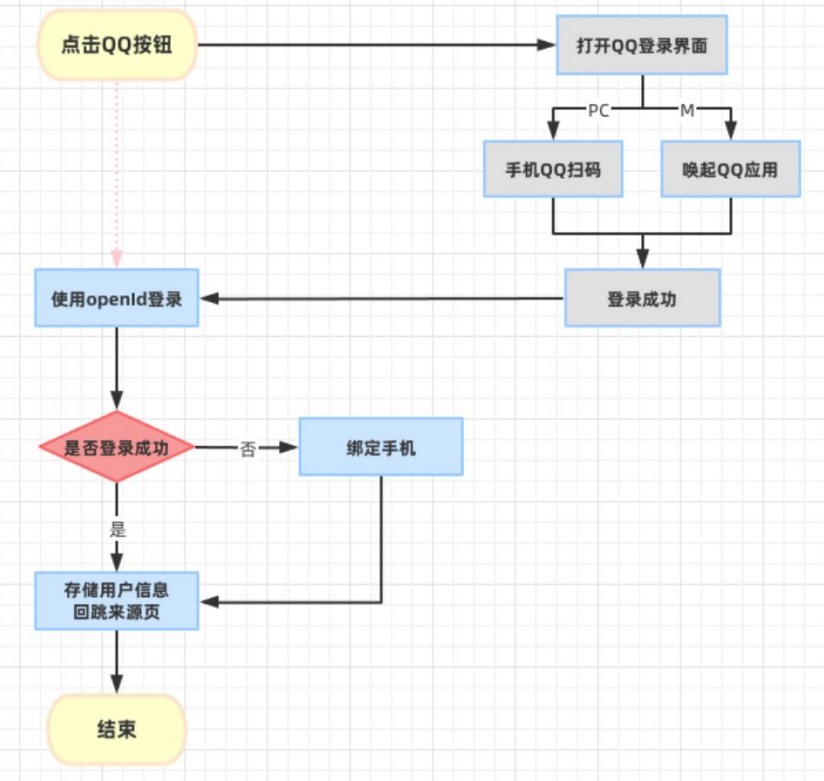
扩展

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

三方登录–QQ登录流程

三方登录-QQ登录流程



准备工作：

1. 注册 QQ 互联且实名认证，拥有已备案的合法域名。
2. 创建 web 应用，配置回跳地址
3. 得到 appid 和 回跳地址

```
# 测试用 appid
# 102015968
# 测试用 redirect_uri
# http://consult-patients.itheima.net/login/callback
```

去QQ登录且回跳 → 使用openId登录 → 验证码 composable → 绑定手机号

三方登录–去QQ登录且回跳

三方登录-去QQ登录且回跳

配置host和白名单 → 设置跳转QQ地址

hosts 文件
127.0.0.1 consult-patients.itheima.net

```ts
export default defineConfig({
  server: {
    port: 80,
    host: true
  },
```

```ts
// 白名单 router.ts
const wihteList = ['/login', '/login/callback']
```

1.2. 放置QQ登录按钮

在html页面需要插入QQ登录按钮的位置，粘贴如下代码：

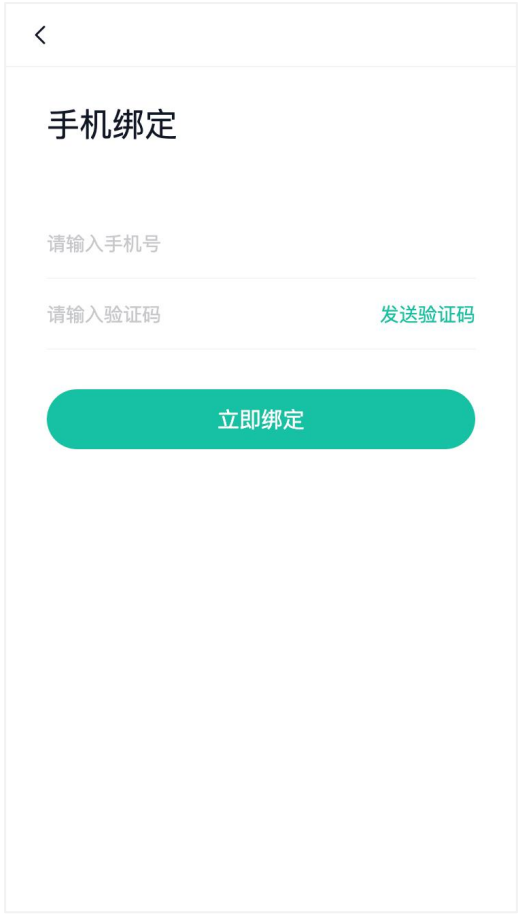
```html
<span id="qqLoginBtn"></span>
<script type="text/javascript">
  QC.Login({
    btnId:"qqLoginBtn"    //插入按钮的节点id
});
</script>
```

```vue
<div class="login-other">
  <van-divider>第三方登录</van-divider>
  <a
    class="icon"
    href="https://graph.qq.com/oauth2.0/authorize?client_id=102015968
  >
    <img src="@/assets/qq.svg" alt="" />
  </a>
</div>
```

```
# 测试用 appid
# 102015968
# 测试用 redirect_uri
# http://consult-patients.itheima.net/login/callback
```

三方登录–使用openId登录

三方登录-使用openId登录



获取openId → 给QC添加类型 → 使用openId登录 → 控制绑定手机界面

```
onMounted(() => {
  if (QC.Login.check()) {
    QC.Login.getMe((id: string) => {
      console.log(id)
    })
  }
})
```

```
type QCType = {
  Login: {
    check(): boolean
    getMe(cb: (openId: string) => void): void
  }
}

declare const QC: QCType
```

```
<template>
  <div class="login-page" v-if="isNeedBind">…
  </div>
</template>

<style lang="scss" scoped>
@import '@/styles/login.scss';
</style>
```

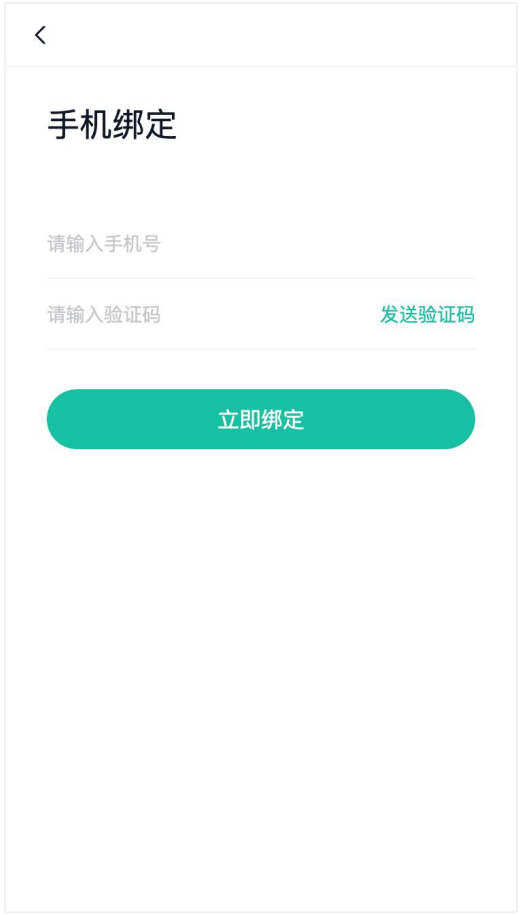
```
export const loginByQQ = (openId: string) =>
  request<User>('/login/thirdparty', 'POST', { openId, source: 'qq' })
```

```
/*global QC*/
import { loginByQQ } from '@/services/user'
import { onMounted, ref } from 'vue'

const openId = ref('')
const isNeedBind = ref(false)
onMounted(() => {
  if (QC.Login.check()) {
    QC.Login.getMe((id: string) => {
      openId.value = id
      loginByQQ(id)
        .then(() => {
          // 使用 openId 登录成功
        })
        .catch(() => {
          isNeedBind.value = true
        })
    })
  }
})
```

三方登录–验证码composable

三方登录-验证码composable



约定入参返参 → 登录页使用 → 绑定手机页使用

```ts
// 发送短信验证码
export const useMobileCode = (
  mobile: Ref<string>,
  type: CodeType = 'login'
) => {
  const time = ref(0)
  const form = ref<FormInstance>()
  let timer: number
  const onSend = async () => {
    // 验证：倒计时 手机号
    if (time.value > 0) return
    await form.value?.validate('mobile')
    await sendMobileCode(mobile.value, type)
    showToast('发送成功')
    time.value = 60
    // 开启倒计时
    if (timer) clearInterval(timer)
    timer = setInterval(() => {
      time.value--
      if (time.value <= 0) clearInterval(timer)
    }, 1000)
  }

  onUnmounted(() => {
    clearInterval(timer)
  })

  return { time, onSend, form }
}
```

```ts
// 验证码
const { form, onSend, time } = useMobileCode(mobile)
```

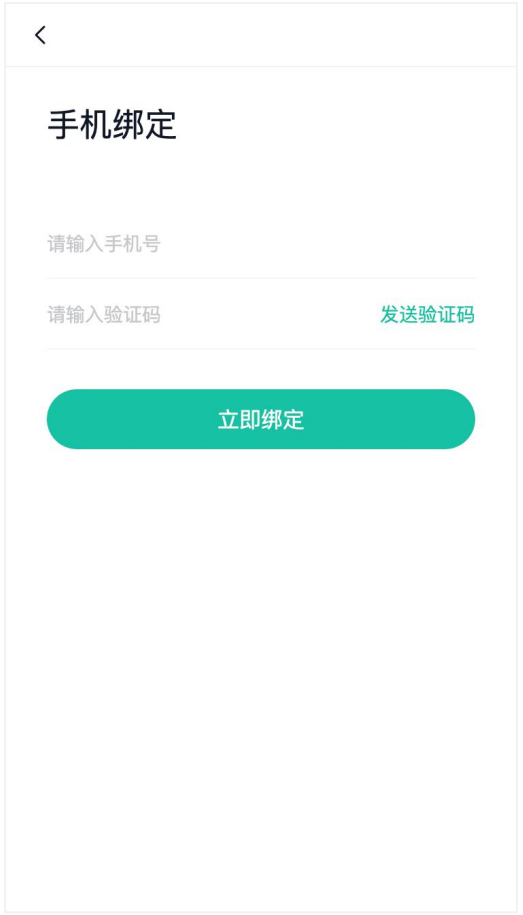
```vue
<van-field
  v-model="mobile"
  :rules="mobileRules"
  name="mobile"
  placeholder="请输入手机号"
></van-field>
<van-field
  v-model="code"
  :rules="codeRules"
  name="code"
  placeholder="请输入验证码"
>
```

```ts
const { form, time, onSend } = useMobileCode(mobile, 'bindMobile')
```

```vue
<template #button>
  <span class="btn-send" :class="{ active: time > 0 }" @click="onSend">
    {{ time > 0 ? `${time}s后再次发送` : '发送验证码' }}
  </span>
</template>
```

三方登录–绑定手机号

三方登录-绑定手机号



绑定手机API → 实现绑定手机号 → 回跳来源页

```ts
export const bindMobile = (data: {
  mobile: string
  code: string
  openId: string
}) => request<User>('/login/binding', 'POST', data)
```

```ts
const mobile = ref('')
const code = ref('')
const bind = async () => {
  const res = await bindMobile({
    mobile: mobile.value,
    code: code.value,
    openId: openId.value
  })
  loginSuccess(res)
}
```

```ts
loginByQQ(id)
  .then((res) => {
    // 使用 openId 登录成功
    loginSuccess(res)
  })
```

```vue
<van-divider>第三方登录</van-divider>
<a
  @click="store.setReturnUrl(route.query.returnUrl as string)"
```

```ts
const returnUrl = ref('')
const setReturnUrl = (url: string) => {
  returnUrl.value = url
}

return { user, setUser, delUser, setReturnUrl, returnUrl }
```

```ts
const store = useUserStore()
const router = useRouter()
const loginSuccess = (res: { data: User }) => {
  store.setUser(res.data)
  router.replace(store.returnUrl || '/user')
  showSuccessToast('登录成功')
  store.setReturnUrl('')
}
```

部署–区分开发环境生产环境

部署-区分开发环境生产环境

- 什么是开发环境和生产环境？
  - pnpm dev 开发环境
  - pnpm build 生产环境
- 我们有什么需求？
  - QQ登录回跳域名
    - 开发 http://consult-patients.itheima.net
    - 生产 https://cp.itheima.net
  - 支付回跳域名
    - 开发 http://consult-patients.itheima.net
    - 生产 https://cp.itheima.net
  - 标题
    - 优医问诊dev
    - 优医问诊

配置环境变量文件 → 项目代码中使用 → html模板页面使用

.env.development

```
VITE_APP_TITLE=优医问诊dev
VITE_APP_CALLBACK=http://consult-patients.itheima.net
```

.env.production

```
VITE_APP_TITLE=优医问诊
VITE_APP_CALLBACK=https://cp.itheima.net
```

```
import { createHtmlPlugin } from 'vite-plugin-html'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [
    createHtmlPlugin(),
```

index.html

```
<script
  src="https://connect.qq.com/qc_jssdk.js"
  data-appid="102015968"
  data-redirecturi="<%=VITE_APP_CALLBACK%>/login/callback"
></script>
```

index.html

```
<title><%=VITE_APP_TITLE%></title>
```

```
interface ImportMetaEnv {
  VITE_APP_TITLE: string
  VITE_APP_CALLBACK: string
}
```

src > router > index.ts > ...

```
// 全局的后置导航
router.afterEach((to) => {
  document.title = `${to.meta.title || ''}-${import.meta.env.VITE_APP_TITLE}`
  NProgress.done()
})
```

src > components > CpPaySheet.vue > ...

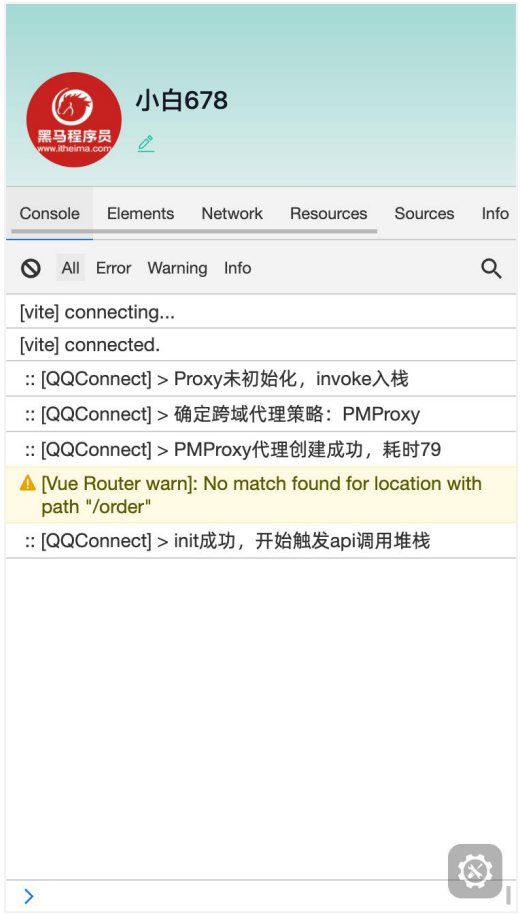
```
    orderId: props.orderId,
    payCallback: import.meta.env.VITE_APP_CALLBACK + props.payCallback
  })
  window.location.href = res.data.payUrl
}
```

src > views > Login > index.vue > ...

```
// QQ登录地址
const url =
  'https://graph.qq.com/oauth2.0/authorize?client_id=102015968&response_type
  encodeURIComponent(import.meta.env.VITE_APP_CALLBACK + '/login/callback')
```

07 部署–真机调试

部署-真机调试

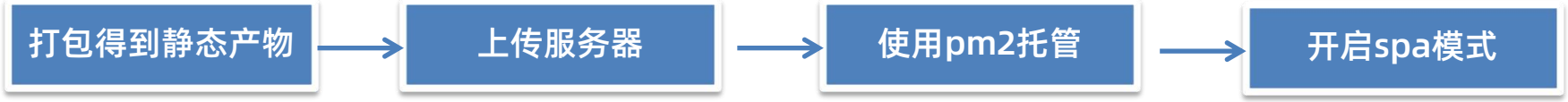


使用eruda → 仅开发环境使用

```
<% if(DEV){ %>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/eruda/2.4.1/eruda.min.js"></script>
  <script>eruda.init()</script>
<% } %>
```

部署–pm2托管

部署-pm2托管（手动）




```
pnpm build
```

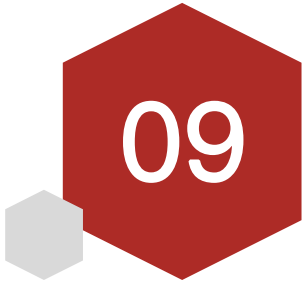
服务器一般是 linux 系统，使用 XFTP 进行文件的上传，或其他工具

```
npm i pm2 -g
```

```
# pm2 serve 目录 端口  --name 服务名称
pm2 serve ./ 8080 --name my-cp-server
```

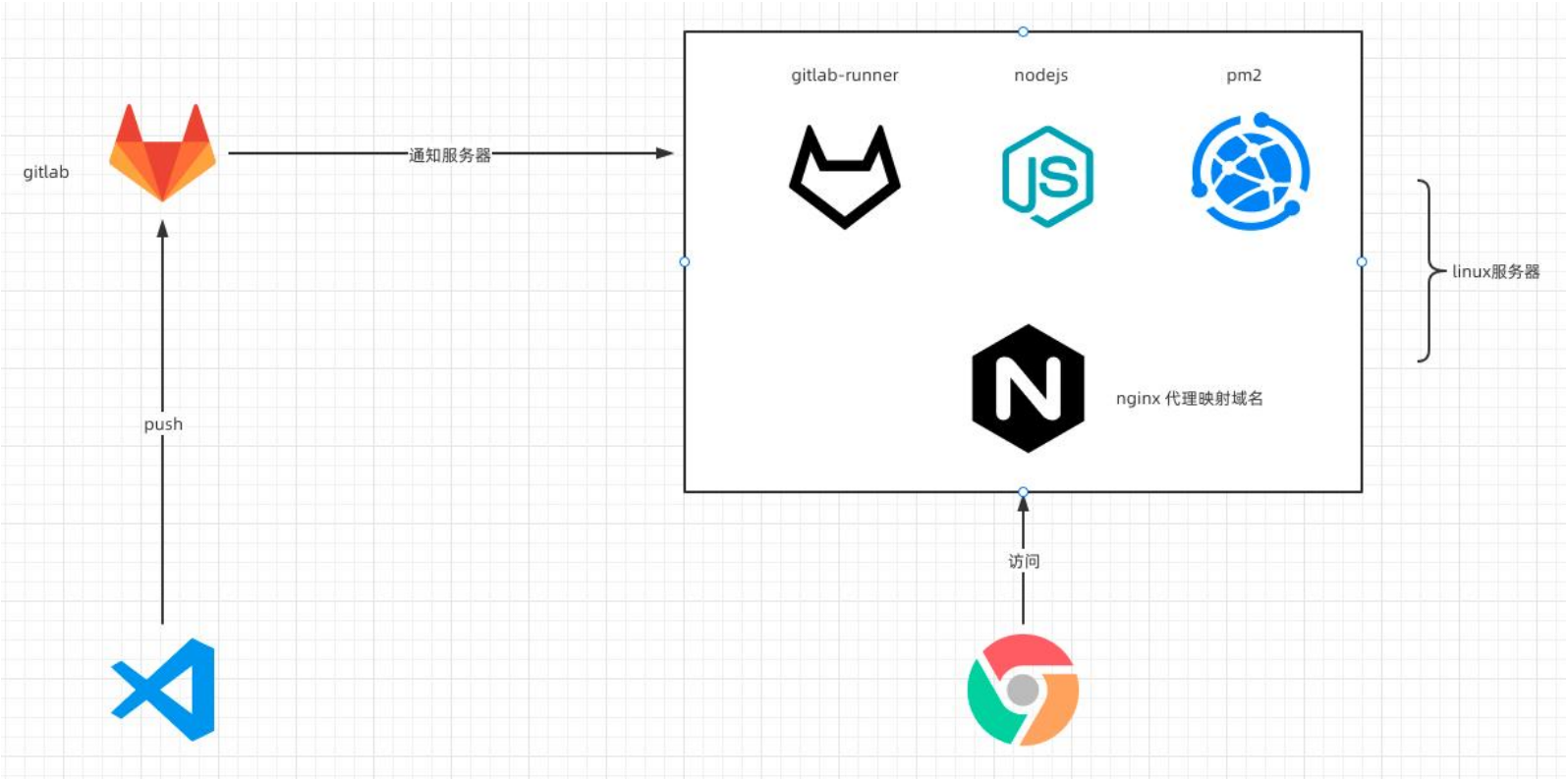
```
pm2 serve --spa ./ 8080 --name my-cp-server
```

```
# 查看服务列表
pm2 list
# 删除服务
pm2 delete my-cp-server
```

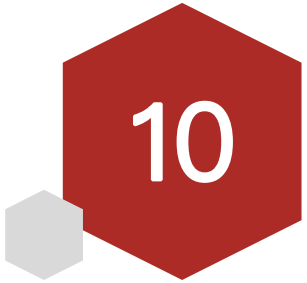


部署–gitlab自动部署

部署-gitlab自动部署(演示)



1. 本地Vscode编写代码，gitlab.yml 配置
2. gitlab是企业版，内部部署
3. Linux服务器
- 安装 gitlab-runner 用于拉取仓库代码
- 安装 Nodejs 用于打包项目
- 安装 pm2 用于启动静态资源托管，守护进程
4. 运维使用 Nginx 进行域名代理
5. 用户通过浏览器访问服务



扩展–mock数据

扩展-mock数据

背景：在开发过程中后台未开发好接口，前端要推进业务进行需要模拟数据

安装mock插件 → vite配置 → 模拟接口 → Mock随机数据

pnpm i vite-plugin-mock mockjs -D

```
viteMockServe({
  mockPath: './src/mock',
  localEnabled: true
})
```

code	number	非必须		请求成功10000标志
message	string	非必须		请求成功
- data	object []	非必须		返回数据
title	string	非必须		消息的标题（聊天-发送消息的医生姓名）
id	string	非必须		消息id
type	string	非必须		消息类型1系统通知2交易通知3聊天消息
content	string	非必须		通知内容
status	string	非必须		是否已读0未读1已读
createTime	string	非必须		创建时间

/patient/message/sys/list GET

```
import Mock from 'mockjs'

const rules = [
  {
    url: '/patient/message/sys/list',
    method: 'get',
    timeout: 1000,
    response: () => {
      return {
        code: 10000,
        message: '模拟数据成功',
        data: []
      }
    }
  }
]

export default rules
```

在配置v好requet基准地址后，需要带上本地开发环境的域名才能走mock

```
const data = []
for (let i = 0; i < 10; i++) {
  data.push(
    Mock.mock({
      id: '@id',
      avatar: '@image("100x100")',
      title: '@ctitle(3,10)',
      content: '@ctitle(10,40)',
      createTime: '@datetime()',
      status: '@integer(0,1)',
      type: '@integer(1,3)',
    })
  )
}
return {
  code: 10000,
  message: '模拟数据成功',
  data
}
```

扩展–单元测试

扩展-单元测试

测试作用：自动化测试能够预防无意引入的 bug
何时测试：越早越好，拖得越久，应用就会有越多的依赖和复杂性，想要开始添加测试也就越困难。
测试类型：单元测试、组件测试、端对端测试

单元测试：检查给定函数、类或组合式函数的输入是否产生预期的输出或副作用。

安装vitest → 配置命令 → 测试filter.ts

```
"scripts": {
  "test": "vitest",
```

```
import { test, expect } from 'vitest'
import { getIllnessTimeText, getConsultFlagText } from '../filter'
import { IllnessTime } from '../../enums/index'

test('getIllnessTimeText', () => {
  const text = getIllnessTimeText(IllnessTime.Month)
  expect(text).toBe('一月内')
})

test('getConsultFlagText', () => {
  const text = getConsultFlagText(0)
  expect(text).toBe('没就诊过')
})
```

扩展–组件测试

扩展-组件测试

对于 视图 的测试：根据输入 prop 和插槽断言渲染输出是否正确。

对于 交互 的测试：断言渲染的更新是否正确或触发的事件是否正确地响应了用户输入事件。

组件测试通常涉及到单独挂载被测试的组件，触发模拟的用户输入事件，并对渲染的 DOM 输出进行断言。安装：happy-dom @testing-library/vue


```
import { test, expect } from 'vitest'
import { render } from '@testing-library/vue'
import CpRadioBtn from '../CpRadioBtn.vue'

test('CpRadioBtn render', async () => {
  const wrapper = render(CpRadioBtn, {
    props: {
      options: [
        { label: '选项一', value: 1 },
        { label: '选项二', value: 2 }
      ],
      modelValue: 1
    }
  })

  // 测试渲染
  wrapper.getByText('选项一')
  wrapper.getByText('选项二')

  // 测试默认激活
  expect(wrapper.queryByText('选项一')?.classList.contains('active')).toBe(true)

  // 双向绑定(测自定义事件)
  wrapper.queryByText('选项二')?.click()
  expect(wrapper.emitted()['update:modelValue'][0]).toEqual([2])
  await wrapper.rerender({
    modelValue: 2
  })
  expect(wrapper.queryByText('选项二')?.classList.contains('active')).toBe(true)
})
```

单独挂载组件 → 测试渲染 → 测试props → 测试自定义事件

传智教育旗下高端IT教育品牌