河南中医药大学信息技术学院（智能医疗行业学院）智能医学工程专业《互联网医疗服务开发》课程

第12章：Web测试与发布

冯顺磊

河南中医药大学信息技术学院（智能医疗行业学院）
河南中医药大学信息技术学院智能医疗教研室
https://aitcm.hactcm.edu.cn
2025/5/16

本章概要

- Web测试
- Web发布

1. Web测试 1.1 什么是Web测试?

- Web测试属于软件测试的范畴，是针对Web服务特征进行的软件测试工作。由于Web应用与用户直接相关，通常需要承受长时间的大量操作，因此Web项目的功能和性能都必须经过可靠的验证，这就要对Web项目开展全面测试。
- Web测试的难点在于Internet和Web媒体的不可预见性，例如测试人员无法准确的定义用户的网络接入状况、使用的浏览器和操作系统、客户计算机的类型和配置信息、用户所处的国家和地区、所使用的语言以及个人文化宗教信仰等，使得Web测试变得更加困难。

1. Web测试 1.1 什么是Web测试?

- 通常Web测试可以分为以下六个部分内容。
  - 功能测试
    - 功能测试是对Web具体功能进行测试，主要包括链接测试、表单测试、数据验证测试、Cookies测试、Web支持系统（如数据库等）的测试、特定功能流程（例如电子商城中的下单、购买、支付、验收、确认等全流程）的测试等。
  - 性能测试
    - 性能测试是对Web在高并发、高压力的情况下服务情况的测试，主要包括连接速度测试、负载测试、压力测试等。
  - 用户界面测试
    - 用户界面测试主要是对Web的UI进行系统的测试，以确保用户访问的UI能够正常传递Web信息。主要包括导航测试、图形测试、动画测试、内容测试以及用户交互测试等。

1. Web测试 1.1 什么是Web测试?

- 通常Web测试可以分为以下六个部分内容。
  - 兼容性测试
    - 兼容性测试主要是针对Web访问者的不可预见性而进行的测试，从而确保任意用户在任何地方通过多样终端均能够正常访问Web。主要包括操作系统兼容性测试、浏览器兼容性测试、分辨率兼容性测试、以太网接入环境兼容性测试、多智能终端兼容性测试、多语言支持测试等。
  - 安全测试
    - 安全测试主要对Web安全性和表单安全性进行测试，从而保障Web能够稳定的提供服务。安全测试包括传输安全、表单安全、日志安全、脚本安全、业务接口安全等方面的测试。
  - 接口测试
    - Web通常情况下都不是孤立存在的，往往会有许多对外部服务的调用。例如位置服务的Web会有对Google、百度等地图的调用，电子商务网站会有对信用卡、支付网关的调用等。对于Web对外部数据接口调用要进行全面测试，以保障业务可用性和安全性。

1. Web测试 1.1 什么是Web测试?

- Web应用因为其复杂的结构特点和不可预见的媒介特征，从而导致其故障分布、故障原因较为复杂，要发现、分析、排除故障通常需要进行多方面的测试。
- Web测试不但需要检查和验证Web应用是否按照设计的要求运行，还要测试Web应用在不同终端的显示是否正常，Web应用是否可用，Web应用是否安全。
- 一般Web测试目的主要有以下几个方面。
  - 验证Web需求和功能是否得到完整实现以及在正常和非正常情况下的功能显示状态；
  - 发现Web的缺陷、错误及不足，进而较为准确地推测出Web应用潜在的缺陷数，获取Web应用的质量信息；
  - 根据当前发现的问题进行分析，为Web应用的下一版本提供支持；
  - 发现影响用户使用的错误，预防用户访问或使用Web应用时可能出现的问题；
  - 通过测试结果数据、测试问题记录等数据，了解并分析Web应用存在的问题，提高Web开发效率；
  - 验证Web是否可以发布并使用。

1. Web测试 1.2 用户界面测试

- 用户界面测试（英文名为User interface testing），简称UI测试，测试用户界面的功能模块的布局是否合理，整体风格是否一致，各个控件的放置位置是否符合客户使用习惯，更重要的是要符合用户操作便捷，导航简单易懂，界面文字正确，命名统一规范，页面美观大方，图文排版整洁等基本要求。
- 界面是软件与用户交互的最直接的层，界面的好坏决定用户对软件的第一印象。界面测试的目标在于确保用户界面向用户提供了适当的访问和浏览测试对象功能的操作。一般界面测试主要包括导航测试、图形测试、内容测试和整体界面测试。

1. Web测试 1.2 用户界面测试

- 导航测试
  - 导航描述了用户在一个页面内操作的方式，在不同的用户接口控制之间（例如按钮、对话框、列表和窗口等），或在不同的连接页面之间。通过考虑下列问题，可以决定一个Web应用系统是否易于导航：导航是否直观、Web系统的主要部分是否可通过主页存取、Web系统是否需要站点地图、搜索引擎或其他的导航帮助等。
  - 在一个页面上放太多的信息往往起到与预期相反的效果。Web应用系统的用户趋向于目的驱动，很快地扫描一个Web应用系统，看是否有满足自己需要的信息，如果没有，就会很快地离开。很少有用户愿意花时间去熟悉Web应用系统的结构，因此Web应用系统导航要尽可能地准确。
  - 导航的另一个重要方面是Web应用系统的页面结构、导航、菜单、连接的风格是否一致。确保用户凭直觉就知道Web应用系统里面是否还有内容，内容在什么地方。
  - Web应用系统的层次一旦决定，就要着手测试用户导航功能，让最终用户参与此测试，效果将更加明显。

1. Web测试 1.2 用户界面测试

- 图形测试
  - 在Web应用系统中，适当的图片和动画既能起到广告宣传的作用，又能起到美化页面的作用。一个Web应用系统的图形可以包括图片、动画、边框、颜色、字体、背景、按钮等。图形测试的内容有以下五个方面。
    - 要确保图形有明确的用途，图片或动画不要胡乱地堆在一起。Web应用系统的图片尺寸要尽量地小，并且要能清楚地说明某件事情，一般都链接到某个具体的页面。
    - 验证所有页面字体的风格是否一致。
    - 背景颜色应该与字体颜色和前景颜色相搭配。
    - 图片大小和质量是很重要的因素，一般采用JPG、PNG压缩，尽量减小到50kb以下。
    - 需要验证的文字与图片的混排是否正确。如果说明文字指向右边的图片，应该确保该图片出现在右边。不要因为使用图片而使窗口和段落排列古怪或者出现孤行。

1. Web测试 1.2 用户界面测试

- 内容测试
  - 内容测试用来检验Web应用系统提供信息的正确性、准确性和相关性。
  - 信息的正确性是指信息是可靠的还是误传的。例如，在商品价格列表中，错误的价格可能引起财务问题甚至导致法律纠纷；信息的准确性是指是否有语法或拼写错误，这种测试通常使用一些文字处理软件来进行，例如使用Microsoft Word的“拼音与语法检查”功能；信息的相关性是指是否在当前页面可以找到与当前浏览信息相关的信息列表或入口，也就是一般Web站点中的“相关文章列表”或“相关推荐”。

1. Web测试 1.2 用户界面测试

- 整体界面测试
  - 整体界面是指整个Web应用系统的页面结构设计，是给用户的一个整体感。例如：当用户浏览Web应用系统时是否感到舒适，是否凭直觉就知道要找的信息在什么地方、整个Web应用系统的设计风格是否一致等。
  - 对整体界面的测试过程，其实是一个对最终用户进行调查的过程。一般Web应用系统采取在主页上做一个调查问卷的形式，来得到最终用户的反馈信息。
  - 对所有的用户界面测试来说，都需要有外部人员（与Web应用系统开发没有联系或联系很少的人员）的参与，最好是最终用户的参与。

1. Web测试 1.3 兼容性测试

- 兼容性测试是指被测项目在特定的硬件平台上、不同的应用软件间、不同的操作系统平台上、不同的网络环境中是否能很好地运行测试。简单的说，兼容性测试是指测试某个新开发的软件在某一个特定环境下与各种软件的协调时，软件能否很好的运行。
- Web兼容性测试主要分为平台兼容性测试、浏览器兼容性测试、分辨率兼容性测试等。

1. Web测试 1.3 兼容性测试

- 平台兼容性测试
  - 市场上有很多不同的操作系统类型，最常见的有Windows、Unix、Mac、Linux等。Web应用系统的最终用户究竟使用哪一种操作系统，取决于用户系统的配置，而Web开发者需要保障的是Web应用能够在所有用户眼里看起来都是完美的，因此需要查看页面在不同平台不同浏览器下的测试截图。
  - 推荐工具
    - BrowserStack
    - BrowserStack是云端跨浏览器及移动应用测试平台，其核心定位是帮助开发者和测试人员在无需搭建本地设备实验室的情况下，高效验证应用在多端环境下的兼容性与稳定性。
    - 提供超过3500种真实桌面和移动浏览器组合（覆盖Chrome、Firefox、Edge、Safari、IE等主流浏览器及新旧版本），以及20,000+真实iOS、Android、Windows、macOS设备，确保测试结果与用户实际使用环境一致。
    - https://www.browserstack.com/

1. Web测试 1.3 兼容性测试

- 浏览器兼容性测试
  - 浏览器是Web客户端最核心的构件，来自不同厂商的浏览器对HTML标签（如表格间距、框架处理）、CSS样式表（如编写规范）、JavaScript（网页元素名称、方法名称等）、ActiveX控件、浏览器插件plug-ins和安全性等很多方面都有着不同程度的支持。另外，框架和层次结构风格在不同的浏览器中也有不同的显示，甚至根本不显示。
  - 主流PC浏览器：Chrome（市占率最高）、Firefox（开源代表）、Edge（Windows默认）、Safari（macOS/iOS默认）、IE（部分企业系统仍在用旧版）。
  - 移动端浏览器：iOS Safari（iPhone/iPad）、Android Chrome（主流安卓设备）、国内定制浏览器（如UC、QQ浏览器）。

1. Web测试 1.3 兼容性测试

- 分辨率兼容性测试
  - 用户终端设备的不同，用户环境的分辨率也不相同，而分辨率对Web页面的展示效果影响很大，不同大小的分辨率会使页面排版、字体样式等的显示显著不同。
  - 推荐工具：
    - Responsive Design Checker
    - 免费在线工具，可预览网站在移动设备、平板电脑、桌面设备和自定义分辨率设备上的展示效果。能并排对比不同设备的网站预览效果，还可前往 Web Vitals 查看基础数据。
    - https://www.websiteplanet.com/zh-hans/webtools/responsive-checker/
    - 浏览器自带的开发者工具

1. Web测试 1.4 功能测试

- 功能测试就是结合规格说明的要求，保证功能上正确无误，其内容包括HTML语法检查、链接检查、表格测试、发送请求以及接受服务回传信息的处理等。
- 根据测试要求的难易程度的不同，功能测试又可分为简单功能测试、任务特征测试、边界测试、强制错误情况测试、探测性测试等，以确保不同层次上的网站或网络应用程序运行的质量。简单功能测试主要做一些链接可达性的检查工作；任务特征测试是根据任务的交互性、不确定性等不同特征，进行有针对性的测试；边界测试是在输入数据域的边界抽取数据进行测试；强制错误情况测试是根据设计时的规格说明，人为输入明显错误的数据，然后观测系统的运行情况，主要测试系统的容错性；探测性测试就是边设计边执行测试，试探性地前进几步并及时调整。
- 网站功能测试相对于其他类型测试来说需要的测试环境要素相对较少，所占用的测试资源也相对较少，只要需求文档明确定义各项功能即可。

1. Web测试 1.4 功能测试

- 链接测试
  - 链接是Web应用系统的一个主要特征，它是在页面之间切换和指导用户访问其他地址页面的主要手段。链接测试可分为三个方面。
    - 测试所有链接是否按指示的那样确实链接到了该链接的页面；
    - 测试链接目标的页面是否存在；
    - 保证Web应用系统上不存在孤立页面，所谓孤立页面是指没有任何链接指向该页面。
  - 链接测试必须在集成测试阶段完成，也就是说，在整个Web应用系统的所有页面开发完成之后进行链接测试。链接测试要注意以下几点。
    - 链接有站内站外之分。在测试环境中，有时站外链接总是不能访问的，因此要区分哪些链接失败是Bug，哪些是测试环境的限制。
    - 表单链接是一种特殊的链接，要确认它传输的参数是否正确。
    - 有些链接是客户端程序生成或者控制的，甚至是动态的。这类链接的测试最好在测试客户端程序时进行。

1. Web测试 1.4 功能测试

- 链接测试
  - 用户要求保证Web应用系统的完整性和可靠性，必须对它所有页面的链接情况用穷举法来测试。如果手工测试需要很大的工作量而且非常枯燥。就需要考虑采用自动化测试工具来完成测试工作。
  - 推进工具
    - Xenu Link Sleuth是一款轻量级、免费的网站死链接检测工具，被广泛用于网站维护、SEO优化等场景。
    - https://download.cnet.com/xenu-s-link-sleuth/3000-10248_4-10020826.html

1. Web测试 1.5 性能测试

- 性能测试是通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。性能测试在软件的质量保证中起着重要的作用，它包括的测试内容丰富多样。中国软件评测中心将性能测试概括为三个方面：应用在客户端性能的测试、应用在网络上性能的测试和应用在服务器端性能的测试。通常情况下，三方面有效、合理的结合，可以达到对系统性能全面的分析和瓶颈的预测。

1. Web测试 1.5 性能测试

- 性能测试的目的是验证软件系统是否能够达到用户提出的性能指标，同时发现软件系统中存在的性能瓶颈，优化软件，最后起到优化系统的目的，其目的包括以下几个方面。
  - 评估系统的能力：测试中得到的负荷和响应时间数据可以被用于验证所计划的模型的能力，并帮助作出决策。
  - 识别体系中的弱点：受控的负荷可以被增加到一个极端的水平，并突破它，从而修复体系的瓶颈或薄弱的地方。
  - 系统调优：重复运行测试，验证调整系统的活动得到了预期的结果，从而改进性能。
  - 检测软件中的问题：长时间的测试执行可导致程序发生由于内存泄露引起的失败，揭示程序中隐含的问题或冲突。
  - 验证稳定性（resilience）、可靠性（reliability）：在一个生产负荷下执行测试一定的时间，从而评估系统稳定性和可靠性是否满足要求。

2. Web发布 2.1 Vite

- Vite是一种新型前端构建工具，能够显著提升前端开发体验。主要由两部分组成开发服务器和构建指令。开发服务器基于原生 ES 模块提供丰富的内建功能，如快速模块热更新（HMR）。构建指令使用预配置的 JavaScript 打包器（Rollup.js） 打包代码，输出用于生产环境的高度优化过的静态资源。Vite 还提供强大的扩展性，可通过插件 API 和 JavaScript API 进行扩展和项目优化，并提供完整的类型支持。Vite 是一个基于 ES modules 的前端构建工具，主要特点包括快速的冷启动、按需编译、热更新、插件化，下面是特点，以下是其详细说明。
  - 极速的服务启动：使用原生ESM文件，无需打包。
  - 轻量快速的热重载：无论应用程序大小如何，都始终极快的模块热重载（HMR）。
  - 丰富的功能：对 TypeScript、JSX、CSS 等支持直接引用。
  - 优化的构建：可选“多页应用”或“库”模式的预配置Rollup.js构建。
  - 通用的插件：在开发和构建之间共享Rollup-superset插件接口。
  - 完全类型化的API：灵活的API和完整TypeScript类型。

2. Web发布 2.1 Vite

- Vite 和传统的打包工具（如 webpack）在构建方式、编译方式、热更新方式、插件化方式和支持的框架等方面都有所不同。Vite 更加轻量、快速、灵活，适合于开发小型应用和组件库，而 webpack 更加适合大型应用的构建和优化。
- Vite 和传统的打包工具（如 webpack）有以下不同点。
  - 构建方式：Vite 采用基于浏览器原生 ES 模块的开发模式，在浏览器请求对应模块时，即时地编译和执行对应的代码，而 webpack 在构建时将所有模块打包成一个或多个bundle.js文件。
  - 编译方式：Vite 根据需要动态地编译模块，而 webpack 将所有模块都打包到一个文件中。
  - 热更新方式：Vite 支持热更新，可以在开发时实时更新修改后的代码，而 webpack 需要重新编译整个应用才能看到修改后的效果。
  - 插件化方式：Vite 支持插件化，可以通过插件扩展 Vite 的功能，而 webpack 则需要通过 loader 和 plugin 扩展其功能。
  - 支持的框架：Vite 支持多种前端框架，包括 Vue、React、Angular 等，而 webpack 则需要通过相应的 loader 和 plugin 来支持不同的框架。

2. Web发布 2.1 Vite

- 项目构建是软件开发过程中的一个重要环节，项目构建是基于代码转换、资源优化、依赖管理、环境变量配置、自动化任务、模块化等，具体解释如下。
  - 代码转换：现代Web开发经常涉及到使用ES6+、TypeScript、CSS预处理器（如Sass、Less）、JSX等高级语言或语法。这些高级特性虽然提高开发效率和代码的可维护性，但浏览器并不直接支持。项目构建过程需要将这些高级代码转换为浏览器能够理解的格式（如JavaScript、CSS）。
  - 资源优化：在开发过程中，为调试和模块化开发，通常会使用多个JS、CSS文件、图片、字体等资源文件。但在生产环境下，过多的网络请求会导致页面加载速度变慢。项目构建过程可以对这些资源进行优化（合并文件、压缩代码、图片优化等），减少请求次数和文件大小，提高页面加载速度。
  - 依赖管理：引入的第三方库和框架依赖项的管理变得复杂且容易出错。Vite提供依赖管理功能，可以自动解析和加载项目中的依赖项，确保正确性和一致性。
  - 环境变量：不同的环境（如开发环境、测试环境、生产环境）需要不同的配置。Vite构建项目根据当前环境设置不同的环境变量，实现灵活的配置管理。
  - 自动化任务：Vite构建项目会执行自动化任务，如代码格式化、代码检查、单元测试等。提高开发效率，减少人为错误。
  - 模块化：现代Web开发强调模块化，即将代码分割成可复用的模块。Vite支持模块化开发，可以自动处理模块之间的依赖关系，确保代码的正确组织和加载。

2. Web发布 2.1 Vite

- 在 Vite 项目中，构建配置涉及多个方面，vite.config.js时Vite的配置文件，其常见的配置项如下。
  - vite.config.js文件配置
    - base：指定项目的公共基础路径。例如，如果设置为“/my-app/”，则所有资源的URL都会基于此路径。
    - server：
      - port：定义开发服务器监听的端口号。
      - host：指定服务器运行的主机，例如“localhost”或特定的IP地址。
      - https：启用或禁用HTTPS协议。
      - proxy：配置代理规则，用于解决跨域请求问题。
    - build：
      - outDir：明确构建输出的目录，默认是“dist”。
      - assetsDir：指定静态资源（如图片、字体）在输出目录中的子文件夹名称。
      - minify：选择代码压缩的方式，“terser”提供更精细的控制，“esbuild”速度更快。
      - rollupOptions：深入定制Rollup的构建选项，例如控制代码拆分、模块格式等。
    - plugins：
      - 例如，“vite-plugin-vue”用于支持Vue框架，“vite-plugin-svg-icons”用于优化SVG图标的处理。

2. Web发布 2.1 Vite

- 在 Vite 项目中，构建配置涉及多个方面，vite.config.js时Vite的配置文件，其常见的配置项如下。
  - 处理 CSS
    - 使用 sass ： 安装sass相关依赖后，通过“vite.config.js”中的css对象进行配置，指定预处理器选项。
    - 引入全局 CSS ： 可以在配置中指定全局样式文件的路径，确保在项目的每个组件中都能应用。
  - 处理静态资源
    - 图片：可以设置不同大小图片的加载策略，如懒加载。
    - 字体：配置字体文件的处理方式，包括字体格式的转换和优化。
  - 环境变量
    - 通过在项目根目录创建“.env”文件来设置不同环境的变量。例如， “.env.development”用于开发环境，“.env.production”用于生产环境。
    - 在代码中使用“import.meta.env.VARIABLE_NAME”来获取环境变量的值。

2. Web发布 2.1 Vite

- 在 Vite 项目中，构建配置涉及多个方面，vite.config.js时Vite的配置文件，其常见的配置项如下。
  - 自定义别名
    - 例如，将“@/”定义为“src/”的别名，方便在代码中更简洁地引用项目中的模块和文件。
    - 通过对这些构建配置的精细调整和优化，使Vite更好地适应项目的特定需求，提高开发效率和构建产物的质量。

JavaScript ▾ 固定高度 复制 设置

```javascript
import { defineConfig } from 'vite';
import vue from '@vitejs/plugin-vue';

export default defineConfig({
  plugins: [vue()],
  root: './',
  base: '/my-app/',
  publicDir: 'public',
  resolve: {
    alias: {
      '@': '/path/to/src',
    },
    extensions: ['.js', '.ts', '.jsx', '.tsx', '.json', '.vue'],
  },
  server: {
    host: 'localhost',
    port: 3000,
    https: false,
    proxy: {
      '/api': {
        target: 'https://example.com',
        changeOrigin: true,
        rewrite: (path) => path.replace(/^\/api/, ''),
      },
    },
  },
  build: {
    outDir: 'dist',
    minify: 'terser',
    terserOptions: {
      compress: {
        drop_console: true,
        drop_debugger: true,
      },
    },
  },
});
```

2. Web发布 2.2 项目发布流程

- 项目准备
  - 测试：在项目发布前，进行全面的测试以确保项目的稳定性和功能完整性。测试包括单元测试、集成测试、系统测试等。
  - 文档准备：编写项目文档，包括用户手册、技术文档、安装指南等，确保用户能够顺利使用项目。
- 构建与打包
  - 使用构建工具生成项目的生产环境版本。
  - 打包项目文件，以便于部署和分发。
- 部署
  - 服务器准备：根据项目需求选择合适的服务器，并配置好相应的环境（如数据库、Web服务器等）。
  - 上传文件：将打包好的项目文件上传到服务器上。
  - 配置反向代理：设置反向代理服务器，以便将用户请求转发到实际的服务器地址，同时可以实现负载均衡、SSL加密等功能。
  - 反向代理在进行项目发布时不是必须要设置的内容，根据实际的部署需求选择。

2. Web发布 2.2 项目发布流程

- 域名与访问设置
  - 配置域名记录，域名记录指向服务器的IP地址。
  - 修改项目中的配置文件，确保项目能够正确识别和使用配置的域名。
- 灰度发布与测试
  - 在部分用户或特定环境中进行灰度发布，收集用户反馈和测试数据。
  - 根据反馈和数据调整项目，优化性能和用户体验。
- 正式发布
  - 完成所有测试和调整后，正式发布访问。
  - 发布后持续监控项目运行情况，及时处理可能出现的问题。

2. Web发布 2.2 项目发布流程

- 稳定性与安全性
  - 确保项目在发布前经过充分的测试，避免出现严重的错误或漏洞，加强项目的安全防护措施，防止网络攻击和数据泄露。
- 备份与恢复
  - 定期备份项目数据和配置文件，防止数据丢失或损坏，制定数据恢复计划，在出现问题时能够迅速恢复项目运行。
- 持续更新与维护
  - 根据项目运行情况和用户反馈，及时修复发现的问题和漏洞，持续优化项目功能和性能，提升用户体验和满意度。
- 合规性
  - 确保项目在发布前符合相关法律法规和行业标准的要求，涉及用户隐私和数据保护的项目，要特别关注数据安全和隐私保护方面的合规性要求。

2. Web发布 2.2 项目发布流程

- 稳定性与安全性
  - 确保项目在发布前经过充分的测试，避免出现严重的错误或漏洞，加强项目的安全防护措施，防止网络攻击和数据泄露。
- 备份与恢复
  - 定期备份项目数据和配置文件，防止数据丢失或损坏，制定数据恢复计划，在出现问题时能够迅速恢复项目运行。
- 持续更新与维护
  - 根据项目运行情况和用户反馈，及时修复发现的问题和漏洞，持续优化项目功能和性能，提升用户体验和满意度。
- 合规性
  - 确保项目在发布前符合相关法律法规和行业标准的要求，涉及用户隐私和数据保护的项目，要特别关注数据安全和隐私保护方面的合规性要求。

2. Web发布 2.3 使用Vite发布管理系统

- 使用Vite发布一个管理系统，是使用Vite作为构建和开发服务器来开发一个管理系统前端项目，并在开发完成后将其构建成可以部署到服务器上的静态文件。使用Vite发布一个管理系统需要准备环境、创建项目、进入项目并安装依赖、开发管理系统、本地开发、 构建项目、部署。具体步骤如下。
  - 环境准备
    - 确保开发环境中已经安装Node.js。
  - 创建项目
    - 使用Vite的命令行工具来创建一个Vue项目“taskManagementSystem”（使用Vue作为前端框架），并配置好Vue的基础环境。

Plain Text ▾ 固定高度 复制 设置

```
1 npm create vite@latest taskManagementSystem -- --template vue
```


2. Web发布 2.3 使用Vite发布管理系统

- 使用Vite发布一个管理系统，是使用Vite作为构建和开发服务器来开发一个管理系统前端项目，并在开发完成后将其构建成可以部署到服务器上的静态文件。使用Vite发布一个管理系统需要准备环境、创建项目、进入项目并安装依赖、开发管理系统、本地开发、 构建项目、部署。具体步骤如下。
  - 进入项目并安装依赖

Plain Text ▾ 固定高度 复制 设置

```
1 cd taskManagementSystem
2 npm install
```

  - 开发管理系统
    - 完成管理系统的开发。

2. Web发布 2.3 使用Vite发布管理系统

- 使用Vite发布一个管理系统，是使用Vite作为构建和开发服务器来开发一个管理系统前端项目，并在开发完成后将其构建成可以部署到服务器上的静态文件。使用Vite发布一个管理系统需要准备环境、创建项目、进入项目并安装依赖、开发管理系统、本地开发、 构建项目、部署。具体步骤如下。
  - 进入项目并安装依赖构建项目
    - 当开发完成后，需要将项目构建成静态文件，在dist/目录下生成构建后的静态文件，包括HTML、CSS、JavaScript和资源文件等，以便部署到生产服务器上，使用以下命令来构建项目。

```
Plain Text ▾   固定高度  复制  设置
1 npm run build
```

  - 部署
    - 默认构建好的项目将保存在“dist”目录下，将“dist”目录中的文件部署至生产服务器。

2. Web发布 2.4 构建配置优化

- Vite构建配置优化是提升应用性能、改善用户体验、提高开发效率、增强可维护性、支持现代Web技术、优化SEO、降低成本以及增强安全性的重要手段。通过合理配置和优化构建过程，可以使得项目更加高效、可靠和易于管理。以下是构建配置优化对应用的影响：
  - 提升加载速度：优化构建配置减少最终打包文件的体积，使用户在访问网站或应用时需要下载的数据更少，加快页面的加载速度，提升用户体验。
  - 改善性能：通过代码分割、按需加载等优化手段，确保用户只加载当前页面或功能所需的代码和资源，减少不必要的计算和内存占用，改善应用的运行性能。
  - 提高构建效率：构建效率影响到开发人员的迭代速度。优化Vite的构建配置可以缩短构建时间，使开发人员能够更快地看到代码变更的效果，从而提高开发效率。
  - 增强可维护性：合理构建配置使项目的结构更清晰，代码模块化，便于迭代和维护。通过自动化的构建和压缩过程，减少手动操作带来的错误和不一致性。

2. Web发布 2.4 构建配置优化

- Vite构建配置优化是提升应用性能、改善用户体验、提高开发效率、增强可维护性、支持现代Web技术、优化SEO、降低成本以及增强安全性的重要手段。通过合理配置和优化构建过程，可以使得项目更加高效、可靠和易于管理。以下是构建配置优化对应用的影响：
  - 支持现代Web技术：Vite作为现代前端构建工具，支持ES模块、HTTP/2、服务器推送等现代Web技术。通过优化构建配置，可以更好地利用这些技术带来的优势，提升应用的性能和用户体验。
  - 提升SEO：较快的加载速度和优化的代码结构对搜索引擎优化（SEO）也有积极的影响。搜索引擎更倾向于排名那些加载速度快、内容组织良好的网站。
  - 降低带宽和存储成本：对于需要部署到云服务器或CDN的应用，较小的包体积意味着更低的带宽消耗和存储成本。
  - 增强安全性：通过移除生产环境中的调试信息（如console.log语句）和不必要的代码，可以减少潜在的安全风险。合理的构建配置可以帮助实现资源的HTTPS传输等安全措施。

2. Web发布 2.4 构建配置优化

- Vite 构建配置优化是提升前端项目构建效率和性能的关键步骤。以下是一些常见的Vite构建配置优化方法：
  - 使用gzip压缩
    - gzip压缩可以显著减少文件大小，加快文件传输速度。在Vite中，可以通过安装“vite-plugin-compression”插件来实现gzip压缩。配置示例如下：

Plain Text ▾ 固定高度 复制 设置

```
import viteCompression from 'vite-plugin-compression';

export default defineConfig({
  plugins: [
    viteCompression({
      verbose: true,  //表示在构建过程中显示详细的压缩信息。
      disable: false,  //表示启用压缩功能。
      deleteOriginFile: false,  //表示不删除原始未压缩的文件。
      threshold: 5120, // 表示只有大于或等于这个大小的文件才会被压缩。
      algorithm: 'gzip',  //表示使用 gzip 算法进行压缩。
      ext: '.gz'  //表示压缩后的文件扩展名为 .gz。
    })
  ]
});
```

2. Web发布 2.4 构建配置优化

- Vite 构建配置优化是提升前端项目构建效率和性能的关键步骤。以下是一些常见的Vite构建配置优化方法：
  - 静态资源打包处理
    - 优化静态资源的打包处理，按分类存放相应的文件，可以提高缓存效率。在Vite中，可以通过配置“build.rollupOptions.output”来实现：

Plain Text ▾ 固定高度 复制 设置

```
build: {
  rollupOptions: {
    output: {
      //控制动态导入的 chunk 文件的命名方式。
      chunkFileNames: 'static/js/[name]-[hash].js',
      //控制入口文件的命名方式。
      entryFileNames: 'static/js/[name]-[hash].js',
      // 控制静态资源（如图片、字体等）的命名方式。
      assetFileNames: 'static/[ext]/[name]-[hash].[ext]',
      // 定义一个函数用于手动指定哪些模块应该被打包到同一个 chunk 中。
      manualChunks(id) {
        if (id.includes('node_modules')) {
          return id.toString().split('node_modules/')[1].split('/')[0].toString(
        }
      }
    }
  }
}
```


2. Web发布 2.4 构建配置优化

- Vite 构建配置优化是提升前端项目构建效率和性能的关键步骤。以下是一些常见的Vite构建配置优化方法：
  - 代码分割与大文件拆分
    - 代码分割可以将代码拆分成多个包，按需加载，减少初始加载时间。在Vite中，可以通过配置“build.chunkSizeWarningLimit”和“manualChunks”来实现：

Plain Text ▾ 固定高度 复制 设置

```
build: {
  chunkSizeWarningLimit: 1500, // 控制 chunk 大小警告的阈值。
  //定义一个对象来手动指定哪些模块应该被打包到同一个 chunk 中
  manualChunks: {
    vue: ['vue', 'vue-router'],
    lodash: ['lodash'],
    // 可以根据实际需要合并其他库或组件
  }
}
```

2. Web发布 2.4 构建配置优化

- Vite 构建配置优化是提升前端项目构建效率和性能的关键步骤。以下是一些常见的Vite构建配置优化方法：
  - 组件按需导入
    - 使用“unplugin-vue-components”插件可以实现Vue组件的按需导入，减少最终打包体积。安装并配置该插件后，Vite会在构建时自动分析并导入用到的组件。

Plain Text ▾ 固定高度 复制 设置

```
// vite.config.js
import { defineConfig } from 'vite';
import Components from 'unplugin-vue-components/vite';
import { AutoImport } from 'unplugin-auto-import/vite';
import { createResolver } from 'unplugin-vue-components/resolvers';

// 定义一个解析器来处理 iView 的组件
const iViewResolver = createResolver({
  // iView 的组件路径前缀
  prefix: 'iView',
  // 匹配组件的正则表达式
  resolveComponentPath: (name) => `iview/es/${name}/index`,
});

export default defineConfig({
  plugins: [
    // 自动导入 Vue 组件
    Components({
      resolvers: [
        iViewResolver,
      ],
      dts: true, // 开启生成 .d.ts 声明文件的支持
    }),
    // 自动导入 Vue 的 Composition API
    AutoImport({
      imports: ['vue', 'vue-router', 'vuex'], // 可以根据需要添加其他库
      resolvers: [],
    }),
    // 其他插件...
  ],
  // 其他配置...
});
```


2. Web发布 2.4 构建配置优化

- Vite 构建配置优化是提升前端项目构建效率和性能的关键步骤。以下是一些常见的Vite构建配置优化方法：
  - 清除console和debugger
    - 在生产环境中，移除console.log和debugger语句可以减小文件体积并避免潜在的安全风险。可以通过配置Terser插件来实现：

Plain Text ▾ 固定高度 复制 设置

```
build: {
  terserOptions: {
    compress: {
//设置为 true 表示在压缩过程中移除所有的 console 调用。
      drop_console: true,
//设置为 true 表示在压缩过程中移除所有的 debugger 语句。
      drop_debugger: true
    }
  }
}
```

2. Web发布 2.4 构建配置优化

- Vite 构建配置优化是提升前端项目构建效率和性能的关键步骤。以下是一些常见的Vite构建配置优化方法：
  - 使用CDN加速
    - 对于第三方库或大型资源，可以考虑使用CDN来加速资源加载。Vite中可以通过安装vite-plugin-cdn-import插件来实现：

Plain Text ▾ 固定高度 复制 设置

```
import { importToCDN } from 'vite-plugin-cdn-import';

export default defineConfig({
  plugins: [
    importToCDN({
      modules: [

        // 需要CDN加速的模块  假设为lodash
        //name: 库的名称。
        //var: 在全局变量中注册的名称。
        //path: CDN 链接地址
         {
          name: 'lodash',
          var: '_',
          path: 'https://cdn.jsdelivr.net/npm/lodash@4.17.21/lodash.min.js'
        }
      ]
    })
  ]
});
```


2. Web发布 2.5 项目监控与维护

- 项目监控与维护是项目管理中的两个重要环节，项目监控确保项目在执行过程中的合规性和有效性，而项目维护则确保项目在交付后能够持续稳定运行并满足用户需求。
- 项目监控
  - 项目发布后前端项目监控在现代Web开发中扮演着至关重要的角色，它有助于确保网站或应用的稳定性、性能优化以及用户体验的持续提升。前端项目监控主要可以从数据监控、性能监控和异常监控三个方面进行实现。
  - 数据监控：数据监控，也被称为埋点统计，主要用于监听用户的操作行为。通过数据监控，了解用户的使用习惯、行为模式以及页面热点区域等信息，为优化提供数据支持。常见的数据监控指标包括。
    - PV（页面浏览量）：用户每次打开或刷新一个页面时，该页面的浏览量就会增加。
    - UV（独立访客数）：在一定时间内，访问网站的不同用户的数量。UV在PV的基础上进行去重处理。
    - 用户在页面停留时间：用户从打开页面到离开页面的时间差，反映了页面对用户的吸引力。
    - 用户来源：用户通过哪个渠道或链接进入当前页面，有助于分析推广效果。
    - 页面操作行为：用户在页面上的点击、滚动、表单填写等行为，通过操作行为分析用户的操作路径和意图。

2. Web发布 2.5 项目监控与维护

- 项目监控
  - 性能监控：性能监控主要关注前端项目在用户端展示的性能，包括页面加载时间、资源加载情况、交互响应速度等。通过性能监控，可以及时发现并解决性能瓶颈，提升用户体验。常见的性能监控指标包括。
    - 首屏加载时间：用户首次看到页面内容所需的时间，包括白屏时间和首屏内容渲染时间。
    - HTTP接口响应时间：前端向服务器发送请求并收到响应的时间。
    - 静态资源加载时间：图片、CSS、JavaScript等静态资源的下载时间。
    - 关键性能指标（KPIs）：如LCP（Largest Contentful Paint，最大内容绘制时间）、FID（First Input Delay，首次输入延迟）、CLS（Cumulative Layout Shift，累计布局偏移）等，这些指标是Google提出的Web Vitals，用于衡量网页的核心性能。

2. Web发布 2.5 项目监控与维护

- 项目监控
  - 异常监控：异常监控主要用于捕获和上报前端代码中的错误和异常，以便能够及时发现并修复问题。常见的异常类型包括JavaScript运行错误、AJAX请求错误、静态资源加载错误等。异常监控的实现通常依赖于以下几种方式。
    - try/catch语句：用于捕获代码中的运行时错误。但try/catch无法捕获语法错误和异步错误。
    - window.onerror事件：当JavaScript运行时发生错误时，会触发该事件。通过为window对象添加onerror事件监听器，可以捕获到大部分运行时错误，并获取到错误信息、错误发生的脚本URL、行号、列号等详细信息。
    - Promise异常捕获：对于使用Promise的异步操作，通过在Promise链的末尾添加catch方法来捕获异步操作中的错误。
    - 第三方监控工具：如Sentry、Bugsnag等，第三方监控工具工具提供了丰富的错误监控和报警功能，可以帮助开发者更高效地管理前端错误。

2. Web发布 2.5 项目监控与维护

- 项目维护
  - 前端项目维护是前端开发中的一个重要环节，涉及多个方面，目的是确保项目的稳定性、可维护性和可扩展性。以下是前端项目维护的关键方面和具体策略。
  - 代码管理与版本控制
    - 使用版本控制系统：将项目代码托管在GitHub、GitLab等平台上，并使用Git作为版本控制系统。便于追踪代码变更历史、回滚历史版本以及团队协作。
    - 保持代码库整洁：定期清理废弃的代码、文件和依赖项，减少项目大小和提高加载速度。
  - 项目结构优化
    - 分层化：将代码结构进行分离，形成清晰的层次，如页面层、组件层、数据层等。提高代码的可读性和可维护性。
    - 配置化：将可配置的部分抽象为配置文件或JSON对象，以便于在不同环境或场景下快速调整。
    - 模块化：遵循单一职责原则，将代码拆分为独立的模块，每个模块负责一个明确的功能。减少代码冗余和提高复用性。

2. Web发布 2.5 项目监控与维护

- 项目维护
  - 性能优化
    - 减少HTTP请求：合并和压缩CSS、JavaScript和图片文件，使用浏览器缓存技术来减少HTTP请求次数。
    - 优化资源加载：使用CDN加速资源加载，优化资源加载顺序，确保关键资源优先加载。
    - 代码优化：避免使用过于复杂的代码结构和大量的嵌套，使用简洁明了的代码风格。
  - 错误监控与调试
    - 集成错误监控工具：使用Sentry、Bugsnag等错误监控工具来捕获和报告前端代码中的错误和异常。
    - 日志记录：在关键操作处添加日志记录，以便于在出现问题时能够追踪到具体的执行流程和上下文信息。
    - 使用开发者工具：熟练使用浏览器的开发者工具进行调试，如Chrome DevTools、Firefox DevTools等。

2. Web发布 2.5 项目监控与维护

- 项目维护
  - 文档与规范
    - 编写文档：包括项目介绍、API文档、使用指南、常见问题等，以便于其他开发者理解和使用项目。
    - 制定代码规范：制定统一的代码编写规范，如命名规范、代码风格、注释规范等，以提高代码的可读性和可维护性。
  - 持续迭代与更新
    - 收集用户反馈：通过用户反馈、用户调查等方式收集用户意见和需求，以便对项目进行持续优化和改进。
    - 跟踪新技术和趋势：关注前端技术的新发展和趋势，如新的框架、库、工具等，及时学习和应用新技术来提高项目的质量和效率。
    - 定期更新和维护：定期更新项目依赖的库和框架，修复已知的安全漏洞和性能问题，确保项目的长期稳定运行。

信创智能医疗系统研发课程体系
河南中医药大学信息技术学院（智能医疗行业学院）

河南中医药大学信息技术学院（智能医疗行业学院）智能医疗教研室
河南中医药大学医疗健康信息工程技术研究所