文章编号：2096-1472(2022)-09-18-04
DOI:10.19644/j.cnki.issn2096-1472.2022.009.005

医院药房管理系统前端工程化设计与实现

李思睿，郑大翔，李志芳

(海南医学院，海南 海口 571199)
✉847520303@qq.com; 2505663420@qq.com; 19143212@qq.com

摘　要：医院药品类别繁多且数量大，合理利用信息化手段，有助于科学地规范、优化药品业务流程，提高医院药品管理工作效率。系统采用前后端分离的开发架构，在Electron平台下使用MVVM(Model−View−ViewModel)模式对其进行详细的技术设计和实现。前端采用Vue.js负责视图渲染，实现数据驱动视图；后端采用Node.js负责业务处理，并提供数据接口。数据库采用MongoDB结合Redis进行缓存和性能优化。系统以组件化、模块化为基准，充分复用开发资源，避免多重操作DOM，实现药房管理系统客户端与后台端细化协作，解决了目前传统药房系统开发效率低下、系统性能差、维护难的问题。

关键词：药品管理；前后端分离；前端工程化；MVVM；Electron
中图分类号：TP399　**文献标识码**：A

Design and Implementation of Front-end Engineering for Hospital Pharmacy Management System

LI Sirui, ZHENG Daxiang, LI Zhifang

(*Hainan Medical University*, *Haikou* 571199, *China*)
✉847520303@qq.com; 2505663420@qq.com; 19143212@qq.com

Abstract: In view of a great variety and large quantities of drugs in hospitals, rational use of information technology will be helpful in scientifically regulating and optimizing the drug business process, and improving the efficiency of hospital drug management. The proposed system adopts a development architecture with front and back ends separated, and uses the MVVM (Model-View-ViewModel) pattern to carry out detailed technical design and implementation under the Electron platform. Vue.js is applied in the front-end for rendering the view and realizing data-driven views; Node.js is used in the back-end for processing the business and providing data interfaces. The application of MongoDB and Redis has contributed to database caching and performance optimization. Based on component and modularization, the development resources are reused fully, which avoids multiple operations of DOM, and realizes the refined collaboration between the client-end and the back-end of the pharmacy management system. The proposed system solves the problems of low development efficiency, poor system performance and difficult maintenance of the existing pharmacy system.

Keywords: drug management; front-end and back-end separation; front-end engineering; MVVM; Electron

1 引言(Introduction)

当前绝大多数的医院信息系统都采用三层架构的开发方式，将整个应用程序划分为界面层、业务逻辑层、数据访问层。在界面层，通常结合DevExpress控件库进行开发，其内容虽然丰富，但加载速度较慢，且样式较为单一。处理业务逻辑时，因为分层需要使用中间层方式(数据访问层)访问数据库，需要对数据进行各种转换和计算，降低了系统性能[1]。当用户访问量增大，处于高并发时，会导致系统响应慢，经常出现未响应状态，甚至崩溃。三层架构实际上不只是三层，随着业务复杂度的提升，少则划分四五层，多至八九层。层级之间缺乏统一的标准，不同的开发者对各个层级的理解不一致，当有新功能需求时，需要开发者在各个层级进行相应开发，同时接口方面也存在衔接不一致的问题。

基金项目：2021校级创新创业训练计划项目(X202111810052)；第三批海南省普通本科高校应用型试点转型专业项目(信息管理与信息系统专业).

随着互联网技术的快速发展，架构技术数不胜数。国内外IT公司在业务层次追求强大功能的同时，更加注重用户体验和系统性能。在开发层次，采用团队协作方式，利用工程化思想，采取前后端分离的开发模式，前端负责对数据的渲染，关注与用户的交互；后端处理业务逻辑，为前端提供数据接口。前端工程化是当今主流的项目开发方案[2]。本文通过对现有医院信息系统的研究，采用解耦的编程思想，设计并实现了药房药库管理系统。系统以药品管理规范化、科学化为目标，提高了医院药房药库整体的工作效率，促进了医院信息化发展[3]。系统主要包括药房管理、药库管理、门诊药房、数据统计、后台管理功能模块。

2　系统架构原理及技术框架(System architecture principle and technical framework)

2.1　系统架构概述

系统采用前后端分离的架构模式，前端主要运用HTML、CSS制作静态页面，结合JavaScript实现页面动态化并对数据进行渲染展示，让页面运行更加流畅，追求更高的用户体验；后端处理业务逻辑，提供数据接口，实现服务的高性能和高并发。前端通过Axios调用后端的Api接口，对后端返回数据进行处理并渲染。这种开发方式降低了前后端代码之间的耦合性，架构体系清晰，便于代码的上线部署和后续维护。

2.2　Electron平台

系统的桌面应用程序基于Electron，它是由GitHub开发的一款跨平台桌面应用开发框架。Electron将Google的Chromium和Node.js合并到一个运行环境中，所以能使用Node.js中几乎所有的模块[4]，通过JavaScript操作系统原生的Api。Electron有且仅只有一个主进程(Main Process)，Main Process通过BrowserWindow实例创建页面并通过Chromium展示，每个独立的Web页面都运行在它自身的渲染进程(Render Process)中[5]，能够完美地使用Vue.js，在各个渲染进程中独立开发Web页面。Electron原理如图1所示。

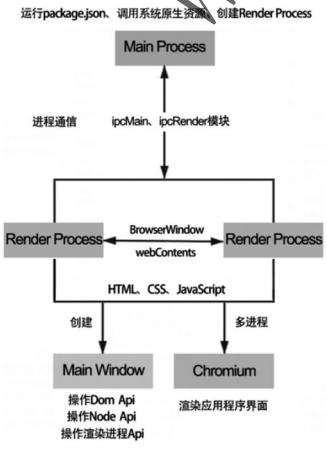

图1 Electron原理图示

Fig.1 Diagram of Electron principle

2.3　MVVM

在传统开发中，页面都是静态渲染的，页面的更新必须要直接操作DOM，大量DOM操作不仅消耗性能且十分烦琐。Vue.js通过MVVM模式实现数据驱动视图，实际上是从数据劫持到发布订阅的一种模式[6]。通过Object.defineProperty这个Api在对象上定义一个新属性或者修改现有的属性，使用该Api下的get方法读取属性获取data，使用set方法写入属性监听data的变化。组件中data一旦发生变化，就会根据修改后的data重新渲染视图，由MVVM去自动更新DOM，开发者就无须直接操作DOM，节约了开发成本，提高了程序的性能。

2.4　Virtual Dom

虽然MVVM能帮助开发者自动更新视图，减少了对DOM的直接操作，但实际上也是要对DOM进行操作的，依然非常耗时。但Virtual Dom先用执行速度更快的JS来模拟DOM结构，在多次的DOM操作中计算出最小的变更，避免了一些毫无意义的操作，最后操作DOM，其本质上是DOM和JS之间的一个缓存[7]。

Virtual Dom通过diff算法比较DOM操作前后的差别，计算出最小变更。对于操作前后的DOM树，diff算法只对它们之间同级比较，若两棵树的tag不相同，删除重建，不再进行下一层级的对比[8]。若一棵树的tag和key都相同，则认为是相同的节点，同样不再进行下一层级的对比。此算法的时间复杂度为O(n)，执行算法时的工作量是十分理想的。

在修改model中的data时，重新编译渲染。在编译过程中会将template模板转变成一个render函数，通过render函数生成Virtual Dom，调用vm._render方法生成一个新的虚拟节点(Vnode)。对原来的Vnode和新生成的Vnode进行diff算法计算出最小变更。这个过程中会实例化一个Watcher，Watcher是Dep实例中的一个对象，Watcher会调用Dep下的Notify方法遍历Dep的Watcher实例数组，通过对应的update方法来更新视图。Vue.js使用了Virtual Dom，避免了回流和重绘的DOM操作，提高了性能。Virtual Dom流程相关的模板渲染过程如图2所示。

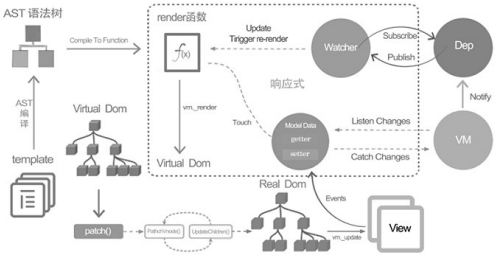

图2 模板渲染过程

Fig.2 Template rendering process

3　系统设计(Systematic design)

3.1　组件化

组件化设计的优势如下：(1)组件标准化。每个组件都有统一的标准，有了标准和规范，各个组件才能更好地结合在一起使用。(2)功能分治。将系统的各个功能都封装到不同的组件中，目的是让每个组件可以独立进行开发，达到解耦[9]。

(3)组件复用。当系统中因为某个功能的改变导致组件不可以使用时，对组件进行回收，更换一个新的组件，或者对原先组件进行优化。(4)组件组合。组件之间通过组合的方式，构成一个功能或者一类功能模块。组件化设计最终的目的是达到高效、协作以及复用。

3.2 模块化

模块是指在系统开发中，用自执行函数将多个函数包裹起来形成闭包，自执行函数通过return将内部函数暴露，这种方式也称单例设计模式。模块化就是把一个或多个函数封装到一个JS中，各个JS之间互不影响，提供方通过export的方式进行暴露，使用方通过import的方式进行导入。模块化最终的目的是解决开发中的命名冲突，避免因各个JS文件中存在同名变量导致的变量同名覆盖问题；同时解决了各个JS之间的依赖问题，并达到代码的复用，提高函数的可维护性，从而提高整体项目的开发效率，降低维护成本。

3.3 组件通信与状态管理

一般情况下，组件只需要操作自己的私有数据就能够满足某一功能的需求，但有些功能比较复杂，需要多个组件共同完成。父组件可以通过属性传递的方式向子组件传递状态(数据)，子组件使用props接收父组件传递的数据，通过this.$emit调用父组件的事件。父组件通过event.$on绑定自定义事件，子组件使用event.$emit调用自定义事件。但是，在组件之间不是父子或兄弟这种简单关系，而是相隔多层的祖孙关系或者没有任何关系，甚至多个组件之间共用一个状态的情况下，使用上述组件通信方式就显得特别笨拙。

采用Vuex对组件的全局状态进行统一管理，实现了组件之间的数据共享。第一步，组件改变数据后通过执行$store.dispatch()方法触发action。第二步，在action处理异步或同步操作后执行$store.commit()方法触发mutation。第三步，在mutation中处理同步操作后更新状态(数据)，存储在state中。组件可以通过$store.getters()方法获取state中的全局对象，也可以通过import将mapGetters辅助函数导入并映射到组件的计算属性(computed)中，在computed中使用扩展运算符...mapGetters将获取的全局对象解构。

Vuex可以集中管理组件间共享的状态(数据)，充分利用Vue.js的细粒度数据响应机制来进行高效的状态管理。存储的数据是响应式的，能够保证在实时存储、视图更新的操作后，存储数据与页面数据保持同步。各步操作采用模块化的方法，分工明确，提高了开发效率，易于后期维护。Vuex组件状态管理如图3所示。

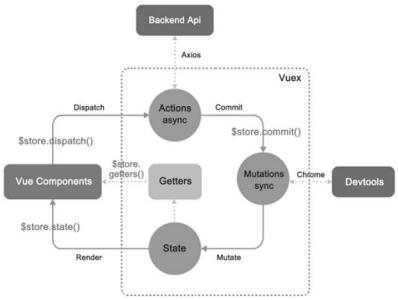

图3 Vuex组件状态管理

Fig.3 Vuex component status management

3.4 安全性设计

3.4.1 密码安全

为避免用户密码信息泄露，危害系统安全，系统采用md5[10]加密算法。md5是哈希算法中的一种信息摘要算法，不同长度的用户密码都会被加密成固定长度的32 位字符，并存储在数据库中。md5加密具有雪崩效应，明文的一丁点修改，都会造成密文的大幅变化，且密文到明文具有不可逆性。

但如果用户密码比较简单，通过彩虹表仍可能被快速破解，故将明文密码拆解为三部分，结合密钥在特定部分加Salt，再进行多次md5加密。Salt具有动态性，用户名具有唯一性，可将用户名作为Salt(如明文拆解1+username+明文拆解2+密钥+明文拆解3)。这样处理前后的密文截然不同，即使Salt和密钥泄露，彩虹表也无法使用，只能根据Salt和密钥对彩虹表进行重新生成，且处理后的密码具有超长位数和复杂度。在原始密码只是纯数字的情况下，分布列范围至少为10的20次方；如果是数字加字母的密码，其分布列范围则达到了62的20次方，几乎不可能破解。

3.4.2 接口访问安全

普通的接口只需要携带指定参数就可以访问后端服务，多次恶意的访问会导致服务器繁忙，甚至崩溃。采用JSON Web Token的认证机制，用户登录成功后会自动生成一个JWT Token，返回并存储到Cookie中，用户访问接口时，附带在请求头中协同用户提交的参数一起发送到服务端，服务端对发来的Token用密钥解析，验证请求是否合法以及判断用户的身份。

JSON Web Token包括JWT头部(Header)、JWT主体(Playload)、JWT签名(Veify Signature)。Header规定了所采用的加密算法和Token的类型，Playload中包含需要传递的数据，Veify Signature对Playload中的数据进行运算，返回一串字符串，再使用Header中规定的加密算法进行加密，生成加密字符串。

本系统采用用户名和姓名作为主体数据，加密算法为HMAC SHA256，Token类型为JWT类型，自设密钥。服务端接收到用户发来的Token信息，使用密钥进行解密，判断Token有效期及Token对应的用户身份，身份验证通过允许访问接口。JWT验证过程如图4所示。

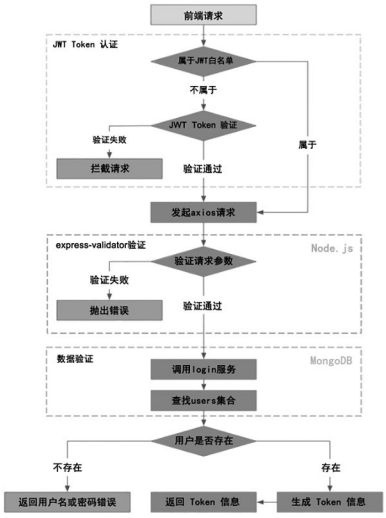

图4 JWT验证过程

Fig.4 JWT validation process

JWT验证过程：(1)可以指定Secure来确保Token只在Https协议下传输，防止传输过程中被窃听。(2)在Cookie中设置HttpOnly，禁止通过JS获取Cookie信息，一定程度上防御了XSS攻击。(3)服务端可以检查用户的Refer和Origin，在Response中缺少Access-Control-Allow-Origin字段或者原始资源URI不明的情况下，可以拒绝处理该请求，防止恶意请求，一定程度上防御了CSRF攻击，减轻服务器的压力[11]。

3.4.3 其他

(1)设置授权机制。对授权码采用md5加密方式，只有设置授权码的管理员和被授予权限的用户才知道真正的授权码。

(2)防御跨站脚本攻击(XSS)。对输入文本的特殊字符进行转义，例如将“<”变为“&lt”，这样被非法嵌入JavaScript的脚本就不能运行。

(3)防御跨伪造攻击(CSRF)。重要请求接口采用POST请求方式，对重要操作增加了验证措施。

(4)采用路由守卫的导航模式。通过函数钩子beforeEach对路由进行全局前置守卫，对登录后才能访问的页面设置了拦截操作，登录后服务端设置Token信息，只有Token存在时才能访问。

3.5 系统主要功能

系统用户主要是医务人员，群体较为固定。根据业务需求，功能模块包括药房管理、药库管理、门诊药房、数据统计、后台管理等。(1)药房管理实现了药品报损、效期管理、药品请领；(2)药库管理实现了药品入库、药品出库、药品盘点、库存查询、药品移库；(3)门诊药房如图5所示，实现了处方发药、状态查询、退药处理；(4)数据统计包括访问量、年度药房药库销售金额、当前库存量、分类药品销量排行、年度药房报损药品金额、年度药品报损原因、年度盘盈盘亏等统计；(5)后台管理包括票据管理、调价管理、用户管理、权限管理。

图5 门诊药房功能模块

Fig.5 Functional modules of outpatient pharmacy

3.6 系统测试及优化

系统采用可视化面板(GUI)对代码进行测试，自动生成测试报告，在4G环境下只需1.59 秒就可以加载完成，但ECharts和Element-UI的依赖体积过大，在一定程度上影响了系统的性能。同时在资源模块中，出现对打包后的JS文件和部分图片文件发出警告的问题。针对这些问题我们做了如下优化：

(1)在依赖方面，通过import方式导入的所有依赖项都会被打包合并到一个文件中，导致文件体积过大。使用config.set方法创建白名单，白名单中的依赖项将不会被合并打包，而在window全局对象查找使用；在public下的index.html文件中添加以link、script的src方式引用CDN资源的配置项，通过修改Webpack的externals节点加载外部的CND资源。但完整引入Element-UI依赖项导致依赖包体积过大，采用按需加载的方式将所需的组件从Element-UI中解构出来，仅对使用的组件进行打包。

(2)在路由方面，使用了路由懒加载技术。安装babel插件，用箭头函数的形式，通过webpackChunkName对路由进行分组，打包到不同的JS中，避免了一次性加载全部路由，只有当用户访问的时候才进行加载。通过Vue.js的内置组件Keep-alive对路由进行缓存，避免了重复渲染，提高了页面的加载效率，增加了用户体验感。

(3)在图片方面，将较大的图片压缩成base64格式，以减少图片体积。对于大量的图片，可以通过v-lazy指令对图片进行懒加载，图片分多批次加载，用户访问时才触发下一次加载。

(4)在代码层面，进行组件通信时，通过event.$on生成自定义事件，框架提供的默认事件在组件生命周期结束时会被自动回收，而开发者定义的自定义事件会一直存在。本系统在组件的生命周期beforeDestory函数中通过evnent.$off销毁自定义事件。此外，还要对全局变量进行回收，避免内存泄

(下转第4页)