计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(4):70–75 [doi: 10.15888/j.cnki.csa.006311]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

前后端分离的终端自适应动态表单设计①

喻莹莹[1,2], 李　新[1], 陈远平[1]

[1](中国科学院 计算机网络信息中心, 北京 100190)
[2](中国科学院大学 计算机与控制学院, 北京 100049)

摘　要: 首先讨论了前后端分离在 Web 开发工作中的优势, 介绍了当下流行的 MVVM 设计模式, 并指出动态表单设计的必要性. 接着, 对动态表单的具体设计从前后端两方面展开详细描述, 前端的实现利用了轻量级 Vue.js 框架的组件特性, 后端则使用 Spring MVC 架构来实现业务模型和数据处理工作并采用 NoSQL 数据库来满足表单字段动态变化的需求. 最后, 对终端自适应的响应式设计方案进行了阐述.

关键词: 前后端分离; Vue; 模块化; 动态表单; 自适应

引用格式: 喻莹莹,李新,陈远平.前后端分离的终端自适应动态表单设计.计算机系统应用,2018,27(4):70–75. http://www.c-s-a.org.cn/1003-3254/6311.html

Design of Terminal Adaptive Dynamic Form Based on Frontend-Backend Separation

YU Ying-Ying[1,2], LI Xin[1], CHEN Yuan-Ping[1]

[1](Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)
[2](Computer and Control Academy, University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Firstly, this study expounds the advantages of frontend-backend separation development of web application, introduces the MVVM design model, and points out the necessity of designing dynamic forms. Then, it describes the specific design of dynamic forms from the frontend to the backend in details. The development of the frontend is based on the lightweight framework of Vue.js taking advantage of the features of its components. The backend uses Spring MVC architecture to implement the business model and data process and utilizes NoSQL database to satisfy the dynamic requirements of form items and fields. At last, this study describes a responsive design about terminals self-adaption.

Key words: frontend-backend separation; Vue; modularization; dynamic form; self-adaption

Web 表单作为网站和用户直接交互的平台, 担负着大量的用户和网页后台数据的更新交互, 在系统开发中显得尤为重要[1]. 传统 Web 表单主要采用前端开发人员编写好静态表单页面, 后端人员根据 html 文件实现 View 层, 前端开发也依赖于后端的完整开发以便获取数据库中的固定格式的数据字段, 这种前后端不分离的开发模式只能满足特定场景下的需求, 操作不灵活、定制困难、功能简单, 一旦需求发生改变, 前后端需要重新对接和开发, 降低了代码的复用性, 这就大大延长了系统的开发周期, 也不利于人员对系统的维护.

同时, 由于移动互联网的飞速发展和浏览器功能的日益完善, 基于浏览器和服务器的 B/S 架构更加受到开发者的欢迎, 很多移动互连设备正在涌现, Web 访问可能会从 PC 终端以及移动终端发起, 但由于移动设备和 PC 在屏幕大小、分辨率、输入方式、启动加载、设备操作方式等方面的差异, 在对 Web 表单进行设计时候必须要考虑如何兼容不同终端的问题, 对不同终端的交互设计和交互方式进行统一, 实现响应式

① 收稿时间: 2017-08-06; 修改时间: 2017-08-22; 采用时间: 2017-09-04; csa 在线出版时间: 2018-03-31

的 Web 表单设计, 给用户提供适合不同终端的产品一致性体验.

针对以上问题, 本文提出了一种基于前后端分离的多终端自适应动态表单设计, 该设计旨在提高 Web 表单开发的效率, 实现前后端开发的异步进行和及时交互, 利用现有的前端组件技术实现表单结构设计的模块化, 满足不同终端用户对表单的个性化定制.

1 前后端分离模式

现有的 Web 开发模式大致分为 3 种: (1) 后端主导的 MVC 框架. 如 Struts、Spring MVC 等. 这种模式做一些同步展现的业务效率很高, 但是遇到同步异步结合的页面与后端开发沟通起来就会比较繁琐, 并且前端重度绑定后端环境, 环境成为影响前端开发效率的重要因素; (2) Ajax 带来的 SPA, 即单页面应用. 使得用户能够在页面不跳转的情况下实现与后台的实时交互, 页面局部刷新, 在这种模式下前后端的分工非常清晰, 但是前端开发需要大量的 JS 代码的组织和 View 层的绑定, 并且对 Ajax 接口的约定要求更加严格, 这就制约了开发效率的提高; (3) 前端采用 MVVM 模式结合后端采用 Spring MVC 进行开发. 前端负责 View 和 Controller 层, 后端负责 Model 层, 处理业务和数据等. 为了降低前端开发的复杂度, 本文采用第三种开发模式 .

相比于传统的 MVC 模式, MVVM 模式实现了对数据的双向绑定, View 的变化会自动更新到 ViewModel, 而 ViewModel 的变化也会自动同步到 View 上显示, 这种自动同步是因为 ViewModel 中的属性实现了 Observer, 当属性值变更时候就能触发对应的操作[2]. MVVM 模式的架构图如图 1 所示.

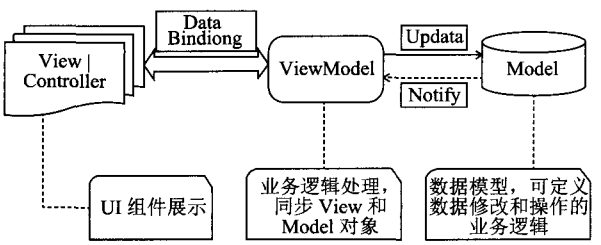

图 1 MVVM 架构

通过前后端统一的 API 接口, 后端程序能够为前端程序 (包括 PC 端程序和移动端应用程序) 提供业务数据和服务的支撑, 不再关注业务具体的展现逻辑. 而前端的 Vue 负责组织数据并展示, 处理用户的请求、实现路由跳转页面等工作. 这种交互模型结构清晰, 关注点分离, 很好的实现了前后端的解耦合.

随着对“前后端分离”概念的深入理解以及相关开发技术的日趋成熟, 这种基于前后端分离的终端自适应动态表单设计将克服传统表单的种种弊端, 为使用者提供更加便捷灵活的跨平台表单服务.

2 动态表单设计

表单应用领域如此广泛和多样化, 为满足不同行业和使用者的需求, 动态表单应运而生, 例如文献[3]中提出的可视化 Web 表单编辑器, 文献[4]中提出的三层架构设计模式, 现在当下对动态表单的设计大多采用 MVC 模式, 前端使用 JavaScript 库 (如 JQuery) 和 Bootstrap 相结合, 利用特定格式的文件 (即模板) 传给后台, 后台使用表单引擎解析前端传过来的文件, 然后解析并生成各种配置文件, 表单视图 jsp 文件, 并部署到服务器的 web 应用目录中. 但是这样的设计存在两方面问题:

(1) JQuery 是一个用来开发 Web 界面的前端库, 直接操作 dom 元素, 较繁琐. 并且, JQuery 有丰富的插件库, 许多功能都是通过开发者自己编写插件来实现的, 但是 JQuery 并不能向后兼容, 这就限制了插件的使用和功能的拓展, 后期维护起来较麻烦.

(2) 前后端不完全分离, 前端的路由跳转需要后端控制, 服务端需要对 Web 端进行处理, 返回完整的 HTML, 增加了服务端的复杂度, 而 Web 端需要加载完整的 HTML, 一定程度上影响了网页的性能, 并且在多终端应用中非常的不友好.

通过前期对动态表单已有技术的调研, 针对以上问题, 本文提出了一种新的解决方案, 使用前端 MVVM 框架进行视图的创建和路由控制, 后端只需要处理相关业务和数据.

动态表单的设计涉及到前端页面的布局以及后端数据的处理. 目前符合 MVVM 的前端框架有很多 , 其中 AngularJS、ReactJS 和 Vue.js 是使用较为普遍的. 但是 AngularJS 过于庞大和全面, 而且涉及到了脏检查, 不容易维护和优化; 而 ReactJS 要求开发者借助 JSX 在 JavaScript 中创建 DOM, 并且不能使用模板; Vue.js 相比于前两者有较为突出的优势, 它采用响应

式编程, 并实现了组件化和模块化, 极大地方便了开发调试和维护工作.

Vue.js 从简单内建到它的设计, 对 DOM API 许多困难的部分进行了封装, 从而直接操作 DOM 元素对象, 去除繁琐的 DOM 操作, 只需要关注数据的源头, 而不用担心 DOM 元素变化之后的绑定变化, 这也是数据驱动的好处之一, 但是有利于后期的维护; 同时 VueJS 官方提供了迁移工具实现 Vue1.0 向 2.0 迁移, 从而实现了兼容; 另外, Vue.js 是基于组件的开发, 降低了各个模块的耦合, 提高了复用性. 相比于 JQuery 等 js 库, Vue 更加适合样式多变的动态表单开发.

同时, Vue 可以轻松的构建起一个无需服务端渲染就可以展示的网站, 不用给后端提供模板, 同时提供前端路由功能 vue-router, 后台不需要再控制路由的跳转, 将前端业务和后端业务分离. 大多数后台应用我们都可以做成 SPA 应用 (单页应用), 而单页应用最主要的特点就是局部刷新, 这通过前端控制路由调用 AJAX, 后台提供接口便可以实现, 在后端没还没有提供接口时, 前端可以将数据固定或者调用本地的 json 文件即可, 这样的方式用户体验更加友好, 网页加载更加快速, 开发和维护成本也降低了不少, 效率明显提升.

通过以上比较, 本文选择了 Vue.js 作为前端开发工具. 后端使用 Spring MVC 框架进行开发, Spring MVC 是 Spring 的一个用于构建 Web 应用的全功能 MVC 模块, 全注解的方式相比于 Struts 更加简单易用, 既能保证系统的安全性、可移植性、可靠性, 又能减轻系统维护人员日后的工作量, 增强系统的复用性和生命力[5].

2.1 整体架构设计

动态表单设计流程如图 2 所示.

首先连接数据库表, 根据用户选择的标签组件提取对应的信息, 显示当前表单标签样式, 接着对表单布局进行设置, 如果不设置, 则为默认布局; 如果设置, 则显示可供选择的几种表单标签样式, 包括文本框、下拉菜单框、多选框、图片、音频视频等. 然后对表单各项进行编辑, 并能够预览显示表单最终样式, 如果用户满足, 则可确认为最终表单布局. 最后保存提交有关的表单信息, 则系统自动把信息保存到数据库表里面.

为了避免繁琐的环境配置工作, 在这里使用 Vue 官方出品的脚手架 Vue-cli 自动构建一个项目, 一键部署开发环境, 根据 package.json 配置文件从 Github 上下载开发环境的依赖包列表和相关的配置文件, 即可进入到单文件组件的开发模式, 同时 Webpack 前端模块打包工具使用 Loader 加载器将项目分解成各个独立的模块进行加载, 还提供了合并压缩文件和加密程序的功能[6].

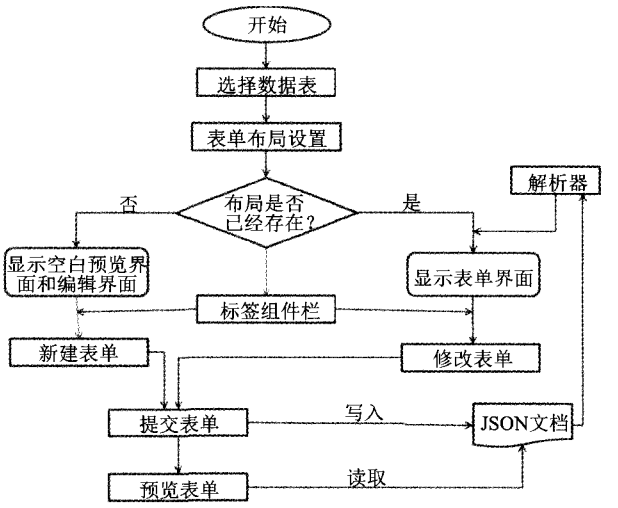

图 2 动态表单设计流程图

下面就从动态表单的前端设计, 后端设计以及相关的数据接口 3 个方面进行阐述.

2.2 前端设计

动态表单由多个子项构成, 每个子项代表实现不同功能的模块. 表单主要功能是提供项目组件供用户选择, 并允许用户对项目组件的各个属性进行编辑, 生成个性化的表单界面, 界面要求美观, 简洁, 易于理解.

动态表单中每个子项都可以抽象为一个活动组件, 每个活动组件都对应 3 个模式组件: 标签组件、预览组件、编辑组件, 这 3 个模式组件展现具体的内容, 而具体的样式和交互由活动 html5 页面根据视觉和交互设计来具体展现. 动态表单功能结构如图 3 所示. 其中, 通过拖动标签组件来创建对应类型的组件, 预览组件用来展现当前组件各项的内容, 编辑组件用来编辑当前选中的组件的各项内容. 点击组件标签, 览组件和编辑组件; 点击这个预览组件, 组件编辑区域会显示对应的编辑组件; 在编辑组件的文本域中可将活动需要的组件拖入预览区域后, 会生成对应的预以对组件各数据项进行编辑, 编辑完成后, 根据 Vue 的数据绑定, 文本域的数据变化就会反应到预览组件的 DOM 上, 从

而更新视图, 显示组件当前的最新内容. 同时 ES6 提供的 for … of 语句也极大地方便了对数组的操作, 并且避免了 DOM 操作, 对应用性能的优化有好处, 同时使用 Vuex 对状态进行集中式管理, 便于对状态的跟踪维护.

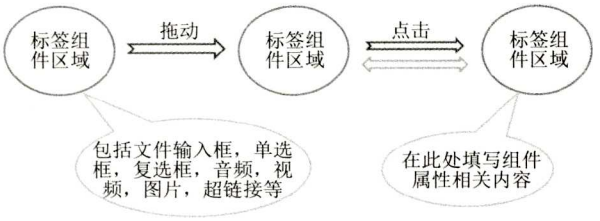

图 3 动态表单功能结构图

由于每个表单控件都可能在不同的场景多次使用, 为了提高提高复用性, 本文采用组件化的方式, 将每个表单项单独写在一个通用的 Vue 组件中, 使用 vuex 临时存储用户输入的数据, 然后根据 type 用 v-if 来渲染对应的表单. 其中对组件的使用代码如下:

```
// template
<div v-for="(item, index) in formData">
<input v-if="item.type === 'input'" v-model='item.val'>
<radio v-if="item.type==='radio'" @click='item.func'></radio>
<radio v-if="item.type==='radio'" @click='item.func'></radio>
<checkbox v-if="item.type === 'checkbox'" @click='item.func'></checkbox>
…
</div>
<input type="button" @click="submit"></input>
```

formData 中存储了表单中组件的信息, 包括组件 type 属性以及相应的 value 值, 通过 v-model 实现 View 层和 Model 层数据的双向绑定, 当点击 submit 按钮时, 我们便可以取到用户的输入值, 经过相应的处理后存入后台数据库中.

最终设计出来的动态表单界面如图 4 所示. 界面的左侧显示了动态表单支持的所有标签组件, 右侧是组件属性的编辑区域, 中间则是设计好的表单的预览区域. 这三个区域全部由 Vue 组件设计完成, 用户只需要根据自己的需求在右边编辑区域对表单属性及内容进行设置, 即可在预览区域看到实时的表单界面效果.

图 4 动态表单界面

2.3 后端设计

后端的实现采用基于 Spring MVC 的 REST 架构, 客户端只需要发送 AJAX 请求, 然后服务器端接受该请求并返回 JSON 数据给客户端, 然后在客户端进行界面渲染. 后端需要做的工作只有访问数据库并给前端提供相应的数据接口即可. 前后端后端交互的架构图如图 5 所示.

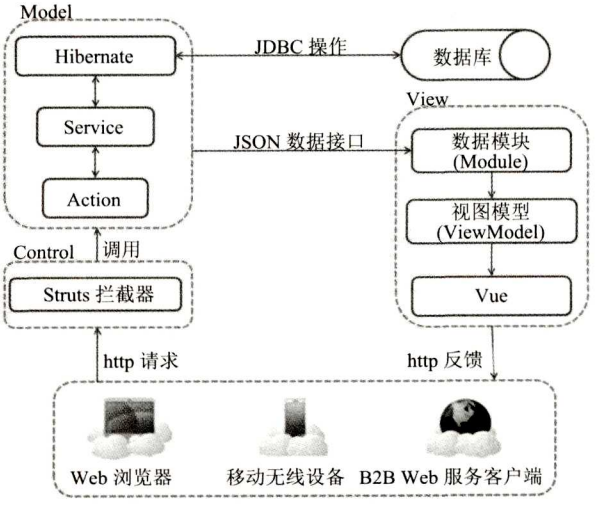

图 5 前后端技术架构图

REST 架构下需要重点解决对象序列化的问题, 服务端将 http 请求中的 JSON 格式参数转换为普通的 Java 对象 (POJO), 同时服务端向浏览器返回结果信息时也需要将普通的 Java 对象 (POJO) 转换为 JSON 字符串才能返回到浏览器中进行渲染. 在具体的实现上 Spring MVC 框架已经为我们提供了这类序列化的特性, 只需要在 Controller 的方法参数中使用@RequestBody 注释定义需要反序列化的参数即可, 若要对 Controller

方法的返回值进行序列化, 则需要在该返回值上使用 @ResponseBody 注释来定义, 然后修改 Spring 配置文件, 使用 Jackson 来提供 JSON 的序列化操作.

考虑到与组件相关的数据格式多变, 类型不统一, 因此对后端数据库的选择也是需要考虑的因素.

作为即时通信的动态表单, 其通信的数据主体是消息, 而消息数据一般是文本、图片、音频、视频等, 数据量大且格式多变, 数据库需要满足高并发读写、高扩展性和高可用性, 同时还要注意海量数据的高效率存储和访问, 传统的关系型数据库的 ACID(原子性、一致性、隔离性、持久性) 四大特性在这里就满足不了对复杂数据项的查询以及存取. 而 NoSQL 数据库有专门的针对应用程序设计的数据模型, 例如 MongoDB 的文档是类似 JSON 的 BSON 格式数据, 可以支持非常复杂的数据结构类型, 并且它还支持模式自由, 使开发者不需要事先定义好数据的结构, 可以根据需要随时添加字段, 同时数据库中也可以存储不同结构的文件[7].

例如我们创建 1 个单行文本的组件, 在右侧设置组件的属性字段, 如果要添加 1 个字段为 defContent 表示文本默认内容, 代码如下:

```
var jsondata= { id: "001",
type: "oneLineText",
title: "题目",
length: "10",
…,
defContent: "title" }
```

则前端页面用 ajax 请求后台的方法传的 data 数据就可以按照如下方式处理:

```
data: JSON.stringify(jsondata)
```

即可将 JSON 字符串传给后端.

由于 MongoDB 中是以 BSON 数据格式进行存储的, 所以首先要将 JSON 字符串转换成 DBObject 或者 Document 对象, 然后再对 MongoDB 数据库进行更新, 这里我们以后者为例进行说明, 关键代码如下:

```
Document document=Document.parse (jsondate);
collection.insertOne (document);
```

同时, 从数据库中取出的数据也需要转换为 JSON 格式才能够被前端解析器解析并渲染浏览器界面, 代码如下:

```
collection.find(document).toArray(function(err, docs){
    console.log(docs); }
```

前端就可以接收到一个包含 JSON 对象的数组.

综合以上分析, 选择 MongoDB 数据库来存储组件的数据项字段, 同时使用 JSON 数据接口进行前后端通信, 从而完成前后端数据的交互.

3 终端自适应

移动互联网风靡全球的时代, 表单的设计不仅需要在 PC 端有完善的功能, 在移动设备上也要有良好的适应性. 由于移动操作系统的多样性, 在屏幕尺寸、分辨率等方面和 PC 端也不尽相同, 同样的内容, 要在大小迥异的屏幕上都呈现出满意的效果, 并不是一件容易的事情. 很多网站的解决方案是为不同的设备提供不同的网页, 于是便出现了同一个系统有 PC 版本、mobile 版本、iPhone/iPad 版本, 这样做固然保证了效果, 但是工作量大, 同时要维护好几个版本, 而且如果一个网站有多个入口, 会大大增加架构设计的复杂度. 针对以上问题, 本文结合 HTML5 和 CSS3 新技术, 提出了 “一次设计, 普遍适用”的设计方案, 让网页自动适应不同大小的屏幕, 根据屏幕宽度自动调整布局.

响应式网页设计的实现主要通过以下 2 个方面进行实现: (1) 使用流式栅格布局; (2) 使用弹性盒子布局模型[8]. Bootstrap 提供了一套响应式、移动设备优先的流式栅格布局, 随着屏幕或者视口尺寸的增加, 系统会自动分为最多为 12 列, 该布局仅仅定义容器的大小, 通过一系列的行与列的组合创建页面布局, 再调整内外边距, 最后结合媒体查询, 就制作出了强大的响应式网络系统. 弹性盒子布局在原有的 DIV+CSS 布局的基础上摒弃了像素单位, 通过相对单位进行布局, 如使用百分比和 em, 弹性盒子由弹性容器和弹性子元素组成, 弹性盒子可以在页面结构不变的情况下进行显示顺序的调整, 以适应多终端的需要[9]. 在传统的布局方式中, block 布局是把块在垂直方向从上到下依次排列的, 而 inline 布局则是在水平方向来排列, 弹性盒子布局没有这样内在的方向限制, 可以由开发人员自由操作 .

以上两种方式都能实现多终端自适应表单的设计, 但是流式栅格布局需要设计者在设计之前就要对表单各功能模块之间的排列和布局有详细的规划, 一旦按照规划制作完成, 就不易进行修改了, 对于表单布局的设计有一定的限制. 而弹性盒子模型的布局模式更加

自由, 不受栅格布局固定模式的限制, 提高了动态表单的交互性. 所以本文选择弹性盒子模型来对表单布局进行设计.

由于浏览器窗口和移动设备窗口的分辨率不同, 必须要为智能手机窗口的布局进行优化. 不仅表格的大小要缩减, 其呈现的顺序也要进行相应的改变. 根据表单的各个模块的功能, 本文采用了弹性盒子布局中的 Holy Grail Layout(圣杯布局). 该布局从上到下分为三个主要的部分, 头部 (header)、躯干 (body)、尾部 (footer). 其中躯干部分有分为水平的三个部分, 依次为左侧栏、主栏和右侧栏分别对应动态表单的组件标签选项、表单预览区域、表单属性设置选项, 主栏部分是自适应的宽度, 左侧栏和右侧栏宽度是固定的. 如果是移动端的小屏幕, 则躯干的三栏就会自动的垂直叠加. 相关代码如下:

```
<div class="body ">
<div class= "content">...</div>
<aside class= "nav">...</aside>
<aside class= "ads">...</aside>
</div>
. body { display: flex;flex: 1; }
. content { flex: 1; }
.nav, . ads {
/* 两个边栏的宽度设为 12em */
flex: 0 0 12em;
}
```

如果是在移动端, 则将布局转换为垂直的结构, 代码如下:

```
@media (max-width: 768px) {
.body { flex-direction: column; flex: 1; }
.nav, .ads, .content { flex: auto; }
}
```

图 6 展现了动态表单从浏览器窗口到移动设备窗口的布局转换.

结合 HTML5+CSS3 Web 表单交互设计框架和模式, 适用性强、界面友好、可维护性高, 是终端自适应动态表单设计的关键部分.

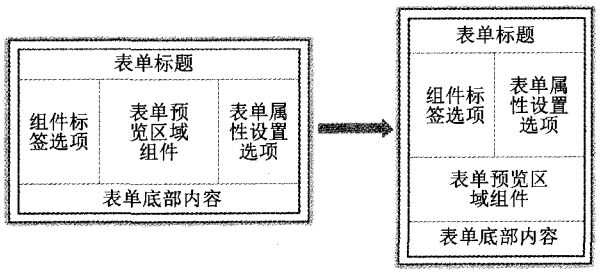

图 6 多终端表单布局图

4 结语与展望

本文从实际需求出发, 设计并初步实现了一种基于前后端分离的动态表单生成系统, 表单根据数据库的字段动态生成, 解决了由于业务变化需要对表单字段进行修改的问题, 并且针对跨平台多终端的问题提出了自适应的技术解决方案, 尝试通过引入当下流行的前端框架和技术手段来适应目前多终端的应用格局.

但是由于前端组件库发展还没有很成熟, 动态表单的设计并没有很完善, 同时两种终端自适应的方案对浏览器的兼容性和开发者的技术水平都有比较高的要求, 因此对于动态表单的设计在以后的工作中还需要不断的改进和完善.

参考文献

1 伍杰华. 基于 CSS3 的 HTML5 网页表单研究与定制. 计算机与信息技术, 2011, (12): 53–55.
2 易剑波. 基于 MVVM 模式的 WEB 前端框架的研究. 信息与电脑, 2016, (19): 76–77, 84. [doi: 10.3969/j.issn.1003-9767.2016.19.041]
3 周晖, 尹建伟, 陈刚, 等. 基于 Struts 框架的 Web 表单快速开发平台. 计算机应用研究, 2004, 21(8): 191–194.
4 吴昶成, 谈华宇, 邱小平. 科研管理系统中动态表单技术的应用与实现. 现代计算机, 2015, (7): 78–80.
5 洪英汉, 刘竹松, 龙桂和. 基于 SSH 框架的动态表单设计与实现. 现代计算机, 2009, (9): 186–188.
6 麦冬, 陈涛, 梁宗湾. 轻量级响应式框架 Vue.js 应用分析. 信息与电脑, 2017, (7): 58–59.
7 吕林. 基于 MongoDB 的应用平台的研究与实现[硕士学位论文]. 北京: 北京邮电大学, 2015.
8 梁仲智. 基于 HTML5 的跨终端 Web 生成系统的设计与实现[硕士学位论文].广州: 中山大学, 2013.
9 谢冠怀. 辨析响应式网页的浮动布局和伸缩盒子布局. 现代计算机, 2014, (15): 42–46, 50.