设计文档

选型

本次项目我们做的是一个Web项目（电影售票系统）。我们采用三层结构作为项目基本架构，即（表示层–>业务逻辑层–>数据访问层）。这种结构有非常多的实现，最后我们定的技术方案是前端Vue.js，后端Django+SQLite的组合。

本身Django是一个综合的Web框架，理论上前端部分应该由它本身的Template系统完成，即前后端不分离。而我们的技术方案选择的是前端使用Vue.js，弃用Django自带的Template系统，即前后端分离。选型原因主要有以下几点：

1.Django自带的Template功能不够强大，不足以支撑我们的需求。

2.前后端分离方便使用Restful API开发。

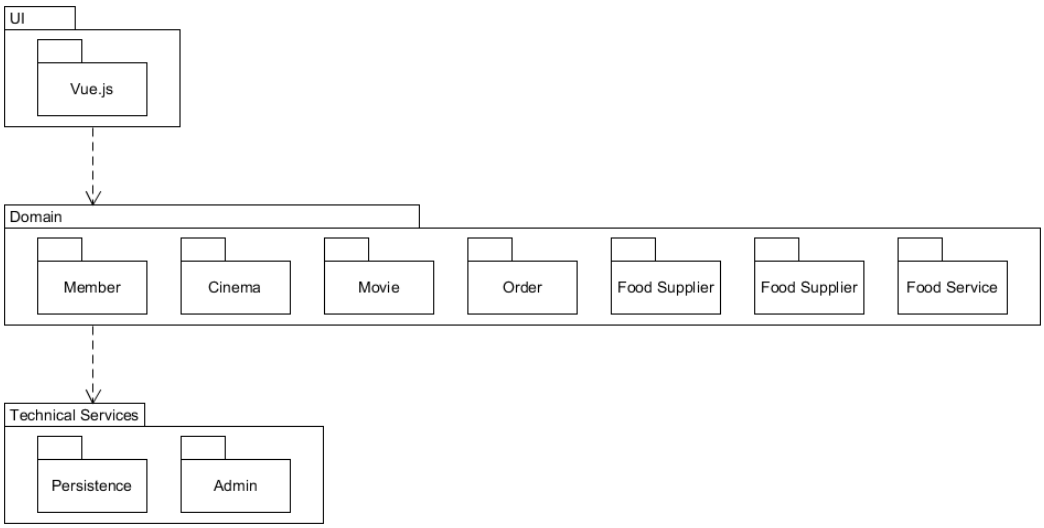
3.前后端分离方便协作开发。

总而言之，前后端分离降低了开发难度，提高了开发效率，因此我们采用这种技术方案。

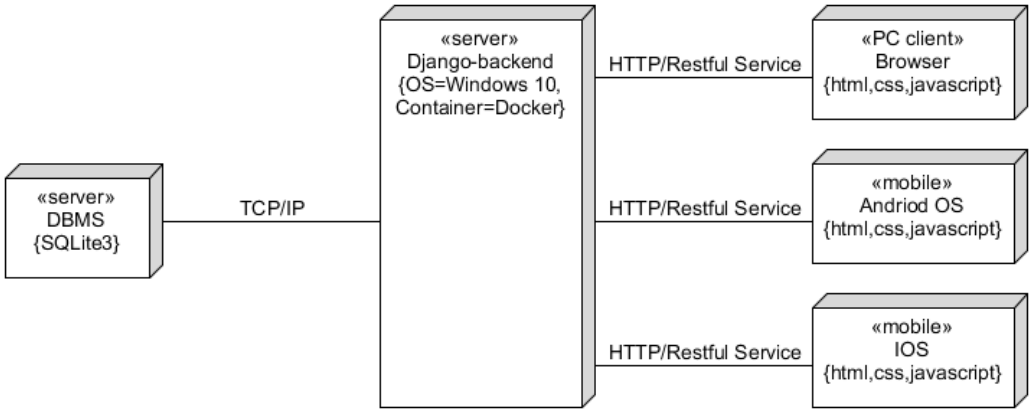
架构设计

项目架构

逻辑视图：



物理视图：

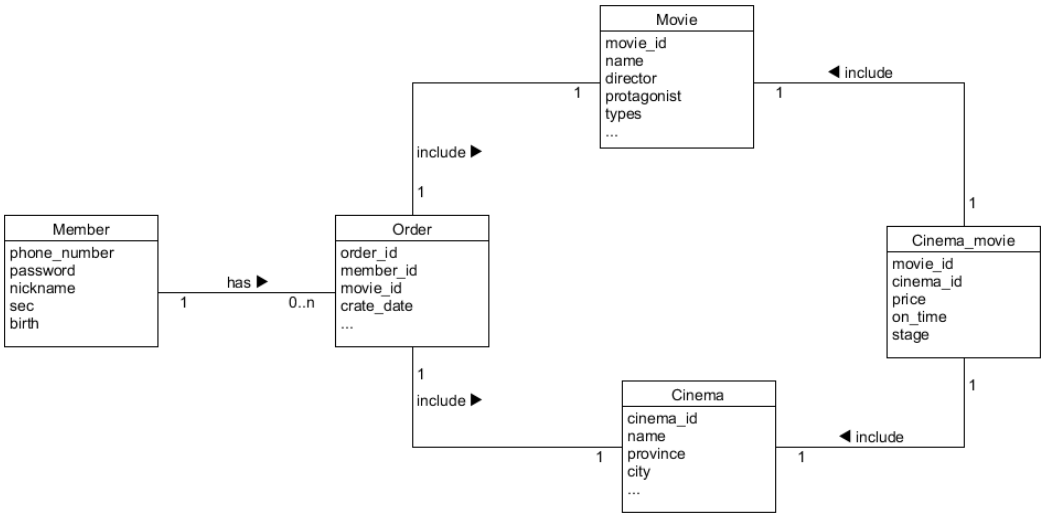


模块划分

项目后端实现主要分为5个模块：

1. Member
2. Movie
3. Cinema
4. Order
5. Cinema_movie

它们之间的关系如下：



软件设计技术

1. 使用了MVC架构,在Django的MTV模式的基础上将前端界面交互、业务逻辑控制以及数据模型分离开。
2. 使用了Rest架构方便前后端分离以及数据、代码的复用，加快开发进度。
3. 使用了面向对象编程，使得代码逻辑清晰，提高了程序的稳定性。

前端设计

技术选型及理由

本项目前端的技术栈为vue全家桶（vue + vue-router）+ axios + webpack

- vue： 当下最火最容易上手的MVVM框架
- vue-router: 前端路由
- axios：很强大很好上手的ajax请求库
- webpack：打包工具

选择这些技术的理由：
vue：
vue是国人开发的一款MVVM框架。它的特点是轻量 ，好上手 。考虑到前端开发同学的技术栈参差不齐，所以统一选用vue来作为开发框架，因为它好上手，学习的曲线是很平滑的，不像react和angular这类MVVM框架学习成本较高。其次就是轻量，vue的作者给广大开发者提供了一个脚手架工具，能够一键生成项目，生成的项目里有vuex，有vue-router，也有eslint这类工具，做到了开箱即用。对比react，react的比较出名的脚手架工具没有vue-cli这么方便，它还得自己根据需求去添加别的第三方插件，而且添加的成本略高，需要去看英文文档和写一些引入代码。综合对比react和angular在这两点上的不同，最后选择了vue。

vue-router：
vue-router是配套vue的一个单页面路由的插件。单页面与传统的网页最大不一样的地方就是，传统网页的路由是有后台去控制的，而单页面网页的路由是放在前台的，在前台中控制。故vue-router是必选的

axios：
选用的原因很简单，就是简单和强大。简单在于可以一行代码就发起一个http请求，支持promise，所以也就可以使用async await来以同步的形式写异步的代码。
强大在于提供了拦截器等功能。方便我们在发起请求前在header增加token，以及在收到回复中，分析结果，统一处理错误。

webpack:
webpack 是当下前端最流行的构建打包工具，利用 webpack 搭配相应的 loader，我们可以在前端项目中使用 es6 进行开发，效率更高。开发完成后，webpack 还可以帮我们完成构建，将代码编译到 es5 以兼容大部分浏览器。webpack 优化了构建打包的算法，优化前端的模块加载，使得构建打包后的产品代码体积更小。

前端架构

前端项目主要是页面样式开发，交互逻辑开发以及开发完后的部署。可分为开发环境和生产环境。而开发环境和生产环境都需要webpack来构建。webpack构建首先需要源代码，其次webpack是配置大于约定的，需要我们写配置文件去告诉它从哪个入口文件开始打包，打包哪种文件，如何打包等等，打包后还会生成一个可用于我们部署到服务器的文件。另外，由于前后端现在是分离的。前后端开发环境不一样，肯定会出现跨域等情况，又或者开发时，热更新可以帮助我们减少重复的操作等等，所以又需要额外的配置文件。故综合以上的分析，我们需要以下4个包

- build：根据config和执行环境来打包构建src中的代码，使其可运行，可调试

- config：项目的webpack打包配置、http代理配置
- src: 源码，平时开发写的代码都在这里
- dist：项目打包后生成的可部署文件

上面也说到我们的源码都在src里面。而一个可维护和可扩展的代码是需要做好文件架构的。根据单一职责原则，我们可以先把代码分割成 assets，router，utils，style和页面的UI和逻辑代码五个部分。
assets主要放一些资源，比如icon和图片。
router主要放前端的路由配置，这里指明了前端的页面的路由路径以及对应的组件。
style放了一些全局的样式和动画的样式，由于浏览器默认的标签样式不统一，为了让我们的网页在各个页面展示的效果一致，我们需要一个reset.css来统一标签的样式，也由于我们会用到一些比较好看的动画效果，有时候自己写比较麻烦，而已经有人做好了轮子，我们直接拿来用就是了，比如我们这次引用的animate.css。
utils就放了全局通用的方法，需要用到的话引入即可，不需要每个组件都写一份。
至于最后的页面UI和逻辑代码。我们这里采用了组件开发的思想，每个页面抽象为一个有很多组件组合而成的。我们需要把可复用的页面UI和逻辑抽象成为一个组件。那需要用到的页面直接使用就好。这样的思想，给我们带来了高复用，高内聚，低耦合的好处。故这部分又可以分为 components和pages两个部分。components就是可复用的组件。pages就是组装这些组件的代码。而且根据页面来命名文件，协作者一看到文件名就知道对应哪个页面，这样也提高了效率。

- components：可复用组件
- pages: 页面组件
- utils：公共工具函数
- style：公共样式、重置样式和动画样式
- assets：图标、图片等静态资源
- router：前端路由

后端设计

技术选型及理由

本项目选择的后端框架为Django框架。

选择Django的理由：
Django是一个基于MVC构造的框架。在Django中，控制器接受用户输入的部分由框架自行处理，所以Django里更关注的是模型（Model）、模板(Template)和视图（Views），称为MTV模式。Django基于MVC的设计十分优美，对象关系映射 (ORM,object-relational mapping)以Python类形式定义数据模型，将模型与关系数据库连接起来，能使我们得到一个非常容易使用的数据库API，同时也可以在Django中使用原始的SQL语句进行数据库操作。这大大方便了后端的开发。除此以外，Django的URL分派非常灵活，可以设计任意的URL，没有框架的特定限定。这使得我们能够更直接地实现Restful API。最后，Django自动化的管理界面不需要我们花大量的工作来创建人员管理和更新内容。Django自带一个ADMIN site,类似于内容管理系统。这使得我们在开发时能方便地进行后端的测试。

后端架构

后端只要负责从前端接受请求，完成业务逻辑并把前端所需要的数据返回给前端。因此后端主要包括了Controller以及Model的内容。前面已经提到，Django的MTV模式使得这些内容的实现大大简化。实现服务逻辑的函数都包含在views.py文件中。数据建模的代码包含在models.py文件中，而调用业务逻辑函数的url则包含在urls.py文件中。运行时，通过runserver启动Django服务器,载入在同一目录下的settings.py。该文件包含了项目中的配置信息。当访问url的时候Django会装载URLConf，按顺序逐个匹配URLConf里的URLpatterns。如果找到则会调用相关联的视图函数，并把HttpRequest对象作为第一个参数(通常是request)。最后该view函数通过对数据库的操作完成前端的请求。如果前端需要某些数据，view函数会把数据封装到一个HttpResponse对象返回给前端。