

【后端部分】

一、选型理由：

1. 选择 Java 的理由

- (1) Java 适合团队开发，软件工程可以相对做到规范。
- (2) Java 虚拟机技术使 Java 成为跨平台语言。
- (3) Java 是一种静态语言，有大量成熟框架支持，语言简单，健壮，不容易出错，可以做一个比较复杂的服务器系统。
- (4) Java 在内存管理方面有一定的优势，可以使用 JVM 的指令来进行性能调优。

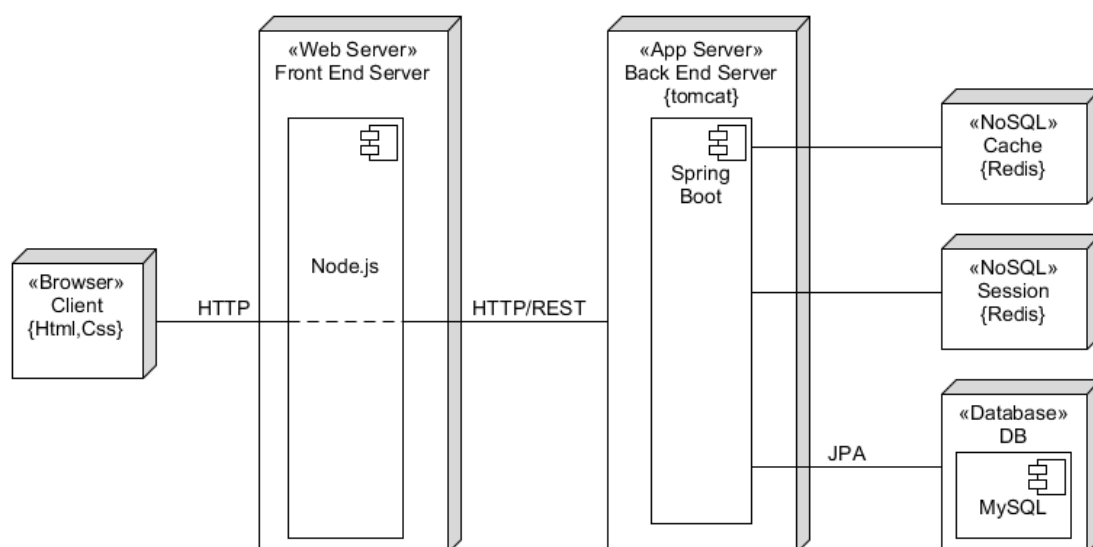
2. 选择 Spring 的理由

- (1) Spring 能有效地组织中间层对象。
- (2) Spring 能消除在许多工程中常见的对 Singleton 的过多使用。
- (3) Spring 能消除各种各样自定义格式的属性能文件的需要，使配置信息一元化。
- (4) Spring 能够帮助我们真正意义上实现 Restful 服务。
- (5) Spring 支持 JDBC 和 O/R Mapping 产品 (Hibernate)。
- (6) MVC Web 框架，提供一种清晰，无侵略性的 MVC 实现方式。
- (7) 简化访问数据库时的例外处理。

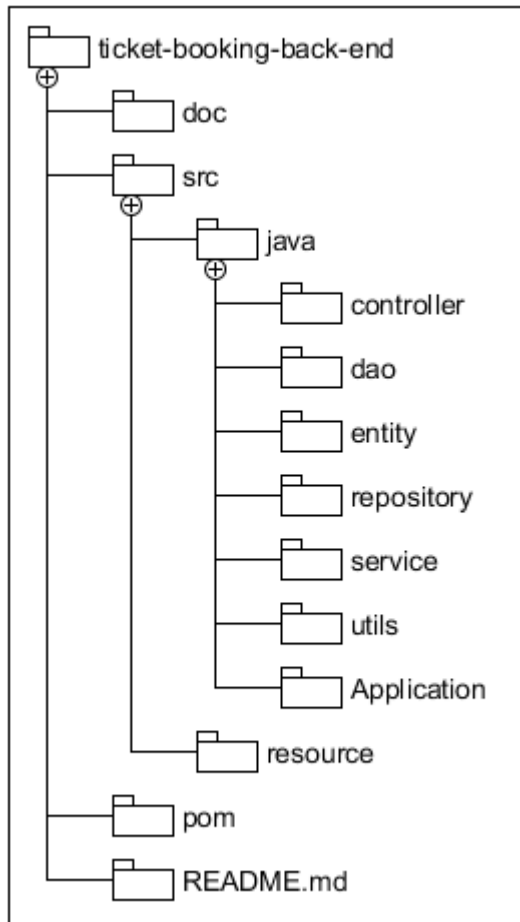
3. 选择 Spring Boot 的理由

- (1) Spring Boot 配置简单，不需要在 xml 里面对许多中间件进行配置，仅需要使用 Spring Boot 特有的配置注解即可完成相关中间件的基本配置。
- (2) Spring Boot 提供嵌入式 HTTP 服务器，如 Tomcat, Jetty 等，以开发和测试 Web 应用程序非常容易。
- (3) Spring Boot 提供了对应用的监控的中间件，有助于开发者对应用活动时期的监控活动。
- (4) Spring Boot 对 Spring 的中间件的简单包装使得开发过程相对简单，有助于小型团队的敏捷开发。

二、架构设计



三、模块划分



四、软件设计技术

(1) Structure Programming。结构化编程出现在项目的各个角落，这里仅仅展示一个例子

```
@GetMapping(value =("/{cinemaId}")
public ResponseEntity<?> getCinemaInfoById(@PathVariable long cinemaId) {
    Cinema cinema = cinemaService.findCinemaById(cinemaId);
    if (cinema != null) {
        List<Schedule> schedules = scheduleService.findSchedulesByCinemaId(cinemaId);
        List<Movie> movies = movieService.findMoviesByCinemaId(cinemaId);
        List<MovieDetail> movieDetails = new ArrayList<>();

        List<ScheduleForm> scheduleForms = new ArrayList<>();
        for (Schedule schedule : schedules) {
            scheduleForms.add(new ScheduleForm(schedule));
        }
        for (Movie movie : movies) {
            movieDetails.add(new MovieDetail(movie, scheduleForms));
        }

        return new ResponseEntity<>(new CinemaInfo(cinema, movieDetails), HttpStatus.OK);
    }
    throw new NotFoundException("电影院不存在");
}
```

(2) Object-Oriented Programming。Java 是一门面向对象的语言，因此面向对象涉及项目的方方面面，这里举个例子

```
@Entity
public class Cinema {

    @Id
    @GeneratedValue
    private long id;

    @Column(nullable = false)
    private String name;

    @Column(nullable = false)
    private String address;

    @Column(nullable = false)
    private float rank;

    public Cinema() {}

    public Cinema(String name, String address, float rank) {
        this.name = name;
        this.address = address;
        this.rank = rank;
    }
}
```

(3) Component-Oriented Programming 和单例模式。Java 中的 javaBean 规范是典型的组件，也是一种单例模式的实现，通过对特定类进行注解，能够标识出具体的 Bean。例如

```
@Service
@Transactional
public class CinemaServiceImpl implements CinemaService {

    private static final int HOT_CINEMAS_NUM = 6;
}
```

(4) RESTful API 设计。部分 API 按照 RESTful 的理念进行设计，例如

```
@GetMapping(value =("/{scheduleId}")
public ResponseEntity<> getScheduleById(@PathVariable long scheduleId) {
    Optional<Schedule> temp = scheduleService.findScheduleById(scheduleId);
    if (!temp.isPresent()) {
        throw new NotFoundException("没有找到对应的档期");
    }
    List<Ticket> ticketList = ticketService.getTicketsByScheduleId(scheduleId);
    if (ticketList == null || ticketList.isEmpty()) {
        throw new NotFoundException("没有票");
    }
    ScheduleTotal responseObject = new ScheduleTotal(temp.get(), ticketList);
    logger.info(responseObject.getStartTime());
    return new ResponseEntity<>(responseObject, HttpStatus.OK);
}
```

【前端部分】

一、选型理由：

1. 选择 Vue.js 的理由

- (1) 响应式编程：mvvm 框架，实现数据的双向绑定。
- (2) 组件化：一切都是组件，组件可以套其他组件，增强了可复用性。
- (3) 模块化：我们用一个模块打包工具来配合 Vue.js，比如 [Webpack](#) 或者 [Browserify](#)，然后再加上 ES2015。每一个 Vue 组件都可以看做一个独立的模块。
- (4) 动画：Vue 自带简洁易用的[过渡动画系统](#)。有很多[获奖的互动类网站](#)是用 Vue 开发的。Vue 的反应式系统也使得它可以用来开发高效的数据驱动的逐帧动画。
- (5) 路由：Vue 本身是不带路由功能的。但是，有 [vue-router](#) 这个可选的库来配合。vue-router 可以将嵌套的路径映射到嵌套的组件，并且提供了细致的路径跳转控制。
- (6) 文档和配套设施：文档和配套设施完善，社区活跃，生态系统完备，易于上手。

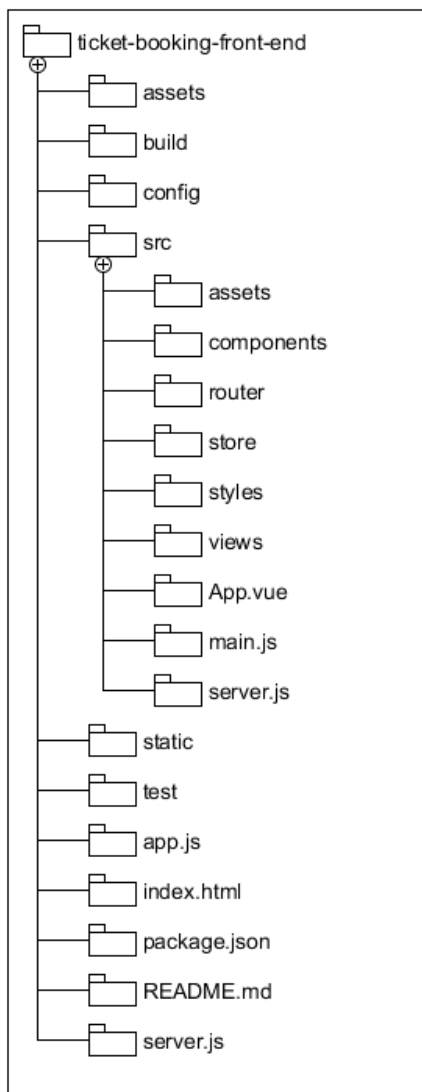
2. 选择 element-UI 的理由

- (1) 支持 Vue 2.x 组件库里最好的了，常用业务组件全面，功能丰富，有英文文档，生态齐全，支持 SSR。
- (2) Element-UI 生态更好，使用频率较高，开发团队实力较高。
- (3) element 有内置过渡动画,使得组件的切换变化,更具动感。
- (4) element 提供了 Sketch 和 Axure 工具 对设计人员友好。

二、架构设计



三、模块划分

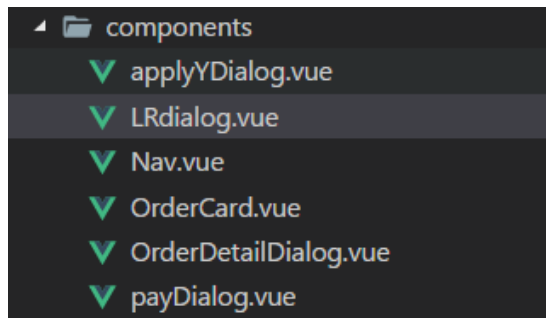


四、软件设计技术

(1) 模块化。使用 webpack 实现模块化：可管理静态资源的依赖，对所有模块一视同仁的依赖管理，引入和管理插件。

```
module: {
  loaders: [
    {
      test: /\.vue$/,
      loader: 'vue'
    },
    {
      test: /\.js$/,
      loader: 'babel',
      include: projectRoot,
      exclude: /node_modules/
    }
  ]
}
```

(2) 组件化。将可复用的元素写在 components 里，提高复用性，使用时仅需 Import 对应组件，便可在当前页面使用；



```
<script>
import card from '@components/ordercard'
export default {
  name: 'order',
  beforeRouteEnter: (to, from, next) => {
    next(vm => {
      if (!vm.$store.getters.getUserInfo.login) {
        vm.$message({
          type: 'error',
          message: '请先登录'
        })
        next({ path: '/home' })
        vm.$store.commit('toggleDiglog', 'LR')
      } else {
        next()
      }
    })
  },
}
```