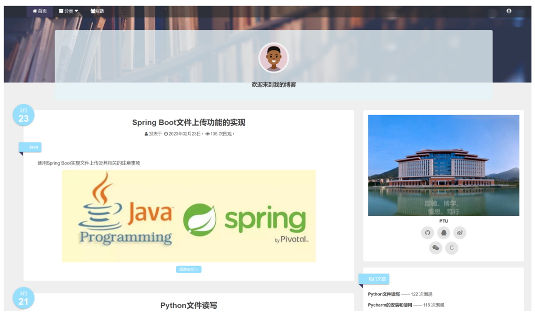
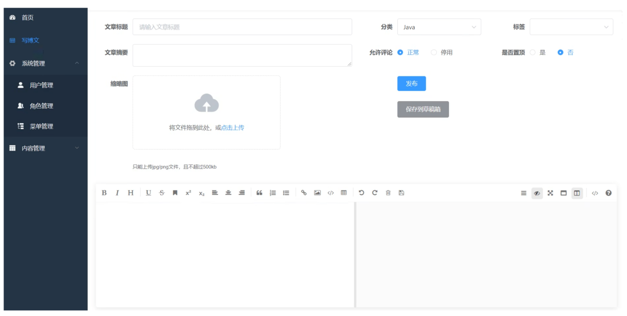
1. 多模块项目创建

博客系统包含前台和后台两套系统，两套系统的前端工程都已经提供好了，接下来只需要进行系统的后端开发


首页
分类
友链
欢迎来到我的博客
2月 23
Spring Boot文件上传功能的实现
发表于 2023年02月23日 • 105 次围观 •
Java
使用Spring Boot实现文件上传及其相关的注意事项
Java Programming
spring by Pivotal
阅读全文>>
厚德、博学、慎思、笃行
PTU
热门文章
Python文件读写 —— 122 次围观
Pycharm的安装和使用 —— 115 次围观
3月 21
Python文件读写


由于两套后端系统的很多代码是可能重复的，为了提高代码复用性，会把这些代码写到一个公共模块中，让前台系统和后台系统分别依赖公共模块。因此，后端创建的是一个多模块项目，包含3个子模块：公共模块framework、前台模块blog、后台模块admin

1. 创建父模块，在父模块的pom.xml中定义项目中所有子模块所使用的依赖项版本号，对其进行统一管理

```xml
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>1.8</java.version>
</properties>

<dependencyManagement>
    <dependencies>
        <!-- SpringBoot的依赖配置-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>2.7.10</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <!--fastjson依赖-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.33</version>
        </dependency>
        <!--jwt依赖-->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.0</version>
        </dependency>
        <!--mybatisPlus依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
```

```
                <version>3.5.3.1</version>
            </dependency>

            <!--阿里云OSS-->
            <dependency>
                <groupId>com.aliyun.oss</groupId>
                <artifactId>aliyun-sdk-oss</artifactId>
                <version>3.10.2</version>
            </dependency>

            <dependency>
                <groupId>com.alibaba</groupId>
                <artifactId>easyexcel</artifactId>
                <version>3.0.5</version>
            </dependency>

            <dependency>
                <groupId>io.springfox</groupId>
                <artifactId>springfox-swagger2</artifactId>
                <version>2.9.2</version>
            </dependency>
            <dependency>
                <groupId>io.springfox</groupId>
                <artifactId>springfox-swagger-ui</artifactId>
                <version>2.9.2</version>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>${java.version}</source>
                    <target>${java.version}</target>
                    <encoding>${project.build.sourceEncoding}</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

2. 创建公共模块 framework，在公共模块的pom.xml中加入项目会使用到的依赖

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!--lombk-->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
```

```xml
        <!--junit-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--SpringSecurity启动器-->
        <!--        <dependency>-->
        <!--            <groupId>org.springframework.boot</groupId>-->
        <!--            <artifactId>spring-boot-starter-security</artifactId>-->
        <!--        </dependency>-->
        <!--redis依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
        <!--fastjson依赖-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
        </dependency>
        <!--jwt依赖-->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
        </dependency>
        <!--mybatisPlus依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
        </dependency>
        <!--mysql数据库驱动-->
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>

        <!--阿里云OSS-->
        <dependency>
            <groupId>com.aliyun.oss</groupId>
            <artifactId>aliyun-sdk-oss</artifactId>
        </dependency>

        <!--AOP-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-aop</artifactId>
        </dependency>

        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>easyexcel</artifactId>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger-ui</artifactId>
        </dependency>
    </dependencies>
```

3. 创建前台模块 blog，在前台模块的pom.xml中使其依赖公共模块

```
<dependencies>
    <dependency>
        <groupId>org.example</groupId>
        <artifactId>framework</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
</dependencies>
```

4. 创建后台模块 admin，在后台模块的pom.xml中也使其依赖公共模块

```
<dependencies>
    <dependency>
        <groupId>org.example</groupId>
        <artifactId>framework</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
</dependencies>
```

2. 博客前台

2.1 准备工作

1. 创建数据库blog，导入SQL脚本blog.sql
2. 使用mybatis-plus-generator，配置好以下信息后，生成表格对应的代码，其中实体类，Mapper，Service复制到公共模块framework的项目所在包下，Controller复制到前台模块blog的项目所在包下

```
// 配置数据库连接信息
String username = "root";
String password = "123456";
String url = "jdbc:mysql://localhost:3306/blog?
serverTimezone=Asia/Shanghai&useUnicode=true&characterEncoding=utf-8";

// 要生成代码的表，多个表用逗号分割
String tableNames = "article,category,link,user,comment";

// 设置生成代码所在的包名
String packageName = "com.my.blog";

// 设置作者
String author = "WH";
```

3. 在前台模块blog的项目所在包下创建启动类BlogApplication

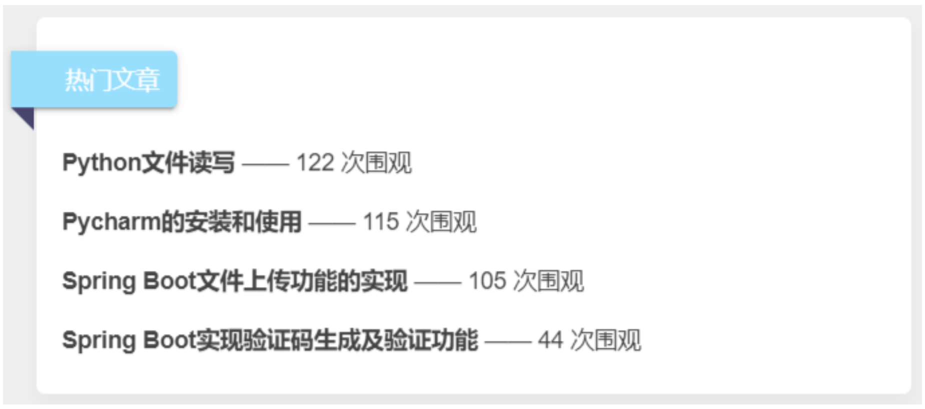
```java
@SpringBootApplication
@MapperScan("com.my.blog.dao")
public class BlogApplication {
    public static void main(String[] args) {
        SpringApplication.run(BlogApplication.class, args);
    }
}
```

4. 在前台模块blog的resources目录下创建application.yml配置文件

```yaml
server:
  port: 7777

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/blog?
serverTimezone=Asia/Shanghai&useUnicode=true&characterEncoding=utf-8
    username: root
    password: 123456
  servlet:
    multipart:
      max-file-size: 2MB
      max-request-size: 5MB

mybatis-plus:
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
  global-config:
    db-config:
      logic-delete-field: delFlag
      logic-delete-value: 1
      logic-not-delete-value: 0
      id-type: auto
```

2.2 热门文章列表

2.2.1 需求


热门文章

Python文件读写 —— 122 次围观

Pycharm的安装和使用 —— 115 次围观

Spring Boot文件上传功能的实现 —— 105 次围观

Spring Boot实现验证码生成及验证功能 —— 44 次围观

需要查询浏览量最高的前10篇文章的信息。要求展示文章标题和浏览量，能让用户自己点击跳转到具体的文章详情进行浏览。

注意：不能把草稿文章查询出来，要按照浏览量进行降序排序。

2.2.2 接口设计

请求方式	请求路径
Get	/article/hotArticleList

响应格式

```
{
    "code":200,
    "data":[
        {
            "id":"1",
            "title":"Spring Boot文件上传功能的实现",
            "viewCount":"105"
        }
    ],
    "msg":"操作成功"
}
```

2.2.3 基础版本代码实现

1. 准备工作

   统一响应类和响应枚举

   ```java
   @JsonInclude(JsonInclude.Include.NON_NULL)
   public class ResponseResult<T> implements Serializable {
       private Integer code;
       private String msg;
       private T data;

       public ResponseResult() {
           this.code = AppHttpCodeEnum.SUCCESS.getCode();
           this.msg = AppHttpCodeEnum.SUCCESS.getMsg();
       }

       public ResponseResult(Integer code, T data) {
           this.code = code;
           this.data = data;
       }

       public ResponseResult(Integer code, String msg, T data) {
           this.code = code;
           this.msg = msg;
           this.data = data;
       }

       public ResponseResult(Integer code, String msg) {
           this.code = code;
           this.msg = msg;
       }

       public static ResponseResult errorResult(int code, String msg) {
           ResponseResult result = new ResponseResult();
           return result.error(code, msg);
   ```

```java
    }

    public static ResponseResult okResult() {
        ResponseResult result = new ResponseResult();
        return result;
    }

    public static ResponseResult okResult(int code, String msg) {
        ResponseResult result = new ResponseResult();
        return result.ok(code, null, msg);
    }

    public static ResponseResult okResult(Object data) {
        ResponseResult result = setAppHttpCodeEnum(AppHttpCodeEnum.SUCCESS,
AppHttpCodeEnum.SUCCESS.getMsg());
        if (data != null) {
            result.setData(data);
        }
        return result;
    }

    public static ResponseResult errorResult(AppHttpCodeEnum enums) {
        return setAppHttpCodeEnum(enums, enums.getMsg());
    }

    public static ResponseResult errorResult(AppHttpCodeEnum enums, String
msg) {
        return setAppHttpCodeEnum(enums, msg);
    }

    public static ResponseResult setAppHttpCodeEnum(AppHttpCodeEnum enums) {
        return okResult(enums.getCode(), enums.getMsg());
    }

    private static ResponseResult setAppHttpCodeEnum(AppHttpCodeEnum enums,
String msg) {
        return okResult(enums.getCode(), msg);
    }

    public ResponseResult<?> error(Integer code, String msg) {
        this.code = code;
        this.msg = msg;
        return this;
    }

    public ResponseResult<?> ok(Integer code, T data) {
        this.code = code;
        this.data = data;
        return this;
    }

    public ResponseResult<?> ok(Integer code, T data, String msg) {
        this.code = code;
        this.data = data;
        this.msg = msg;
        return this;
    }
    public ResponseResult<?> ok(T data) {
        this.data = data;
        return this;
    }

    public Integer getCode() {
        return code;
    }

    public void setCode(Integer code) {
        this.code = code;
    }

    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
        this.msg = msg;
    }

    public T getData() {
        return data;
    }

    public void setData(T data) {
        this.data = data;
    }
}
```

```java
public enum AppHttpCodeEnum {
    SUCCESS(200, "操作成功"),
    NEED_LOGIN(401, "需要登录后操作"),
    NO_OPERATOR_AUTH(403, "无权限操作"),
    SYSTEM_ERROR(500, "出现错误"),
    USERNAME_EXIST(501, "用户名已存在"),
    PHONENUMBER_EXIST(502, "手机号已存在"),
    EMAIL_EXIST(503, "邮箱已存在"),
    REQUIRE_USERNAME(504, "必需填写用户名"),
    LOGIN_ERROR(505, "用户名或密码错误");

    int code;
    String msg;

    AppHttpCodeEnum(int code, String errorMessage) {
        this.code = code;
        this.msg = errorMessage;
    }

    public int getCode() {
        return code;
    }

    public String getMsg() {
        return msg;
    }
}
```

2. 代码实现

```java
@Controller
@RequestMapping("/article")
public class ArticleController {

    @Autowired
    private IArticleService articleService;

    @GetMapping("/hotArticleList")
    @ResponseBody
    public ResponseResult hotArticleList(){
        ResponseResult result =  articleService.hotArticleList();
        return result;
    }
}
```

```java
public interface IArticleService extends IService<Article> {
    ResponseResult hotArticleList();
}
```

```java
@Service
public class ArticleServiceImpl extends ServiceImpl<ArticleMapper, Article>
implements IArticleService {

    @Autowired
    ArticleMapper articleMapper;

    @Override
    public ResponseResult hotArticleList() {
        //查询热门文章 封装成ResponseResult返回
        LambdaQueryWrapper<Article> queryWrapper = new LambdaQueryWrapper<>
();
        //必须是正式文章
        queryWrapper.eq(Article::getStatus, 0);
        //按照浏览量进行排序
        queryWrapper.orderByDesc(Article::getViewCount);
        //最多只查询10条
        Page<Article> page = new Page(1, 10);
        articleMapper.selectPage(page, queryWrapper);

        List<Article> articles = page.getRecords();
        return ResponseResult.okResult(articles);
    }
}
```

配置MybatisPlus分页插件

```java
@Configuration
public class MybatisPlusConfig {

    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor() {
        MybatisPlusInterceptor mybatisPlusInterceptor = new
MybatisPlusInterceptor();
        mybatisPlusInterceptor.addInnerInterceptor(new
PaginationInnerInterceptor());
        return mybatisPlusInterceptor;
    }
}
```

3. 解决跨域问题。跨域问题指的是在浏览器中，当一个网页向不同域名、不同端口、不同协议的服务器发送请求时，会被浏览器拦截，因为这种行为可能会引起安全问题

```java
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        // 设置允许跨域的路径
        registry.addMapping("/**")
                // 设置允许跨域请求的域名
                .allowedOriginPatterns("*")
                // 是否允许cookie
                .allowCredentials(true)
                // 设置允许的请求方式
                .allowedMethods("GET", "POST", "DELETE", "PUT")
                // 设置允许的header属性
                .allowedHeaders("*")
                // 跨域允许时间
                .maxAge(3600);
    }
}
```

2.2.4 字面值处理

实际项目中都不允许直接在代码中使用字面值。都需要定义成常量来使用。这种方式有利于提高代码的可维护性。

```java
public class SystemConstants{
    /**
     *  文章是草稿状态
     */
    public static final int ARTICLE_STATUS_DRAFT = 1;
    /**
     *  文章是正常发布状态
     */
    public static final int ARTICLE_STATUS_NORMAL = 0;
}
```

```java
@Override
public ResponseResult hotArticleList() {
    ...
    queryWrapper.eq(Article::getStatus, SystemConstants.ARTICLE_STATUS_NORMAL);
    ...
}
```

2.2.5 使用VO优化

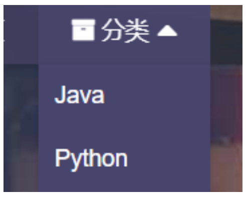
目前我们的响应是不符合接口文档标准的，多返回了很多字段。这是因为我们查询的结果是直接用Article封装的，Article中字段比较多。我们在项目中一般还要用VO来封装查询出来的结果。一个接口对应一个VO，这样即使接口响应字段要修改也只要改VO即可。

```java
@Data
public class HotArticleVo {
    private Long id;
    //标题
    private String title;

    //访问量
    private Long viewCount;
}
```

```java
@Override
public ResponseResult hotArticleList() {
    ...
    List<Article> articles = page.getRecords();

    //bean拷贝
    List<HotArticleVo> articleVos = new ArrayList<>();
    for (Article article : articles) {
        HotArticleVo vo = new HotArticleVo();
        BeanUtils.copyProperties(article, vo);
        articleVos.add(vo);
    }
    return ResponseResult.okResult(articleVos);
}
```

2.3 查询分类列表

2.3.1 需求



页面上需要展示分类列表，用户可以点击具体的分类查看该分类下的文章列表。

注意： ①要求只展示有发布正式文章的分类 ②必须是正常状态的分类

2.3.2 接口设计

请求方式	请求路径
Get	/category/getCategoryList

响应格式

```
{
    "code":200,
    "data":[
        {
            "id":"1",
            "name":"Java"
        },
        {
            "id":"2",
            "name":"Python"
        }
    ],
    "msg":"操作成功"
}
```

2.3.3 代码实现

CategoryController

```
@Autowired
private ICategoryService categoryService;

@GetMapping("/getCategoryList")
@ResponseBody
public ResponseResult getCategoryList() {
    return categoryService.getCategoryList();
}
```

ICategoryService

```
ResponseResult getCategoryList();
```

CategoryServiceImpl

```
@Override
public ResponseResult getCategoryList(){...};
```

CategoryVo

```
@Data
public class CategoryVo {
    private Long id;
    private String name;
}
```

2.4 分类查询文章列表

2.4.1 需求

在首页和分类页面都需要查询文章列表。

首页：查询所有的文章

分类页面：查询对应分类下的文章

要求：①只能查询正式发布的文章 ②置顶的文章要显示在最前面

2.4.2 接口设计

请求方式	请求路径	请求参数
Get	/article/articleList	pageNum, pageSize, categoryId

响应格式

```
{
    "code":200,
    "data":{
        "rows":[
            {
                "categoryName":"Java",
                "createTime":"2023-02-23 23:20:11",
                "id":"1",
                "summary":"使用Spring Boot实现文件上传及其相关的注意事项",

 "thumbnail":"https://img1.baidu.com/it/u=4026470308,2412268569&fm=253&fmt=auto&app=138&f=JPEG?w=824&h=500",
                "title":"Spring Boot文件上传功能的实现",
                "viewCount":"105"
            },
            {
                "categoryName":"Python",
                "createTime":"2023-03-21 14:58:30",
                "id":"2",
                "summary":"Python读写文本文件学习笔记",
                "thumbnail":"https://cdn2.byhy.net/imgs/gh/36462795_36383834-5d7bc92c-15c8-11e8-8821-da79d117d45c.png",
                "title":"Python文件读写",
                "viewCount":"122"
            }
        ],
        "total":"2"
    },
    "msg":"操作成功"
}
```

2.4.3 代码实现

ArticleController

```
@GetMapping("/articleList")
@ResponseBody
public ResponseResult articleList(Integer pageNum, Integer pageSize, Long
categoryId) {
    return articleService.articleList(pageNum, pageSize, categoryId);
}
```

IArticleService

```
ResponseResult articleList(Integer pageNum, Integer pageSize, Long categoryId);
```

ArticleServiceImpl

```
@Override
public ResponseResult articleList(Integer pageNum, Integer pageSize, Long
categoryId) {...}
```

ArticleListVo

```
@Data
public class ArticleListVo {
    private Long id;
    private String title;
    private String summary;
    private String categoryName;
    private String thumbnail;
    private Long viewCount;
    private LocalDateTime createTime;
}
```

PageVo

```
@Data
@AllArgsConstructor
public class PageVo {
    private List rows;
    private Long total;
}
```

2.4.4 FastJson配置

在WebConfig中配置FastJson消息转换器，优化日期时间显示

```
// 使用@Bean注入fastJsonHttpMessageConvert
@Bean
public HttpMessageConverter fastJsonHttpMessageConverters() {
    // 定义一个Convert转换消息的对象
    FastJsonHttpMessageConverter fastConverter = new
FastJsonHttpMessageConverter();
    FastJsonConfig fastJsonConfig = new FastJsonConfig();
    fastJsonConfig.setSerializerFeatures(SerializerFeature.PrettyFormat);
    fastJsonConfig.setDateFormat("yyyy-MM-dd HH:mm:ss");

    SerializeConfig.globalInstance.put(Long.class, ToStringSerializer.instance);

    fastJsonConfig.setSerializeConfig(SerializeConfig.globalInstance);
    fastConverter.setFastJsonConfig(fastJsonConfig);
    HttpMessageConverter<?> converter = fastConverter;
    return converter;
}

@Override
public void configureMessageConverters(List<HttpMessageConverter<?>> converters)
{
    converters.add(fastJsonHttpMessageConverters());
}
```

2.5 文章详情接口

2.5.1 需求

在文章列表点击阅读全文时能够跳转到文章详情页面，可以让用户阅读文章正文。

2.5.2 接口设计

请求方式	请求路径
Get	/article/{id}

响应格式：

```
{
    "code":200,
    "data":{
        "categoryId":"1",
        "categoryName":"Java",
        "content":"文章内容",
        "createTime":"2023-02-23 23:20:11",
        "id":"1",
        "isComment":"0",
        "title":"Spring Boot文件上传功能的实现",
        "viewCount":"105"
    },
    "msg":"操作成功"
}
```

2.5.3 代码实现

ArticleController

```
@GetMapping("/{id}")
@ResponseBody
public ResponseResult getArticleDetail(@PathVariable("id") Long id) {
    return articleService.getArticleDetail(id);
}
```

IArticleService

```
ResponseResult getArticleDetail(Long id);
```

ArticleServiceImpl

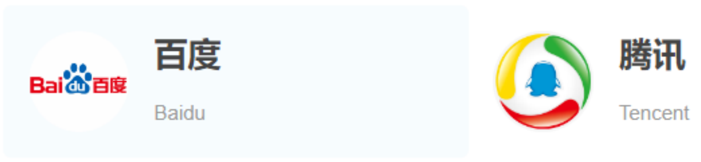
```
@Override
public ResponseResult getArticleDetail(Long id) {...}
```

ArticleDetailVo

```
@Data
public class ArticleDetailVo {
    private Long categoryId;
    private String categoryName;
    private String content;
    private LocalDateTime createTime;
    private Long id;
    private String isComment;
    private String title;
    private Long viewCount;
}
```

2.6 友链查询

2.6.1 需求



在友链页面要查询出所有的审核通过的友链。

2.6.2 接口设计

请求方式	请求路径
Get	/link/getAllLink

响应格式:

```
{
    "code":200,
    "data":[
        {
            "address":"https://www.baidu.com",
            "description":"Baidu",
            "id":"1",
            "logo":"https://storage-public.zhaopin.cn/15427893323/3b46ed.jpg",
            "name":"百度"
        },
        {
            "address":"https://www.qq.com",
            "description":"Tencent",
            "id":"2",

"logo":"https://bpic.51yuansu.com/pic3/cover/00/69/38/58abe_610.jpg",
            "name":"腾讯"
        }
    ],
    "msg":"操作成功"
}
```

2.6.3 代码实现

LinkController

```
@Autowired
private ILinkService linkService;

@GetMapping("/getAllLink")
@ResponseBody
public ResponseResult getAllLink(){
    return linkService.getAllLink();
}
```

ILinkService

```
ResponseResult getAllLink();
```

LinkServiceImpl

```
@Override
public ResponseResult getAllLink() {...}
```

LinkVo

```
@Data
public class LinkVo {
    private Long id;
    private String name;
    private String logo;
    private String description;
    private String address;
}
```