

1 背景

在日常生活中，内容管理系统占到所有系统的80%以上，比如个人博客，腾讯新闻，今日头条，搜狐新闻等。

这些都是我们日常生活中经常使用到的系统，它的核心就是可以概括为内容（资讯），一篇文章就是一个资讯，大家越来越喜欢将生活中的所见所闻简述出来。

如图：



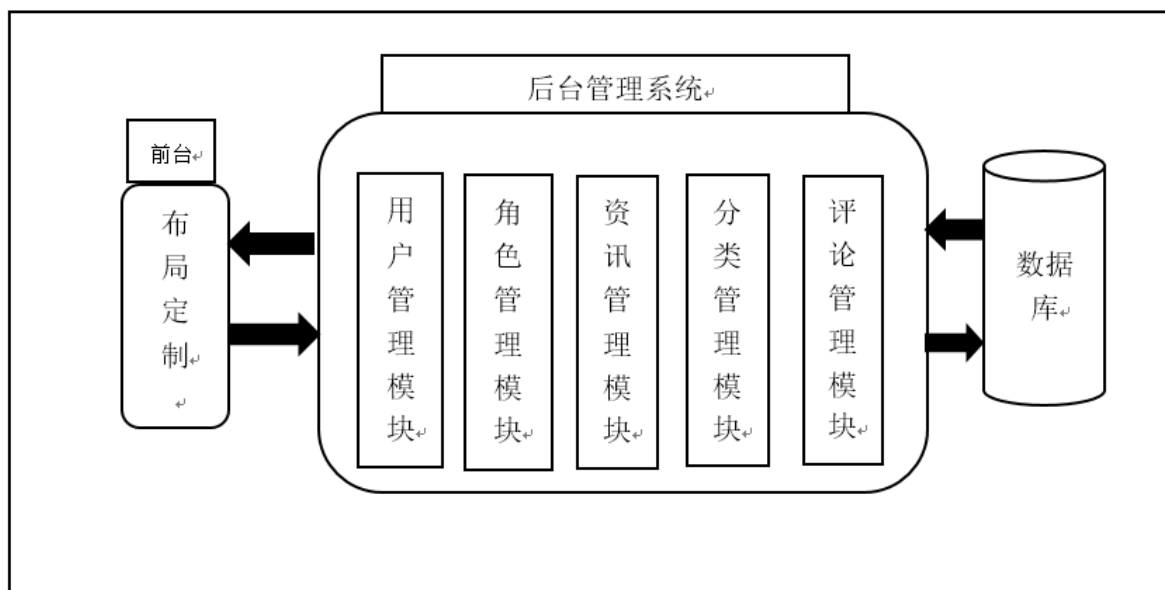
可以看出，目前来讲对于资讯系统的需求量是非常大的

通用的看点资讯管理系统，采用前后台分离开发的技术，将系统分为前台页面展示以及后台资讯管理系统。对于后台系统是基本上是通用的，而前台可根据不同客户的需求去修改定制。

2 简介

看点资讯系统最核心的数据是内容，例如文章、新闻等，所以后台主要功能是实现这些内容的管理，例如栏目管理、评论管理、用户管理以及角色管理等，对于前台可以灵活得展示后台的数据。

例如，



3 技术

后台采用的技术：

- springboot
- spring
- springmvc
- springdata-jpa

数据库：

- mysql

文档管理：

- swagger

架构选取：

- 三层架构

4 功能

注意，Controller中的返回值类型，统一采用自定义的Result类型

```
1 package com.briup.cms.util;  
2  
3 import java.io.Serializable;  
4  
5 /**  
6  * 统一Controller中RESTFul风格接口返回的结果
```

```

7  */
8  public class Result implements Serializable {
9
10     private static final long serialVersionUID = 1L;
11
12     private Integer code;
13     private String msg;
14     private Object data;
15
16     private Result() {}
17
18     private Result(Integer code, String msg) {
19         this.code = code;
20         this.msg = msg;
21     }
22
23     private void setResultCode(ResultCode code) {
24         this.code = code.code();
25         this.msg = code.message();
26     }
27
28     /**
29      * 操作失败, 自定义code和msg
30      */
31     public static Result failure(Integer code, String msg) {
32         Result result = new Result(code,msg);
33         return result;
34     }
35
36     /**
37      * 操作成功, 没有返回的数据
38      */
39     public static Result success() {
40         Result result = new Result();
41         result.setResultCode(ResultCode.SUCCESS);
42         return result;
43     }
44
45     /**
46      * 操作成功, 有返回的数据
47      */
48     public static Result success(Object data) {
49         Result result = new Result();
50         result.setResultCode(ResultCode.SUCCESS);
51         result.setData(data);
52         return result;
53     }
54
55     /**
56      * 操作失败, 没有返回的数据
57      */
58     public static Result failure(ResultCode resultCode) {
59         Result result = new Result();
60         result.setResultCode(resultCode);
61         return result;
62     }
63
64     /**

```

```

65      * 操作失败，有返回的数据
66      */
67      public static Result failure(ResultCode resultCode, Object data) {
68          Result result = new Result();
69          result.setResultCode(resultCode);
70          result.setData(data);
71          return result;
72      }
73
74      public Integer getCode() {
75          return code;
76      }
77
78      public void setCode(Integer code) {
79          this.code = code;
80      }
81
82      public String getMsg() {
83          return msg;
84      }
85
86      public void setMsg(String msg) {
87          this.msg = msg;
88      }
89
90      public Object getData() {
91          return data;
92      }
93
94      public void setData(Object data) {
95          this.data = data;
96      }
97
98  }
99

```

```

1  package com.briup.cms.util;
2
3  /**
4   * 统一并自定义返回状态码，如有需求可以另外增加
5   */
6  public enum ResultCode {
7
8      /* 默认成功状态码 */
9      SUCCESS(1, "操作成功"),
10
11     /* 默认失败状态码 */
12     ERROR(2, "操作失败，未知指定错误信息"),
13
14     /* 参数错误: 10001-19999 */
15     PARAM_IS_INVALID(10001, "参数无效"),
16     PARAM_IS_BLANK(10002, "参数为空"),
17     PARAM_TYPE_BIND_ERROR(10003, "参数类型错误"),
18     PARAM_NOT_COMPLETE(10004, "参数缺失"),
19
20

```

```

21      /* 用户错误: 20001-29999 */
22      USER_NOT_LOGIN(20001, "用户未登录"),
23      USER_LOGIN_ERROR(20002, "账号不存在或密码错误"),
24      USER_ACCOUNT_FORBIDDEN(20003, "账号已被禁用"),
25      USER_NOT_EXIST(20004, "用户不存在"),
26      USER_HAS_EXISTED(20005, "用户已存在"),
27
28      /* 业务错误: 30001-39999 */
29      SPECIFIED_QUESTIONED_USER_NOT_EXIST(30001, "业务逻辑出现问题"),
30
31      /* 系统错误: 40001-49999 */
32      SYSTEM_INNER_ERROR(40001, "系统内部错误, 请稍后重试"),
33
34      /* 数据错误: 50001-599999 */
35      DATA_NONE(50001, "数据未找到"),
36      DATA_WRONG(50002, "数据错误"),
37      DATA_EXISTED(50003, "数据已存在"),
38
39
40      /* 接口错误: 60001-69999 */
41      INTERFACE_INNER_INVOKE_ERROR(60001, "内部系统接口调用异常"),
42      INTERFACE_OUTTER_INVOKE_ERROR(60002, "外部系统接口调用异常"),
43      INTERFACE_FORBID_VISIT(60003, "该接口禁止访问"),
44      INTERFACE_ADDRESS_INVALID(60004, "接口地址无效"),
45      INTERFACE_REQUEST_TIMEOUT(60005, "接口请求超时"),
46
47      /* 权限错误: 70001-79999 */
48      PERMISSION_NO_ACCESS(70001, "无访问权限");
49
50
51      private Integer code;
52
53      private String message;
54
55      ResultCode(Integer code, String message) {
56          this.code = code;
57          this.message = message;
58      }
59
60      public Integer code() {
61          return this.code;
62      }
63
64      public String message() {
65          return this.message;
66      }
67  }
68

```

4.1 用户模块

对应的实体类为 **User**

1、新增用户

- 参数1：User

2、编辑用户

- 参数1：User

3、查询用户（分页）

- 参数1：Integer num(页数)
- 参数2：Integer size（每页显示多少条数据）

4、删除用户（批量，注销用户）

- 参:1：List<Long> ids

5、禁用或解封用户信息（更新用户状态）

- 参数1：Long id
- 参数2：String status

6、根据用户名获取用户信息

- 参数1：String username

7、登录

- 参数1：String username
- 参数2：String password

4.2 角色模块

对应的实体类为 **Role**

1、新增角色

- 参数1：Role r

2、编辑角色

- 参数1：Role r

3、查询角色（分页）

- 参数1：Integer num(页数)
- 参数2：Integer size（每页显示多少条数据）

4、批量删除角色

- 参数1：List<Long> ids

4.3 资讯模块

资讯在这里主要指的是文章，对应的实体类为 `Article`

1、新增资讯

- 参数1：Article article

2、编辑资讯

- 参数1：Article article

3、查询资讯（分页）

- 参数1：Integer num(页数)
- 参数2：Integer size（每页显示多少条数据）

4、批量删除资讯

- 参数1：List<Long> ids

5、审核资讯

- 参数1：Long id
- 参数2：String status

6、根据类别ID查询，分页获取资讯信息，并且按照阅读量降序排

- 参数1：Long categoryId
- 参数2：Integer pageNum
- 参数3：Integer pageSize

7、根据用户分页获取资讯信息

- 参数1：Long userId
- 参数2：Integer pageNum
- 参数3：Integer pageSize

4.4 类别模块

类别也就是栏目：`Category`

1、新增类别

- 参数1：Category category

2、编辑类别

- 参数1：Category category

3、查询类别（分页）

- 参数1：Integer num(页数)
- 参数2：Integer size（每页显示多少条数据）

4、批量删除类别

- 参数1：List<Long> ids

5、更新类别序号

- 参数1：Long id
- 参数2：Long no

6、按照序号升序分页获取类别信息

- 参数1：Integer num(页数)
- 参数2：Integer size（每页显示多少条数据）

4.5 评论模块

对应的实体类为 `Comment`

1、新增评论

- 参数1：Comment comment

2、编辑评论

- 参数1：Comment comment

3、批量删除评论

- 参数1：List<Long> ids

4、查询评论（分页）

- 参数1：Integer num(页数)
- 参数2：Integer size（每页显示多少条数据）

5、分页获取指定文章下所有的评论

- 参数1：Long articleId
- 参数2：Integer num
- 参数3：Integer size

5 字典

1、用户表，cms_user

字段名	类型	约束	说明
id	int	PK	编号
username	varchar	NN,UK	用户名
password	varchar	NN	密码
phone	varchar		手机
realname	varchar		真实姓名
gender	varchar		性别
birthday	datetime		生日
register_time	datetime		注册时间
status	varchar	NN	状态（正常0、禁用1）
image	varchar		头像地址
role_id	int	FK	角色ID

2、角色表，cms_role

字段名	类型	约束	说明
id	int	PK	编号
name	varchar	NN	角色名
description	varchar		角色描述信息

3、文章类别表，cms_category

字段名	类型	约束	说明
id	int	PK	编号
name	varchar	NN	类别名称
description	varchar		类别描述信息
no	int		类别序号
parent_id	int	FK	父类别ID

4、文章表，cms_article

字段名	类型	约束	说明
id	int	PK	编号
title	varchar	NN	标题
content	longtext	NN	内容
publish_time	datetime		发布时间
status	int		状态（未审核0，审核通过1，审核失败2）
read_times	int		阅读量
thumb_up	int		点赞数
thumb_down	int		被踩数
user_id	int	FK	所属用户ID
category_id	int	FK	所属类别ID

5、评论表，cms_comment

字段名	类型	约束	说明
id	int	PK	编号
content	varchar	NN	评论内容
time	varchar	NN	评论时间
user_id	int	FK	评论人ID
article_id	int	FK	评论的文章ID
parent_id	int	FK	父评论ID，可以评论进行再评论

注意，因为项目中使用了JPA，可以在实体类中配置完关系映射后，让JPA根据配置自动创建表

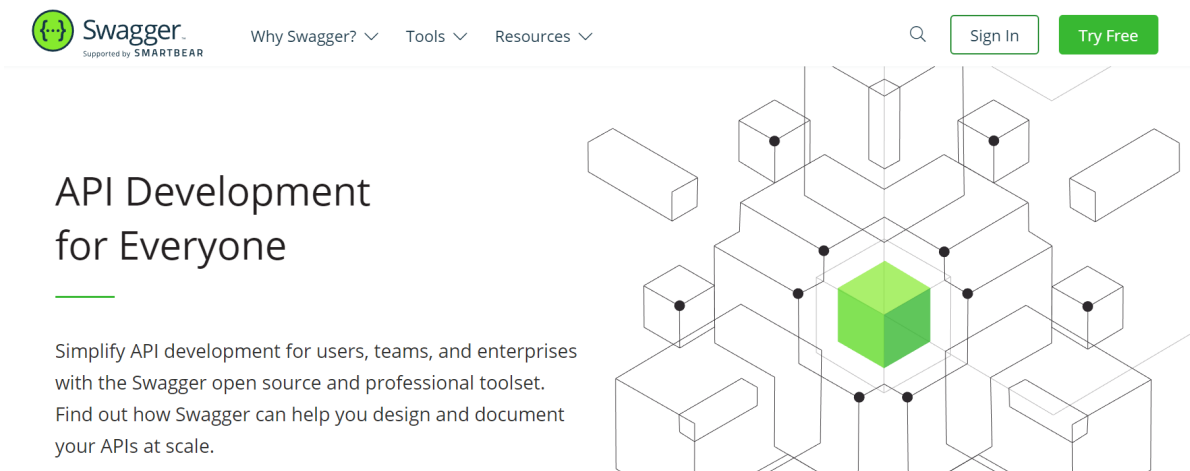
6 对接

6.1 概述

项目采用前后台分离的架构进行开发，后台可以使用Swagger，生成在线API文档，方便前端人员对接使用

Swagger 是一个规范和完整的框架，用于生成、描述、调用和可视化 RESTful 风格的 Web 服务。

Swagger官网



配置生成的在线API文档样例：



springfox，是一个开源的API Doc的框架，它的前身是swagger-springmvc，可以将我们的Controller中的方法以文档的形式展现。

springfox官网



Quick Links

Samples

- [Springfox demo repository](#)

springfox-swagger2，它是整合springmvc和swagger2的一个项目，项目中使用swagger时，要引入它的依赖



Springfox Swagger2

JSON API documentation for spring based applications
























License	Apache 2.0
Tags	io api swagger
Used By	1,039 artifacts

Version		Repository	Usages	Date
3.0.x	3.0.0	Central	19	Jul, 2020
2.10.x	2.10.5	Central	9	Jun, 2020
	2.10.4	Central	1	Jun, 2020
	2.10.3	Central	0	Jun, 2020
	2.10.2	Central	0	Jun, 2020
	2.10.1	Central	0	Jun, 2020
	2.10.0	Central	1	Jun, 2020
	2.9.2	Central	582	Jun, 2018
2.9.x	2.9.1	Central	4	Jun, 2018
2.8.x	2.8.0	Central	166	Jan, 2018
2.7.x	2.7.0	Central	141	May, 2017
2.6.x	2.6.1	Central	99	Nov, 2016
	2.6.0	Central	14	Oct, 2016
2.5.x	2.5.0	Central	34	Jun, 2016
2.4.x	2.4.0	Central	30	Mar, 2016
2.3.x	2.3.1	Central	6	Jan, 2016
	2.3.0	Central	3	Dec, 2015

6.2 使用

在springboot中，使用Swagger非常简单，引入依赖，并做出少量的固定配置即可

例如，新建项目springboot-swagger，如下

- ▼  springboot-swagger [boot]
 - ▼  src/main/java
 - ▼  com.briup.cms
 - ▼  config
 - >  Swagger2Config.java
 - ▼  exception
 - >  GlobalExceptionHandler.java
 - >  ServiceException.java
 - ▼  util
 - >  ExceptionUtil.java
 - >  Result.java
 - >  ResultCode.java
 - ▼  web.controller
 - >  HelloController.java
 - >  Application.java
 - ▼  src/main/resources
 -  application.properties
 -  src/test/java
 - >  JRE System Library [JavaSE-1.8]
 - >  Maven Dependencies
 - >  src
 -  target
 -  pom.xml

1、pom文件，引入依赖swagger2的依赖

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5     http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7     <parent>
8         <groupId>org.springframework.boot</groupId>
9         <artifactId>spring-boot-starter-parent</artifactId>
10        <version>2.4.0</version>
11        <relativePath/> <!-- lookup parent from repository -->
12    </parent>
13    <groupId>com.briup.demo</groupId>
14    <artifactId>springboot-swagger</artifactId>
15    <version>0.0.1-SNAPSHOT</version>
16    <name>springboot-swagger</name>
17    <description>Demo project for Spring Boot</description>
18    <properties>

```

```

18     <java.version>1.8</java.version>
19 </properties>
20
21 <dependencies>
22     <dependency>
23         <groupId>org.springframework.boot</groupId>
24         <artifactId>spring-boot-starter-web</artifactId>
25     </dependency>
26
27     <!-- 引入swagger相关依赖 -->
28     <dependency>
29         <groupId>io.springfox</groupId>
30         <artifactId>springfox-swagger2</artifactId>
31         <version>2.9.2</version>
32     </dependency>
33
34     <dependency>
35         <groupId>io.springfox</groupId>
36         <artifactId>springfox-swagger-ui</artifactId>
37         <version>2.9.2</version>
38     </dependency>
39
40 </dependencies>
41
42 <build>
43     <plugins>
44         <plugin>
45             <groupId>org.springframework.boot</groupId>
46             <artifactId>spring-boot-maven-plugin</artifactId>
47         </plugin>
48     </plugins>
49 </build>
50
51 </project>
52

```

2、使用配置类，对swagger进行配置

```

1  package com.briup.cms.config;
2
3  import org.springframework.context.annotation.Bean;
4  import org.springframework.context.annotation.Configuration;
5
6  import springfox.documentation.builders.ApiInfoBuilder;
7  import springfox.documentation.builders.PathSelectors;
8  import springfox.documentation.builders.RequestHandlerSelectors;
9  import springfox.documentation.service.ApiInfo;
10 import springfox.documentation.spi.DocumentationType;
11 import springfox.documentation.spring.web.plugins.Docket;
12 import springfox.documentation.swagger2.annotations.EnableSwagger2;
13
14 //开始springboot中对swagger2的支持
15 @EnableSwagger2
16 @Configuration

```

```

17 public class Swagger2Config {
18
19     //配置需要扫描的Controller的包路径
20     @Bean
21     public Docket createRestApi() {
22         return new Docket(DocumentationType.SWAGGER_2)
23             .apiInfo(apiInfo())
24             .select()
25
26     .apis(RequestHandlerSelectors.basePackage("com.briup.cms.web.controller"))
27         .paths(PathSelectors.any())
28         .build();
29     }
30
31     //swagger界面中显示的基本信息
32     private ApiInfo apiInfo() {
33         return new ApiInfoBuilder()
34             .title("swagger在线API")
35             .description("欢迎访问briup官网, http://www.briup.com")
36             .termsOfServiceUrl("http://www.briup.com")
37             .version("1.0")
38             .build();
39     }
40 }

```

注意，swagger中基本都是固定的配置，按照自己的项目情况，进行修改字符串变量即可

3、Controller中使用swagger相关注解，进行辅助说明

```

1 package com.briup.cms.web.controller;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 import com.briup.cms.util.Result;
7
8 import io.swagger.annotations.Api;
9 import io.swagger.annotations.ApiImplicitParam;
10 import io.swagger.annotations.ApiImplicitParams;
11 import io.swagger.annotations.ApiOperation;
12
13 @Api(tags = "测试模块")
14 @RestController
15 public class HelloController {
16
17     @ApiOperation(value = "hello测试", notes = "第一个swagger测试程序")
18     @ApiImplicitParams({
19         @ApiImplicitParam(name = "name", value = "用户名", dataType =
20 "String", required = true, defaultValue = "tom", paramType = "query"),
21     })
22     @GetMapping("/hello")
23     public Result hello(String name) {
24         if("tom".equals(name)) {

```



```

24         throw new RuntimeException("异常测试...");
25     }
26     return Result.success("hello world!" + name);
27 }
28
29 }
30

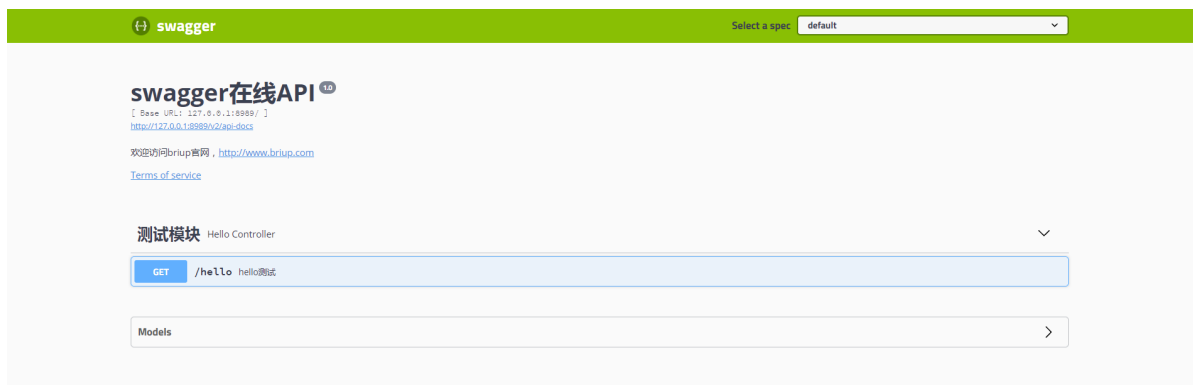
```

其中：

- `@Api`，用来指定当前API模块的名称
- `@ApiOperation`，用来设置API方法的简介说明
- `@ApiImplicitParams`，用来设置API方法的参数，可以有多个参数
- `@ApiImplicitParam`，用来设置一个参数的详细信息
 - name，参数的名称
 - value，参数的介绍
 - dataType，参数的数据类型，例如String
 - required，是否为必须参数
 - defaultValue，默认填入输入框的值
 - paramType，参数的类型：path、query、body、header、form

注意，这些注解配置，都是可以不写的，swagger也能自动扫描这个Controller及其处理方法，并生成文档

4、启动项目，访问固定地址：<http://127.0.0.1:8989/swagger-ui.html>



GET

/hello hello测试

第一个swagger测试程序

Parameters

Try it out

Name

Description

name required

string
(query)

用户名

Default value: tom

Responses

Response content type */*

Code

Description

200

OK

Example Value

Model

{

"code": 0,

"data": {},

"msg": "string"

}

name required

string
(query)

用户名

tom

Execute

Clear

Responses

Response content type */*

Curl

curl -X GET "http://127.0.0.1:8989/hello?name=tom" -H "accept: */*"

Request URL

http://127.0.0.1:8989/hello?name=tom

Server response

Code

Details

200

Response body

{

"code": 500,

"msg": "服务器意外情况: 异常测试...",

"data": null

}

Download

Response headers

connection: keep-alive
content-type: application/json
date: Fri, 20 Nov 2020 02:52:34 GMT
keep-alive: timeout=60
transfer-encoding: chunked

name
string
(query)

用户名

mary

Execute

Clear

Responses

Response content type */*

Curl

curl -X GET "http://127.0.0.1:8989/hello?name=mary" -H "accept: */*"

Request URL

http://127.0.0.1:8989/hello?name=mary

Server response

Code

Details

200

Response body

```
{
  "code": 1,
  "msg": "操作成功",
  "data": "hello world!mary"
}
```

Download

Response headers

connection: keep-alive
content-type: application/json
date: Fri, 20 Nov 2020 02:55:00 GMT
keep-alive: timeout=60
transfer-encoding: chunked

此时，就可以使用swagger的在线文档，对Controller中暴露出来的API进行测试了。该项目中其他类，都是起到辅助作用，对本例中swagger的使用，没有任何影响。具体代码可查看项目实例。