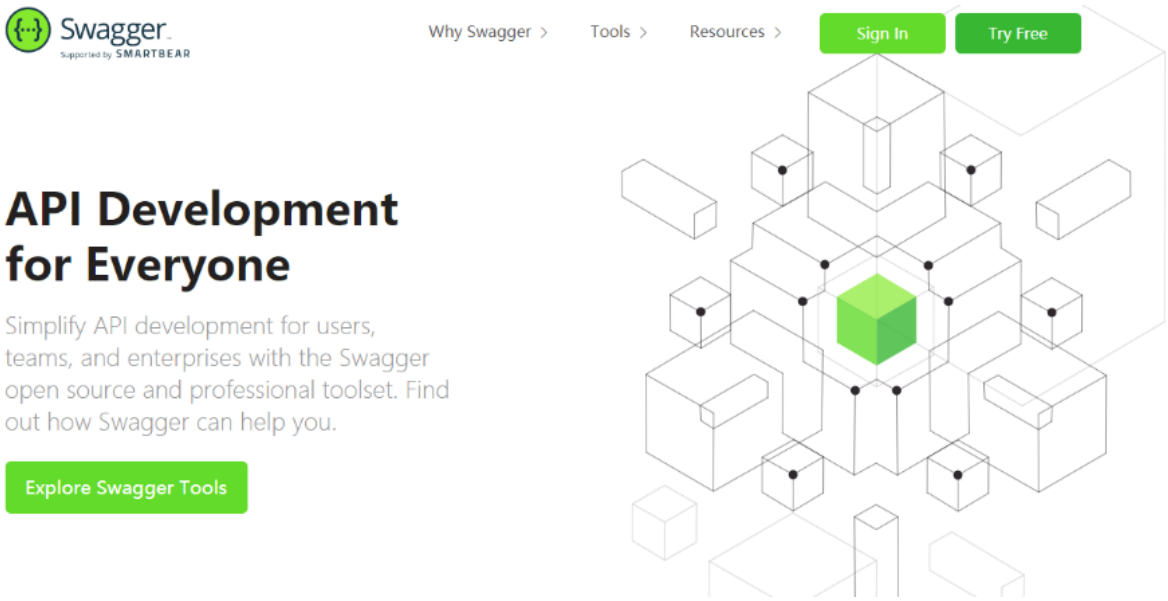
集成Swagger



学习目标：

- 了解Swagger的概念及作用
- 掌握在项目中集成Swagger自动生成API文档

1、Swagger简介

前后端分离

- 前端 -> 前端控制层、视图层
- 后端 -> 后端控制层、服务层、数据访问层
- 前后端通过API进行交互
- 前后端相对独立且松耦合

产生的问题

- 前后端集成，前端或者后端无法做到“及时协商，尽早解决”，最终导致问题集中爆发

解决方案

- 首先定义schema [计划的提纲]，并实时跟踪最新的API，降低集成风险

Swagger

- 号称世界上最流行的API框架
- Restful Api 文档在线自动生成器 => **API 文档 与API 定义同步更新**
- 直接运行，在线测试API
- 支持多种语言 （如：Java，PHP等）
- 官网：https://swagger.io/

2、SpringBoot集成

SpringBoot集成Swagger => **springfox**，两个jar包

- **Springfox-swagger2**
- swagger-springmvc

使用Swagger

要求：jdk 1.8 + 否则swagger2无法运行

步骤：

1. 新建一个SpringBoot-web项目
2. 添加Maven依赖

```
<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger2 -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger-ui -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

3. 编写HelloController，测试确保运行成功！
4. 要使用Swagger，我们需要编写一个配置类-SwaggerConfig来配置 Swagger

```
@Configuration //配置类
@EnableSwagger2// 开启Swagger2的自动配置
public class SwaggerConfig {
}
```

5. 访问测试：http://localhost:8080/swagger-ui.html ，可以看到swagger的界面；

3、配置Swagger

1. Swagger实例Bean是Docket，所以通过配置Docket实例来配置Swaggger。

```
@Bean //配置docket以配置Swagger具体参数
public Docket docket() {
    return new Docket(DocumentationType.SWAGGER_2);
}
```

2. 可以通过apiInfo()属性配置文档信息

```
//配置文档信息
private ApiInfo apiInfo() {
    Contact contact = new Contact("联系人名字", "http://xxx.xxx.com/联系人访问链接", "联系人邮箱");
    return new ApiInfo(
            "Swagger学习", // 标题
            "学习演示如何配置Swagger", // 描述
            "v1.0", // 版本
```

```
            "http://terms.service.url/组织链接", // 组织链接
            contact, // 联系人信息
            "Apach 2.0 许可", // 许可
            "许可链接", // 许可连接
            new ArrayList<>()// 扩展
    );
}
```

3. Docket 实例关联上 apiInfo()

```
@Bean
public Docket docket() {
    return new Docket(DocumentationType.SWAGGER_2).apiInfo(apiInfo());
}
```

4. 重启项目，访问测试 http://localhost:8080/swagger-ui.html 看下效果；

4、配置扫描接口

1. 构建Docket时通过select()方法配置怎么扫描接口。

```
@Bean
public Docket docket() {
    return new Docket(DocumentationType.SWAGGER_2)
        .apiInfo(apiInfo())
        .select()// 通过.select()方法，去配置扫描接口,RequestHandlerSelectors配置如何扫描接口

.apis(RequestHandlerSelectors.basePackage("com.kuang.swagger.controller"))
        .build();
}
```

2. 重启项目测试，由于我们配置根据包的路径扫描接口，所以我们只能看到一个类
3. 除了通过包路径配置扫描接口外，还可以通过配置其他方式扫描接口，这里注释一下所有的配置方式：

```
any() // 扫描所有，项目中的所有接口都会被扫描到
none() // 不扫描接口
// 通过方法上的注解扫描，如withMethodAnnotation(GetMapping.class)只扫描get请求
withMethodAnnotation(final Class<? extends Annotation> annotation)
// 通过类上的注解扫描，如.withClassAnnotation(Controller.class)只扫描有controller注解的类中的接口
withClassAnnotation(final Class<? extends Annotation> annotation)
basePackage(final String basePackage) // 根据包路径扫描接口
```

4. 除此之外，我们还可以配置接口扫描过滤：

```
@Bean
public Docket docket() {
    return new Docket(DocumentationType.SWAGGER_2)
        .apiInfo(apiInfo())
        .select()// 通过.select()方法，去配置扫描接口,RequestHandlerSelectors配置如何扫描接口

.apis(RequestHandlerSelectors.basePackage("com.kuang.swagger.controller"))
        // 配置如何通过path过滤,即这里只扫描请求以/kuang开头的接口
        .paths(PathSelectors.ant("/kuang/**"))
        .build();
}
```

5. 这里的可选值还有

```
any() // 任何请求都扫描
none() // 任何请求都不扫描
regex(final String pathRegex) // 通过正则表达式控制
ant(final String antPattern) // 通过ant()控制
```

5、配置开关Swagger

1. 通过enable()方法配置是否启用swagger，如果是false，swagger将不能在浏览器中访问了

```
@Bean
public Docket docket() {
    return new Docket(DocumentationType.SWAGGER_2)
        .apiInfo(apiInfo())
        .enable(false) //配置是否启用Swagger，如果是false，在浏览器将无法访问
        .select()// 通过.select()方法，去配置扫描接口,RequestHandlerSelectors配置如何扫描接口

.apis(RequestHandlerSelectors.basePackage("com.kuang.swagger.controller"))
        // 配置如何通过path过滤,即这里只扫描请求以/kuang开头的接口
        .paths(PathSelectors.ant("/kuang/**"))
        .build();
}
```

2. 如何动态配置当项目处于test、dev环境时显示swagger，处于prod时不显示?

```
@Bean
public Docket docket(Environment environment) {
    // 设置要显示swagger的环境
    Profiles of = Profiles.of("dev", "test");
    // 判断当前是否处于该环境
    // 通过 enable() 接收此参数判断是否要显示
    boolean b = environment.acceptsProfiles(of);

    return new Docket(DocumentationType.SWAGGER_2)
        .apiInfo(apiInfo())
        .enable(b) //配置是否启用Swagger，如果是false，在浏览器将无法访问
        .select()// 通过.select()方法，去配置扫描接口,RequestHandlerSelectors配置如何扫描接口
```

```
13
.apis(RequestHandlerSelectors.basePackage("com.kuang.swagger.controller"))
14          // 配置如何通过path过滤,即这里只扫描请求以/kuang开头的接口
15          .paths(PathSelectors.ant("/kuang/**"))
16          .build();
17  }
```

3. 可以在项目中增加一个dev的配置文件查看效果！

6、配置API分组

1. 如果没有配置分组，默认是default。通过groupName()方法即可配置分组：

```
1  @Bean
2  public Docket docket(Environment environment) {
3      return new Docket(DocumentationType.SWAGGER_2).apiInfo(apiInfo())
4          .groupName("hello") // 配置分组
5          // 省略配置....
6  }
```

2. 重启项目查看分组

3. 如何配置多个分组？配置多个分组只需要配置多个docket即可：

```
1  @Bean
2  public Docket docket1() {
3      return new Docket(DocumentationType.SWAGGER_2).groupName("group1");
4  }
5  @Bean
6  public Docket docket2() {
7      return new Docket(DocumentationType.SWAGGER_2).groupName("group2");
8  }
9  @Bean
10 public Docket docket3() {
11     return new Docket(DocumentationType.SWAGGER_2).groupName("group3");
12 }
```

4. 重启项目查看

7、实体配置

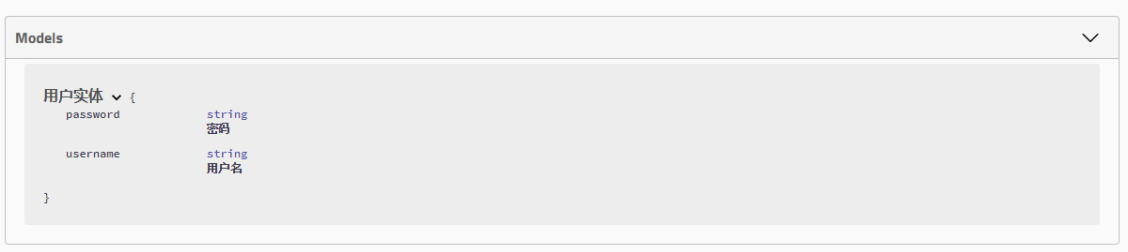
1. 新建一个实体类

```
1  @ApiModel("用户实体")
2  public class User {
3      @ApiModelProperty("用户名")
4      public String username;
5      @ApiModelProperty("密码")
6      public String password;
7  }
```

2. 只要这个实体在**请求接口**的返回值上（即使是泛型），都能映射到实体项中：

```java
@RequestMapping("/getUser")
public User getUser(){
    return new User();
}
```

3. 重启查看测试



注：并不是因为@ApiModel这个注解让实体显示在这里了，而是只要出现在接口方法的返回值上的实体都会显示在这里，而@ApiModel和@ApiModelProperty这两个注解只是为实体添加注释的。

@ApiModel为类添加注释

@ApiModelProperty为类属性添加注释

8、常用注解

Swagger的所有注解定义在io.swagger.annotations包下

下面列一些经常用到的，未列举出来的可以另行查阅说明：

Swagger注解	简单说明
@Api(tags = "xxx模块说明")	作用在模块类上
@ApiOperation("xxx接口说明")	作用在接口方法上
@ApiModel("xxxPOJO说明")	作用在模型类上：如VO、BO
@ApiModelProperty(value = "xxx属性说明",hidden = true)	作用在类方法和属性上，hidden设置为true可以隐藏该属性
@ApiParam("xxx参数说明")	作用在参数、方法和字段上，类似@ApiModelProperty

我们也可以给请求的接口配置一些注释

```java
@ApiOperation("狂神的接口")
@PostMapping("/kuang")
@ResponseBody
public String kuang(@ApiParam("这个名字会被返回")String username){
    return username;
}
```

这样的话，可以给一些比较难理解的属性或者接口，增加一些配置信息，让人更容易阅读！

相较于传统的Postman或Curl方式测试接口，使用swagger简直就是傻瓜式操作，不需要额外说明文档(写得好本身就是文档)而且更不容易出错，只需要录入数据然后点击Execute，如果再配合自动化框架，可以说基本就不需要人为操作了。

Swagger是个优秀的工具，现在国内已经有很多的中小型互联网公司都在使用它，相较于传统的要先出Word接口文档再测试的方式，显然这样也更符合现在的快速迭代开发行情。当然了，提醒下大家在正式环境要记得关闭Swagger，一来出于安全考虑二来也可以节省运行时内存。

9、其他皮肤

我们可以导入不同的包实现不同的皮肤定义：

1、默认的 **访问** http://localhost:8080/swagger-ui.html

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

2、bootstrap-ui **访问** http://localhost:8080/doc.html

```
<!-- 引入swagger-bootstrap-ui包 /doc.html-->
<dependency>
    <groupId>com.github.xiaoymin</groupId>
    <artifactId>swagger-bootstrap-ui</artifactId>
    <version>1.9.1</version>
</dependency>
```

3、Layui-ui **访问** http://localhost:8080/docs.html

```
<!-- 引入swagger-ui-layer包 /docs.html-->
<dependency>
    <groupId>com.github.caspar-chen</groupId>
    <artifactId>swagger-ui-layer</artifactId>
    <version>1.1.3</version>
</dependency>
```

4、mg-ui **访问** http://localhost:8080/document.html

```
<!-- 引入swagger-ui-layer包 /document.html-->
<dependency>
    <groupId>com.zyplayer</groupId>
    <artifactId>swagger-mg-ui</artifactId>
    <version>1.0.6</version>
</dependency>
```

异步任务

1. 创建一个service包
2. 创建一个类AsyncService

   异步处理还是非常常用的，比如我们在网站上发送邮件，后台会去发送邮件，此时前台会造成响应不动，直到邮件发送完毕，响应才会成功，所以我们一般会采用多线程的方式去处理这些任务。

   编写方法，假装正在处理数据，使用线程设置一些延时，模拟同步等待的情况；

```java
@Service
public class AsyncService {

    public void hello(){
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("数据处理中....");
    }
}
```

3. 编写controller包
4. 编写AsyncController类

   我们去写一个Controller测试一下

```java
@RestController
public class AsyncController {

    @Autowired
    AsyncService asyncService;

    @GetMapping("/hello")
    public String hello(){
        asyncService.hello();
        return "success";
    }

}
```

5. 访问http://localhost:8080/hello进行测试，3秒后出现success，这是同步等待的情况。

   问题：我们如果想让用户直接得到消息，就在后台使用多线程的方式进行处理即可，但是每次都需要自己手动去编写多线程的实现的话，太麻烦了，我们只需要用一个简单的办法，在我们的方法上加一个简单的注解即可，如下：

6. 给hello方法添加@Async注解；

```java
//告诉Spring这是一个异步方法
@Async
public void hello(){
    try {
        Thread.sleep(3000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("数据处理中....");
}
```

   SpringBoot就会自己开一个线程池，进行调用！但是要让这个注解生效，我们还需要在主程序上添加一个注解@EnableAsync ，开启异步注解功能；

```java
@EnableAsync //开启异步注解功能
@SpringBootApplication
public class SpringbootTaskApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringbootTaskApplication.class, args);
    }

}
```

7. 重启测试，网页瞬间响应，后台代码依旧执行！

定时任务

项目开发中经常需要执行一些定时任务，比如需要在每天凌晨的时候，分析一次前一天的日志信息，Spring为我们提供了异步执行任务调度的方式，提供了两个接口。

- TaskExecutor接口
- TaskScheduler接口

两个注解：

- @EnableScheduling
- @Scheduled

cron表达式：

字段	允许值	允许的特殊字符
秒	0-59	, - * /
分	0-59	, - * /
小时	0-23	, - * /
日期	1-31	, - * ? / L W C
月份	1-12	, - * /
星期	0-7或SUN-SAT 0,7是SUN	, - * ? / L C #

特殊字符	代表含义
,	枚举
-	区间
*	任意
/	步长
?	日/星期冲突匹配
L	最后
W	工作日
C	和calendar联系后计算过的值
#	星期，4#2，第2个星期三

1. 创建一个ScheduledService

   我们里面存在一个hello方法，他需要定时执行，怎么处理呢?

```java
@Service
public class ScheduledService {

    //秒    分    时     日    月    周几
    //0 * * * * MON-FRI
    //注意cron表达式的用法;
    @Scheduled(cron = "0 * * * * 0-7")
    public void hello(){
        System.out.println("hello.....");
    }
}
```

2. 这里写完定时任务之后，我们需要在主程序上增加@EnableScheduling 开启定时任务功能

```java
@EnableAsync //开启异步注解功能
@EnableScheduling //开启基于注解的定时任务
@SpringBootApplication
public class SpringbootTaskApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringbootTaskApplication.class, args);
    }

}
```

3. 我们来详细了解下cron表达式；

   http://www.bejson.com/othertools/cron/

4. 课堂练习

```
/*
【0 0/5 14,18 * * ?】每天14点整和18点整，每隔5分钟执行一次
【0 15 10 ? * 1-6】每个月的周一-周六10:15分执行一次
【0 0 2 ? * 6L】每个月的最后一个周六凌晨2点执行一次
【0 0 2 LW * ? 】每个月的最后一个工作日凌晨2点执行一次
【0 0 2-4 ? * 1#1】每个月的第一个周一凌晨2点到4点期间，每个整点都执行一次
*/
```

邮件任务

邮件发送，在我们的日常开发中，也非常的多，Springboot也帮我们做了支持

- 邮件发送需要引入spring-boot-start-mail
- SpringBoot 自动配置MailSenderAutoConfiguration
- 定义MailProperties内容，配置在application.yml中
- 自动装配JavaMailSender
- 测试邮件发送

演示

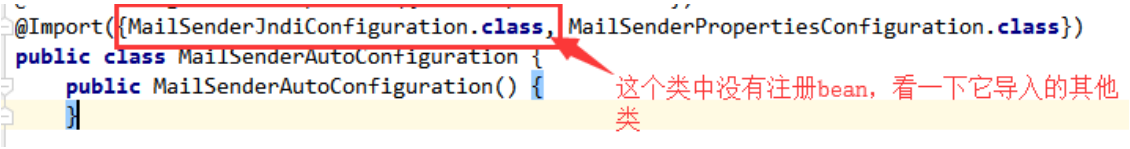
1. 引入pom依赖

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

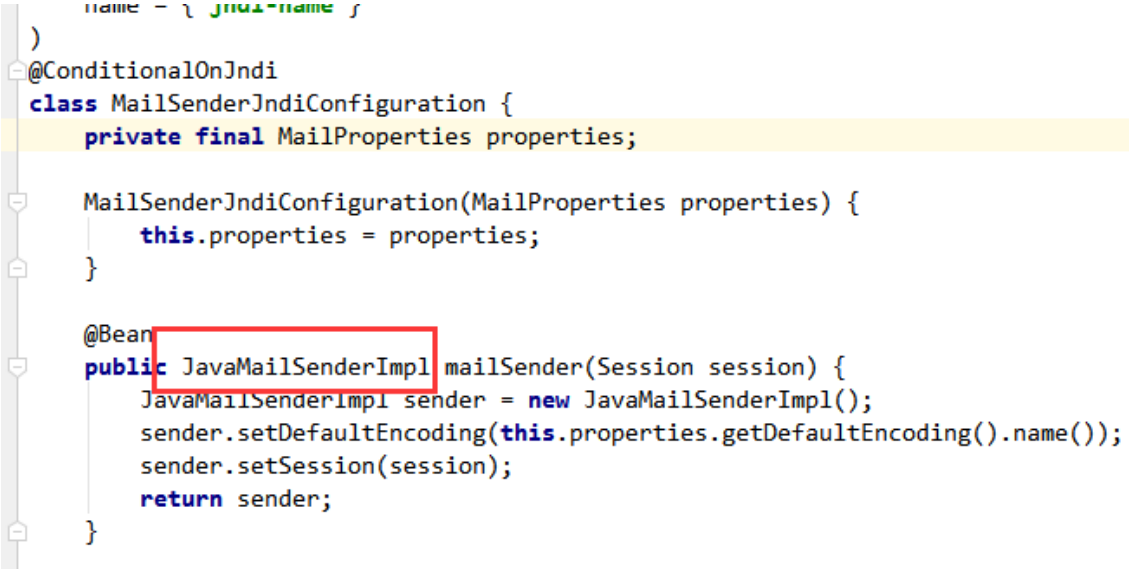
看它引入的依赖，可以看到 jakarta.mail

```xml
<dependency>
    <groupId>com.sun.mail</groupId>
    <artifactId>jakarta.mail</artifactId>
    <version>1.6.4</version>
    <scope>compile</scope>
</dependency>
```

2. 查看自动配置类：MailSenderAutoConfiguration



这个类中存在bean，JavaMailSenderImpl



然后我们去看下配置文件

```java
@ConfigurationProperties(
    prefix = "spring.mail"
)
public class MailProperties {
    private static final Charset DEFAULT_CHARSET;
    private String host;
    private Integer port;
    private String username;
    private String password;
    private String protocol = "smtp";
    private Charset defaultEncoding;
    private Map<String, String> properties;
    private String jndiName;
}
```

3. 配置文件：

```
spring.mail.username=24736743@qq.com
spring.mail.password=yhkrgtqwbnrcbhcj
spring.mail.host=smtp.qq.com
# qq需要配置ssl
spring.mail.properties.mail.smtp.ssl.enable=true
```

4. Spring单元测试

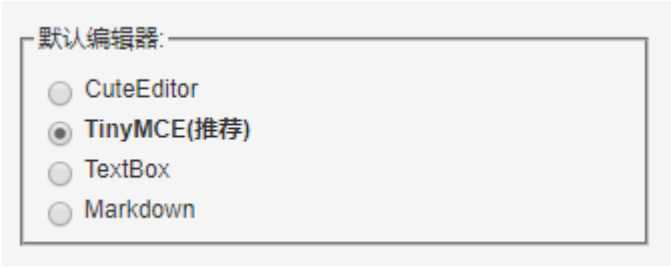
```
@Autowired
JavaMailSenderImpl mailSender;

@Test
public void contextLoads() {
    //邮件设置1：一个简单的邮件
    SimpleMailMessage message = new SimpleMailMessage();
    message.setSubject("通知-明天来狂神这听课");
    message.setText("今晚7:30开会");

    message.setTo("24736743@qq.com");
    message.setFrom("24736743@qq.com");
    mailSender.send(message);
}

@Test
public void contextLoads2() throws MessagingException {
    //邮件设置2：一个复杂的邮件
    MimeMessage mimeMessage = mailSender.createMimeMessage();
    MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, true);

    helper.setSubject("通知-明天来狂神这听课");
    helper.setText("<b style='color:red'>今天 7:30来开会</b>",true);

    //发送附件
    helper.addAttachment("1.jpg",new File(""));
    helper.addAttachment("2.jpg",new File(""));

    helper.setTo("24736743@qq.com");
    helper.setFrom("24736743@qq.com");

    mailSender.send(mimeMessage);
}
```

富文本编辑器

1、简介

思考：我们平时在博客园，或者CSDN等平台进行写作的时候，有同学思考过他们的编辑器是怎么实现的吗?

在博客园后台的选项设置中，可以看到一个文本编辑器的选项：



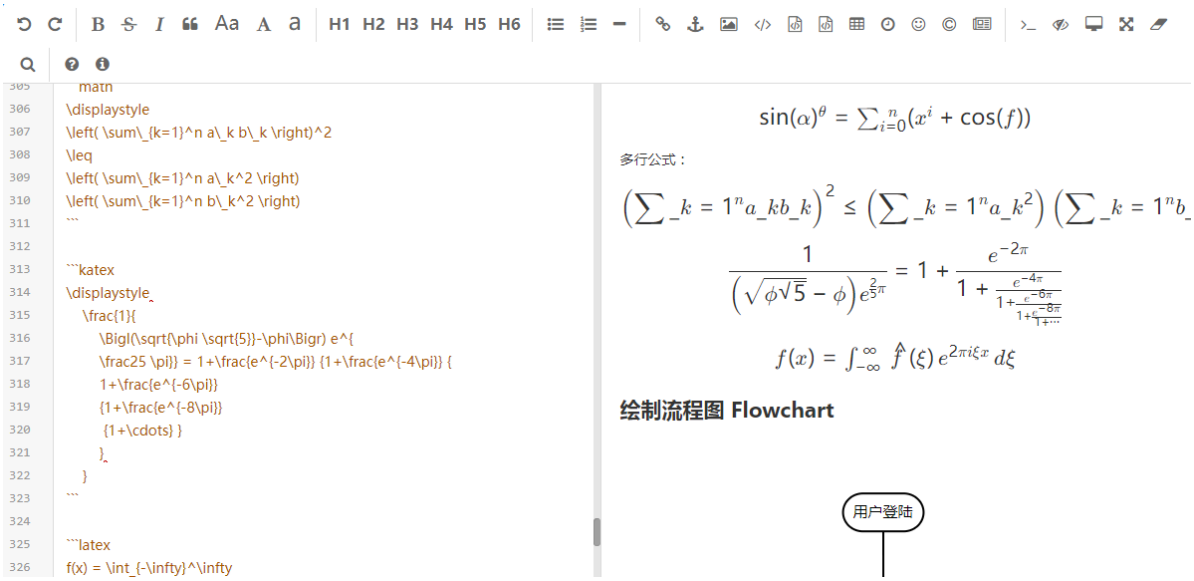
其实这个就是富文本编辑器，市面上有许多非常成熟的富文本编辑器，比如：

- **Editor.md**——功能非常丰富的编辑器，左端编辑，右端预览，非常方便，完全免费
  - 官网：https://pandao.github.io/editor.md/
- **wangEditor**——基于javascript和css开发的 Web富文本编辑器， 轻量、简洁、界面美观、易用、开源免费。
  - 官网：http://www.wangeditor.com/
- **TinyMCE**——TinyMCE是一个轻量级的基于浏览器的所见即所得编辑器，由JavaScript写成。它对IE6+和Firefox1.5+都有着非常良好的支持。功能齐全，界面美观，就是文档是英文的，对开发人员英文水平有一定要求。
  - 官网：https://www.tiny.cloud/docs/demo/full-featured/
  - 博客园
- **百度ueditor**——UEditor是由百度web前端研发部开发所见即所得富文本web编辑器，具有轻量，功能齐全，可定制，注重用户体验等特点，开源基于MIT协议，允许自由使用和修改代码，缺点是已经没有更新了
  - 官网：https://ueditor.baidu.com/website/onlinedemo.html
- **kindeditor**——界面经典。
  - 官网：http://kindeditor.net/demo.php
- **Textbox**——Textbox是一款极简但功能强大的在线文本编辑器，支持桌面设备和移动设备。主要功能包含内置的图像处理和存储、文件拖放、拼写检查和自动更正。此外，该工具还实现了屏幕阅读器等辅助技术，并符合WAI-ARIA可访问性标准。
  - 官网：https://textbox.io/
- **CKEditor**——国外的，界面美观。
  - 官网：https://ckeditor.com/ckeditor-5/demo/
- **quill**——功能强大，还可以编辑公式等
  - 官网：https://quilljs.com/
- **simditor**——界面美观，功能较全。
  - 官网：https://simditor.tower.im/
- **summernote**——UI好看，精美
  - 官网：https://summernote.org/
- **jodit**——功能齐全
  - 官网：https://xdsoft.net/jodit/
- **froala Editor**——界面非常好看，功能非常强大，非常好用（非免费）
  - 官网：https://www.froala.com/wysiwyg-editor

总之，目前可用的富文本编辑器有很多……这只是其中的一部分

2、Editor.md

我这里使用的就是 `Editor.md`，作为一个资深码农，Mardown必然是我们程序猿最喜欢的格式，看下面，就爱上了！



我们可以在官网下载它：https://pandao.github.io/editor.md/ ， 得到它的压缩包！

解压以后，在 `examples` 目录下面，可以看到他的很多案例使用！学习，其实就是看人家怎么写的，然后进行模仿就好了！

我们可以将整个解压的文件倒入我们的项目，将一些无用的测试和案例删掉即可！

3、基础工程搭建

> 数据库设计

`article：文章表`

字段		备注
id	int	文章的唯一ID
author	varchar	作者
title	varchar	标题
content	longtext	文章的内容

建表SQL：

```
CREATE TABLE `article` (
  `id` int(10) NOT NULL AUTO_INCREMENT COMMENT 'int文章的唯一ID',
  `author` varchar(50) NOT NULL COMMENT '作者',
  `title` varchar(100) NOT NULL COMMENT '标题',
  `content` longtext NOT NULL COMMENT '文章的内容',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

> 基础项目搭建

1、新建一个SpringBoot项目（模块：web，mysql驱动，mybatis，thymeleaf、lombok.....）

```yaml
spring:
  datasource:
    username: root
    password: 123456
    #?serverTimezone=UTC解决时区的报错
    url: jdbc:mysql://localhost:3306/springboot?serverTimezone=UTC&useUnicode=true&characterEncoding=utf-8
    driver-class-name: com.mysql.cj.jdbc.Driver
```

```xml
<resources>
    <resource>
        <directory>src/main/java</directory>
        <includes>
            <include>**/*.xml</include>
        </includes>
        <filtering>true</filtering>
    </resource>
</resources>
```

2、实体类：

```java
//文章类
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Article implements Serializable {

    private int id; //文章的唯一ID
    private String author; //作者名
    private String title; //标题
    private String content; //文章的内容

}
```

3、mapper接口：

```java
@Mapper
@Repository
public interface ArticleMapper {
    //查询所有的文章
    List<Article> queryArticles();

    //新增一个文章
    int addArticle(Article article);

    //根据文章id查询文章
    Article getArticleById(int id);

    //根据文章id删除文章
    int deleteArticleById(int id);

}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
```

```
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.kuang.mapper.ArticleMapper">

    <select id="queryArticles" resultType="Article">
       select * from article
    </select>

    <select id="getArticleById" resultType="Article">
       select * from article where id = #{id}
    </select>

    <insert id="addArticle" parameterType="Article">
        insert into article (author,title,content) values (#{author},#{title},#{content});
    </insert>

    <delete id="deleteArticleById" parameterType="int">
        delete from article where id = #{id}
    </delete>

</mapper>
```

既然已经提供了 myBatis 的映射配置文件，自然要告诉 spring boot 这些文件的位置

```
mybatis:
  mapper-locations: classpath:com/kuang/mapper/*.xml
  type-aliases-package: com.kuang.pojo
```

编写一个Controller测试下，是否ok；

4、文章编辑整合

1、导入 editor.md 资源， 删除多余文件

2、编辑文章页面 editor.html、需要引入 jQuery；

```
<!DOCTYPE html>
<html class="x-admin-sm" lang="zh" xmlns:th="http://www.thymeleaf.org">

<head>
    <meta charset="UTF-8">
    <title>秦疆'Blog</title>
    <meta name="renderer" content="webkit">
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
    <meta name="viewport" content="width=device-width,user-scalable=yes, minimum-scale=0.4, initial-scale=0.8,target-densitydpi=low-dpi" />
    <!--Editor.md-->
    <link rel="stylesheet" th:href="@{/editormd/css/editormd.css}"/>
    <link rel="shortcut icon" href="https://pandao.github.io/editor.md/favicon.ico" type="image/x-icon" />
</head>

```

```
<body>

<div class="layui-fluid">
    <div class="layui-row layui-col-space15">
        <div class="layui-col-md12">
            <!--博客表单-->
            <form name="mdEditorForm">
                <div>
                    标题：  <input type="text" name="title">
                </div>
                <div>
                   作者：  <input type="text" name="author">
                </div>
                <div id="article-content">
                    <textarea name="content" id="content"
style="display:none;"> </textarea>
                </div>
            </form>

        </div>
    </div>
</div>
</body>

<!--editormd-->
<script th:src="@{/editormd/lib/jquery.min.js}"></script>
<script th:src="@{/editormd/editormd.js}"></script>

<script type="text/javascript">
    var testEditor;

    //window.onload = function(){ }
    $(function() {
        testEditor = editormd("article-content", {
            width : "95%",
            height : 400,
            syncScrolling : "single",
            path : "../editormd/lib/",
            saveHTMLToTextarea : true,     // 保存 HTML 到 Textarea
            emoji: true,
            theme: "dark",//工具栏主题
            previewTheme: "dark",//预览主题
            editorTheme: "pastel-on-dark",//编辑主题
            tex : true,                    // 开启科学公式TeX语言支持，默认关闭
            flowChart : true,              // 开启流程图支持，默认关闭
            sequenceDiagram : true,        // 开启时序/序列图支持，默认关闭,
            //图片上传
            imageUpload : true,
            imageFormats : ["jpg", "jpeg", "gif", "png", "bmp", "webp"],
            imageUploadURL : "/article/file/upload",
            onload : function() {
                console.log('onload', this);
            },
            /*指定需要显示的功能按钮*/
            toolbarIcons : function() {
                return ["undo","redo","|",

 "bold","del","italic","quote","ucwords","uppercase","lowercase","|",
```

```
                    "h1","h2","h3","h4","h5","h6","|",
                    "list-ul","list-ol","hr","|",
                    "link","reference-link","image","code","preformatted-text",
                    "code-block","table","datetime","emoji","html-entities","pagebreak","|",
                    "goto-line","watch","preview","fullscreen","clear","search","|",
                    "help","info","releaseIcon", "index"]
            },

            /*自定义功能按钮，下面我自定义了2个，一个是发布，一个是返回首页*/
            toolbarIconTexts : {
                releaseIcon : "<span bgcolor=\"gray\">发布</span>",
                index : "<span bgcolor=\"red\">返回首页</span>",
            },

            /*给自定义按钮指定回调函数*/
            toolbarHandlers:{
                releaseIcon : function(cm, icon, cursor, selection) {
                    //表单提交
                    mdEditorForm.method = "post";
                    mdEditorForm.action = "/article/addArticle";//提交至服务器的路径
                    mdEditorForm.submit();
                },
                index : function(){
                    window.location.href = '/';
                },
            }
        });
    });
</script>

</html>
```

3、编写Controller，进行跳转，以及保存文章

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

    @GetMapping("/toEditor")
    public String toEditor(){
        return "editor";
    }

    @PostMapping("/addArticle")
    public String addArticle(Article article){
        articleMapper.addArticle(article);
        return "editor";
    }

}
```

> 图片上传问题

1、前端js中添加配置

```
//图片上传
imageUpload : true,
imageFormats : ["jpg", "jpeg", "gif", "png", "bmp", "webp"],
imageUploadURL : "/article/file/upload", // //这个是上传图片时的访问地址
```

2、后端请求，接收保存这个图片，需要导入 FastJson 的依赖！

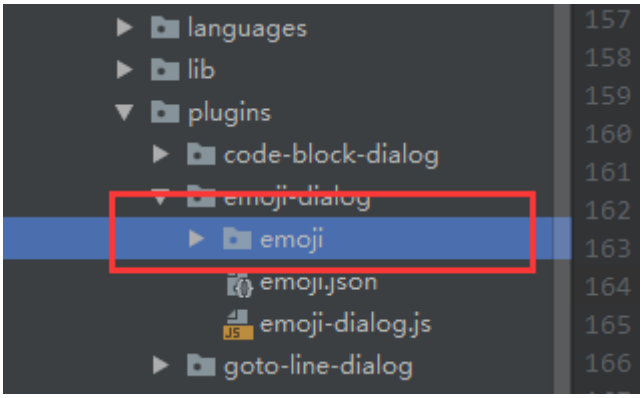
```
//博客图片上传问题
@RequestMapping("/file/upload")
@ResponseBody
public JSONObject fileUpload(@RequestParam(value = "editormd-image-file", required = true) MultipartFile file, HttpServletRequest request) throws IOException {
    //上传路径保存设置

    //获得SpringBoot当前项目的路径：System.getProperty("user.dir")
    String path = System.getProperty("user.dir")+"/upload/";

    //按照月份进行分类：
    Calendar instance = Calendar.getInstance();
    String month = (instance.get(Calendar.MONTH) + 1)+"月";
    path = path+month;

    File realPath = new File(path);
    if (!realPath.exists()){
        realPath.mkdir();
    }

    //上传文件地址
    System.out.println("上传文件保存地址："+realPath);

    //解决文件名字问题：我们使用uuid;
    String filename = "ks-"+UUID.randomUUID().toString().replaceAll("-", "");
    //通过CommonsMultipartFile的方法直接写文件（注意这个时候）
    file.transferTo(new File(realPath +"/"+ filename));

    //给editormd进行回调
    JSONObject res = new JSONObject();
    res.put("url","/upload/"+month+"/"+ filename);
    res.put("success", 1);
    res.put("message", "upload success!");

    return res;
}
```

3、解决文件回显显示的问题，设置虚拟目录映射！在我们自己拓展的MvcConfig中进行配置即可！

```
@Configuration
public class MyMvcConfig implements WebMvcConfigurer {

    // 文件保存在真实目录/upload/下，
    // 访问的时候使用虚路径/upload，比如文件名为1.png，就直接/upload/1.png就ok了。
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/upload/**")

.addResourceLocations("file:"+System.getProperty("user.dir")+"/upload/");
    }

}
```

> 表情包问题

自己手动下载，emoji 表情包，放到图片路径下：



修改editormd.js文件

```
// Emoji graphics files url path
editormd.emoji      = {
    path   : "../editormd/plugins/emoji-dialog/emoji/",
    ext    : ".png"
};
```

5、文章展示

1、Controller 中增加方法

```
@GetMapping("/{id}")
public String show(@PathVariable("id") int id,Model model){
    Article article = articleMapper.getArticleById(id);
    model.addAttribute("article",article);
    return "article";
}
```

2、编写页面 article.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
    <title th:text="${article.title}"></title>
</head>
<body>

<div>
    <!--文章头部信息：标题，作者，最后更新日期，导航-->
    <h2 style="margin: auto 0" th:text="${article.title}"></h2>
    作者：<span style="float: left" th:text="${article.author}"></span>
    <!--文章主体内容-->
    <div id="doc-content">
        <textarea style="display:none;" placeholder="markdown" th:text="${article.content}"></textarea>
    </div>

</div>

<link rel="stylesheet" th:href="@{/editormd/css/editormd.preview.css}" />
<script th:src="@{/editormd/lib/jquery.min.js}"></script>
<script th:src="@{/editormd/lib/marked.min.js}"></script>
<script th:src="@{/editormd/lib/prettify.min.js}"></script>
<script th:src="@{/editormd/lib/raphael.min.js}"></script>
<script th:src="@{/editormd/lib/underscore.min.js}"></script>
<script th:src="@{/editormd/lib/sequence-diagram.min.js}"></script>
<script th:src="@{/editormd/lib/flowchart.min.js}"></script>
<script th:src="@{/editormd/lib/jquery.flowchart.min.js}"></script>
<script th:src="@{/editormd/editormd.js}"></script>

<script type="text/javascript">
    var testEditor;
    $(function () {
        testEditor = editormd.markdownToHTML("doc-content", {//注意：这里是上面DIV的id
            htmlDecode: "style,script,iframe",
            emoji: true,
            taskList: true,
            tocm: true,
            tex: true, // 默认不解析
            flowChart: true, // 默认不解析
            sequenceDiagram: true, // 默认不解析
            codeFold: true
        });});
</script>
</body>
</html>
```

重启项目，访问进行测试！