Swagger简介

接口文档对于前后端开发人员都十分重要。尤其近几年流行前后端分离后接口文档又变成重中之重。接口文档固然重要，但是由于项目周期等原因后端人员经常出现无法及时更新，导致前端人员抱怨接口文档和实际情况不一致。很多人员会抱怨别人写的接口文档不规范，不及时更新。如果接口文档可以实时动态生成就不会出现上面问题，Swagger可以完美地解决上面的问题。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，可用于接口的文档在线自动生成以及功能测试。

Open API 是什么

Open API 规范(OpenAPI Specification)以前叫做 Swagger 规范，是REST API 的 API 描述格式。Open API 文件允许描述整个 API，包括：

- 每个访问地址的类型。POST 或 GET

- 每个操作的参数。包括输入输出参数
- 认证方法
- 连接信息，声明，使用团队和其他信息。

Open API 规范可以使用YAML或 JSON格式进行编写，这样更利于我们和机器进行阅读。

Swagger和Open API

Swagger 是一套围绕 Open API 规范构建的开源工具，可以帮助设计，构建，记录和使用 REST API。Swagger工具包括的组件：

- Swagger Editor ： 基于浏览器编辑器，可以在里面编写 Open API规范。类似 Markdown具有实时预览描述文件的功能。
- Swagger UI：将 Open API 规范呈现为交互式 API 文档。用可视化UI展示描述文件。
- Swagger Codegen：将 OpenAPI 规范生成为服务器存根和客户端库。通过 Swagger Codegen 可以将描述文件生成 html 格式和 cwiki 形式的接口文档，同时也可以生成多种言语的客户端和服务端代码。
- Swagger Inspector：和 Swagger UI 有点类似，但是可以返回更多信息，也会保存请求的实际参数数据。
- Swagger Hub：集成了上面所有项目的各个功能，你可以以项目和版本为单位，将你的描述文件上传到 Swagger Hub 中。在 SwaggerHub 中可以完成上面项目的所有工作，需要注册账号，分免费版和收费版。使用 Swagger，就是把相关的信息存储在它定义的描述文件里面（yml 或 json 格式），再通过维护这个描述文件可以去更新接口文档，以及生成各端代码

实时效果反馈

1.关于Swagger和open API Specification（OAS）说法正确的是

A Swagger是一组工具，它能帮助开发人员设计、构建RestfulAPI文档

B OAS是一种用于描述Restful API的格式

C OAS以前叫作Swagger Sepcification

D 以上都对

答案

1=>D

通过Springfox使用Swagger

使用 Swagger 时如果碰见版本更新，只需要更改Swagger的描述文件即可。但是在频繁地更新项目版本时很多开发人员认为即使修改描述文件（yml 或 json）也有一定的工作负担，久而久之就直接修改代码，而不去修改描述文件了，这样基于描述文件生成接口文档也就失去了意义。

由于Spring的流行，Marty Pitt 编写了一个基于Spring 的组件 swagger-springmvc。Spring-fox 就是根据这个组件发展而来的全新项目。Spring-fox 是根据代码生成接口文档，所以正常的进行更新项目版本，修改代码即可，而不需要跟随修改描述文件。Spring-fox 利用自身 AOP 特性，把 Swagger 集成进来，底层还是

Swagger。但是使用起来确方便很多。所以在实际开发中，都是直接使用 spring-fox。官网地址：

```
https://github.com/springfox/springfox
```

创建springboot项目

项目中 controller 中包含一个 Handler，用于测试项目，保证程序可以正确运行。

```
@RestController
@RequestMapping( "/people")
public class DemoController {

    @RequestMapping( "/getPeople")
    public People getPeople(Long id, String
name){
        People peo = new People();
        peo.setId(id);
        peo.setName(name);
        peo.setAddress("海淀");
        return peo;
    }
}
```

导入Spring-fox 依赖

在项目的pom.xml中导入Spring-fox依赖。选择版本2.9.2，所以导入的依赖也是这个版本。其中 springfox-swagger2 是核心内容的封装。springfox-swagger-ui 是对 swagger-ui 的封装。

```
<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>
```

添加注解

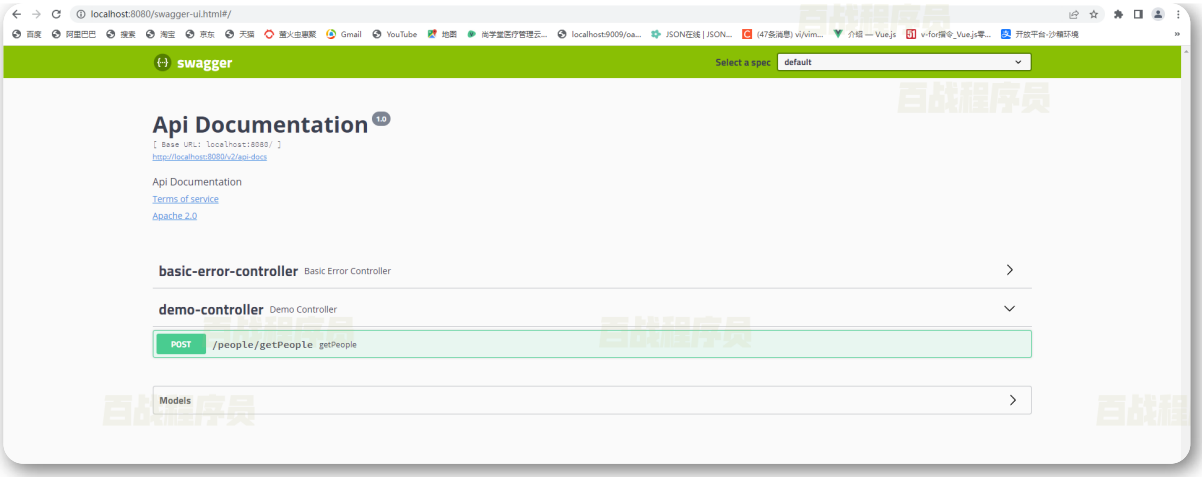
在 SpringBoot 的启动类中添加@EnableSwagger2 注解。添加此注解后表示对当前项目中全部控制器进行扫描，应用Swagger2。

```
@SpringBootApplication
@EnableSwagger2
public class MyApp {
    public static void main(String [] args){

SpringApplication.run(MyApp.class,args);
    }
}
```

访问 swagger-ui

启动项目后在浏览器中输入http://ip:port/swagger-ui.html即可以访问到 swagger-ui 页面，在页面中可以可视化的进行操作项目中所有接口。



实时效果反馈

1.在springboot项目中使用Springfox需要引入的依赖有

A springfox-swagger2

B springfox-swagger-ui

C springfox-boot-starter

D 以上都对

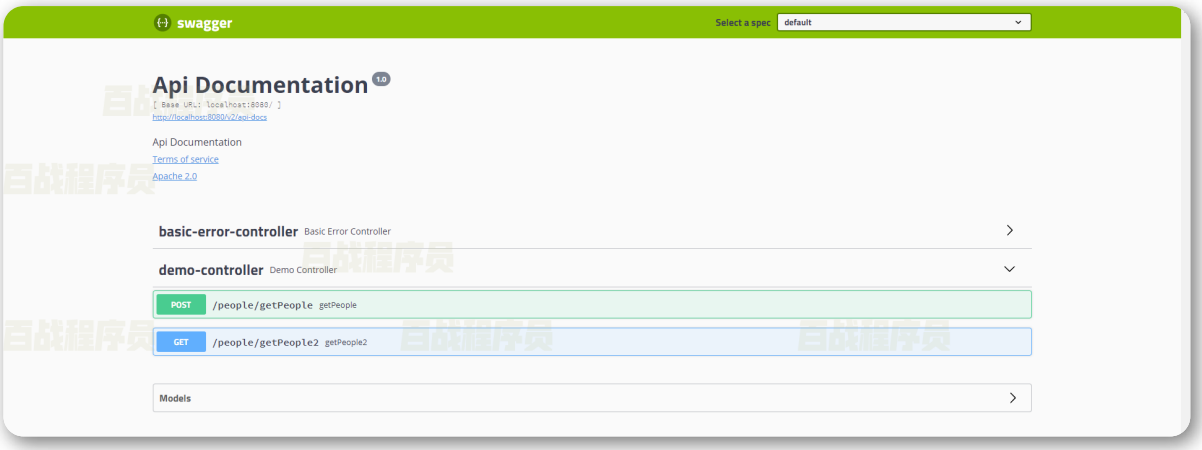
答案

1=>C

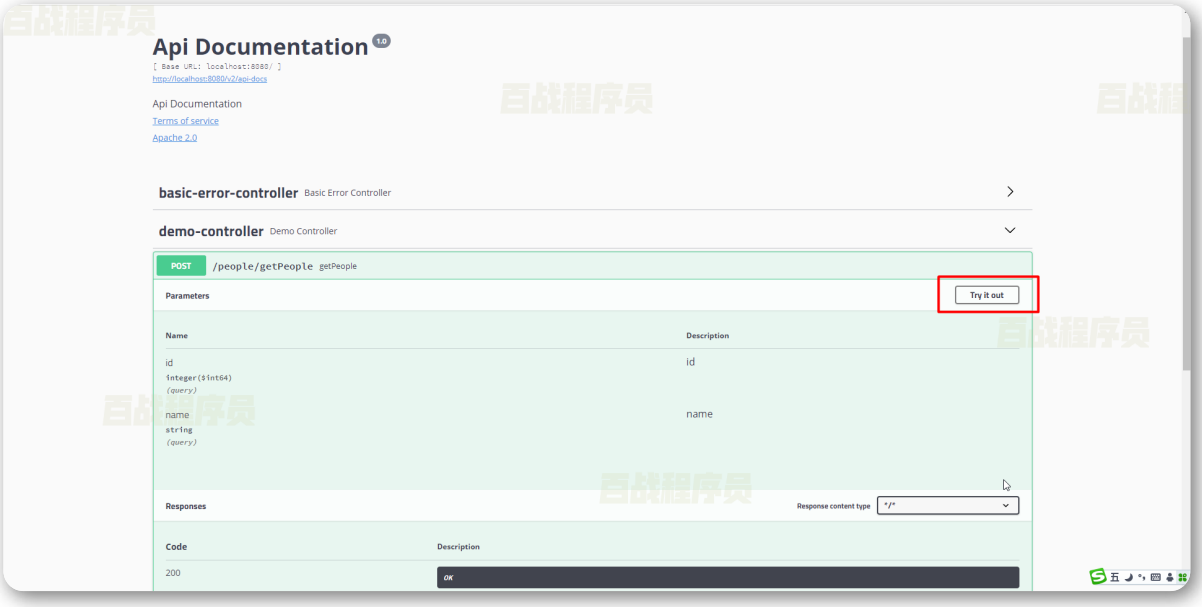
SwaggerUI的使用和配置

访问 swagger-ui.html 可以在页面中看到所有需要生成接口文档的控制器名称。

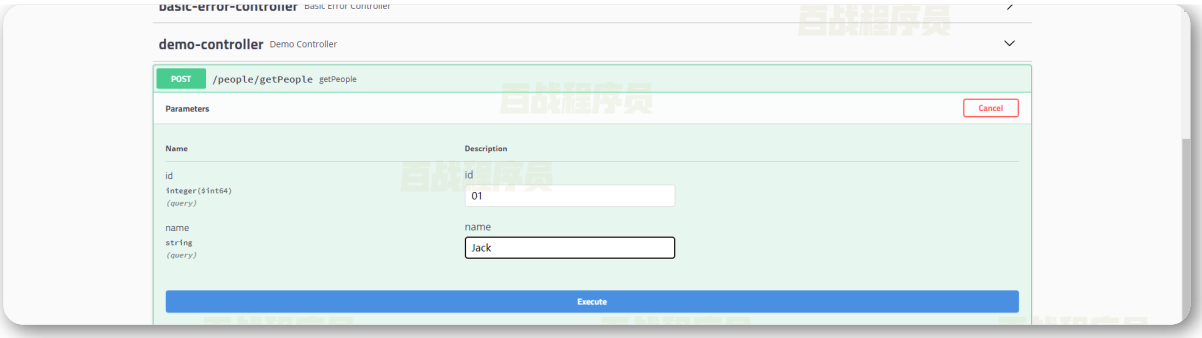
每个控制器包含该控制器下所有的方法及访问方式。如果使用的是@RequestMapping 进行映射，将显示下面的所有请求方式。如果使用@PostMapping 将只有 Post 方式可以能访问，下面也就只显示Post 的一个。



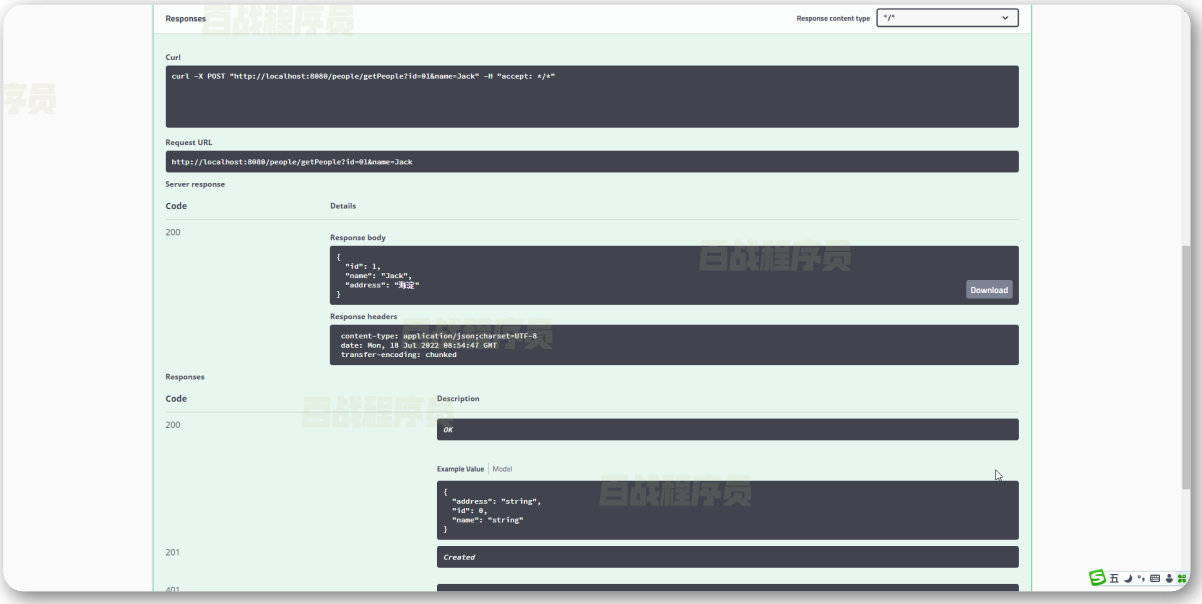
点击某个请求方式中 try it out



输入参数值，完成后点击 Execute 按钮



下面会出现 Request URL 以及响应结果。



Swagger的配置

可以在项目中创建 SwaggerConfig，进行配置文档内容

```
@Configuration
public class SwaggerConfig {

    @Bean
    public Docket getDocket(){
        return new
Docket(DocumentationType.SWAGGER_2)
                .apiInfo(swaggerDemoApiInfo())
                .select()
```

```
                .build();
    }

    private ApiInfo swaggerDemoApiInfo(){
        return new ApiInfoBuilder()
            .contact(new Contact("北京尚学堂",
"http://www.bjsxt.com","xxx@163.com"))
        //文档标题
        .title("这里是Swagger的标题")
        //文档描述
        .description("这里是Swagger的描述")
        //文档版本
        .version("1.0.0")
        .build();
    }
}
```

- 配置基本信息

配置项	含义
Docket	摘要对象，通过对象配置描述文件的信息。
apiInfo	设置描述文件中 info，参数类型 ApiInfo
select()	返回 ApiSelectorBuilder 对象，通过对象调用 build()可以创建 Docket 对象
ApiInfoBuilder	ApiInfo 构建器

显示效果如下：

- 设置扫描的包

可以通过 apis()方法设置哪个包中内容被扫描

```
@Bean
public Docket getDocket() {
 return new Docket(DocumentationType.
SWAGGER_2)
    .apiInfo(swaggerDemoApiInfo())
    .select()
    .apis(RequestHandlerSelectors.basePackage (
"com.bjsxt.controller"))
    .build();
}
```

- 自定义注解设置不需要生成接口文档的方法

注解名称自定义

```
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME )
public @interface NotIncludeSwagger {
}
```

添加规则

通过下面的代码设置生成规则

```
public ApiSelectorBuilder
apis(Predicate<RequestHandler>
selector)
```

下面的方法表示不允许的条件

```
public static <T> Predicate<T> not(Predicate<T>
predicate)
```

withMethodAnnotation：表示此注解是方法级别注解

```
@Bean
public Docket getDocket(){
    return w new
Docket(DocumentationType.SWAGGER_2)
        .apiInfo(swaggerDemoApiInfo())
        .select()
        .paths(allowPaths())

.apis(not((withMethodAnnotation(NotIncludeSwagg
er.class)))
        .build();
}
```

添加NotIncludeSwagger注解

在不需要生成接口文档的方法上面添加@NotIncludeSwagger 注解后，该方法将不会被 Swagger 进行生成在接口文档中。

```
@NotIncludeSwagger
@RequestMapping("/getPeople2")
public People getPeople2(Integer id, String
name, String    address){
    People peo = new People();
    peo.setId(id);
    peo.setName(name);
    peo.setAddress(address);
    return peo;
}
```

设置范围

通过下面的代码

```
public ApiSelectorBuilder
paths(Predicate<String> selector)
```

可以设置满足什么样规则的 url 被生成接口文档。可以使用正则表达式进行匹配。

下面例子中表示只有以/demo/开头的 url 才能被 swagger 生成接口文档。

```
@Bean
public Docket getDocket(){
    return new Docket(DocumentationType.
SWAGGER_2 )
        .apiInfo(swaggerDemoApiInfo())
        .select()
        .paths(allowPaths())
        .build();
}
private Predicate<String> allowPaths(){
    return or(regex("/demo/.*"));
}
```

如何希望全部扫描可以使用

```
paths(PathSelectors.any())
```

实时效果反馈

1.通过哪个方法设置扫描的包

A apiinfo

B select

C apis

D path

答案

1=>C

Swagger2 常用注解

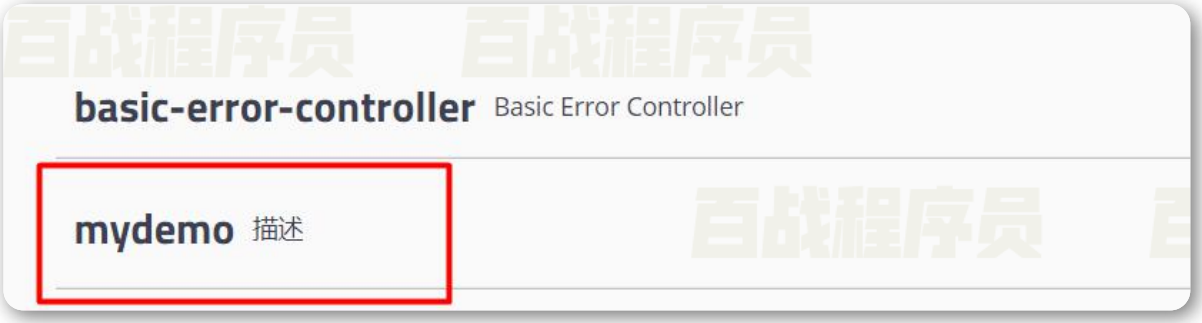
- Api

注解名称	参数	含义
@Api		类上注解。控制整个类生成接口信息的内容
	tags	类的名称，可以有多个值，多个值表示多个副本
	description	描述

代码示例如下：

```
@RestController
@RequestMapping("/people")
@Api(tags = {"mydemo"},description = "描述")
public class DemoController {...}
```

在 swagger-ui.html 中显示效果



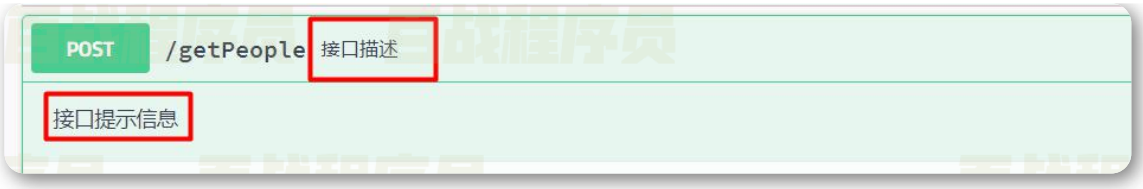
- ApiOperation

注解名称	参数	含义
@ApiOperation		写在方法上，对方法进行总体描述
	value	接口描述
	notes	提示信息

代码示例：

```
@ApiOperation(value="接口描述",notes = "接口提示信息)
```

在 swagger-ui 中显示效果



- ApiParam

注解名称	参数	含义
@ApiParam		写在方法参数前面。用于对参数进行描述或说明是否为必添项等说明
	name	参数名称
	value	参数描述
	required	是否是必须

在代码中示例：

```
public People getPeople(Integer id,
            @ApiParam(value="姓名",required =
 true)
            String name, String address)
```

swagger-ui 显示效果如下：

- ApiModel

注解名称	参数	含义
@ApiModel		是类上注解，主要应用 Model，也就是说这个注解一般都是写在实体类上
	value	名称
	description	描述

代码示例：

```
@ApiModel(value = "人类",description = "描述")
public class People {...}
```

swagger-ui.html 效果展示

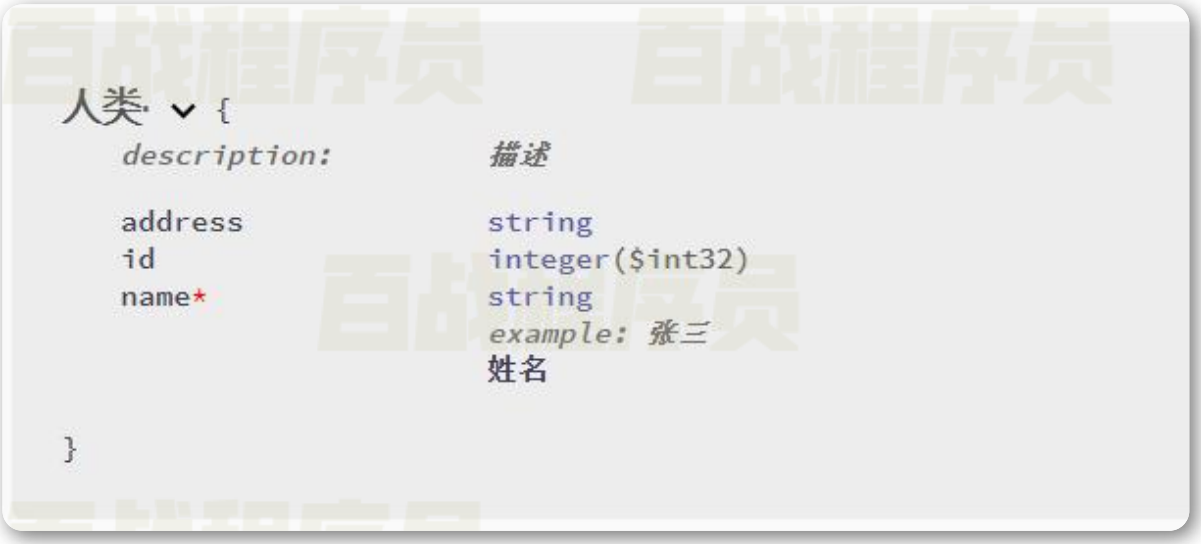
- ApiModelProperty

注解名称	参数	含义
@ApiModelProperty		一般用于方法、属性的说明
	value	字段说明
	name	重写属性名
	required	是否必须
	example	示例内容
	hidden	是否隐藏

代码示例：

```
@ApiModelProperty(value = "姓名", name = "name",
required =  true, example = "张三")
private String  name;
```

swagger-ui.html 效果展示



- ApiIgnore

注解名称	含义
@ApiIgnore	用于方法或类或参数上，表示这个方法或类被忽略，和之前讲解的自定义注解@NotIncludeSwagger 效果类似。只是这个注解是 Swagger 内置的注解，而@NotIncludeSwagger 是我们自定义的注解

- ApiImplicitParam

注解名称	参数	含义
@ApiImplicitParam		用在方法上，表示单独的请求参数，总体功能和@ApiParam 类似。
	value	对接口中的参数进行简单必要的描述
	name	描述接口中参数的名称
	required	是否必须
	paramType	接口中的参数应该用在哪个地方，常用的位置有：header、query 、path
	dataType	接口中参数的类型

代码示例：

```
@PostMapping( "/getPeople")
@ApiImplicitParam(name = "address",value = "地址",required = true,paramType = "query",dataType = "string"
public People getPeople(Integer id,
@ApiParam(value="姓名", required =  true) String name, String address){
    ...
}
```

swagger-ui.html 效果展示

如果希望在方法上配置多个参数时，使用@ApiImplicitParams 进行配置。示例如下：

```
@ApiImplicitParams(value=
{@ApiImplicitParam(name= "id",value = "编
号",required = true),@ApiImplicitParam(name=
"name",value = "姓名",required =  true)})
```

实时效果反馈

1.swagger常用注解中写在方法上对方法进行描述的注解是

A ApiModelProperty

B ApiOperation

C ApiParam

D ApiModel

答案

1=>B