@RestController 注解和@Controller 注解

@RestController 和@Controller 的区别

@RestController 注解相当于@ResponseBody + @Controller 合在一起的作用。

（1）如果只是使用@RestController 注解 Controller 层，那么 Controller 层中的方法无法返回到 jsp 页面，配置的视图解析器 InternalResourceViewResolver 不起作用，返回的数据就是 return 语句里面的内容。

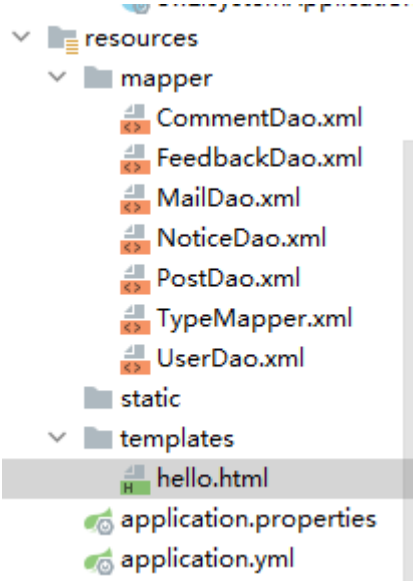
例如：本来应该到 success.jsp 页面的，则其显示 success

（2）如果需要返回到指定页面，则需要用 @Controller 配合视图解析器 InternalResourceViewResolver 才行。

（3）如果需要返回 JSON、XML 或者自定义 mediaType 到页面，则需要在对应的方法上加上 @ResponseBody 注解。

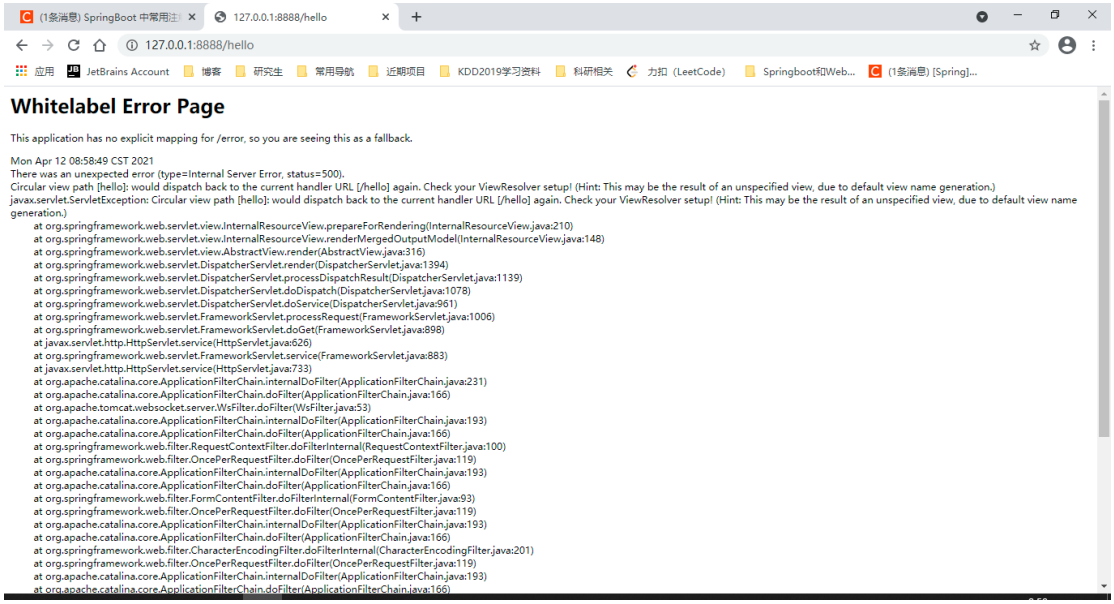
@Controller 处理 http 请求

```
6   @Controller
7   public class TestController {
8
9       @RequestMapping("/hello")
10      public String hello(){
11          return "hello";
12      }
13  }
14
```



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
你好你好
</body>
</html>
```

如果直接使用@Controller 这个注解，当运行该 SpringBoot 项目后，在浏览器中输入：127.0.0.1:8080/hello，会得到如下错误提示。



出现这种情况的原因在于：没有使用模板。即@Controller 用来响应页面，必须配合模板来使用。SpringBoot 支持多种模板引擎盖包括：（1）FreeMaker（2）Groovy（3）Thymeleaf（4）Velocity（5）JSP

如果我们在 pom 文件中加入模板依赖，比如 Thymeleaf。

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

在浏览器中输入同样的地址就会把 hello.html 页面解析出来。

@RestController 处理 Http 请求

Spring4 之后新加入的注解，原来返回 json 数据需要@ResposeBody 和@Controller 配合。即@RestController 是@ResponseBody 和@Controller 的组合注解。

```java
@RestController
public class TestController {

    @RequestMapping("/hello")
    public String hello(){
        return "hello";
    }
}
```

在浏览器中输入 127.0.0.1:8080/hello，之后

显示的并不是 hello.html 页面的内容，而是 return 语句的内容，这就是说，该注解仅返回数据，不解析视图。

联想一下“前后端分离和前后端不分离的本质区别”文档，你可能会更加的理解一点。为何我们在前后端分离的项目中 Controller 层的类中总是可以看到@RestController 的注解；而在前后端不分离的项目中，又总会看到@Controller 注解呢？原因可能如此：前后端分离的项目后端不需要渲染视图上的数据，数据渲染在前端完成，因此后端做的只是返回数据即可；而在不分离的项目中，后端需要完成完成视图的数据渲染工作，所以需要@Controller 配合。具体可以看下图：

前后端分离项目服务端接口
Return 语句返回的数据

```java
@RestController
@RequestMapping("/post")
public class PostController {

    @Resource
    private PostService postService;
    @Resource
    private CommentService commentService;
    @Autowired
    private TypeService typeService;
    @Resource
    private NoticeService noticeService;
    @RequestMapping("/queryAll")
    public JsonObject<Post> queryAll(@RequestParam(defaultValue = "1") Integer page,
                                     @RequestParam(defaultValue = "10") Integer pageSize,
                                     Post post){
        //创建返回的对象
        JsonObject<Post> jsonObject = new JsonObject<>();
        //查询结果列表
        PageInfo<Post> postPageInfo = postService.findAll(page,pageSize,post);
        //填充返回的前端数据对象
        jsonObject.setCode(0);
        jsonObject.setMsg("ok");
        jsonObject.setData(postPageInfo.getList());
        jsonObject.setCount(postPageInfo.getTotal());

        return jsonObject;
    }
```

前后端分离项目服务端接口

```
import ...

@Controller
public class TypeShowController {

    @Autowired
    private TypeService typeService;
    @Autowired
    private BlogService blogService;


    @GetMapping("/types/{id}")
    public String types(@PageableDefault(size = 8, sort = {"updateTime"},direction = Sort.Direction.D
                        @PathVariable Long id, Model model){
        List<Type> types = typeService.listTypeTop( size: 1000);
        if (id == -1){
            id = types.get(0).getId();
        }
        BlogQuery blogQuery = new BlogQuery();
        blogQuery.setTypeId(id);
        model.addAttribute( s: "types",types);
        model.addAttribute( s: "page",blogService.listBlog(pageable,blogQuery));
        model.addAttribute( s: "activeTypeId",id);
        return "types";
    }
}
```

Return 返回的是视图页面。