

springboot 项目中定义了很多类，我们在 rest 返回中直接返回或者在返回对象中使用这些类，spring 已经使用 jackson 自动帮我们完成这些的 to json。但是有时候自动转的 json 内容太多，或者格式不符合我们的期望，因此需要调整类的 to json 过程，或者说希望自定义类的 json 过程。

解决办法：

使用@JsonIgnoreProperties、@JsonIgnore、@JsonFormat。

@JsonIgnore 注解用来忽略某些字段，可以用在变量或者 Getter 方法上，用在 Setter 方法时，和变量效果一样。这个注解一般用在我们要忽略的字段上。

@JsonIgnoreProperties(ignoreUnknown = true)，将这个注解写在类上之后，就会忽略类中不存在的字段。这个注解还可以指定要忽略的字段，例如@JsonIgnoreProperties({ "password", "secretKey" })

@JsonFormat 可以帮我们完成格式转换。例如对于 Date 类型字段，如果不适用 JsonFormat 默认在 rest 返回的是 long，如果我们使用@JsonFormat(timezone = "GMT+8", pattern = "yyyy-MM-dd HH:mm:ss")，就返回"2018-11-16 22:58:15"

@Scheduled(fixedRate = 5000) 五秒执行一次

@Scheduled(fixedDelay = 5000) 执行完一次之后过五秒再继续第二次

使用 SpringApplication.run(Application.class)执行主函数

[@EnableScheduling](#) ensures that a background task executor is created. Without it, nothing gets scheduled.

Spring boot 框架构建， URL 路由， 参数管理， 自定义配置， JPA， Thymeleaf 模板， 表单处理， 分页处理， 开发者工具。

Spring boot 框架构建?

框架, 简化新 spring 应用的初始搭建以及开发过程, 使用自动配置, 使开发人员不再需要样板化的配置 (XML Configuration)。

Spring 的争议: 大量的 XML 配置, 复杂的依赖管理。

Springboot: 自动配置, 起步依赖, 命令行界面, actuator(监控运行程序状态的组件)

解决了什么问题? 搭建, 配置, 编码, 部署, 监控简单。。

能干什么? 基于模板的 Web Application 开发, restful 服务开发, 微服务

Spring 构建: initializr, IDEA, spring boot CLI

Spring boot 启动:

1. java 类文件启动 (@SpringBootApplication 注解 + SpringApplication.run() + 运行 main 方法)
2. mvn spring-boot:run 启动 在项目目录下运行此命令
3. java -jar 命令启动 mvn package->进入到 package 所在目录, 然后 java -jar

URL 路由

Restful API

@RestController 注解

@RequestMapping 注解

@RequestMapping 简写形式

@Controller 注解

Restful API????例子

GET http://localhost:8080/api/vi/books 获取读书清单列表

POST http://localhost:8080/api/vi/books 新增一个清单

GET 、 books/{id} 获取一条读书清单

PUT /books 更新一个读书清单

DELETE books/{id} 删除一个读书清单

@RestController

@RequestMapping(value = “/say”, method=RequestMethod.GET)

不标记请求类型就是支持所有请求类型

Requestmapping 的简写形式:

@PostMapping(“/say”) 代表了一个 post 请求

Controller 注解?

返回的字符串, 再去找对应的模板 thymeleaf,如何找到?

模板在 resource 下面的 template 定义

所以 controller 用来标记返回的是 view 也就是个模板, 类似一个 HTML 文档,

如果返回一个 JSON, 返回一个 string, 用 restcontroller

@ResponseBody 标记之后, 如果是字符串就返回字符串, 如果是实体就返回一个 JSON

参数管理 @PathVariable

```
@GetMapping("/books/{id}/{username}")

public Object getOne(@PathVariable long id, @PathVariable String username){

    System.out.println(id);

    System.out.println(username);

    Map<String, Object> book = new HashMap<>();

    book.put("name", "hulianwangshijieguan");

    book.put("isbn", "45646546");

    book.put("author", "wangshuai");

    return book;

}
```

参数处理 requestparam 表单接收

POST http://localhost:8080/api/v1/books 新增一个清单

http://localhost:8080/api/v1/books

POST http://localhost:8080/api/v1/books Send Save

Params Authorization Headers (9) **Body** Pre-request Script Tests Cookies Code Comments (0)

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	name	shijianyimin			
<input checked="" type="checkbox"/>	author	wangshuai			
<input checked="" type="checkbox"/>	isbn	4546546			
	Key	Value	Description		

```
@PostMapping("/books")

public Object post(@RequestParam("name") String name, @RequestParam("author") String author,
@RequestParam("isbn") String isbn){

    Map<String,Object> book = new HashMap<>();

    book.put("name", name);

    book.put("author", author);

    book.put("isbn", isbn);

    return book;

}
```

GET http://localhost:8080/api/v1/books 获取读书清单列表

:

```

@GetMapping("/books")
public Object getAll(@RequestParam("page") int page, @RequestParam(value = "size", defaultValue = "10") int
size) {
    Map<String, Object> book1 = new HashMap<>();
    book1.put("name", "hulianwangshijieguan");
    book1.put("isbn", "45646546");
    book1.put("author", "wangshuai");

    Map<String, Object> book2 = new HashMap<>();
    book2.put("name", "hulianwangshijieguan");
    book2.put("isbn", "456465467");
    book2.put("author", "wangshuail");
    List<Map> contents = new ArrayList<>();
    contents.add(book1);
    contents.add(book2);
    Map<String, Object> pagemap = new HashMap<>();
    pagemap.put("page", page);
    pagemap.put("size", size);
    pagemap.put("content", contents);
    return pagemap;
}

```

总结:

GET <http://localhost:8080/api/vi/books/{id}> 一般用 pathvariable

RequestParam 一般用于提交一个表单，比如分页方式获取数据。
<http://localhost:8080/api/v1/books?page=1>

自定义配置

配置文件，自定义属性配置，多环境配置

改端口号 port，在 resource 文件夹下修改 application.properties 文件，写入：

Server.port = 8081

将公有的部分 URL 加入根目录下，即修改根目录，使用：

```
server.servlet.context-path=/api
```

定义日志级别，日志位置：

```
logging.level.root = WARN
```

```
logging.level.com.lrm = DEBUG
```

```
logging.file.path=
```

在 properties 或者 yml 配置文件里写入常量，例如 book.name/author/isbn...

然后在 controller 类里增加这几个常量，并注解@Value(),即可直接使用

Controller 里

```
@Value("${book.name}")
```

```
private String name;
```

```
@Value("${book.author}")
```

```
private String author;
```

```
@Value("${book.isbn}")
```

```
private String isbn;
```

配置文件里

```
book.name=hulianwangshijieguan
```

```
book.author=wangshuai
```

```
book.isbn=45646546
```

自定义属性配置：

如何直接注入一个类的对象？？？ Autowired 用于注入一个实体对象，

@Component 标记 class，使 spring boot 识别它，识别了才能注入此类的一个实体对象

@ConfigurationProperties（prefix=?）用于表明类与配置文件中的哪个前缀匹配

```
public class HelloController {
```

```
    @Autowired
```

```
    private Book book;
```

```
@GetMapping("/books/{id}")
    public Object getOne(@PathVariable long id){
        return book;
    }
}
```

```
@Component
@ConfigurationProperties(prefix = "book")
public class Book {

    private String name;
    //    @Value("${book.author}")
    private String author;
    //    @Value("${book.isbn}")
    private String isbn;
    public Book(){

    }

    public void setName(String name) {
        this.name = name;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }

    public String getName() {
        return name;
    }

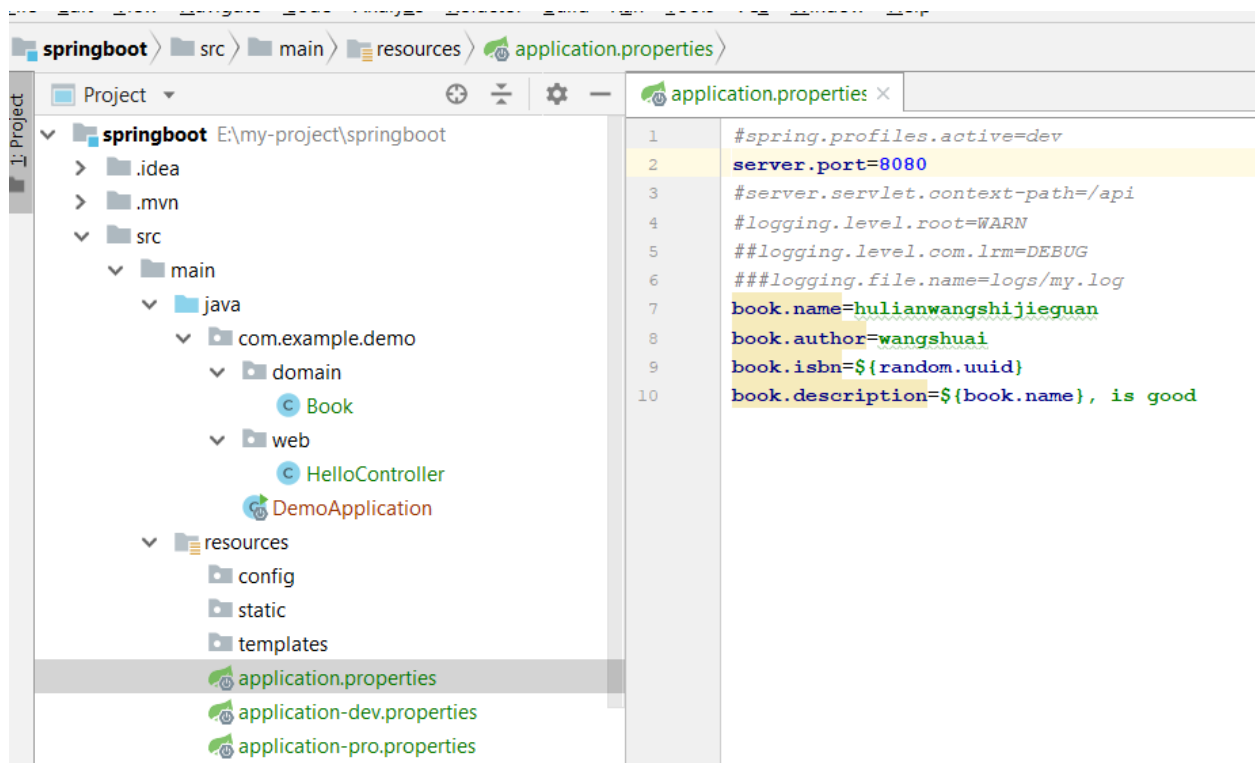
    public String getAuthor() {
        return author;
    }

    public String getIsbn() {
```



```
        return isbn;
    }
}
```

多环境配置:



建立了开发测试生产三个环境，使用 `spring.profiles.active=dev` 指定使用开发环境配置。

JPA 数据库操作

数据持久化-> 保存到数据库

配置, 基本查询, 复杂查询, 自定义查询, 自定义更新, 事务

配置:

JPA : java persistent API, 在 hibernate 中使用。

Spring data JPA: 基于 JPA 进一步简化了数据访问层的实现, 只需要编写 repository

如何使用 spring data JPA? ? ?

引入 pom 文件

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

改写 properties 配置文件:

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

```
spring.datasource.url=jdbc:mysql://localhost:3306/mysql?useUnicode=true&characterEncoding=utf-8
```

```
spring.datasource.username=root
```

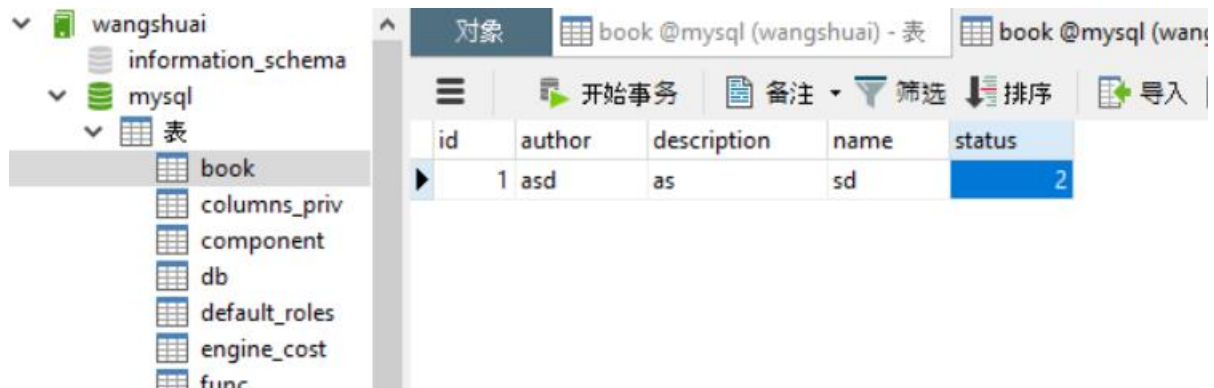
```
spring.datasource.password=root
```

```
spring.jpa.hibernate.ddl-auto=update
```

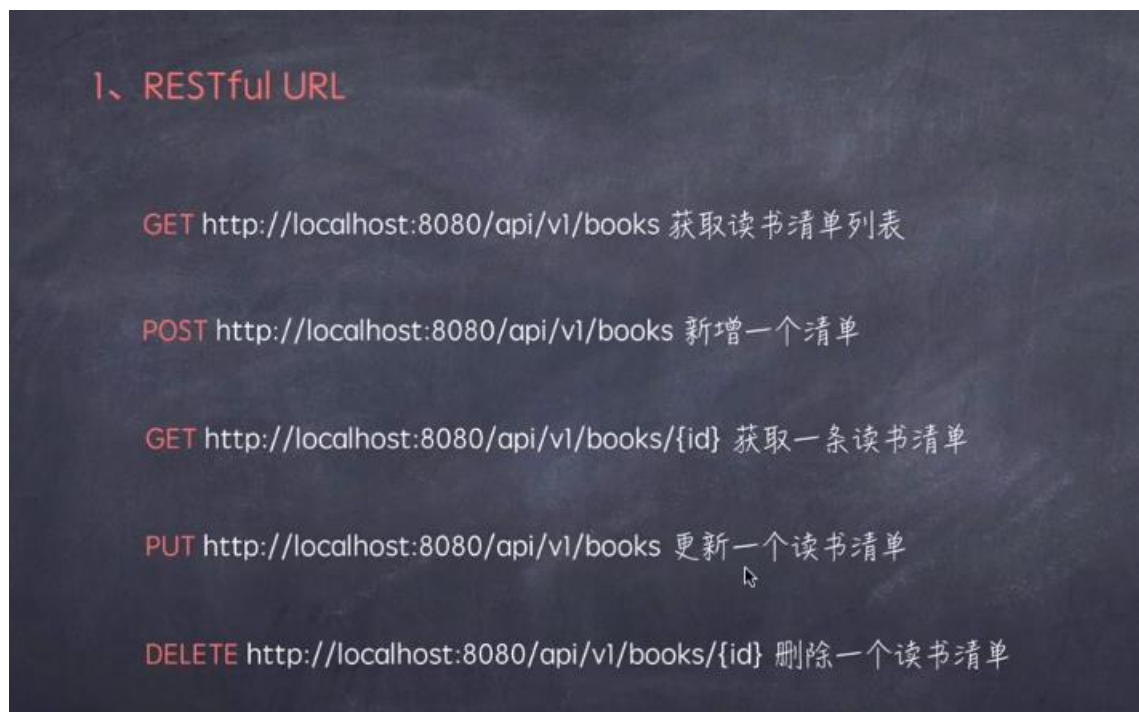
```
spring.jpa.show-sql=true
```

打开 navicat 可视化 mysql 数据库

运行 application



JPA 基本查询: 增删改查



步骤: 首先通过 `bookrepository` 接口继承 `JpaRepository`

然后明确 `web` 层和 `service` 层, `service` 层用来定义 `CRUD` 操作具体实现, 通过 `bookRepository`, `web` 层用来直接调用写好的方法。

具体代码:

```
1 package com.example.demo.domain;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 //用它来进行数据库操作
5 public interface BookRepository extends JpaRepository<Book, Long> {
6
7 }
8
```

ple / demo / service / BookService /

```
BookService.java x
4 import com.example.demo.domain.BookRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.data.domain.Example;
7 import org.springframework.stereotype.Service;
8 import java.util.List;
9 //定义数据库的一些业务，当前例子只需要查询
10 @Service
11 public class BookService {
12     @Autowired
13     private BookRepository bookRepository; // 拥有JpaRepository的一些基本方法
14     public List<Book> findAll() { return bookRepository.findAll(); }
15     //提交一各书单
16     public Book save(Book book) { return bookRepository.save(book); }
17     // 获取一条书单
18     public Book findOne(long id) {
19         return bookRepository.findById(id).get();
20     }
21     //删除一个书单 返回空值
22     public void deleteOne(long id) {
23         bookRepository.deleteById(id);
24     }
25 }
```

```
BookApp.java x
8 @RestController
9 @RequestMapping("/api/v1")
10 public class BookApp {
11     @Autowired
12     private BookService bookService;
13     // return all the read list
14     @GetMapping("/books")
15     public List<Book> getAll() { return bookService.findAll(); }
16     @PostMapping("/books")
17     public Book post(@RequestParam String name,
18                     @RequestParam String author,
19                     @RequestParam String description,
20                     @RequestParam int status){
21         Book book = new Book();
22         book.setName(name);
23         book.setAuthor(author);
24         book.setDescription(description);
25         book.setStatus(status);
26         return bookService.save(book);
27     }
28     //获取一条书单信息
29     @GetMapping("/books/{id}")
30     public Book getOne(@PathVariable long id) { return bookService.findOne(id); }
```

```
BookApp.java x
31 @GetMapping("/books/{id}")
32 public Book getOne(@PathVariable long id) { return bookService.findOne(id); }
33 // 更新一条书单
34 @PutMapping("/books")
35 public Book updateOne(@RequestParam long id,
36                      @RequestParam String name,
37                      @RequestParam String author,
38                      @RequestParam String description,
39                      @RequestParam int status){
40     Book book = new Book();
41     book.setName(name);
42     book.setAuthor(author);
43     book.setDescription(description);
44     book.setStatus(status);
45     book.setId(id);
46     return bookService.save(book);
47 }
48 //删除一个书单
49 @DeleteMapping("/books/{id}")
50 public void deleteOne(@PathVariable long id){
51     bookService.deleteOne(id);
52 }
53 }
54 }
55 }
```

JPA 复杂查询

```
BookRepository.java x BookService.java x BookApp.java x
1 package com.example.demo.domain;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 import java.util.List;
6
7 //用它来进行数据库操作
8 public interface BookRepository extends JpaRepository<Book, Long> {
9     List<Book> findByAuthor(String author);
10    List<Book> findByAuthorAndStatus(String author, int status);
11    List<Book> findByDescriptionEndsWith(String des);
12 }
13
```

```
re / demo / service / BOOKSERVICE /
- BookRepository.java x BookService.java x BookApp.java x
18 public Book save(Book book) { return bookRepository.save(book); }
21 // 获取一条书单
22 public Book findOne(long id) {
23
24     return bookRepository.findById(id).get();
25 }
26 //删除一个书单 返回空值
27 public void deleteOne(long id) { bookRepository.deleteById(id); }
30 // 根据author查询书单列表
31 public List<Book> findByAuthor(String author) {
32     return bookRepository.findByAuthor(author);
33 }
34
35 public List<Book> findByAuthorAndStatus(String author, int status) {
36     return bookRepository.findByAuthorAndStatus(author, status);
37 }
38 public List<Book> findByDescriptionEndsWith(String des) {
39     return bookRepository.findByDescriptionEndsWith(des);
40 }
41 }
42
```

```
le / demo / web / BookApp /
- BookRepository.java x BookService.java x BookApp.java x
42     BOOK book = new BOOK();
43     book.setName(name);
44     book.setAuthor(author);
45     book.setDescription(description);
46     book.setStatus(status);
47     book.setId(id);
48     return bookService.save(book);
49 }
50 //删除一个书单
51 @DeleteMapping("/books/{id}")
52 public void deleteOne(@PathVariable long id) { bookService.deleteOne(id); }
55
56 @PostMapping("/books/by")
57 public List<Book> findBy(@RequestParam String description){
58     // return bookService.findByAuthor(author);
59     return bookService.findByDescriptionEndsWith(description);
60 }
61
62
63 }
64
```

JPA 自定义查询

```
BookRepository.java x BookService.java x BookApp.java x
1 package com.example.demo.domain;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.data.jpa.repository.Query;
5
6 import java.util.List;
7
8 //用它来进行数据库操作
9 public interface BookRepository extends JpaRepository<Book, Long> {
10     List<Book> findByAuthor(String author);
11     List<Book> findByAuthorAndStatus(String author, int status);
12     List<Book> findByDescriptionEndsWith(String des);
13     //自定义查询语句注解
14     @Query(value = "select * from Book where length(name) > ?1", nativeQuery = true)
15     List<Book> findByJPQL(int len);
16 }
17
```

```
BookRepository.java × BookService.java × BookApp.java ×
30 // 根据author查询所有书
31 public List<Book> findByAuthor(String author){
32     return bookRepository.findByAuthor(author);
33 }
34
35 public List<Book> findByAuthorAndStatus(String author, int status){
36     return bookRepository.findByAuthorAndStatus(author, status);
37 }
38 public List<Book> findByDescriptionEndsWith(String des){
39     return bookRepository.findByDescriptionEndsWith(des);
40 }
41
42 // find by customized method
43 public List<Book> findByJPQL(int len){
44     return bookRepository.findByJPQL(len);
45 }
46 }
47
```

```
demo / demo / web / BOOKApp /
BookRepository.java × BookService.java × BookApp.java ×
42 BOOK book = new BOOK();
43 book.setName(name);
44 book.setAuthor(author);
45 book.setDescription(description);
46 book.setStatus(status);
47 book.setId(id);
48 return bookService.save(book);
49 }
50 //删除一个书单
51 @DeleteMapping("/books/{id}")
52 public void deleteOne(@PathVariable long id) { bookService.deleteOne(id); }
53
54
55
56 @PostMapping("/books/by")
57 public List<Book> findBy(@RequestParam int len){
58     // return bookService.findByAuthor(author);
59     return bookService.findByJPQL(len);
60 }
61
62
63 }
```


JPA 自定义更新

如果只需要修改表中某一列数据，就不需要调用 JPARepository 里面的 save 方法，可以自定义更新

例子，修改 ID 为 18 的书记 status 为 2

```
BookRepository.java × BookService.java × BookApp.java ×
import org.springframework.data.jpa.repository.Query;
import java.util.List;

//用它来进行数据库操作
public interface BookRepository extends JpaRepository<Book, Long> {
    List<Book> findByAuthor(String author);
    List<Book> findByAuthorAndStatus(String author, int status);
    List<Book> findByDescriptionEndsWith(String des);
    //自定义查询语句注解
    @Query(value = "select * from Book where length(name) > ?1", nativeQuery = true)
    List<Book> findByJPQL(int len);
}

@Modifying
@Query("update Book b set b.status = ?1 where id = ?2")
int updateByJPQL(int status, long id);
}
```

```
e / demo / service / BOOKService /
BookRepository.java × BookService.java × BookApp.java ×
public List<Book> findByAuthorAndStatus(String author, int status){
    return bookRepository.findByAuthorAndStatus(author, status);
}

public List<Book> findByDescriptionEndsWith(String des){
    return bookRepository.findByDescriptionEndsWith(des);
}

// find by customized method
public List<Book> findByJPQL(int len){
    return bookRepository.findByJPQL(len);
}

// 自定义更新
@Transactional
public int updateByJPQL(int status, long id){
    return bookRepository.updateByJPQL(status, id);
}
}
```

```
BookRepository.java × BookService.java × BookApp.java ×
46         book.setStatus(status);
47         book.setId(id);
48         return bookService.save(book);
49     }
50     //删除一个书单
51     @DeleteMapping("/books/{id}")
52     public void deleteOne(@PathVariable long id) { bookService.deleteOne(id); }
53
54
55
56     @PostMapping("/books/by")
57     public int findBy(@RequestParam int status, @RequestParam long id){
58         //         return bookService.findByAuthor(author);
59         return bookService.updateByJPQL(status, id);
60     }
61
62
63
64 }
65
```

数据库事务

使用@Transaction 在 service 层标记一个方法,那么方法中的所有 SQL 操作都在一个事务中,要么全部成功,要么全部失败。

Thymeleaf 模板的使用-》渲染界面展示

使用 thymeleaf3,取值, 静态资源处理, 判断, 迭代, URL

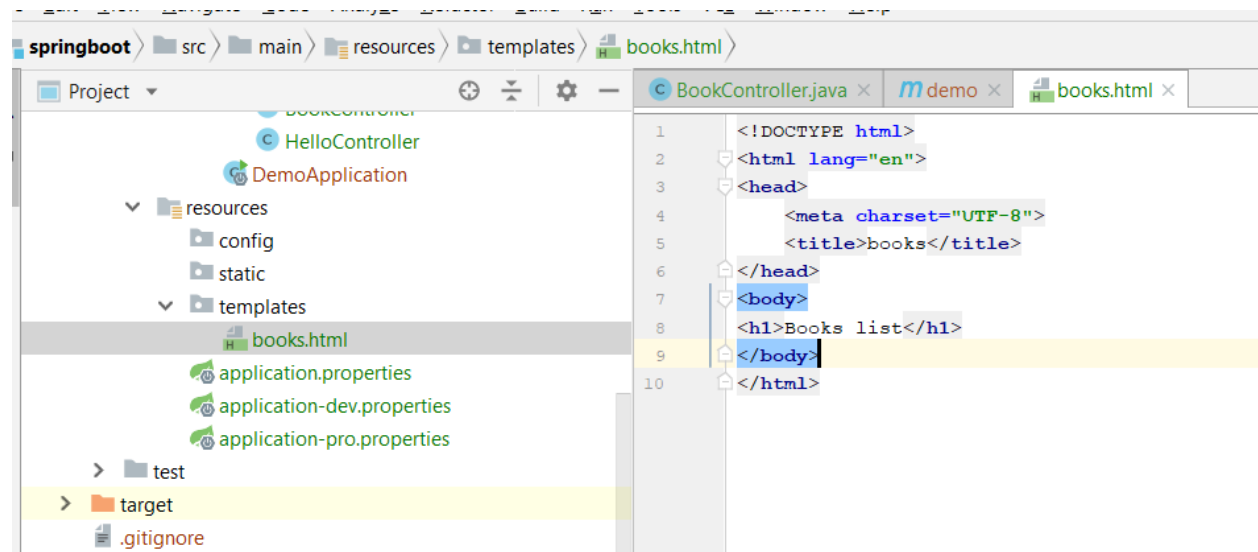
回忆-》》》 controller 关键字标记了之后, 会自动去找对应名字的模板

如何使用?

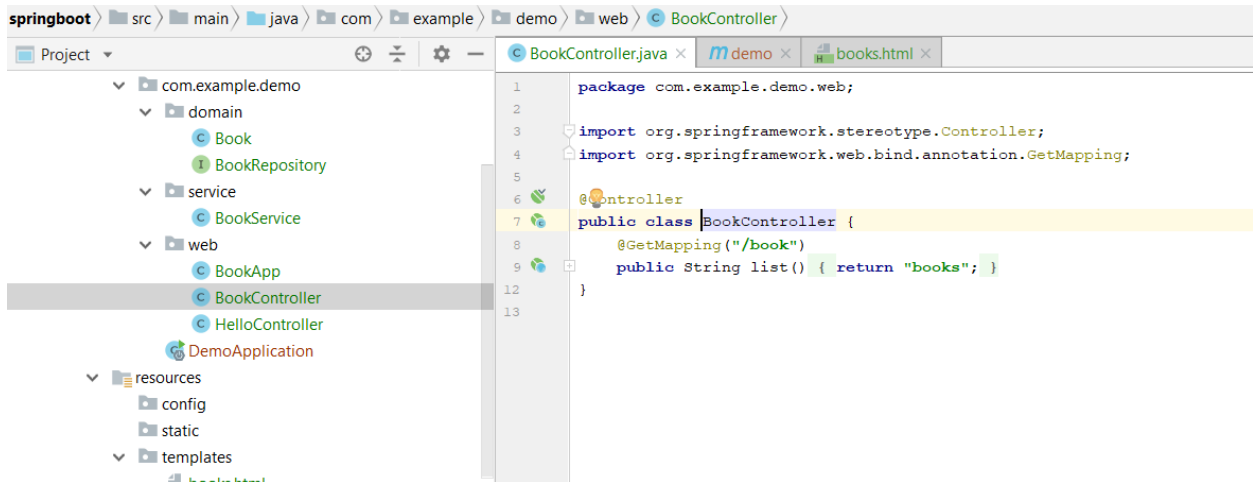
Pom 引入

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

创建 template 模板



建立 bookcontroller 并且附加@Controller 关键词



就会返回对应的 books 模板

Spring boot2 默认支持 thymeleaf3 不需要再手动引入

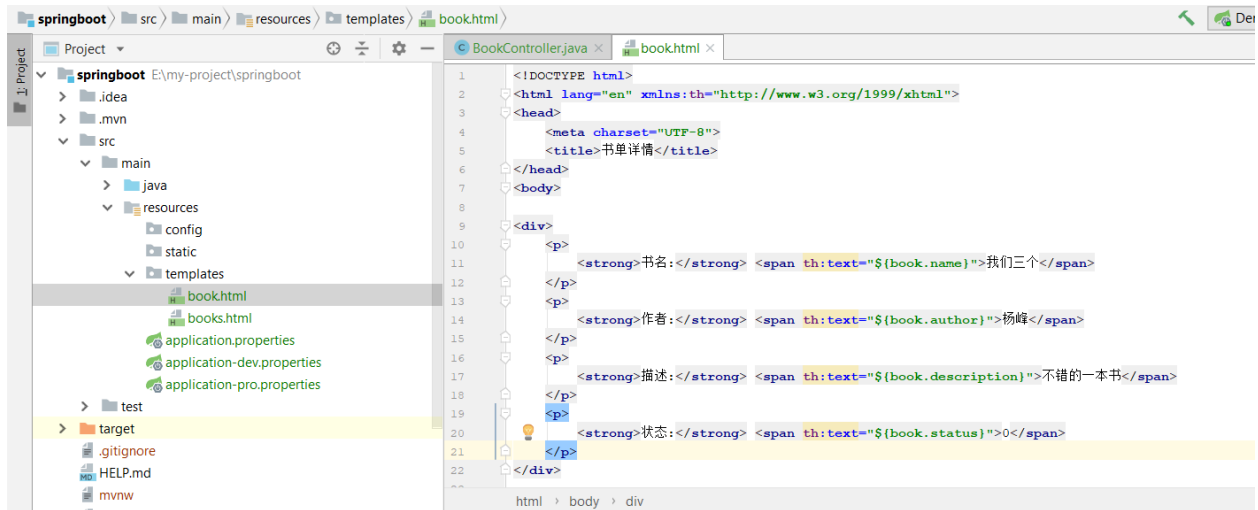
Thymeleaf 取值

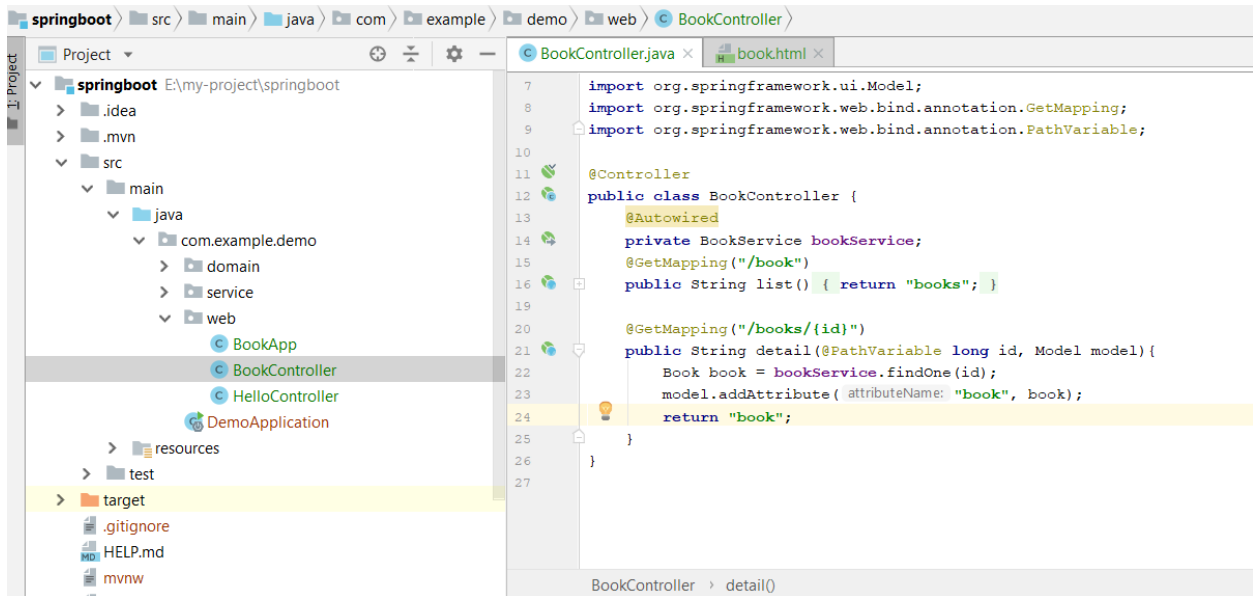
如何使用 thymeleaf3? ? ?

前后端分离, JSP 没有前后端分离

如何使用 spring boot 实现简单的书单管理? ? ?

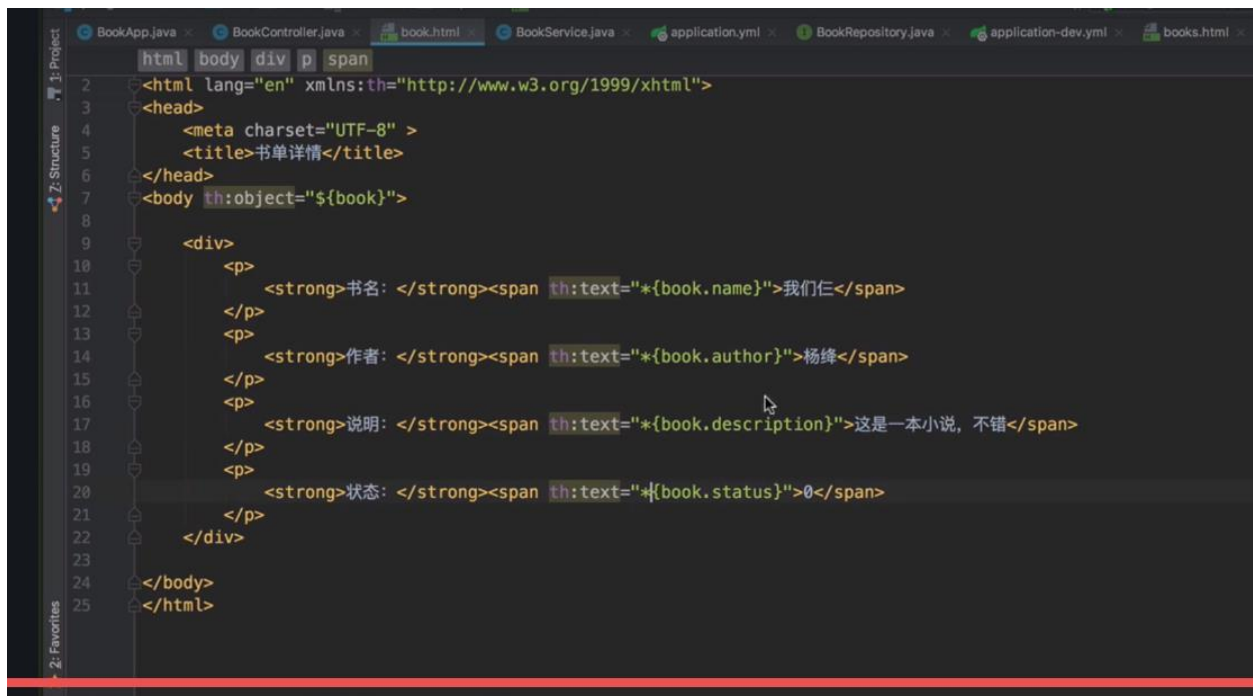
如何从数据库取得值, 然后渲染到页面上? ? ?





通过 `th:text="${book.name}"`

另一种方式



都可以

每次都可以省略写 book 了

Thymeleaf 静态资源处理

首先把静态资源放在正确的位置，static 文件夹下。

HTML 文件里面引入 bootstrap，使用相对路径。

`th:href="@{/css/bootstrap.min.css}"`

`th:src="@{/js/bootstrap.min.js}"`

```
<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">

<head>

    <title>书单详情</title>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap -->

    <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" href="../static/css/bootstrap.min.css">

</head>

<body>

<div class = "container" style="max-width: 700px;">

    <h2 class="page-header" style="margin-top: 50px;">详情</h2>

    <div class = "well">

        <p>

            <strong>书名:</strong> <span th:text="${book.name}">我们三个</span>

        </p>

        <p>

            <strong>作者:</strong> <span th:text="${book.author}">杨峰</span>

        </p>

        <p>
```

```

        <strong>描述:</strong> <span th:text="${book.description}">不错的一本书</span>
    </p>
    <p>
        <strong>状态:</strong> <span th:text="${book.status}">0</span>
    </p>
</div>
</div>

<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
<script th:src="@{/js/bootstrap.min.js}" src="../../static/js/bootstrap.min.js"></script>

</body>
</html>

```

Thymeleaf 判断

使用 switch

```

<p th:switch="${book.status}">
    <strong>状态:</strong>

    <span th:case="0">响度</span>
    <span th:case="1">在读</span>
    <span th:case="2">已读</span>

</p>

```

另一个场景：

想读的时候，给出提示

已读或者在读：给出出一个绿色的提示，不错 开始行动了

使用 bootstrap 模板的判断

```

<div class="alert alert-success" th:unless="*{status == 0}">
    <strong>不错</strong>，你已经开始行动了

```

```
</div>
<div class="alert alert-warning" th:if="*{status == 0}">
    <strong>注意</strong>, 你还没开始行动
</div>
```

Switch 可以判断多个元素，也可以直接使用 if,unless 判断

Thymeleaf 迭代

获取书单的列表，把书单的所有信息在一个页面显示出来

将需要循环的标签放在需要循环的元素上面，

比如要把所有书单显示出来，循环书单，将每一本书放到每一行里面。

所以在<tr></tr>里循环

```
<tr th:each="book,iterStat:${books}">
```

接着使用\$取出对应的属性

```
<td th:text="${iterStat.count}">1</td>
<td th:text="${book.name}">书名</td>
<td th:text="${book.author}">作者</td>
<td th:text="${book.description}">说明</td>
<td th:text="${book.status}">状态</td>
```



```
BookController.java x books.html x
4 <h2 class="page-header">书单列表</h2>
5 <table class="table table-bordered">
6   <thead>
7     <tr>
8       <th>#</th>
9       <th>书名</th>
0       <th>作者</th>
1       <th>说明</th>
2       <th>状态</th>
3     </tr>
4   </thead>
5   <tbody>
6     <tr th:each="book,iterStat:${books}">
7       <td th:text="${iterStat.count}">1</td>
8       <td th:text="${book.name}">书名</td>
9       <td th:text="${book.author}">作者</td>
0       <td th:text="${book.description}">说明</td>
1       <td th:text="${book.status}">状态</td>
2     </tr>
3   </tbody>
4   <!--<tr>-->
5     <!--<td>1</td>-->
```

Books list

书单列表

#	书名	作者	说明	状态
1	fgfgfg	fg	fg	0
2	we2	ui	sd	1
3	we22	ui	sd	1
4	we223	ui	sd	2
5	we223456	ui	sd	1
6	ginger	ws	xiaojjj	2
7	kkk	ginger	ginger is beautiful	0
8	huojin	ws	this is a physical book	0
9	dfdfdf	yajle	youxu	2

iterStat 的属性如下:

```
count: 计数 从1开始  
index: 索引 从0开始  
size: 列表总条数  
even/odd: 偶数 / 奇数  
first/last: 第一条 / 最后一条
```

THymeleaf URL 使用

点击书单列表如何能跳转到详情页面???

```
<!--<td><a href="#" th:text="${book.name}"  
th:href="@{/books/{id} (id=${book.id})}">书名  
</a></td>-->
```

```
<td><a href="#" th:text="${book.name}"  
th:href="@{'/books/' + ${book.id}}">书名  
</a></td>
```

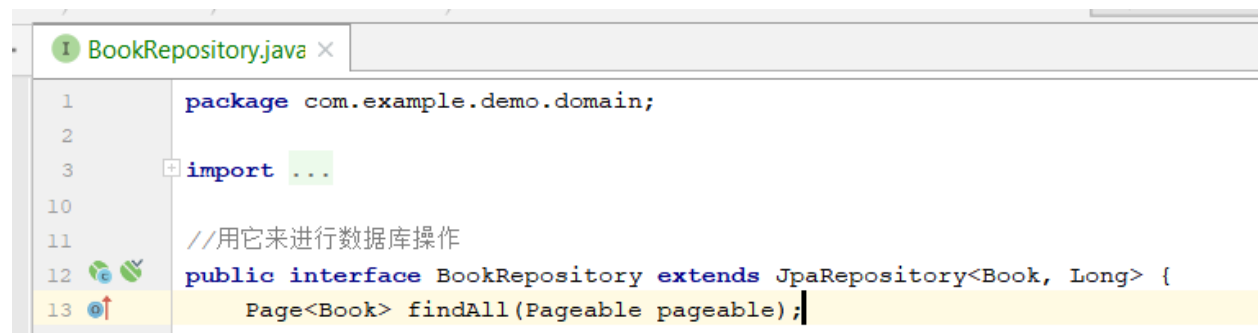
表单处理

分页处理:

分页查询方法:

目前数据全部在第一页渲染出来，所以需要分页获取

分页查询



```
1 package com.example.demo.domain;
2
3 import ...
10
11 //用它来进行数据库操作
12 public interface BookRepository extends JpaRepository<Book, Long> {
13     Page<Book> findAll(Pageable pageable);
```

自动重启:

File settings->compiler->build automatically

Maintenance->allow app auto running