Evan

Only let oneself become strong enough, good enough, can afford the life that you want to.

使用 JSONDoc 记录 Spring Boot RESTful API

标签： Spring Boot JSONDoc RESTful API

2017-06-11 13:37 216人阅读 评论(0) 收藏 编辑 删除

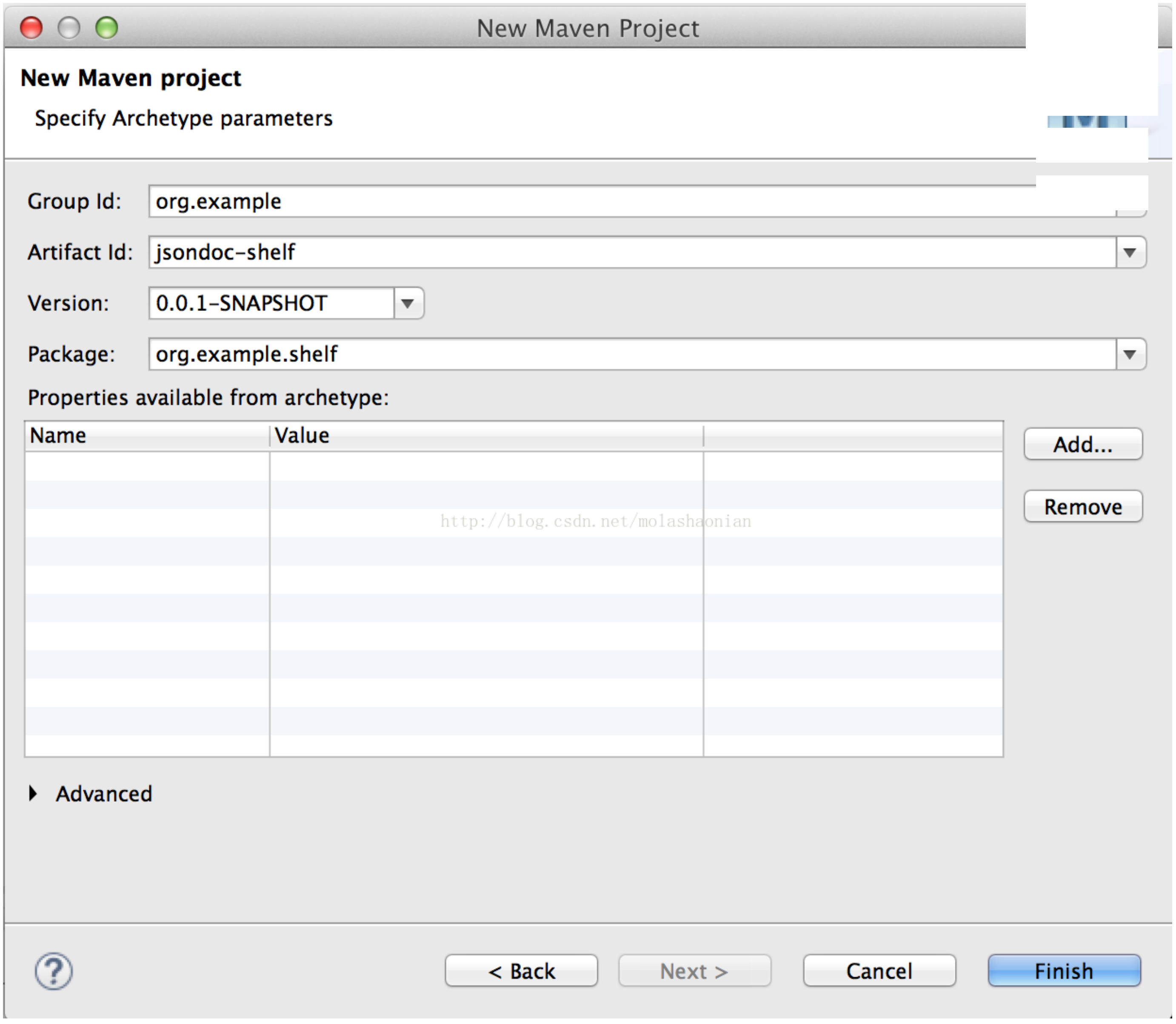
分类： Spring Boot（1） RESTful（1）

目录(?) [-]

这个博文可以分为两部分：第一部分我将编写一个**spring** Boot RESTful API，第二部分将介绍如何使用JSONDoc来记录创建的API。做这两个部分最多需要15分钟，因为使用Spring Boot创建一个API非常简单快捷，并且使用JSONDoc Spring Boot启动器和UI webjar进行记录也是如此。我将跳过这个例子的**测试**创建，因为主要目标是如何记录API而不是编写和测试它。

编写API

我们首先根据快速入门的原型创建Maven项目



New Maven project

Specify Archetype parameters

Group Id: org.example
Artifact Id: jsondoc-shelf
Version: 0.0.1-SNAPSHOT
Package: org.example.shelf

Properties available from archetype:

Name	Value

并声明API所需的依赖关系：

- spring-boot-starter-web
- spring-boot-starter-data-jpa
- h2

我还添加了 Lombok 保持我的代码更干净。所得的pom看起来像这样：

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <groupId>org.example</groupId>
        <artifactId>jsondoc-shelf</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>jar</packaging>

        <name>jsondoc-shelf</name>
        <url>http://maven.apache.org</url>

        <properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        </properties>

        <dependencies>

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

```
				<version>1.2.0.RELEASE</version>
			</dependency>

			<dependency>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-starter-data-jpa</artifactId>
				<version>1.2.0.RELEASE</version>
			</dependency>

			<dependency>
				<groupId>com.h2database</groupId>
				<artifactId>h2</artifactId>
				<version>1.3.176</version>
			</dependency>

			<dependency>
				<groupId>org.projectlombok</groupId>
				<artifactId>lombok</artifactId>
				<version>1.14.8</version>
			</dependency>

			<dependency>
				<groupId>junit</groupId>
				<artifactId>junit</artifactId>
				<version>4.11</version>
				<scope>test</scope>
			</dependency>

		</dependencies>
</project>
```

这个应用程序将是一个管理简单货架的服务的集合。将有两个实体：

- Book
- Author

创建 Entities 和 Controllers

为此，我将创建通常的组件来管理持久层和控制器层：

- 一个包名为 `model` 将包含 `Book` 和 `Author`
- 一个包名为 `repository` 将包含 `BookRepository` 和 `AuthorRepository`
- 一个包名为 `controller` 将包含 `BookController` 和 `AuthorController`

对于这个例子，我将跳过 Service 层。我还将创建一个 `DatabasePopulator` 类，实现 `CommandLineRunner` ，以便在启动时将在内存数据库中存在一些数据。我们来看看实体，存储库和控制器的代码：

Entities

```
package org.example.shelf.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

import lombok.Data;
import lombok.EqualsAndHashCode;

@Entity
@Data
@EqualsAndHashCode(exclude = "id")
public class Book {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;

	@Column(name = "title")
```

```
        private String title;

        @ManyToOne
        @JoinColumn(name = "author_id")
        private Author author;

}
```

```
package org.example.shelf.model;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;

import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.NoArgsConstructor;
import lombok.ToString;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Entity
@Data
@NoArgsConstructor
@ToString(exclude = "books")
@EqualsAndHashCode(of = "name")
public class Author {

        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        private Long id;

        @Column(name = "name")
        private String name;

        @JsonIgnore
        @OneToMany(mappedBy = "author", fetch = FetchType.LAZY, cascade = CascadeType.ALL)
        private List<Book> books = new ArrayList<Book>();

}
```

Repositories

```
package org.example.shelf.repository;

import org.example.shelf.model.Book;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}
```

```
package org.example.shelf.repository;

import org.example.shelf.model.Author;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

public interface AuthorRepository extends JpaRepository<Author, Long> {

}
```

Controllers

```
package org.example.shelf.controller;

import java.util.List;

import org.example.shelf.flow.ShelfFlowConstants;
import org.example.shelf.model.Book;
import org.example.shelf.repository.BookRepository;
import org.jsondoc.core.annotation.Api;
import org.jsondoc.core.annotation.ApiBodyObject;
import org.jsondoc.core.annotation.ApiMethod;
import org.jsondoc.core.annotation.ApiPathParam;
import org.jsondoc.core.annotation.ApiResponseObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.util.UriComponentsBuilder;

@RestController
@RequestMapping(value = "/books", produces = MediaType.APPLICATION_JSON_VALUE)
public class BookController {

        @Autowired
        private BookRepository bookRepository;

        @RequestMapping(value = "/{id}", method = RequestMethod.GET)
        public Book findOne(@PathVariable Long id) {
                return bookRepository.findOne(id);
        }

        @RequestMapping(method = RequestMethod.GET)
        public List<Book> findAll() {
                return bookRepository.findAll();
        }

        @RequestMapping(method = RequestMethod.POST, consumes = MediaType.APPLICATION_JSON_VALUE)
        @ResponseStatus(value = HttpStatus.CREATED)
        public ResponseEntity<Void> save(@RequestBody Book book, UriComponentsBuilder uriComponentsBuilder) {
                bookRepository.save(book);

                HttpHeaders headers = new HttpHeaders();
                headers.setLocation(uriComponentsBuilder.path("/books/{id}").buildAndExpand(book.getId()).toUri());
                return new ResponseEntity<Void>(headers, HttpStatus.CREATED);
        }

        @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
        @ResponseStatus(value = HttpStatus.OK)
        public void delete(@PathVariable Long id) {
                Book book = bookRepository.findOne(id);
                bookRepository.delete(book);
        }

}
```

```
package org.example.shelf.controller;

import java.util.List;

import org.example.shelf.flow.ShelfFlowConstants;
import org.example.shelf.model.Author;
import org.example.shelf.repository.AuthorRepository;
import org.jsondoc.core.annotation.Api;
import org.jsondoc.core.annotation.ApiBodyObject;
import org.jsondoc.core.annotation.ApiMethod;
import org.jsondoc.core.annotation.ApiPathParam;
import org.jsondoc.core.annotation.ApiResponseObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.util.UriComponentsBuilder;

@RestController
@RequestMapping(value = "/authors", produces = MediaType.APPLICATION_JSON_VALUE)
public class AuthorController {

        @Autowired
        private AuthorRepository authorRepository;

        @RequestMapping(value = "/{id}", method = RequestMethod.GET)
        public Author findOne(@PathVariable Long id) {
                return authorRepository.findOne(id);
        }

        @RequestMapping(method = RequestMethod.GET)
        public List<Author> findAll() {
                return authorRepository.findAll();
        }

        @RequestMapping(method = RequestMethod.POST, consumes = MediaType.APPLICATION_JSON_VALUE)
        @ResponseStatus(value = HttpStatus.CREATED)
        public ResponseEntity<Void> save(@RequestBody Author author, UriComponentsBuilder uriComponentsBuilder) {
                authorRepository.save(author);

                HttpHeaders headers = new HttpHeaders();
                headers.setLocation(uriComponentsBuilder.path("/authors/{id}").buildAndExpand(author.getId()).toUri());
                return new ResponseEntity<Void>(headers, HttpStatus.CREATED);
        }

        @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
        @ResponseStatus(value = HttpStatus.OK)
        public void delete(@PathVariable Long id) {
                Author author = authorRepository.findOne(id);
                authorRepository.delete(author);
        }

}
```

Database populator

```
package org.example.shelf;

import org.example.shelf.model.Author;
import org.example.shelf.model.Book;
import org.example.shelf.repository.AuthorRepository;
import org.example.shelf.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Configuration;

@Configuration
public class DatabasePopulator implements CommandLineRunner {

        @Autowired
        private AuthorRepository authorRepository;

        @Autowired
        private BookRepository bookRepository;

        public void run(String... arg0) throws Exception {
                Author horbny = new Author();
                horbny.setId(1L);
                horbny.setName("Nick Horby");

                Author smith = new Author();
                smith.setId(2L);
                smith.setName("Wilbur Smith");

                authorRepository.save(horbny);
                authorRepository.save(smith);

                Book highFidelty = new Book();
                highFidelty.setId(1L);
                highFidelty.setTitle("High fidelty");
                highFidelty.setAuthor(horbny);

                Book aLongWayDown = new Book();
                aLongWayDown.setId(2L);
                aLongWayDown.setTitle("A long way down");
                aLongWayDown.setAuthor(horbny);

                Book desertGod = new Book();
                desertGod.setId(3L);
                desertGod.setTitle("Desert god");
                desertGod.setAuthor(smith);

                bookRepository.save(highFidelty);
                bookRepository.save(aLongWayDown);
                bookRepository.save(desertGod);
        }

}
```

现在是编写主类来运行应用程序的时候了。 `Shelf` 在这种情况下，我会称之为Spring Boot，这很简单：

```
package org.example.shelf;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@EnableAutoConfiguration
@EnableJpaRepositories
@ComponentScan
public class Shelf {

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

}
```

通过运行这个类，我们可以实际验证应用程序是否响应请求。您可以通过使用 curl 轻松测试 API 的工作：

```
curl -i http://localhost:8080/books/1
curl -i http://localhost:8080/books

curl -i http://localhost:8080/authors/1
curl -i http://localhost:8080/authors
```

用JSONDoc记录API

这是有趣的和新的部分，即使用JSONDoc库来注释代码并自动生成其文档。要做到这一点，你必须声明JSONDoc依赖关系，并在你的类中插入一些代码。让我们看看如何做到这一点：

声明JSONDoc依赖关系

只需添加两个依赖关系到pom文件：

```
<dependency>
        <groupId>org.jsondoc</groupId>
        <artifactId>spring-boot-starter-jsondoc</artifactId>
        <version>1.1.3</version>
</dependency>

<dependency>
        <groupId>org.jsondoc</groupId>
        <artifactId>jsondoc-ui-webjar</artifactId>
        <version>1.1.3</version>
</dependency>
```

在主类中启用JSONDoc

使用JSONDoc启动器，您可以通过添加 `@EnableJSONDoc` 到 `Shelf` 类中来启用文档生成，如下所示：

```
package org.example.shelf;

import org.jsondoc.spring.boot.starter.EnableJSONDoc;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@EnableAutoConfiguration
@EnableJpaRepositories
@EnableJSONDoc
@ComponentScan
public class Shelf {

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

}
```

配置JSONDoc

接下来要做的是配置JSONDoc来扫描您的控制器，对象和流类。要做到这一点，只需添加一些条目到 `application.properties` 文件（ `src/main/resources` 如果你没有它创建它）

```
jsondoc.version=1.0
jsondoc.basePath=http://localhost:8080
jsondoc.packages[0]=org.example.shelf.model
jsondoc.packages[1]=org.example.shelf.controller
```

文档控制器

JSONDoc可以从Spring注释中获取几个信息来构建文档。无论如何，它是一个选择加入的过程，这意味着JSONDoc将仅在使用自己的注释注释时才扫描类和方法。例如，要正确记录 `BookController` ，这里是如何使用JSONDoc注释：

```
package org.example.shelf.controller;

import java.util.List;

import org.example.shelf.flow.ShelfFlowConstants;
import org.example.shelf.model.Book;
import org.example.shelf.repository.BookRepository;
import org.jsondoc.core.annotation.Api;
import org.jsondoc.core.annotation.ApiBodyObject;
import org.jsondoc.core.annotation.ApiMethod;
import org.jsondoc.core.annotation.ApiPathParam;
import org.jsondoc.core.annotation.ApiResponseObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.util.UriComponentsBuilder;

@RestController
@RequestMapping(value = "/books", produces = MediaType.APPLICATION_JSON_VALUE)
@Api(description = "The books controller", name = "Books services")
public class BookController {

        @Autowired
        private BookRepository bookRepository;

        @ApiMethod
        @RequestMapping(value = "/{id}", method = RequestMethod.GET)
        public @ApiResponseObject Book findOne(@ApiPathParam(name = "id") @PathVariable Long id) {
                return bookRepository.findOne(id);
        }

        @ApiMethod
        @RequestMapping(method = RequestMethod.GET)
        public @ApiResponseObject List<Book> findAll() {
                return bookRepository.findAll();
        }

        @ApiMethod
        @RequestMapping(method = RequestMethod.POST, consumes = MediaType.APPLICATION_JSON_VALUE)
        @ResponseStatus(value = HttpStatus.CREATED)
        public @ApiResponseObject ResponseEntity<Void> save(@ApiBodyObject @RequestBody Book book, UriComponentsBuilder uriCompo
nentsBuilder) {
                bookRepository.save(book);

                HttpHeaders headers = new HttpHeaders();
            headers.setLocation(uriComponentsBuilder.path("/books/{id}").buildAndExpand(book.getId()).toUri());
                return new ResponseEntity<Void>(headers, HttpStatus.CREATED);
        }

        @ApiMethod
        @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
        @ResponseStatus(value = HttpStatus.OK)
        public void delete(@ApiPathParam(name = "id") @PathVariable Long id) {
                Book book = bookRepository.findOne(id);
                bookRepository.delete(book);
        }

}
```

同样的 `AuthorController` 。

文件对象

接下来要做的就是把一些JSONDoc注释也需要被记录在案，在这种情况下，对象 `Book` 和 `Author` 。这是 `Book` 类：

```
package org.example.shelf.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

import org.jsondoc.core.annotation.ApiObject;
import org.jsondoc.core.annotation.ApiObjectField;

import lombok.Data;
import lombok.EqualsAndHashCode;

@Entity
@Data
@EqualsAndHashCode(exclude = "id")
@ApiObject
public class Book {

        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        @ApiObjectField(description = "The book's ID")
        private Long id;

        @Column(name = "title")
        @ApiObjectField(description = "The book's title")
        private String title;

        @ManyToOne
        @JoinColumn(name = "author_id")
        @ApiObjectField(description = "The book's author")
        private Author author;

}
```

而且在这种情况下 Author 也是如此。

检查点：启动应用程序

在开始记录流程之前，让我们启动应用程序，看看会发生什么：

- 如果你去 http://localhost:8080/jsondoc 你会看到一个json，这是由JSONDoc生成的，它代表了基于控制器方法和模型对象上的工作的文档
- 如果你去 http://localhost:8080/jsondoc-ui.html 你会看到JSONDoc UI。只需复制并粘贴 http://localhost:8080/jsondoc 到输入字段中，并在清晰的用户界面中获取文档

这是一个很好的时机，需要一些时间来探索界面，并在界面上玩API。

文件流

按照流程我的意思是一些API方法的后续执行，旨在实现一个目标，即可以购买一本书，或浏览目录并获取图书详细信息。在这种情况下，流程可能涉及几种方法，API用户可能需要知道哪个是正确的调用方法序列来实现目标。在这个例子中，我不能想到有意义的流程，但是让我们假设我想要记录浏览框架的方法顺序，并通过我选择的一本书获取作者的细节，所以这个用例的结果流是就像是：

- 获取书籍清单
- 选择一本书并获得其细节
- 得到这本书的作者

要记录此流程，您只需按照以下步骤操作：

1. 创建一个包含应用程序流的类。此类仅用于文档目的，不会在您的应用程序中实际使用。使用注释来注释这个类 @ApiFlowSet ，这使得JSONDoc了解在构建文档时应该考虑到这个类。
2. 在这个类中创建假的方法，注释为 @ApiFlow 。方法的正文以及它的返回类型和参数可以是void，因为方法签名服务器只是作为 @ApiFlow 注释的钩子
3. 决定标识JSONDoc产生文档内的每一个API方法中，例如一个ID的 findAll 方法的 BookController 可有一个像ID BOOK_FIND_ALL
4. 将这个ID内部ID的 @ApiMethod 注释和内部 api methodid 的 @ApiFlowStep 注解
5. 如果将流类放在一个单独的包中，请记住 application.properties 使用该值更新该文件

我们来看看我是怎么做到的 这是持有应用程序流程的类：

```
package org.example.shelf.flow;

import org.jsondoc.core.annotation.ApiFlow;
import org.jsondoc.core.annotation.ApiFlowSet;
import org.jsondoc.core.annotation.ApiFlowStep;

@ApiFlowSet
public class ShelfFlows {

        @ApiFlow(
                name = "Author detail flow",
                description = "Gets an author's details starting from the book's list",
                steps = {
                        @ApiFlowStep(apimethodid = ShelfFlowConstants.BOOK_FIND_ALL),
                        @ApiFlowStep(apimethodid = ShelfFlowConstants.BOOK_FIND_ONE),
                        @ApiFlowStep(apimethodid = ShelfFlowConstants.AUTHOR_FIND_ONE)
                }
        )
        public void authorDetailFlow() {

        }

}
```

这是包含注释中要引用的方法ID的类：

```
package org.example.shelf.flow;

public class ShelfFlowConstants {

        // Book IDs
        public final static String BOOK_FIND_ALL = "BOOK_FIND_ALL";
        public final static String BOOK_FIND_ONE = "BOOK_FIND_ONE";
        public final static String BOOK_SAVE = "BOOK_SAVE";
        public final static String BOOK_DELETE = "BOOK_DELETE";

        // Author IDs
        public final static String AUTHOR_FIND_ALL = "AUTHOR_FIND_ALL";
        public final static String AUTHOR_FIND_ONE = "AUTHOR_FIND_ONE";
        public final static String AUTHOR_SAVE = "AUTHOR_SAVE";
        public final static String AUTHOR_DELETE = "AUTHOR_DELETE";

}
```

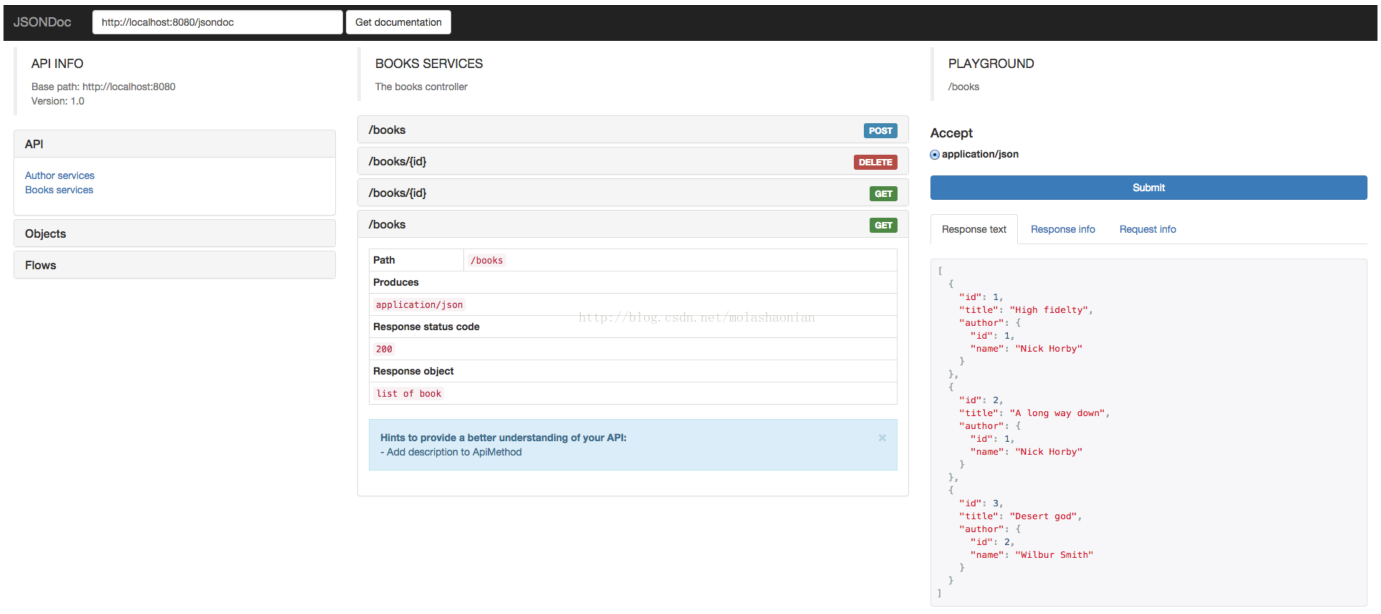
这是 `BookController` ，指定了id属性后：

```
package org.example.shelf.controller;

import java.util.List;

import org.example.shelf.flow.ShelfFlowConstants;
import org.example.shelf.model.Book;
import org.example.shelf.repository.BookRepository;
import org.jsondoc.core.annotation.Api;
import org.jsondoc.core.annotation.ApiBodyObject;
import org.jsondoc.core.annotation.ApiMethod;
import org.jsondoc.core.annotation.ApiPathParam;
import org.jsondoc.core.annotation.ApiResponseObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.util.UriComponentsBuilder;

@RestController
@RequestMapping(value = "/books", produces = MediaType.APPLICATION_JSON_VALUE)
@Api(description = "The books controller", name = "Books services")
public class BookController {

        @Autowired
        private BookRepository bookRepository;

        @ApiMethod(id = ShelfFlowConstants.BOOK_FIND_ONE)
        @RequestMapping(value = "/{id}", method = RequestMethod.GET)
        public @ApiResponseObject Book findOne(@ApiPathParam(name = "id") @PathVariable Long id) {
                return bookRepository.findOne(id);
        }

        @ApiMethod(id = ShelfFlowConstants.BOOK_FIND_ALL)
        @RequestMapping(method = RequestMethod.GET)
        public @ApiResponseObject List<Book> findAll() {
                return bookRepository.findAll();
        }

        @ApiMethod(id = ShelfFlowConstants.BOOK_SAVE)
        @RequestMapping(method = RequestMethod.POST, consumes = MediaType.APPLICATION_JSON_VALUE)
        @ResponseStatus(value = HttpStatus.CREATED)
        public @ApiResponseObject ResponseEntity<Void> save(@ApiBodyObject @RequestBody Book book, UriComponentsBuilder uriCompo
nentsBuilder) {
                bookRepository.save(book);

                HttpHeaders headers = new HttpHeaders();
            headers.setLocation(uriComponentsBuilder.path("/books/{id}").buildAndExpand(book.getId()).toUri());
                return new ResponseEntity<Void>(headers, HttpStatus.CREATED);
        }

        @ApiMethod(id = ShelfFlowConstants.BOOK_DELETE)
        @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
        @ResponseStatus(value = HttpStatus.OK)
        public void delete(@ApiPathParam(name = "id") @PathVariable Long id) {
                Book book = bookRepository.findOne(id);
                bookRepository.delete(book);
        }

}
```

最后的 `application.properties` 文件，用新的包：

```
jsondoc.version=1.0
jsondoc.basePath=http://localhost:8080
jsondoc.packages[0]=org.example.shelf.model
jsondoc.packages[1]=org.example.shelf.controller
jsondoc.packages[2]=org.example.shelf.flow
```

现在是再次启动应用程序的时候，转到 `http://localhost:8080/jsondoc-ui.html` ，插入 `http://localhost:8080/jsondoc` 输入框并获取文档。请享用！



API INFO
Base path: http://localhost:8080
Version: 1.0

API
Author services
Books services

Objects

Flows

BOOKS SERVICES
The books controller

/books POST
/books/{id} DELETE
/books/{id} GET
/books GET

Path	/books
Produces	
application/json	
Response status code	
200	
Response object	
list of book	

Hints to provide a better understanding of your API:
- Add description to ApiMethod

PLAYGROUND
/books

Accept
(●) application/json

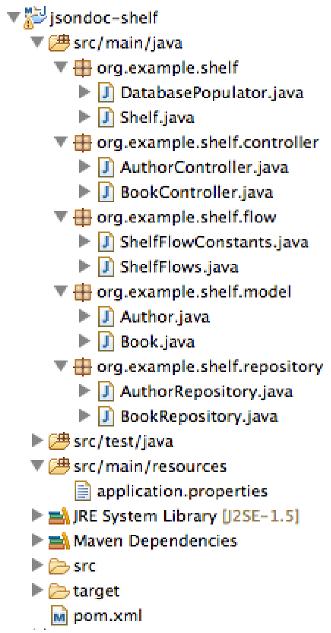
Submit

Response text | Response info | Request info

```
[
  {
    "id": 1,
    "title": "High fidelty",
    "author": {
      "id": 1,
      "name": "Nick Horby"
    }
  },
  {
    "id": 2,
    "title": "A long way down",
    "author": {
      "id": 1,
      "name": "Nick Horby"
    }
  },
  {
    "id": 3,
    "title": "Desert god",
    "author": {
      "id": 2,
      "name": "Wilbur Smith"
    }
  }
]
```

资源

这是项目的结构：



```
▼ jsondoc-shelf
  ▼ src/main/java
    ▼ org.example.shelf
      ▶ DatabasePopulator.java
      ▶ Shelf.java
    ▼ org.example.shelf.controller
      ▶ AuthorController.java
      ▶ BookController.java
    ▼ org.example.shelf.flow
      ▶ ShelfFlowConstants.java
      ▶ ShelfFlows.java
    ▼ org.example.shelf.model
      ▶ Author.java
      ▶ Book.java
    ▼ org.example.shelf.repository
      ▶ AuthorRepository.java
      ▶ BookRepository.java
  ▶ src/test/java
  ▼ src/main/resources
      application.properties
  ▶ JRE System Library [J2SE-1.5]
  ▶ Maven Dependencies
  ▶ src
  ▶ target
    pom.xml
```

链接

您可以在https://github.com/fabiomaffioletti/jsondoc-samples上看到这个和其他示例

您可以在https://github.com/fabiomaffioletti/jsondoc上查看JSONDoc的完整源代码

和http://jsondoc.eu01.aws.af.cm/jsondoc.jsp的演示

顶 踩
0 0

- 上一篇 Nginx负载均衡
- 下一篇 Restful 接口传递参数

相关文章推荐

- Spring Boot中使用Swagger2构建强大的RESTful...
- spring3 的restful API RequestMapping介绍
- 微服务架构 - Spring Boot中使用Swagger2构建...
- Spring MVC实现的RESTful webservice服务器并...
- restful api的spring实现
- 基于Spring-WS的Restful API的集成测试
- 使用Spring Boot 创建微服务
- Spring Boot学习笔记 - 整合Swagger2自动生成R...
- Spring Boot使用redis做数据缓存
- Spring boot 中使用swagger-ui实现 restful-api

猜你在找

【直播】机器学习&深度学习系统实战（唐宇迪）
【直播回放】深度学习基础与TensorFlow实践（王琛）
【直播】机器学习之凸优化（马博士）
【直播】机器学习之概率与统计推断（冒教授）
【直播】TensorFlow实战进阶（智亮）
【直播】Kaggle 神器：XGBoost 从基础到实战（冒教授）
【直播】计算机视觉原理及实战（屈教授）
【直播】机器学习之矩阵（黄博士）
【直播】机器学习之数学基础
【直播】深度学习30天系统实训（唐宇迪）

查看评论

暂无评论

发表评论

用 户 名：	molashaonian
评论内容：	

提交

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场