

A Developer Diary

{about:"code learn and share"}



[Home](#)

[Data Science](#)

[Java](#)

[JavaScript](#)

[jBPM](#)

[Tools](#)

[Tips](#)

[About](#)

April 4, 2016 By [Abhisek Jana](#)

How to create RESTful Webservices using Spring Boot

SPRING BOOT
REST





In this part of Spring Boot series, we will learn How to create RESTful Webservices using Spring Boot. We will also look into Actuator and its usefulness in an Application. **Spring boot has made bootstrapping Java application incredibly simple.** You probably already know about RESTful webservice, however we will see how easily we can create them using Spring Boot annotations.

We have already added the **Web** as a dependency in our project. You can see the following being added in the **pom** file. In case you want to add any additional features, you can add in the pom file directly and IntelliJ IDEA or Eclipse will automatically add the dependent jars.

pom.xml	XHTML
<pre>1 <dependency> 2 <groupId>org.springframework.boot</groupId> 3 <artifactId>spring-boot-starter-web</artifactId> 4 </dependency></pre>	

The @SpringBootApplication annotation is the starting point of Spring Boot application. We will initialize the application by calling the following line from our main method.

	Java
<pre>1 @SpringBootApplication 2 public class Application { 3 public static void main(String[] args) { 4 SpringApplication.run(Application.class, args); 5 } 6 }</pre>	

Let's create a Student Application where we can add/delete/view student data. We will use static variable to hold the data for our application. We will start by creating a **Student** class, then we will create a **StudentService** class and expose the methods there as REST service.

	Java
<pre>1 package com.adeveloperdiary;</pre>	

```

2
3 import java.util.Date;
4
5 public class Student {
6     private long id;
7     private String name;
8
9     public Student() {
10    }
11
12    public Student(String name, String subject) {
13        this.id = (new Date()).getTime();
14        this.name = name;
15        this.subject = subject;
16    }
17
18    private String subject;
19
20    public long getId() {
21        return id;
22    }
23
24    public String getName() {
25        return name;
26    }
27
28    public void setName(String name) {
29        this.name = name;
30    }
31
32    public String getSubject() {
33        return subject;
34    }
35
36    public void setSubject(String subject) {
37        this.subject = subject;
38    }
39
40    @Override
41    public String toString() {
42        return "Student{" +
43            "id=" + id +
44            ", name='" + name + '\'' +
45            ", subject='" + subject + '\'' +
46            '}';
47    }
48 }

```

We will create a static hash map to keep the student data. We will initialize that with some static values at the beginning. Let's add two new Students in our **hmStudent** map.

	Java
1	@SpringBootApplication
2	public class Application {
3	
4	public static HashMap<Long,Student> hmStudent;
5	
6	public static void main(String[] args) {
7	hmStudent=new HashMap<Long,Student>();
8	
9	Student one=new Student("John","math");
10	hmStudent.put(new Long(one.getId()),one);
11	
12	SpringApplication.run(Application.class, args);
13	
14	Student two=new Student("Jane","history");
15	hmStudent.put(new Long(two.getId()),two);
16	}
17	}

@RequestMapping

Create a `StudentService` class, annotate that using `@RestController` and `@RequestMapping`. Set the value for `@RequestMapping` to `/rest/student`, this will enforce to have all the services under `StudentService` class to have this as parent path. Next create a function named `getAllStudents()` and set the value for the `@RequestMapping` to `/`.

	Java
1	@RestController
2	@RequestMapping(value="/rest/student")
3	class StudentService{
4	
5	@RequestMapping(value="/",method = RequestMethod.GET)
6	public HashMap<Long,Student> getAllStudents(){
7	return Application.hmStudent;
8	}
9	}

Now start the application and access the

`http://localhost:8080/rest/student/` URL. You should be able to see the following JSON.

1	{
2	{
3	"id":1459658467056,
4	"name":"John",

```

5      "subject": "math"
6    }
7    ,
8    {
9      "id": 1459658467056,
10     "name": "Jane",
11     "subject": "history"
12   }
13 }

```

@RequestParam

We will try out different ways of sending data from the Browser to the Server. We will first use request parameter to send the data. Our URL would look like this.

`http://localhost:8080/rest/student/add?name=Joe&subject=english`

We are using `@RequestParam` annotation here, if you set a `defaultValue` it will also make the parameter optional. By default everything is mandatory. You can also set the `required=false` if needed.

	Java
1	<code>@RequestMapping(value="/add",method = RequestMethod.POST)</code>
2	<code>public Student addStudent(@RequestParam(value="name") String name</code>
3	<code>,@RequestParam(value="subject",defaultValue = "unknown") Str</code>
4	
5	<code>Student student=new Student(name,subject);</code>
6	<code>Application.hmStudent.put(new Long(student.getId()),student);</code>
7	<code>return student;</code>
8	
9	<code>}</code>

In order to test the POST service we created we would need a REST Client. I am using a plugin in Chrome itself, its called **Advanced REST Client**. Submit the URL using POST and you should see the below response.

```

1 {
2     "id": 1459661501207
3     "name": "Joe"
4     "subject": "english"
5 }

```

Now if you invoke `http://localhost:8080/rest/student/` you will get total 3 students.

@RequestBody

Now lets update the Students. This time we will pass the JSON Student object, Spring will use Jackson to map that to our Student class automatically. We need to use `@RequestBody` for this.

```
Java
1 @RequestMapping(value="/update",method = RequestMethod.PUT)
2 public Student updateStudent(@RequestBody Student student) throws
3
4     if(Application.hmStudent.containsKey(new Long(student.getId()))
5         Application.hmStudent.put(new Long(student.getId()),student
6     }else{
7         throw new Exception("Student "+student.getId()+" does not e
8     }
9
10    return student;
11 }
```

We need to set the **Content-Type** to `application/json` and send the JSON as the request body. See below :

The screenshot shows a REST client interface with the following details:

- URL: `http://localhost:8080/rest/student/update`
- Method: `PUT` (selected)
- Headers: `Content-Type: application/json`
- Payload: `{"id":1459663189412,"name":"New Name","subject":"New Subject"}`

We should get the following response back with the updated object.

```
1 {
2     "id": 1459663189412
3     "name": "New Name"
4     "subject": "New Subject"
```

```
5  
6 }
```

Notice, in case the id is not available our code would throw an exception. Spring will automatically convert that to JSON response. Let's use an invalid id as 1234. Our service will return the following response. Our custom error has been embedded in the error here.

```
1 {  
2     "timestamp": 1459663812487  
3     "status": 500  
4     "error": "Internal Server Error"  
5     "exception": "java.lang.Exception"  
6     "message": "Student 1234 does not exists"  
7     "path": "/rest/student/update"  
8 }
```

@PathVariable

Now its time to delete our Student data. This time we will use Path Variable to send the data as part of the URL itself. Spring provides `@PathVariable` annotation for this. The URL would look like:

`http://localhost:8080/rest/student/delete/1459664087939`

```
Java  
1 @RequestMapping(value="/delete/{id}",method = RequestMethod.DELETE)  
2 public Student deleteStudent(@PathVariable long id) throws Except  
3  
4     Student student;  
5  
6     if(Application.hmStudent.containsKey(new Long(id))){  
7         student=Application.hmStudent.get(new Long(id));  
8         Application.hmStudent.remove(new Long(id));  
9     }else{  
10         throw new Exception("Student "+id+" does not exists");  
11     }  
12     return student;  
13 }
```

Also lets add a `getStudent()` by id method.

<pre> 1 @RequestMapping(value="/{id}",method = RequestMethod.GET) 2 public Student getStudent(@PathVariable long id){ 3 return Application.hmStudent.get(new Long(id)); 4 } </pre>	Java
--	------

So we have used `@RequestParam`, `@RequestBody` and `@PathVariable` annotation to send the data from client to server. Use them as needed in your project.

Spring Boot Actuator

I want to quickly introduce **Actuator** here. Actuator is a very helpful library which provides runtime details of our application. To start with, we can find whether our App and all of its components (Like DB, Cache, MQ etc) are up or down. It also shows all the services available for an application. Additionally all properties, including server details, dump and many other details are easily available is your browser using Actuator.

In case you missed, here is the dependency details to be included in the pom.xml file.

	XHTML
<pre> 1 <dependency> 2 <groupId>org.springframework.boot</groupId> 3 <artifactId>spring-boot-actuator</artifactId> 4 <version>1.3.3.RELEASE</version> 5 </dependency> </pre>	

Application Health

We have already added actuator in out spring boot application, so just access <http://localhost:8080/health>

<pre> 1 { 2 "status": "UP", 3 "diskSpace": { 4 "status": "UP", 5 "total": 249779191808, </pre>
--


```

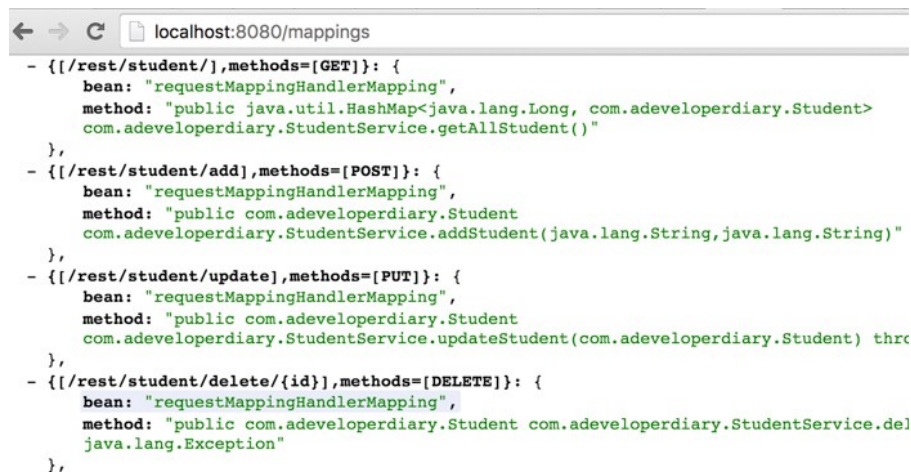
6      "free": 183181832192,
7      "threshold": 10485760
8    }
9  }

```

List of Services

Access the URL to get all the Request Mappings.

<http://localhost:8080/mappings>. This is very useful and when multiple teams working together it can really be very helpful.



```

- {[/rest/student/],methods=[GET]}: {
  bean: "requestMappingHandlerMapping",
  method: "public java.util.HashMap<java.lang.Long, com.adeveloperdiary.Student>
com.adeveloperdiary.StudentService.getAllStudent()"
},
- {[/rest/student/add],methods=[POST]}: {
  bean: "requestMappingHandlerMapping",
  method: "public com.adeveloperdiary.Student
com.adeveloperdiary.StudentService.addStudent(java.lang.String,java.lang.String)"
},
- {[/rest/student/update],methods=[PUT]}: {
  bean: "requestMappingHandlerMapping",
  method: "public com.adeveloperdiary.Student
com.adeveloperdiary.StudentService.updateStudent(com.adeveloperdiary.Student) thr
",
- {[/rest/student/delete/{id}],methods=[DELETE]}: {
  bean: "requestMappingHandlerMapping",
  method: "public com.adeveloperdiary.Student com.adeveloperdiary.StudentService.del
java.lang.Exception"
},

```

You can customize actuator to display or restrict information that should/shouldn't be available. For getting the full list, check this URL.

URL	Desc
env	Exposes properties from Spring's ConfigurableEnvironment.
beans	Displays a complete list of all the Spring beans in your application.
configprops	Displays a collated list of all <code>@ConfigurationProperties</code> .
dump	Performs a thread dump.
health	Shows application health information (when the application is secure, a simple 'status' when accessed over an

unauthenticated connection or full message details when authenticated).

logfile	Returns the contents of the logfile (if logging.file or logging.path properties have been set). Only available via MVC. Supports the use of the HTTP Range header to retrieve part of the log file's content.
metrics	Shows 'metrics' information for the current application.
mappings	Displays a collated list of all <code>@RequestMapping</code> paths.
trace	Displays trace information (by default the last few HTTP requests).

Consuming a RESTful Web Service

We may have to consume a RESTful service from our backend itself. So integration of REST Service from backend server is also very important and useful. Lets learn how Spring Boot can help there. We create another Spring Boot app to consume our StudentService.

In order to have both the server up and running we need to use another port other than 8080. Once you have created another Spring Boot App, open the `application.properties` and add the following line there.

```
server.port=8081
```

We will create a simple Service named, `SchoolService`. Before we add any method there, lets create our Student Class here so that the data can be mapped to the attributes automatically. Everything will be as is, however just

add one annotation to the top of the class. This will help us to ignore any unused variables.

	Java
1	@JsonIgnoreProperties(ignoreUnknown = true)
2	public class Student {
3	. . .
4	}

Now we will create a service named `getStudents()`. Spring Boot provides `RestTemplates` for consuming REST Services. The Student object will be automatically mapped. Here is a very simple basic implementation.

	Java
1	@RestController
2	@RequestMapping(value="/rest/school")
3	class SchoolService {
4	
5	@RequestMapping(value="/students/{id}")
6	public Student getStudent(@PathVariable String id){
7	
8	String URL="http://localhost:8080/rest/student/"+id;
9	
10	RestTemplate template=new RestTemplate();
11	
12	Student student =template.getForObject(URL,Student.class);
13	
14	return student;
15	}
16	}

Our URL would look like the following.

`http://localhost:8081/rest/school/students/1459668282416`

Here is the response.

1	{
2	id: 1459668282416,
3	name: "Jane",
4	subject: "history"
5	}

Consuming a RESTful Web Service in Java is a much larger topic than we can discuss here. I hope you have got the initial idea of it. I will have detailed post on this topic later.

Conclusion

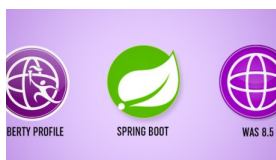
There are few topics I have not discussed here, like uploading/downloading a file using REST service. However this tutorial should get you started. Next we will learn how to deploy our service in IBM Application Servers.

This is the 3rd part of Getting Started with Spring Boot Series. Find all the parts here.

1. [An Introduction to Spring Boot](#)
2. [How to Create Spring Boot Application Step by Step](#)
3. [How to create RESTful Webservices using Spring Boot](#)
4. [How to deploy Spring Boot application in IBM Liberty and WAS 8.5](#)



Related



How to deploy Spring Boot application in IBM Liberty and WAS 8.5

April 4, 2016
In "Spring Boot"



How to Create Spring Boot Application Step by Step

April 4, 2016
In "Spring Boot"



An Introduction to Spring Boot

March 27, 2016
In "Spring Boot"

Filed Under: [Spring Boot](#) | Tagged With: [Code](#), [example](#), [Learn](#), [Microservices](#), [REST](#), [RESTful](#), [Spring Annotation](#), [Spring Boot](#), [WebService](#)

Comments



Ala says

April 18, 2016 at 1:50 am

Thanks a lot – you absolutely helped me to start with SpringBoot.



A Developer Diary says

April 18, 2016 at 1:54 am

You are most welcome Ala, glad to hear that this tutorial was helpful to you.

Thanks for your feedback !



Iron says

December 2, 2016 at 2:54 am

Where can I get complete reference of Spring Boot??,Since am new to it!!

Thank you



Sumit says

June 10, 2016 at 8:24 am

Thanks for providing so simple example for starting spring boot.



A Developer Diary says

June 10, 2016 at 4:52 pm

Hi Sumit,
Appreciate your feedback on this !

Thanks,
A Developer Diary



Anonymous says

August 31, 2016 at 4:18 pm

Hey A Developer Diary,
Great Job man!! Your approach of explaining things is good.
Correction- In Application health field, I guess the URL should be
"http://localhost:8080/health" instead of "http://localhost:8080/heath", I
guess. I was confused thinking it to be a new command.



adeveloperdiary says

October 18, 2016 at 7:35 pm

Appreciate your feedback... I have corrected the typo. Thank you very
much.



Pradipta Maitra says

September 27, 2016 at 8:57 am

Awesome A Developer Diary!

Any example of end-to-end examples using JavaScript/SpringBoot (with JPA)/Mongo (or any DB) will be of great help.



adeveloperdiary says

October 18, 2016 at 7:36 pm

Hi Pradipta,

Thank you for your feedback !

Sure, I will have end-to-end example posted soon.

Thanks,

A Developer Diary



Yolpo says

October 20, 2016 at 7:17 am

Hi,

Thanks for all, very usefull post. I can't wait for the end-to-end examples :).

Best regards



Siva Prakash says



October 18, 2016 at 4:59 pm

when i specify server.port=8081 port in application.properties file my application is not started. if i change to 8083 it is started but it says

Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request processing failed; nested exception is org.springframework.web.client.ResourceAccessException: I/O error on GET request for "http://localhost:8080/rest/student/1476824007402": Connection refused; nested exception is java.net.ConnectException: Connection refused] with root cause

java.net.ConnectException: Connection refused



adeveloperdiary says

October 18, 2016 at 7:39 pm

Hi Siva,

If the server.port is defined as 8083 then you need to connect to 8083 and not 8080, unless you have a different service in 8080.

Let me know if this helps.

Thanks,

A Developer Diary



artegento says

November 6, 2016 at 4:08 am

very good article, i find out that, in order to use hasMap, you need to redefine hasCode, other way, in my case, at least, i was seeing just only record.



Samer Najjar says

November 23, 2016 at 9:12 am

Thanks alot



Usman Haider says

December 8, 2016 at 10:15 am

Great tutorial for a start to SpringBoot



Matt says

January 25, 2017 at 2:14 pm

This tutorial is a jem, thank you sir!



Yu says

February 4, 2017 at 8:46 pm

Thanks for your material, this is really helpful for a guy who needs to study spring boot. Is it possible to get some tutorial about how to use JDBC in spring boot?Appreciated.



arsa says

February 7, 2017 at 7:11 am

Hi Thanks a lot for working example.

Anyway my JSON output has additional element ID as below.

```
{
  1486465076953: {
    "id": 1486465076953,
    "name": "Jane",
    "subject": "history"
  }
}
```

How to hide it so I could have output as yours.

```
{
  "id": 1486465076953,
  "name": "Jane",
  "subject": "history"
}
```



Gireesh Bhat says

February 16, 2017 at 8:22 am

No worries, its just a HashMap data, first ID is the key of HashMap..

Anyways a very good example for newbiess(like me)..

Thank you 😊



Zubin says

February 26, 2017 at 3:58 pm

Thanks for the article. It helped me a lot.
Just to add one point for usage of Actuator.
We need to add below line in application.properties file.
management.security.enabled=false



vivek says

March 11, 2017 at 11:16 pm

Wow!!! Excellent tutorial!!! Great job!! Gave me a headstart to SpringBoot



Midza says

April 30, 2017 at 4:38 pm

Thanx a lot



solomon says

May 17, 2017 at 8:54 am

i was glad when start the tutorial but get stacked when "404 page not found" to i try the code on browser



Raj says

June 18, 2017 at 6:34 pm

Awesome !! Great job. Implementing this was a breeze

Top Posts

[How to Create Spring Boot Application Step by Step](#)

134,062 views | 9 comments

[How to integrate React and D3 – The right way](#)

44,626 views | 30 comments

[How to create RESTFul Webservices using Spring Boot](#)

43,800 views | 24 comments

[How to convert XML to JSON in Java](#)

25,992 views | 4 comments

[Get started with jBPM KIE and Drools Workbench – Part 1](#)

25,322 views | 14 comments

[How to deploy Spring Boot application in IBM Liberty and WAS 8.5](#)

25,123 views | 8 comments

An Introduction to Spring Boot

24,113 views | 9 comments

How to Create Stacked Bar Chart using d3.js

19,723 views | 15 comments

Create a simple Donut Chart using D3.js

16,749 views | 5 comments

How to easily encrypt and decrypt text in Java

13,738 views | 8 comments

Search in this website