Spring MVC @RequestMapping Annotation Example with Controller, Methods, Headers, Params, @RequestParam, @PathVariable

APRIL 2, 2018 BY PANKAJ — 20 COMMENTS

@RequestMapping is one of the most widely used **Spring MVC** annotation. `org.springframework.web.bind.annotation.RequestMapping` annotation is used to map web requests onto specific handler classes and/or handler methods.

`@RequestMapping` can be applied to the controller class as well as methods. Today we will look into various usage of this annotation with example and other annotations `@PathVariable` and `@RequestParam`.

Table of Contents [hide]

Spring @RequestMapping

1. **@RequestMapping with Class**: We can use it with class definition to create the base URI. For example:

```
@Controller
@RequestMapping("/home")
public class HomeController {

}
```

Now /home is the URI for which this controller will be used. This concept is very similar to servlet context of a web application.

2. **@RequestMapping with Method**: We can use it with method to provide the URI pattern for which handler method will be used. For example:

```
@RequestMapping(value="/method0")
@ResponseBody
public String method0(){
        return "method0";
}
```

Above annotation can also be written as `@RequestMapping("/method0")`. On a side note, I am using @ResponseBody to send the String response for this web request, this is done to keep the example simple. Like I always do, I will use these methods in Spring MVC application and test them with a simple program or script.

3. **@RequestMapping with Multiple URI**: We can use a single method for handling multiple URIs, for example:

```
@RequestMapping(value={"/method1","/method1/second"})
@ResponseBody
```

```
public String method1(){
        return "method1";
}
```

If you will look at the source code of **RequestMapping annotation**, you will see that all of it's variables are arrays. We can create String array for the URI mappings for the handler method.

4. **@RequestMapping with HTTP Method**: Sometimes we want to perform different operations based on the HTTP method used, even though request URI remains same. We can use @RequestMapping method variable to narrow down the HTTP methods for which this method will be invoked. For example:

```
@RequestMapping(value="/method2", method=RequestMethod.POST)
@ResponseBody
public String method2(){
        return "method2";
}

@RequestMapping(value="/method3", method={RequestMethod.POST,RequestMethod.GET})
@ResponseBody
public String method3(){
        return "method3";
}
```

5. **@RequestMapping with Headers**: We can specify the headers that should be present to invoke the handler method. For example:

```
@RequestMapping(value="/method4", headers="name=pankaj")
@ResponseBody
public String method4(){
        return "method4";
}

@RequestMapping(value="/method5", headers={"name=pankaj", "id=1"})
@ResponseBody
public String method5(){
        return "method5";
}
```

6. **@RequestMapping with Produces and Consumes**: We can use header `Content-Type` and `Accept` to find out request contents and what is the mime message it wants in response. For clarity,

@RequestMapping provides **produces** and **consumes** variables where we can specify the request content-type for which method will be invoked and the response content type. For example:

```
@RequestMapping(value="/method6", produces={"application/json","application/xml"},
consumes="text/html")
@ResponseBody
public String method6(){
        return "method6";
}
```

Above method can consume message only with **Content-Type as text/html** and is able to produce messages of type **application/json** and **application/xml**.

Spring @PathVariable

7. **@RequestMapping with @PathVariable**: RequestMapping annotation can be used to handle dynamic URIs where one or more of the URI value works as a parameter. We can even specify Regular Expression for URI dynamic parameter to accept only specific type of input. It works with **@PathVariable annotation** through which we can map the URI variable to one of the method arguments. For example:

```
@RequestMapping(value="/method7/{id}")
@ResponseBody
public String method7(@PathVariable("id") int id){
        return "method7 with id="+id;
}

@RequestMapping(value="/method8/{id:[\\d]+}/{name}")
@ResponseBody
public String method8(@PathVariable("id") long id, @PathVariable("name") String
name){
        return "method8 with id= "+id+" and name="+name;
}
```

Spring @RequestParam

8. **@RequestMapping with @RequestParam for URL parameters**: Sometimes we get parameters in the request URL, mostly in GET requests. We can use @RequestMapping with **@RequestParam annotation** to retrieve the URL parameter and map it to the method argument. For example:

```
@RequestMapping(value="/method9")
@ResponseBody
```

```
public String method9(@RequestParam("id") int id){
        return "method9 with id= "+id;
}
```

For this method to work, the parameter name should be "id" and it should be of type int.

9. **@RequestMapping default method**: If value is empty for a method, it works as default method for the controller class. For example:

```
@RequestMapping()
@ResponseBody
public String defaultMethod(){
        return "default method";
}
```

As you have seen above that we have mapped `/home` to `HomeController`, this method will be used for the default URI requests.

10. **@RequestMapping fallback method**: We can create a fallback method for the controller class to make sure we are catching all the client requests even though there are no matching handler methods. It is useful in sending custom 404 response pages to users when there are no handler methods for the request.

```
@RequestMapping("*")
@ResponseBody
public String fallbackMethod(){
        return "fallback method";
}
```

Spring RestMapping Test Program

We can use Spring RestTemplate to test the different methods above, but today I will use cURL commands to test these methods because these are simple and there are not much data flowing around.

I have created a simple shell script springTest.sh to invoke all the above methods and print their output. It looks like below.

```
#!/bin/bash

echo "curl http://localhost:9090/SpringRequestMappingExample/home/method0";
curl http://localhost:9090/SpringRequestMappingExample/home/method0;
```

```
printf "\n\n*****\n\n";

echo "curl http://localhost:9090/SpringRequestMappingExample/home";
curl http://localhost:9090/SpringRequestMappingExample/home;
printf "\n\n*****\n\n";

echo "curl http://localhost:9090/SpringRequestMappingExample/home/xyz";
curl http://localhost:9090/SpringRequestMappingExample/home/xyz;
printf "\n\n*****\n\n";

echo "curl http://localhost:9090/SpringRequestMappingExample/home/method1";
curl http://localhost:9090/SpringRequestMappingExample/home/method1;
printf "\n\n*****\n\n";

echo "curl http://localhost:9090/SpringRequestMappingExample/home/method1/second";
curl http://localhost:9090/SpringRequestMappingExample/home/method1/second;
printf "\n\n*****\n\n";
```

Note that I have deployed my web application on Tomcat-7 and it's running on port 9090. **SpringRequestMappingExample** is the servlet context of the application. Now when I execute this script through command line, I get following output.

```
pankaj:~ pankaj$ ./springTest.sh
curl http://localhost:9090/SpringRequestMappingExample/home/method0
method0

*****

curl http://localhost:9090/SpringRequestMappingExample/home
default method

*****

curl http://localhost:9090/SpringRequestMappingExample/home/xyz
fallback method

*****

curl http://localhost:9090/SpringRequestMappingExample/home/method1
method1

*****
```

Most of these are self understood, although you might want to check default and fallback methods. That's all for **Spring RequestMapping Example**, I hope it will help you in understanding this annotation and it's various features. You should download the sample project from below link and try different scenarios to explore it further.

Download Spring MVC RequestMapping Project

« PREVIOUS
Spring Tutorial – Spring Core Framework Tutorials

NEXT »
Spring Hibernate Integration Example Tutorial (Spring 4 + Hibernate 3 and Hibernate 4)

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: SPRING

Comments

arun singh says
JUNE 14, 2018 AT 11:20 PM
can some please help me answering
how is a url template with integer value generated: “http://localhost:8080/springmvc/hello/101”
It is about @PathVariable.
How to pass integer value along with url in the above form? How does this happen dynamically?
Thanks
Reply

Sheela says
APRIL 24, 2018 AT 5:40 AM
How to make the requestParam key as case insensitive?
For Example : public String welcome(@RequestParam(“orgID”) String orgID, ModelMap model)
The request parameter “orgID” should be case insensitive. How to do this ?.
I should be able to give the query-string as “orgid” or “orgId”. The parameter should be completely case-insensitive.
Reply

arun singh says
APRIL 21, 2018 AT 5:25 PM
thanks
Reply

Prakash says
MARCH 4, 2018 AT 7:01 AM
Thanking you
Reply

NAVEENSNH says
JANUARY 10, 2018 AT 12:23 AM
Thank you sir very nice explanation on each annotations
Reply

Luiz Couto says
SEPTEMBER 7, 2017 AT 5:05 AM
Very simple and well explained, thanks !
Reply

Rohit Singh says
SEPTEMBER 6, 2017 AT 11:07 AM
Good post Pankaj. This was very crisp and clear □
Reply

Fin says
OCTOBER 31, 2016 AT 2:55 AM
I got an problem to get url param like this:
`http://mydomain.com&sort[customer_name]=asc`
I want to sort param by :
customer_name = field param will be sorting
asc = ascending sorting
@RequestMapping(value="/listsort",method = RequestMethod.GET)
@ResponseBody
public String listAll(@RequestParam(value="sort[customer_name]", required=false) String sort){
return "Test with sort field = "+sort;
}
Reply

Rasu says
AUGUST 17, 2016 AT 5:36 AM
This is my fallback method,
@RequestMapping("*")
@ResponseBody
public String fallBackMethod()

{
return “Error404”;
}
here, Error404 is my JSP page. if i gave any wrong URI it displays Error404 msg only, but it did’t display my Error404 JSP page,
Now i want to display my Error404 JSP page if i gave any wrong URI
Reply

Pankaj says
JUNE 29, 2017 AT 2:57 AM
Here in this case you have to define model if you want to send this msg to pass to Jsp page and use any of resource view handler for ex . InternalResourceViewResolver
/WEB-INF/pages/
.jsp
Reply

vasanth says
JUNE 13, 2016 AT 12:18 AM
Hello Pankaj… Good snippet to understand the concepts. Thanks.
Reply

Shiva kumar says
APRIL 17, 2016 AT 7:13 AM
in method level request mapping that method return some string like method1, method2, where we need to configure relevant jsp to forward? In my case I want to redirect to customers list and my method is like
@RequestMapping(value = “customers”, method = RequestMethod.GET)
public String getCustomers(Model model, HttpServletRequest request) {
……..
……..
return “customers”
}
is it enough to have customers.jsp or need to configure this “/customers” to customers.jsp
Reply

Rahul says
JANUARY 31, 2016 AT 11:23 PM
Thank you so much, it helped clearing my doubts.
Reply

madhusmita says
DECEMBER 23, 2015 AT 3:45 AM
very nice.thank u.
Reply

Vijay G says
AUGUST 22, 2015 AT 9:52 PM
I have to use dynamic URI but it fails for me. Assume the method value in controller is given below,
Controller:
@RequestMapping(value="/updateSpecifiCertificate/{id}")
@ResponseBody
public String method7(@PathVariable("id") int id){
return "method7 with id="+id;
}
jsp:
/updateSpecifiCertificate/${conCertBean.certificateId}
when i click the above link, it binds the dynamic id with the url and throws 404 as below,
The requested resource (/updateSpecifiCertificate/CERT22) is not available.
How to call my controller in this case, Please help me i have tried so many ways.....
Thank You So Much...
Vijay
Reply

Mak says
OCTOBER 28, 2015 AT 5:15 AM
You declare id parameter as int.
Reply

madhu says
MARCH 13, 2015 AT 2:10 AM

Hi,
How to handle the multiple actions in single jsp using springMVC like
……
……
……
……
……
……
……
……
Thanks,
Madhu R.
Reply

Rajesh says
JANUARY 23, 2015 AT 5:27 AM
while passing json code through restfull service by using soap ui tool.getting 415 error
1)json:
[{
“AuthToken” : “token”,
“empName”: “Name”,
“empNo”: 111,
“locName”: “kolkata”,
“locNumber”: 713156
}]
2)bean class:
private String AuthToken ;
private String empName;
private String name;
private String empNo;
private String locName ;
private String locNumber;
public void setAuthToken(String AuthToken) {
this.AuthToken = AuthToken;
}
public String getAuthToken() {
return AuthToken;
}
public void setEmpName(String empName) {
this.empName = empName;
}

```
public String getEmpName() {
return empName;
}
public void setName(String name) {
this.name = name;
}
public String getName() {
return name;
}
public void setEmpNo(String empNo) {
this.empNo = empNo;
}
public String getEmpNo() {
return empNo;
}
public void setLocName(String locName) {
this.locName = locName;
}
public String getLocName() {
return locName;
}
public void setLocNumber(String locNumber) {
this.locNumber = locNumber;
}
public String getLocNumber() {
return locNumber;
}
}
```

3)controller:

```
@RequestMapping(value="/upload", method={RequestMethod.POST},consumes={"application/json"
},produces={"application/xml" })
@ResponseBody
public NurseInfo uploadNote(@Validated @RequestBody NurseInfo nurseInfo) throws Exception {
// TODO Auto-generated method stub
System.out.println(nurseInfo);
System.out.println(nurseInfo.getEmpNo());
String json=""+nurseInfo.getEmpName()+"";
System.out.println(json);
return nurseInfo;
}
```

4)Error:

Apache Tomcat/7.0.16 – Error report

```
<!–H1 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:22px;} H2 {font-
family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:16px;} H3 {font-
```

family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:14px;} BODY {font-family:Tahoma,Arial,sans-serif;color:black;background-color:white;} B {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;} P {font-family:Tahoma,Arial,sans-serif;background:white;color:black;font-size:12px;}A {color : black;}A.name {color : black;}HR {color : #525D76;}–>

HTTP Status 415 –

type

Status report

message

description

The server refused this request because the request entity is in a format not supported by the requested resource for the requested method ().

Apache Tomcat/7.0.16

Thanks in advancs

Reply

divyesh shani says

JANUARY 13, 2015 AT 10:48 AM

It works fine but what if multiple json array is there (array inside array and again inside array)??
how can i create this type of web services with Spring RestTemplate??

Reply

Ram says

NOVEMBER 13, 2014 AT 10:06 PM

Nice one..

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

SPRING FRAMEWORK

Spring Tutorial

Spring Core

- Spring Framework
- Spring Dependency Injection
- Spring IoC and Bean
- Spring Bean Life Cycle
- Spring REST
- Spring REST XML
- Spring RestTemplate
- Spring AOP
- Spring AOP Method Profiling
- Spring Annotations
- Spring @Autowired
- Spring @RequestMapping

Spring MVC

- Spring MVC Example

- Spring Primefaces Hibernate
- SpringRoo Primefaces Hibernate
- Spring JSF
- Spring JDF Hibernate

Miscellaneous

- Spring Data MongoDB
- Spring Interview Questions

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial