Theory: Rest controller

33 minutes 0 / 5 problems solved

§1. @RestController

Basically, in web-based applications, a client communicates with the server using its API, i.e. various methods available from outside via HTTP (HyperText Transfer Protocol) requests. A **controller** is a part of the application that handles these API methods.

The `@RestController` annotation is usually on top of the class. It declares that a class will provide the exact endpoints (URLs you request for) to access the REST methods. The class itself and its methods can tell which requests are suitable. All the appropriate requests will be sent to the concrete method of this class.

Now we'll explore how to work with the two most popular methods: `POST` and `GET` as well as corresponding annotations, `@PostMapping` and `@GetMapping`.

§2. @PostMapping

You can place this one before the method of a `@RestController` class. The method will provide the `POST` request handling. With this REST method, we can send some information to the request body. Usually, the method saves this information on the server for future use.

We need to tell the controller what type of information this will be and how we can access it in a handling method. As it will be passed in a body of the request, we can use the `@RequestBody` annotation before the parameters of the method, like this:

```
1 @PostMapping
2 public String addLine(@RequestBody String line)
```

We will use the basic project. If you don't have such an application, just visit the Spring Initializr site and generate it with Gradle and Java.

Let's consider a server that represents a personal to-do list of tasks. Each task in the list has a unique id, name, and description with the "done/completed" flag. As a trivial example, it can be presented in a table like this:

N	done	name
1	+	Dentist
		Visit dr Anderson
2		Dance class
		Remember to pick up Steven to class
3		

TODO list example

First, let's declare the `Task` class:

Current topic:
Rest controller

Topic depends on:
HTTP messages
JSON
Stage 6
Instance methods
Annotations
ArrayList
Basic project structure

Topic is required for:
Passing JSON to server
Exception handling
Bean Validation

Table of contents:
↑ Rest controller
§1. @RestController
§2. @PostMapping
§3. What is inside the @RestController
§4. @GetMapping
§5. Conclusion
Feedback & Comments

```
package com.hyperskill.wardrobe;

public class Task {

    private int id;
    private String name;
    private String description;
    private boolean completed;


    public Task() {

    }



public Task(int id, String name, String description, boolean completed) {

        this.id = id;

        this.name = name;

        this.description = description;

        this.completed = completed;

    }



    // getters and setters

}
```

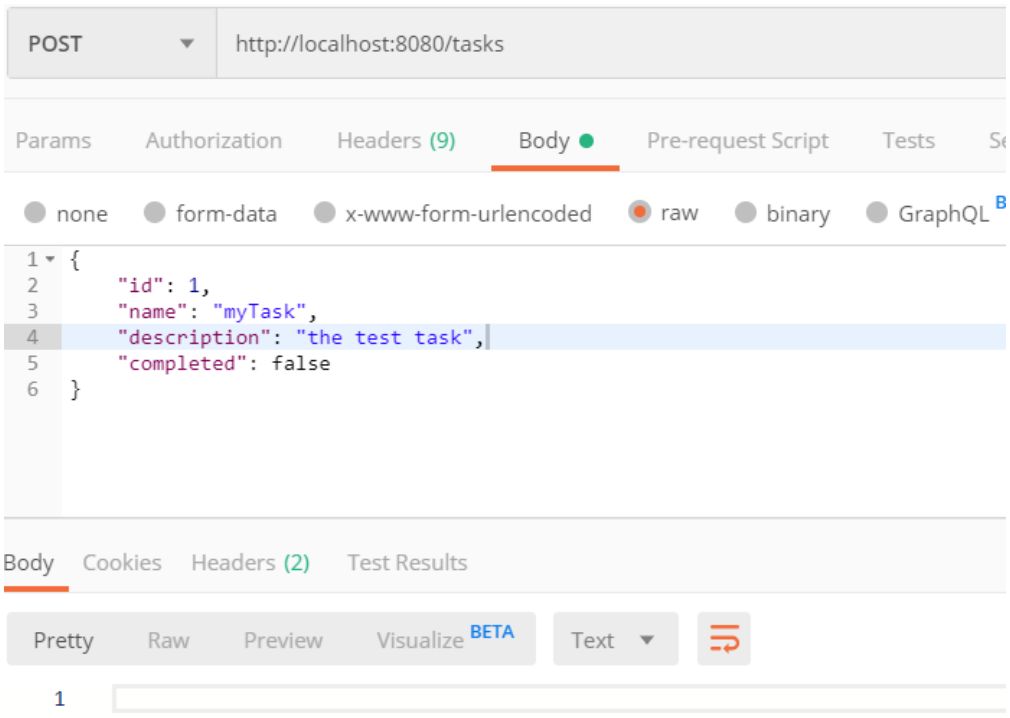
Now we can write a method for adding a task to the list.

> Note that we are using `ArrayList` here to save information on the side of the server. It is not a common way to do it because usually, web-application keeps the information in a database. We will use `ArrayList` here to simplify the process of trying the `@RestController`'s in action.

Then we will pass the information about a task in a request body and save the object on a server:

```
@RestController
public class TaskController {

    private List<Task> tasks = new ArrayList<>();

    public TaskController() {
    }

    @PostMapping(path = "/tasks")

    public void addTask(@RequestBody Task task){

        tasks.add(task);

    }

}
```

Now let’s try it out in any rest client:



Trying the POST method

§3. What is inside the @RestController

Now let's look at the `@RestController` itself. This annotation is a wrapper, so it consists of two annotations combining their functions:

- `@Controller`: the class will contain the handler methods;
- `@ResponseBody`: the response object of each of the handler's methods will be represented in a JSON format.

The following declarations would be equal:

```
@RestController
public class TaskController{}


@Controller
@ResponseBody
public class TaskController{}
```

In the next section, we will discover how we can use these inner annotations.

§4. @GetMapping

The second basic function of any server is providing users with saved information, usually as a response to their `GET` requests. Continuing the to-do list story, the `GET` method would return the information about a certain task.

There are **few notable points about the `GET` handling methods**, despite the heading annotations.

1. We cannot pass a body with a `GET` request.
2. Here comes the functionality of a `@ResponseBody` inside the `@RestController`: you can return any type from the method, and Spring will try to represent it in a JSON format (or it will inform you if it can't find a way to do it).
3. When we want to get a description of the task, there must be a way to specify what task exactly we are interested in (e.g., by its ID). The most common way to do that is adding a `@PathVariable` to a request path. The method, handling the `GET` request, would have access to the passed ID and would be able to understand which task to return.

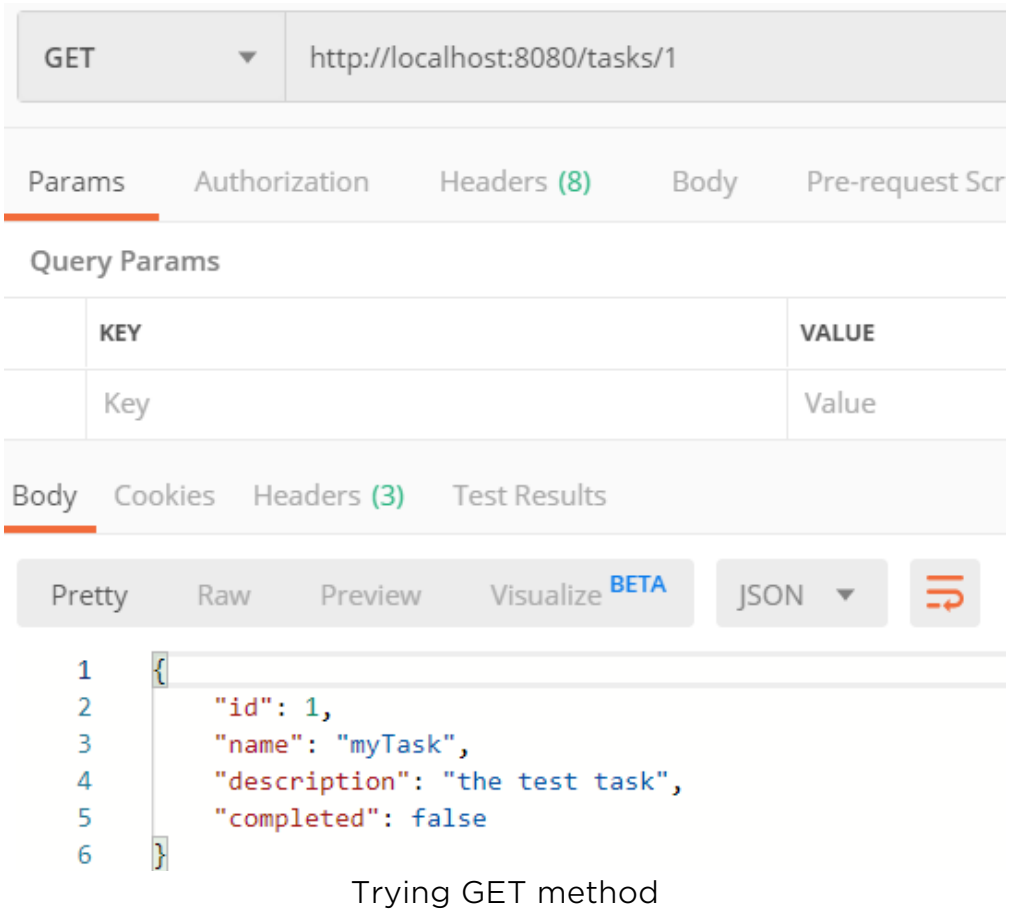
```
@GetMapping("/lines/{id}")
public String getLine(@PathVariable String id)
```

Note that you can use `@PathVariable` in both `GET` and `POST` methods.

Explore one of the ways of providing the `GET` method handler in a snippet below:

```
@RestController
public class TaskController {

    private List<Task> tasks = new ArrayList<>();

    public TaskController() {
    }

    @GetMapping(path = "/tasks/{id}")

    public Task getTask(@PathVariable int id){

        return tasks.get(id - 1);

    }

}
```

Now we can put it all together and test the `GET` method for the first task we added before:



Trying GET method

§5. Conclusion

A controller is one of the first parts that a request meets in the web application. You have learned how to define the main two methods in a `@RestController` annotated class. Now you can use `@PostMapping` annotated methods to send information to a server and `@GetMapping` methods to get it. Explore the picture to revise what parts of the client-server communication are controlled by the mentioned annotations.

POST and GET processes annotations

On the one hand, web-app developers need to keep the handlers short and clear: it helps to find the right handler and create a correct request quickly.

On the other hand, almost all of the web apps are clients for other web apps. It means that they will call controllers of other applications. That's why you will also need to be able to read foreign handlers easily and figure out what request it can handle.

Report a typo

119 users liked this theory. **34** didn't like it. **What about you?**

Start practicing

Comments (21) Hints (6) Useful links (5) Show discussion