Theory: Passing JSON to server

13 minutes 0 / 5 problems solved

Skip this topic Start practicing

1160 users solved this topic. Latest completion was about 3 hours ago.

§1. Sending information to a server

When we want to pass some information to any web-application server, we most likely would use the `POST` methods. Although we can send any information of any type, even an unformatted raw string, JSON is one of the easiest formats to create and to read for both humans and servers.

Let's see what we need to change in `@RestController` to accept JSONs to methods.

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.

§2. Passing a primitive JSON

Imagine you're creating a controller for a bot. The first command we need to implement is a greeting method for a new user. The method should accept some user information and return a personal "Hello" back. We will provide a username in a following simple JSON:

```
{
 "id" : "1234",
 "name" : "Unicorn",
 "enabled" : false,
 "phone" : "987654321"
}
```

This object can be represented with the following class:

```
public class UserInfo {

    private int id;
    private String name;
    private String phone;
    private boolean enabled;

    // getters and setters


    UserInfo() {}

}
```

Now we can pass a `UserInfo` object to the `BotController`:

```
@RestController
public class BotController {

    @PostMapping(value = "/greet", consumes = "application/json")
    public String greet(@RequestBody UserInfo userInfo) {
        if (userInfo.isEnabled()) {

           return String.format("Hello! Nice to see you, %s!", userInfo.getName()
);

        } else return String.format("Hello! Nice to see you, %s! Your account is d
isabled", userInfo.getName());
    }

}
```

Which important things are new here?

- "consumes =" parameter of `@PostMapping` annotation shows the acceptable type of the method; for JSONs, the value **application/json** is used;

Current topic:

Passing JSON to server

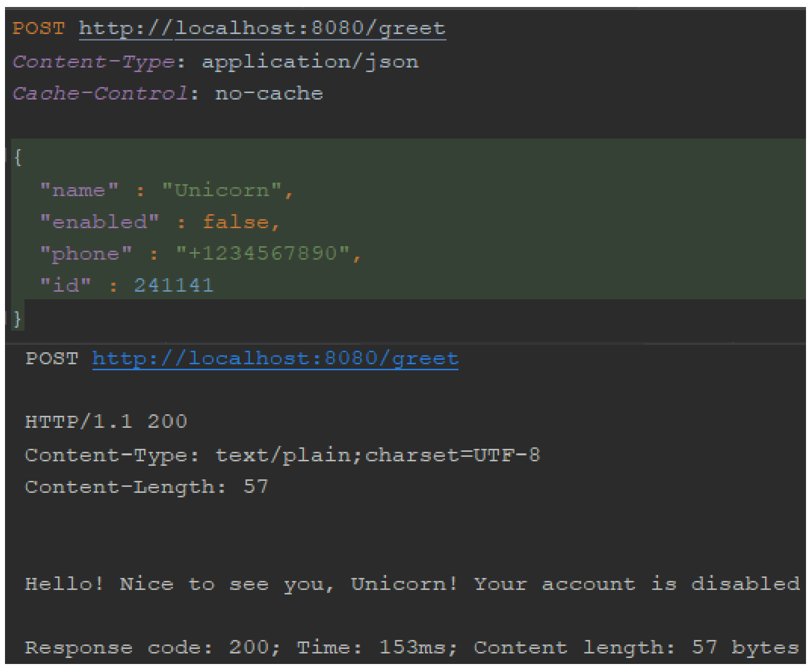
Topic depends on:

✕ Rest controller

Table of contents:

- `@RequestBody` annotation before the handling method argument means that a method accepts a body;
- The class representing a passing object must have setters for its fields and the names of the fields should be the same as they are in a JSON.

Now let's test the method :



```
POST http://localhost:8080/greet
Content-Type: application/json
Cache-Control: no-cache

{
  "name" : "Unicorn",
  "enabled" : false,
  "phone" : "+1234567890",
  "id" : 241141
}
```

```
POST http://localhost:8080/greet

HTTP/1.1 200
Content-Type: text/plain;charset=UTF-8
Content-Length: 57


Hello! Nice to see you, Unicorn! Your account is disabled

Response code: 200; Time: 153ms; Content length: 57 bytes
```

Correct JSON request and answer

You can see that our Unicorn user successfully got its notification about the disabled account.

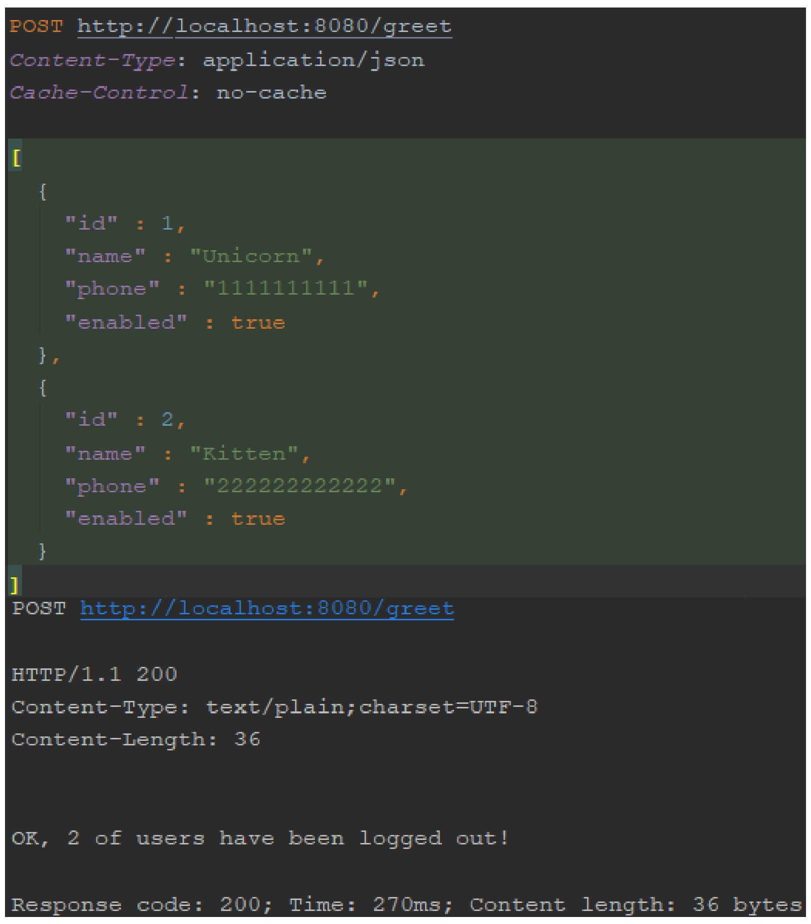
§3. Passing an array

JSON can also be an array of objects. To accept this kind of body, we need to use a Collection as a handling method parameter, e.g., an `ArrayList`. Let's consider a method that logs out all of the given users from the list:

```
public class BotController {

    @PostMapping(value = "/greet", consumes = "application/json")
    public String greet(@RequestBody List<UserInfo> userInfoList) {
        //logging out users
        return String.format("OK, %d of users have been logged out!", userInfoList.size());
    }
}
```

Then see the result of passing an array of `UserInfo` objects to this method:



```
POST http://localhost:8080/greet
Content-Type: application/json
Cache-Control: no-cache

[
  {
    "id" : 1,
    "name" : "Unicorn",
    "phone" : "1111111111",
    "enabled" : true
  },
  {
    "id" : 2,
    "name" : "Kitten",
    "phone" : "222222222222",
    "enabled" : true
  }
]
```

```
POST http://localhost:8080/greet

HTTP/1.1 200
Content-Type: text/plain;charset=UTF-8
Content-Length: 36


OK, 2 of users have been logged out!

Response code: 200; Time: 270ms; Content length: 36 bytes
```

Passing a JSON array result

The list was passed correctly and we got the expected message.

§4. In conclusion

So, there are few steps to make a `@RestController` able to accept JSON as a body:

1. add `consumes = "application/json"` to `@PostMapping` annotation to specify JSON as an acceptable format;
2. create a class to map a body to an object, **remember to add setters;**
3. add `@RequestBody` in front of the handler method's argument.

After these steps, your method will be able to read a passed JSON as a usual object.

Report a typo

91 users liked this theory. **4** didn't like it. **What about you?**

Start practicing

Comments (10) Hints (0) Useful links (1) Show discussion