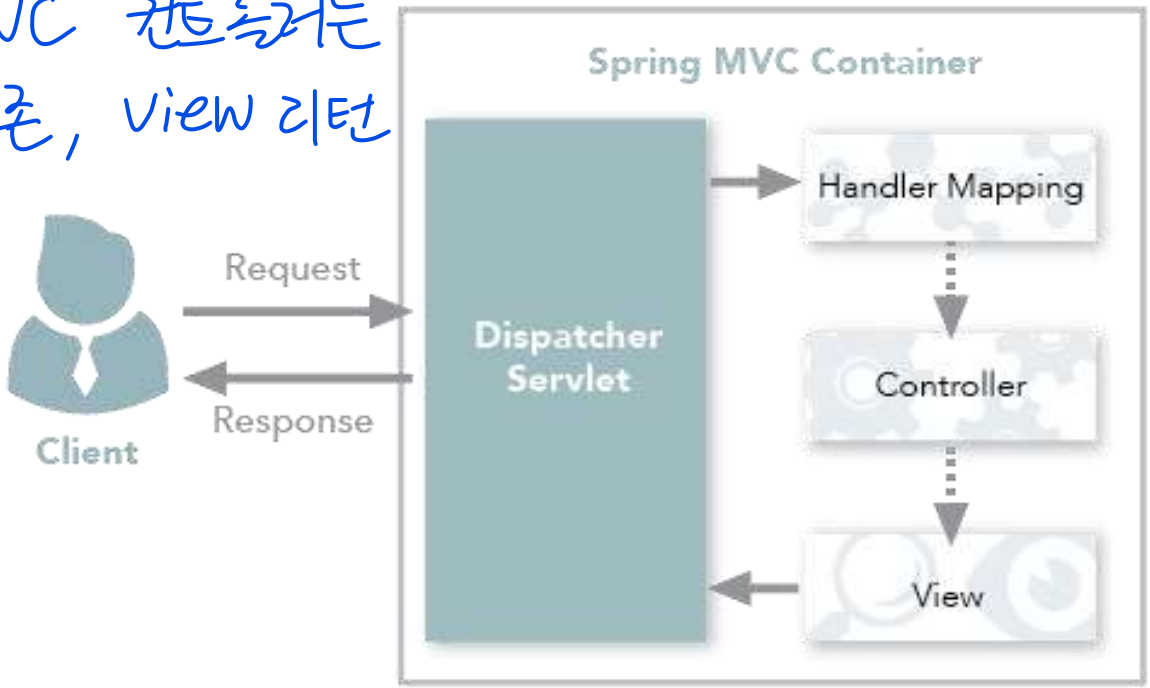
Restful Web Service with Spring

1. Spring MVC Traditional Workflow

The traditional MVC controller relies on the <u>View</u> technology

전통적인 MVC 컨트롤러는
view에 의존, view 리턴



2. RESTful Web Service Controller

반면에 이 컨트롤러는 객체(자바)를 리턴

- Controller simply returns the object
- The object data is written directly to the HTTP response as JSON/XML

body에 JSON/XML 포맷으로 작성됨, 작성본은 클라이언트로 보내짐

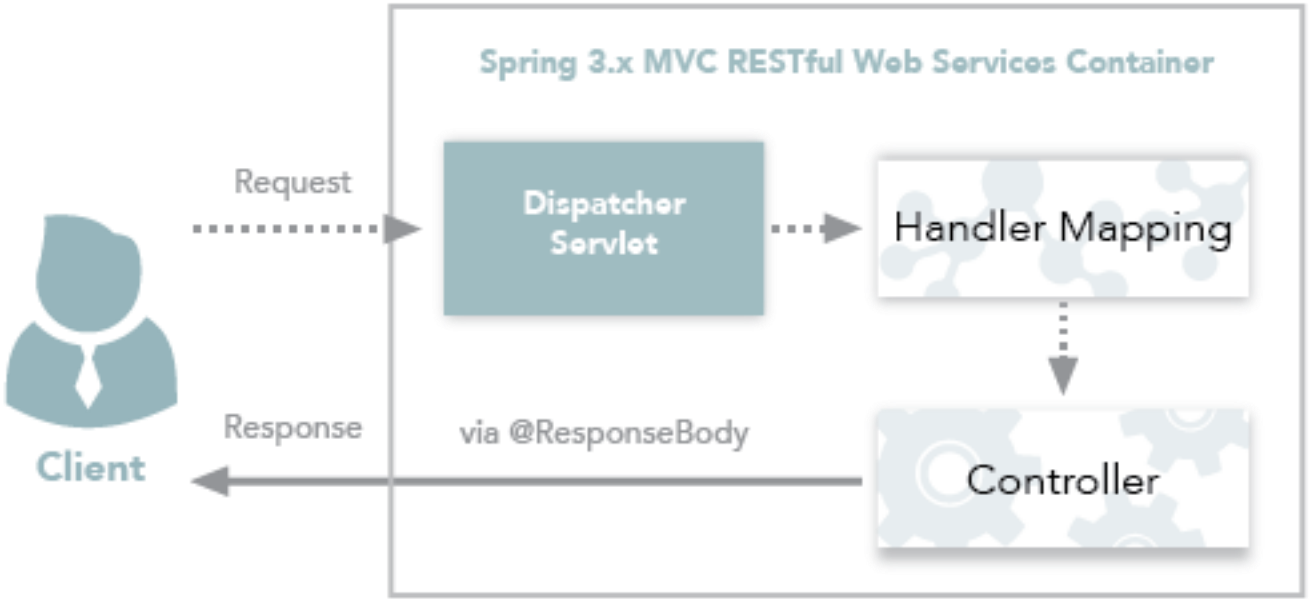
- @RestController , @RequestBody, ResponseEntity & @PathVariable are all you need to know to implement a REST API in Spring

객체를 줄세워서 보낸다 (sevialigation)

1) @ResponseBody Annotation

객체를 body에 넣어줌

- Each method in the Controller class must be annotated with @ResponseBody



@ResponseBody Annotation

```
@Controller
@RequestMapping("/api")
public class RestApiController {

    @Autowired
    UserService userService;

    @GetMapping("/users")
    @ResponseBody
    public List<User> listAllUsers() {
        return userService.getAllUsers();
    }
}
```

@ResponseBody Annotation

- If a method is annotated with @ResponseBody, Spring will bind the return value to outgoing HTTP response body

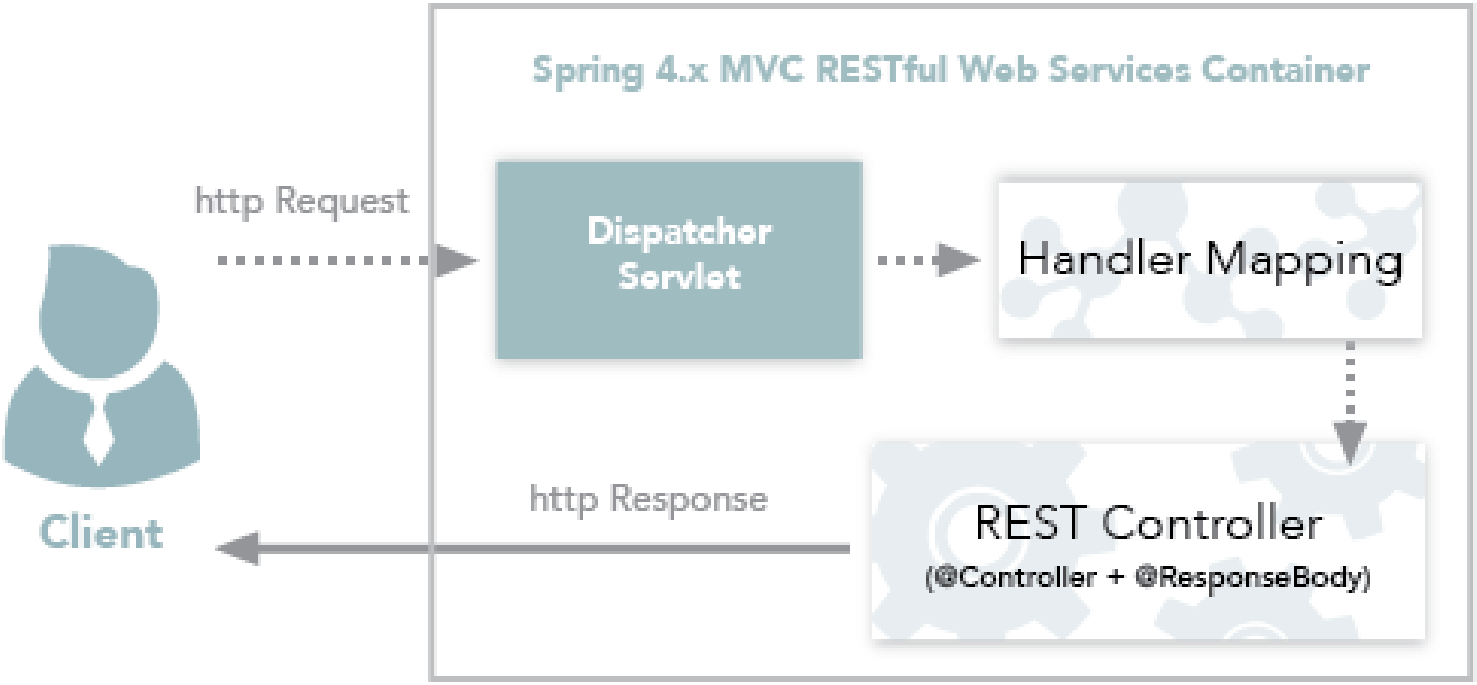
- While doing that, Spring will [behind the scenes] use HTTP Message converters to convert the <u>return value</u> to HTTP <u>response body</u> [serialize the object to response body]

2) @RestController Annotation

@Controller 와 @ResponseBody 의 합

- Spring 4.0 introduced @RestController
  - a specialized version of the controller which is a convenience annotation that does nothing more than add the @Controller and @ResponseBody annotations ∴ 앞으로는 @Controller 대신 쓸 수 있음
  - By annotating the controller class with @RestController annotation, you no longer need to add @ResponseBody to all the request mapping methods

@RestController Annotation

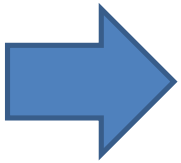


@RestController Annotation

```
@Controller
@RequestMapping("/api")
public class RestApiController {

    @Autowired
    UserService userService;

    @GetMapping("/users")
    @ResponseBody
    public List<User> listAllUsers() {
        return userService.getAllUsers();
    }
}
```



```
@RestController
@RequestMapping("/api")
public class RestApiController {

    @Autowired
    UserService userService;

    @GetMapping("/users")
    @ResponseBody
    public List<User> listAllUsers() {
        return userService.getAllUsers();
    }
}
```

3) @PathVariable Annotation

- This annotation indicates that a method parameter should be bound to a URI template variable [the one in '{}']

GET /api/users/{id}

URI template variable

```
@RestController
@RequestMapping("/api")
public class UserController {

    @GetMapping("/users/{id}")
    public ResponseEntity<User>
            getUserById(@PathVariable Long id) {
        ...
    }
}
```

여기 들어있는 값을

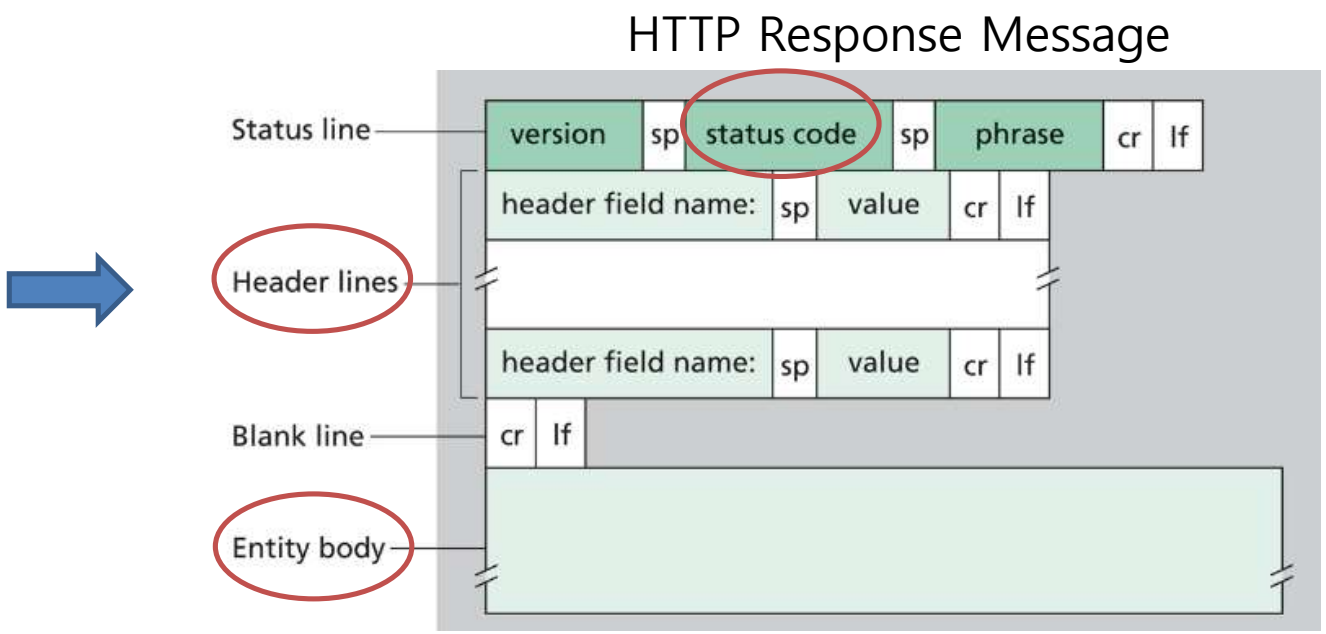
조회할 수 있다

4) ResponseEntity<T> class

response message의 [ ] 을 명시해줄 수 있다

- It represents the entire HTTP response
- You can specify status code, headers, and body
- It comes with several constructors to carry the information you want to sent in HTTP Response

ResponseEntity



ResponseEntity<T> class

Entity 클래스의 다양한 생성자들

Constructor	Description
ResponseEntity (**HttpStatus** status)	Create a new ResponseEntity with the given status code, and no body nor headers.
ResponseEntity (**MultiValueMap**<**String**,**String**> headers, **HttpStatus** status)	Create a new ResponseEntity with the given headers and status code, and no body.
ResponseEntity (**T** body, **HttpStatus** status)	Create a new ResponseEntity with the given body and status code, and no headers.
ResponseEntity (**T** body, **MultiValueMap**<**String**,**String**> headers, **HttpStatus** status)	Create a new ResponseEntity with the given body, headers, and status code.

ResponseEntity<T> class

우리가 지정한대로 얘가 response message 를 만들어준다

```
@RestController
@RequestMapping("/api")
public class RestApiController {

  @Autowired
  UserService userService;

  @RequestMapping(value = "/users", method = RequestMethod.GET)
  public ResponseEntity<List<User>> listAllUsers() {

    List<User> users = userService.findAllUsers();
    if (users.isEmpty()) {
      return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
    return new ResponseEntity<List<User>>(users, HttpStatus.OK);
  }
}
```

5) @RequestBody Annotation

- If a method parameter is annotated with @RequestBody, Spring will bind the incoming HTTP request to that parameter

객체의 내용을 response의 body로 보내는 건 @Responsebody

- While doing that, Spring will [behind the scenes] use HTTP Message converters to convert the HTTP request body into domain object [deserialize request body to domain object]

그런데 이것은 반대로, 사용자가 입력한 json(이메일?) 같은 것들이 객체로 오는 것이다 (desevialigation)

@RequestBody Annotation

HTTP Request Message

```
POST /api/users HTTP/1.1
Host: hansung.ac.kr
Content-Type: application/json

{
    "id": 123,
    "name": "Alice Kim",
    "email": "alice.kim@hansung.ac.kr"
}
```

```java
@RequestMapping(value = "/users", method = RequestMethod.POST)
public ResponseEntity<Void> createUser(@RequestBody User user,
                        UriComponentsBuilder ucBuilder) {
        …
        return new ResponseEntity<Void>(headers, HttpStatus.CREATED);
}
```

3. RESTful Web Services CRUD Example

합치면 서버에서
제공하는 API?

HTTP Method	URI	Operation
GET	/api/users	returns a list of users
GET	/api/users/1	returns the user with ID 1
POST	/api/users	creates a new user
PUT	/api/users/3	updates the user with ID 3
DELETE	/api/users/4	deletes the user with ID 4
DELETE	/api/users	deletes all the users

1) Dependency Management [pom.xml]

Java object <-> JSON data

객체 → JSON

= sevialigation →

```
<!-- jackson-databind -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.10.0</version>
</dependency>
```

Utilizes the Jackson Databind library
for automatic serialization /deserialization

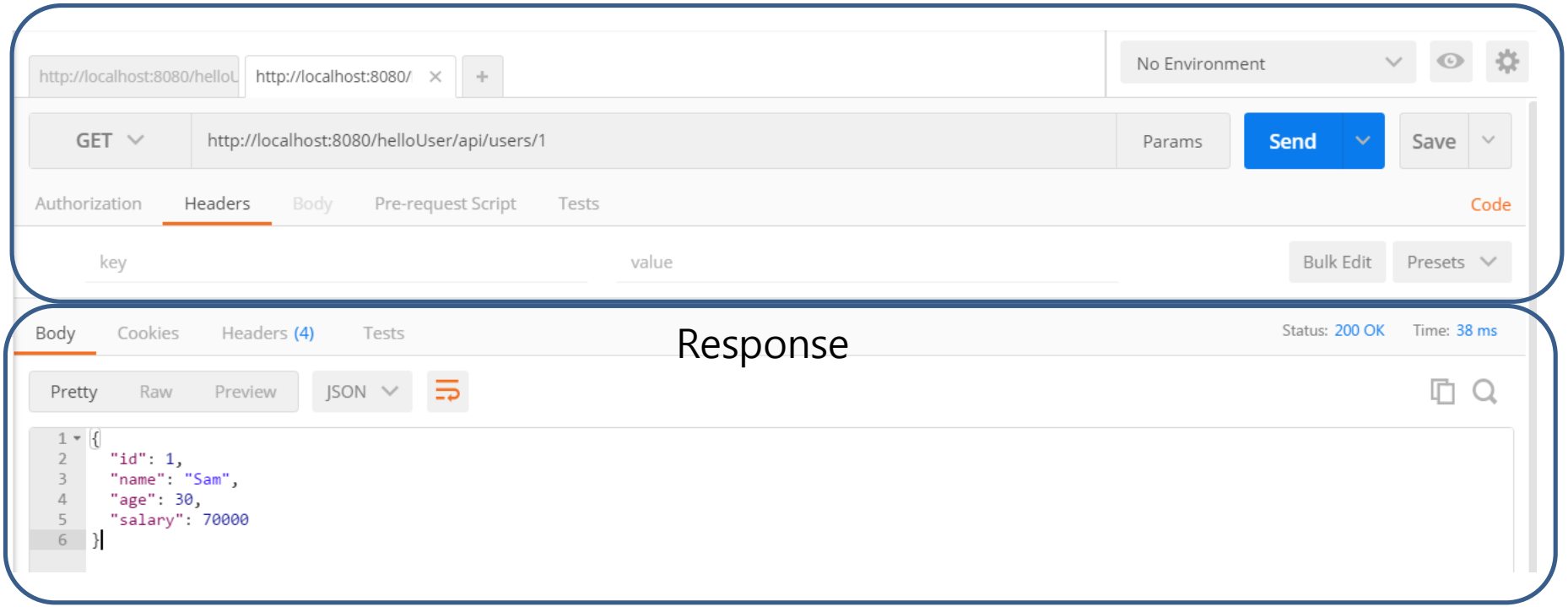
@RequestBody: Converts an HTTP request body
into a Java object(deserialization)

@ResponseBody: Converts a Java object
into the HTTP response body(serialization)

2) Rest Client
(Postman)

postman으로 클라이언트 역할을 대체할 수 있다

Request



Response

3) Rest API

```java
@RestController
@RequestMapping("/api")
public class RestApiController {

    @Autowired
    UserService userService;

    //-------------------Retrieve All Users---------------
    @RequestMapping(value = "/users", method = RequestMethod.GET)
    public ResponseEntity<List<User>> listAllUsers() {

        List<User> users = userService.findAllUsers();
        if (users.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.NO_CONTENT); // HttpStatus.NOT_FOUND?
        }
        return new ResponseEntity<List<User>>(users, HttpStatus.OK);
    }
```

Rest API

```java
//-------------------Retrieve Single User---------
@RequestMapping(value = "/users/{id}", method = RequestMethod.GET)
public ResponseEntity<User> getUser(@PathVariable("id") long id) {

    User user = userService.findById(id);
    if (user == null) {
        throw new UserNotFoundException(id);
    }
    return new ResponseEntity<User>(user, HttpStatus.OK);
}

// --------------Create a User-----------------------------------

@RequestMapping(value = "/users", method = RequestMethod.POST)
public ResponseEntity<Void> createUser(@RequestBody User user,  UriComponentsBuilder ucBuilder) {

    if (userService.isUserExist(user)) {
        throw new UserDuplicateException(user);
    }
    userService.saveUser(user);

    HttpHeaders headers = new HttpHeaders();
    headers.setLocation(ucBuilder.path("/api/users/{id}").buildAndExpand(user.getId()).toUri());

    return new ResponseEntity<Void>(headers, HttpStatus.CREATED);
}
```

예외처리는 컨트롤러 안에서

Rest API

```
// -------------- Update a User ----------------------------------------

@RequestMapping(value = "/users/{id}", method = RequestMethod.PUT)
public ResponseEntity<User> updateUser(@PathVariable("id") long id, @RequestBody User user) {

    User currentUser = userService.findById(id);

    if (currentUser == null) {
        throw new UserNotFoundException(id);
    }

    currentUser.setName(user.getName());
    currentUser.setAge(user.getAge());
    currentUser.setSalary(user.getSalary());

    userService.updateUser(currentUser);
    return new ResponseEntity<User>(currentUser, HttpStatus.OK);
}
```

Rest API

```
// ------------------- Delete a User----------------------------------------

@RequestMapping(value = "/users/{id}", method = RequestMethod.DELETE)
public ResponseEntity<User> deleteUser(@PathVariable("id") long id) {

    User user = userService.findById(id);
    if (user == null) {
        throw new UserNotFoundException(id);
    }
    userService.deleteUserById(id);
    return new ResponseEntity<User>(HttpStatus.NO_CONTENT);
}

// ------------------- Delete All Users-----------------------------

@RequestMapping(value = "/users", method = RequestMethod.DELETE)
public ResponseEntity<User> deleteAllUsers() {
    userService.deleteAllUsers();
    return new ResponseEntity<User>(HttpStatus.NO_CONTENT);
}
```