Spring RESTFul Web Services

Basic Spring 5.0

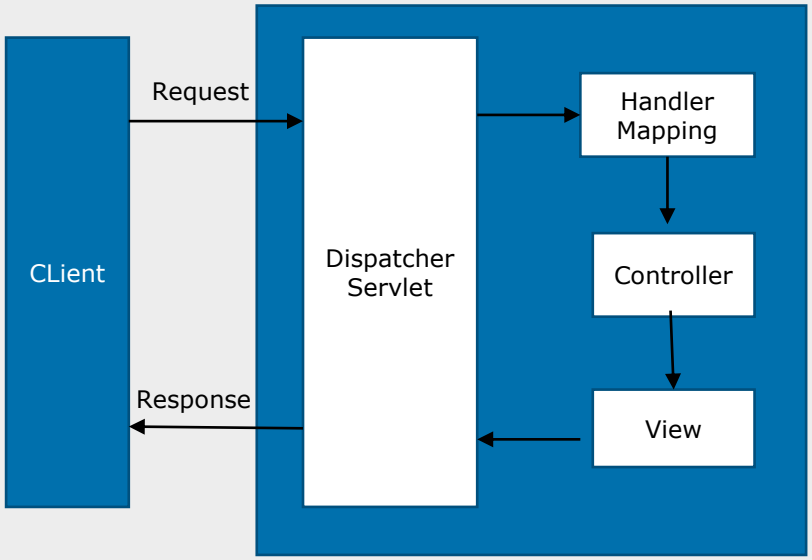
Capgemini

Lesson Objectives

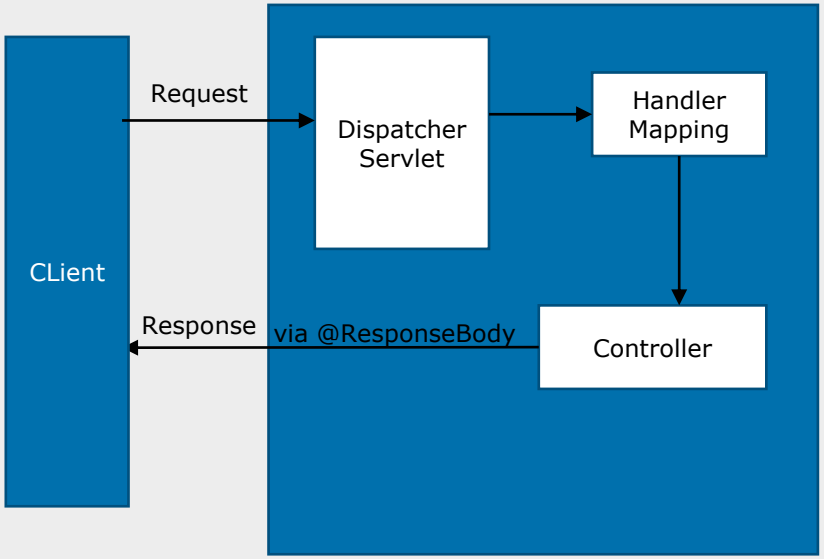
Introduction to Spring MVC framework

- *Spring MVC traditional workflow vs Spring MVC REST Workflow*
- Using the @ResponseBody Annotation
- Life cycle of a Request in Spring MVC Restful
- Why REST Controller ?
- *Spring 4.x MVC RESTful Web Services Workflow*
- REST Controller
- RESTful URLs – HTTP methods
- Cross-Origin Resource Sharing (CORS)
- @RequestBody annotation

7.1 Spring MVC traditional workflow vs Spring MVC REST Workflow



Spring MVC traditional workflow



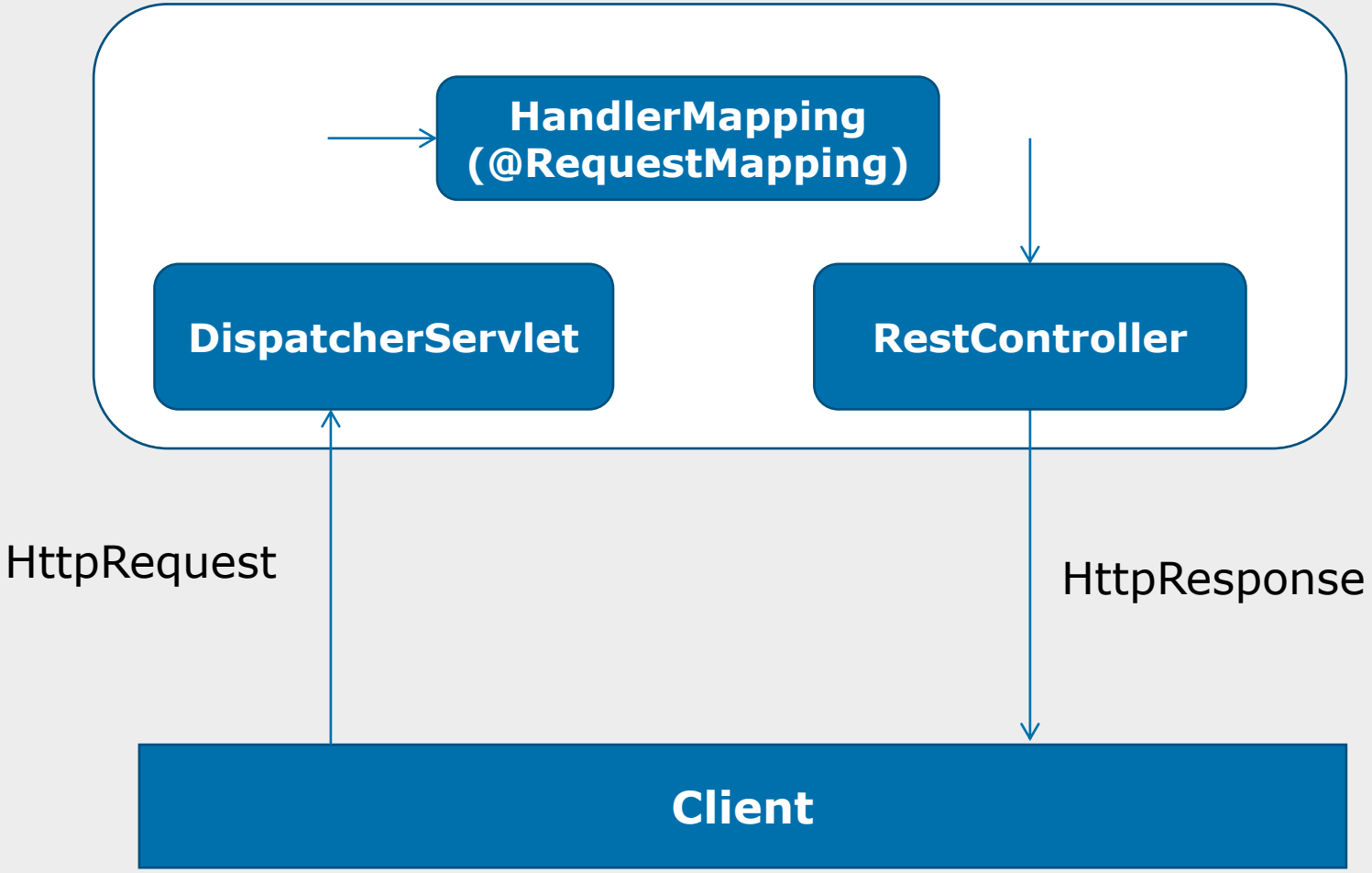
Spring3.x MVC REST Workflow

7.2 Using the @ResponseBody Annotation

When you use the @ResponseBody annotation on a method, Spring converts the return value and writes it to the http response automatically.

```
@Controller
@RequestMapping("employees")
public class EmployeeController {

    Employee employee = new Employee();

@RequestMapping(value = "/{name}", method = RequestMethod.GET, produces =
"application/json")
    public @ResponseBody Employee getEmployeeInJSON(@PathVariable String name) {

   employee.setName(name);
   employee.setEmail("employee1@genuitec.com");

   return employee;

    }

    @RequestMapping(value = "/{name}.xml", method = RequestMethod.GET, produces =
"application/xml")
    public @ResponseBody Employee getEmployeeInXML(@PathVariable String name) {

   employee.setName(name);
   employee.setEmail("employee1@genuitec.com");

   return employee;

    }
```

7.3 Life cycle of a Request in Spring MVC Restful



7.4 Why REST Controller ?

Traditional Spring MVC controller and the RESTful web service controller differs in the way the HTTP response body is created

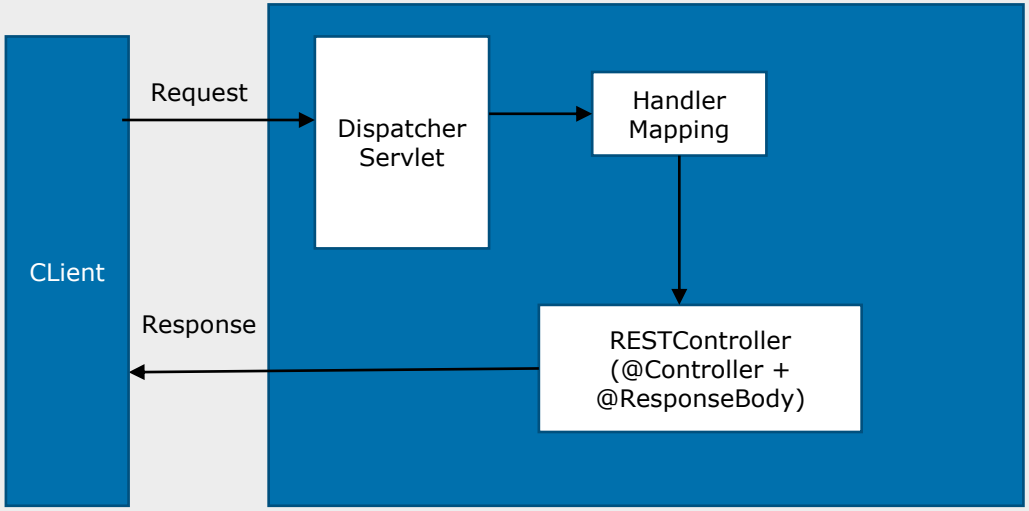
Traditional MVC controller relies on the View technology

RESTful controller simply returns the object and the object data is written directly to the HTTP response as JSON/XML

Spring 4.0 introduced @RestController

No longer need to add @ResponseBody

7.5 Spring 4.x MVC RESTful Web Services Workflow



7.6 Spring 4 support for RESTful web services

In Spring 4 REST is built on the top of MVC

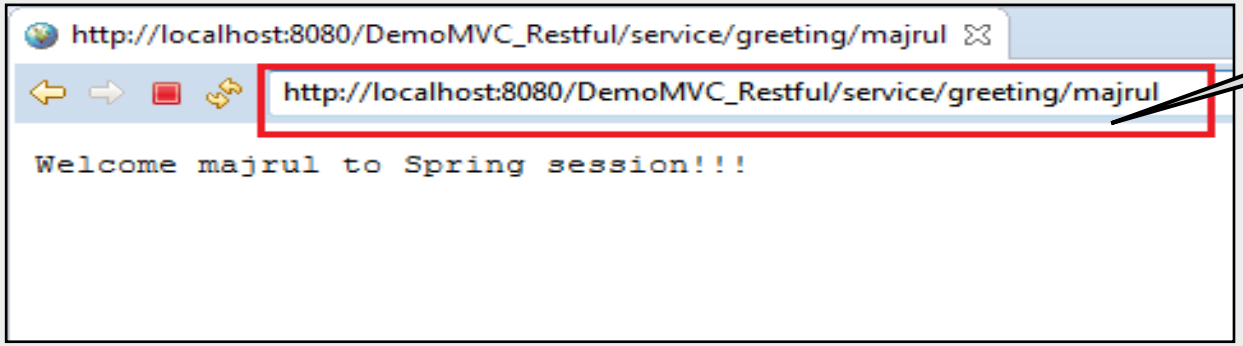
- REST methods: GET, PUT, DELETE, and POST, can be handled by Controllers
- Using @PathVariable annotation controllers can handle requested parameterized URLs

7.7 REST Controller

```
@RestController
@RequestMapping("/service/greeting")
public class SpringRestController {
        @RequestMapping(value = "/{name}", method = RequestMethod.GET)
        public String sayHello(@PathVariable Optional<String> name) {
                String result = "Welcome " + name + " to Spring session!!!";
                return result;
        }
}
```

No "name" required. "Do not repeat yourself" with Java 8 version

output



REST Controller

```
@RestController
@RequestMapping("employees")
public class EmployeeController {

    Employee employee = new Employee();

    @RequestMapping(value = "/{name}", method = RequestMethod.GET, produces =
"application/json")
    public Employee getEmployeeInJSON(@PathVariable String name) {

   employee.setName(name);
   employee.setEmail("employee1@genuitec.com");

   return employee;

    }

    @RequestMapping(value = "/{name}.xml", method = RequestMethod.GET, produces =
"application/xml")
    public Employee getEmployeeInXML(@PathVariable String name) {

   employee.setName(name);
   employee.setEmail("employee1@genuitec.com");

   return employee;

    }

}
```

RESTful URLs – HTTP methods

7.8 Cross-Origin Resource Sharing (CORS)

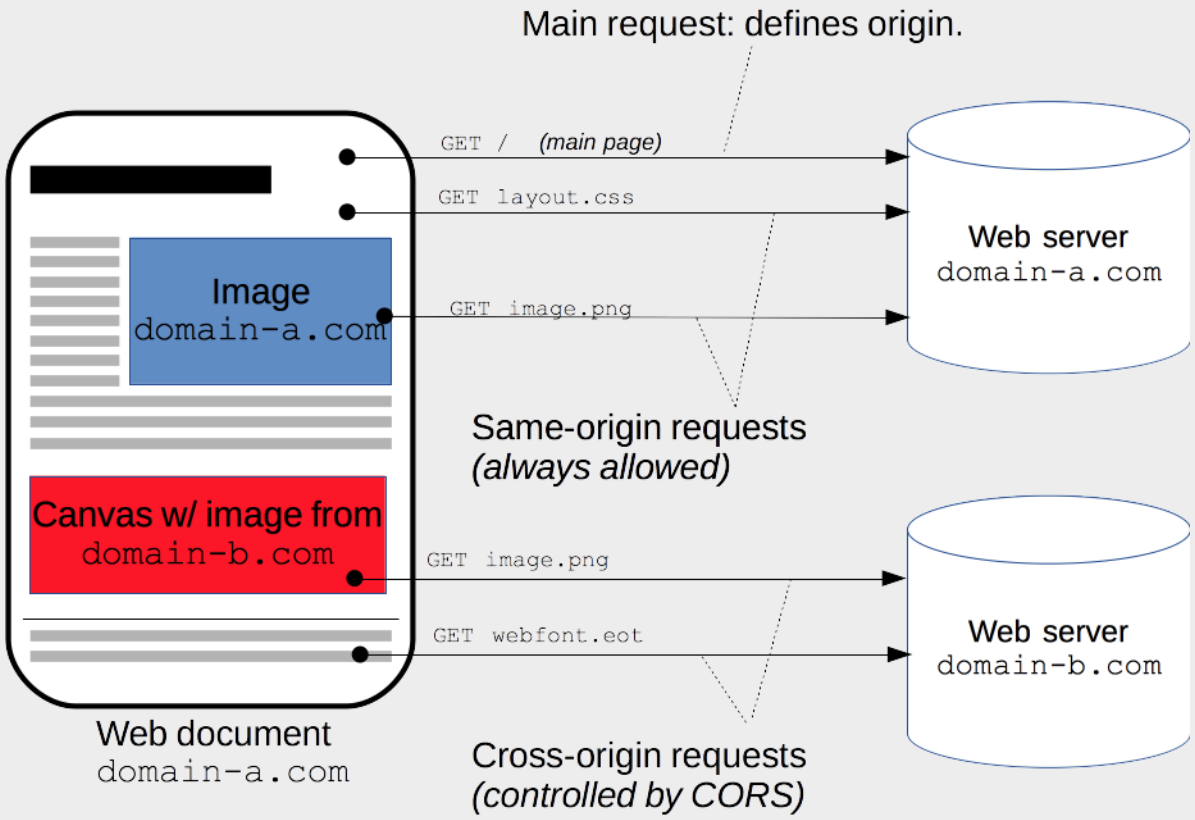
- Cross-Origin Resource Sharing (CORS) is a mechanism that uses additional HTTP headers to let a user agent gain permission to access selected resources from a server on a different origin (domain) than the site currently in use.

- A user agent makes a **cross-origin HTTP request** when it requests a resource from a different domain, protocol, or port than the one from which the current document originated.

- An example of a cross-origin request: A HTML page served from http://domain-a.com makes an <img> src request for http://domain-b.com/image.jpg. Many pages on the web today load resources like CSS stylesheets, images, and scripts from separate domains, such as content delivery networks (CDNs)

Cross-Origin Resource Sharing (CORS)

For security reasons, browsers restrict cross-origin HTTP requests initiated from within scripts. For example, XMLHttpRequest and the Fetch API follow the same-origin policy. This means that a web application using those APIs can only request HTTP resources from the same domain the application was loaded from unless CORS headers are used.



Cross-Origin Resource Sharing (CORS)

- We can add an @CrossOrigin annotation to your @RequestMapping annotated handler method in order to enable CORS on it.
- By default @CrossOrigin allows all origins and the HTTP methods specified in the @RequestMapping annotation
- It is also possible to enable CORS for the whole controller. So, we can even use both controller-level and method-level CORS configurations
- In addition to fine-grained, annotation-based configuration you'll probably want to define some global CORS configuration as well.

Cross-Origin Resource Sharing (CORS)

```
@CrossOrigin(origins = "http://localhost:4200")
@RestController
public class CountryController {
@Autowired
ICountryService service;

//@CrossOrigin(origins = "http://localhost:4200")
@RequestMapping(value = "/countries/search/{id}",method =
RequestMethod.GET,headers="Accept=application/json")
public Country getCounty(@PathVariable int id) {
return service.searchCountry(id);
}}
```

6.9 @RequestBody annotation

- If a method parameter is annotated with @RequestBody, Spring will bind the incoming HTTP request body(for the URL mentioned in @RequestMapping for that method) to that parameter.
- While doing that, Spring will use HTTP Message converters to convert the HTTP request body into domain object **[deserialize request body to domain object]**, based on **Accept** header present in request.

```
@RestController
public class EmployeeController {
@Autowired
IEmployeeService empservice;
@RequestMapping(value ="/employee/create/", consumes =
MediaType.APPLICATION_JSON_VALUE,
headers="Accept=application/json",method =
RequestMethod.POST)
public List<Employee> createEmployee(@RequestBody Employee
emp) {

        empservive.addEmployee(emp);
        return empservice.getAllEmployee();
} }
```

Demo: SpringRESTDemos

SpringRESTDemo

Summery

We have so far learnt

- *Spring MVC traditional workflow vs Spring MVC REST Workflow*
- Using the @ResponseBody Annotation
- Life cycle of a Request in Spring MVC Restful
- Why REST Controller ?
- *Spring 4.x MVC RESTful Web Services Workflow*
- REST Controller
- RESTful URLs – HTTP methods

Lab

Lab 2

Review Question

Question 1: Which of the following are true?

- Option1 : Resource classes are POJOs that have at least one method annotated with @Path
- Option 2: Resource methods are methods of a resource class annotated with a request method designator such as @GET, @PUT, @POST, or @DELETE
- Option 3: @FormParam binds query parameters value to a Java method

Question 2: The @Path annotation's value is a relative URI path indicating where the Java class will be hosted?

- True
- False

Review Question

Question 3: _____ specifies a media type a resource can generate.

- @PUT
- @POST
- @Produces
- @Consumes