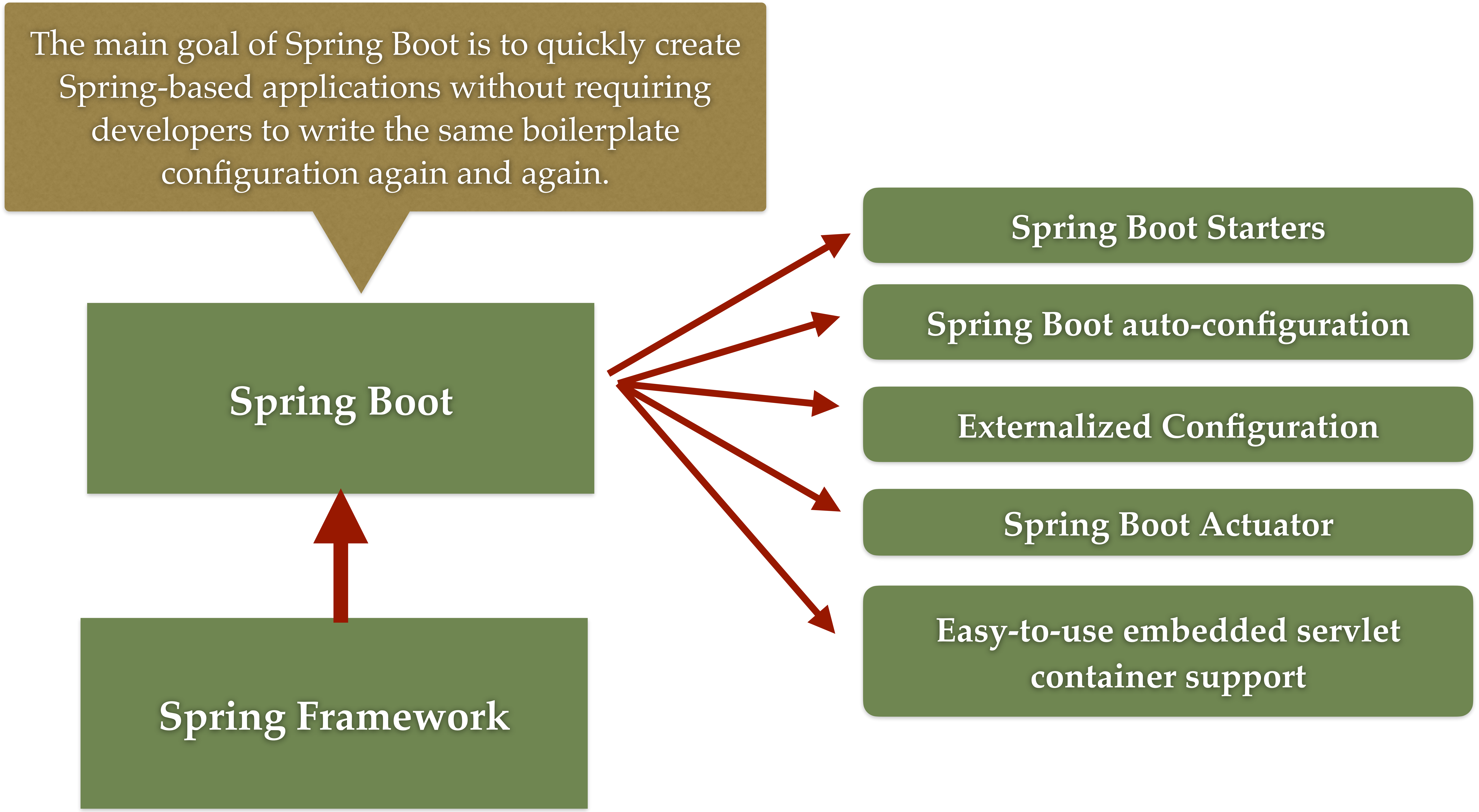
Spring Boot Fundamentals

By Ramesh Fadatare (Java Guides)

What is Spring Boot?



The main goal of Spring Boot is to quickly create Spring-based applications without requiring developers to write the same boilerplate configuration again and again.
Spring Boot
Spring Framework
Spring Boot Starters
Spring Boot auto-configuration
Externalized Configuration
Spring Boot Actuator
Easy-to-use embedded servlet container support

Spring Framework

1. **Spring is a very popular Java framework for building web and enterprise applications.**

2. **Spring is very popular for several reasons:**

    - dependency injection

    - Easy to use but powerful database transaction management capabilities

    - Good Integration with other Java frameworks like JPA/Hibernate ORM, Struts/JSF/etc. web frameworks

    - Web MVC framework for building web applications

What is a Problem?

Basically, Spring-based applications have a lot of configurations.

For example:

When we develop Spring MVC web application using Spring MVC then we need to configure:

Component scan, Dispatcher Servlet, View resolver, Web jars(for delivering static content) among other things.

When we use Hibernate/JPA in the same Spring MVC application then we would need to configure a

Data source, Entity manager factory/session factory, Transaction manager among other things.

When you use cache, message queue, NoSQL in the same Spring MVC application then we need to configure:

Cache configuration

Message queue configuration

NoSQL database configuration

One more major problem - We need to maintain all integration of different Jar dependencies and it's compatible versions

What is a Solution

> Spring Boot is the solution

> Spring Boot automatically configures the configurations based on the jar dependencies that we add to our project.

Spring Boot Starters

1. These starters are pre-configured with the most commonly used library dependencies so you don't have to search for the compatible library versions and configure them manually.

2. For example, when we add the **spring-boot-starter-web dependency**, it will by default pull all the commonly used libraries while developing Spring MVC applications, such as spring-webmvc, jackson-json, validation-api, and tomcat.

3. One more example, the **spring-boot-starter-data-jpa** starter module includes all the dependencies required to use Spring Data JPA, along with Hibernate library dependencies, as Hibernate is the most commonly used JPA implementation.

Spring Boot Auto Configuration

Spring Boot auto-configuration attempts to automatically configure Spring application based on the jar dependencies that you have added to project.

Example 1: if you add **spring-boot-starter-web** Jar dependency to your Spring boot application, Spring Boot assumes you are trying to build a SpringMVC-based web application and automatically tries to register Spring beans such as **DispatcherServlet, ViewResolver** if it is not already registered.

Example 2: If you add **spring-boot-starter-data-jpa** starter dependency then it assume that you are trying to use Hibernate to develop DAO layer so Spring boot automatically register the Spring beans such as **Data source, Entity manager factory/ session factory, Transaction manager.**

Spring Boot auto-configuration attempts to automatically configure Spring application based on the jar dependencies that you have added to project.

@RestController Annotation

1. In order to develop REST web services using Spring MVC, we need to use @Controllerand @ResponseBody annotations.

2. Spring 4.0 introduced @RestController, a specialized version of the @Controller which is a convenience annotation that does nothing more than adding the @Controller and @ResponseBody annotations.

3. In order to create Restful web services using Spring MVC, you need annotate a Java class with @RestController annotation.

@ResponseBody Annotation

1. The @ResponseBody annotation tells a controller that the object returned is automatically serialized into JSON and passed back into the HttpResponse object.

2. When you use the @ResponseBody annotation on a method, Spring uses HTTPMessageConverters to converts the return value based on the MIME types and writes it to the HTTP response automatically.

@RequestMapping Annotation

1. @RequestMapping is the most common and widely used annotation in Spring MVC. It is used to map web requests onto specific handler classes and/or handler methods.

2. @RequestMapping can be applied to the controller class as well as methods.

@RequestMapping Annotation Examples

1. @RequestMapping with Class
2. @RequestMapping with Method
3. @RequestMapping with Multiple URI
4. @RequestMapping with HTTP Method
5. @RequestMapping with Headers
6. @RequestMapping with Producers and Consumers

HTTP method-specific shortcut variants of @RequestMapping annotation

1. **@GetMapping** - shortcut for @RequestMapping(method = RequestMethod.GET)
2. **@PostMapping** - shortcut for @RequestMapping(method = RequestMethod.POST)
3. **@PutMapping** - shortcut for @RequestMapping(method = RequestMethod.PUT)
4. **@DeleteMapping** - shortcut for @RequestMapping(method =RequestMethod.DELETE)
5. **@PatchMapping** - shortcut for @RequestMapping(method = RequestMethod.PATCH)

@GetMapping Annotation

1. The GET HTTP request is used to get a single or multiple resources and @GetMapping annotation for mapping HTTP GET requests onto specific handler methods.

2. Specifically, @GetMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET).

ResponseEntity

1. ResponseEntity represents the whole HTTP response: status code, headers, and body. As a result, we can use it to fully configure the HTTP response.

2. If we want to use it, we have to return it from the endpoint; Spring takes care of the rest.

3. ResponseEntity is a generic type. Consequently, we can use any type as the response body.

@PostMapping Annotation

1. The POST HTTP method is used to create a resource and @PostMapping annotation for mapping HTTP POST requests onto specific handler methods.

2. Specifically, @PostMapping is a composed annotation that acts as a shortcut for **@RequestMapping(method = RequestMethod.POST).**

@RequestBody Annotation

1. The @RequestBody annotation is responsible for retrieving the HTTP request body and automatically converting it to the Java object.

2. Annotation indicating a method parameter should be bound to the body of the web request. The body of the request is passed through an HttpMessageConverter to resolve the method argument depending on the content type of the request.

@PutMapping Annotation

1. The PUT HTTP method is used to update the resource and @PutMapping annotation for mapping HTTP PUT requests onto specific handler methods.

2. Specifically, @PutMapping is a composed annotation that acts as a shortcut for **@RequestMapping(method = RequestMethod.PUT).**

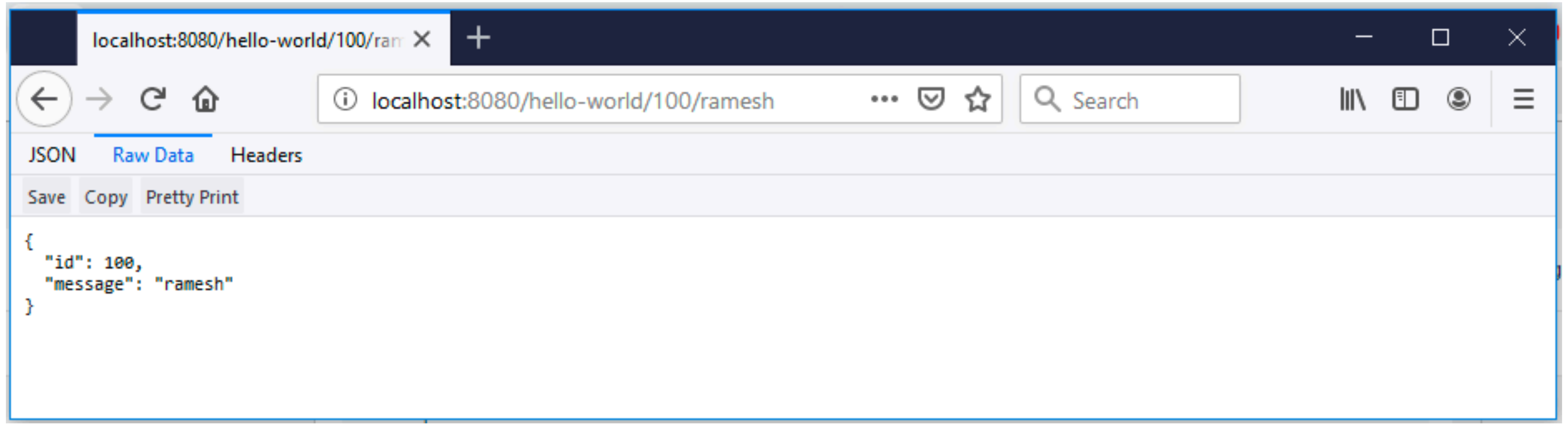
@DeleteMapping Annotation

1. The DELETE HTTP method is used to delete the resource and @DeleteMapping annotation for mapping HTTP DELETE requests onto specific handler methods.

2. Specifically, @DeleteMapping is a composed annotation that acts as a shortcut for **@RequestMapping(method = RequestMethod.DELETE).**

@PathVariable Annotation

1. Spring boot @PathVariable annotation used on a method argument to bind it to the value of a URI template variable.


localhost:8080/hello-world/100/ramesh

JSON Raw Data Headers

Save Copy Pretty Print

```
{
  "id": 100,
  "message": "ramesh"
}
```

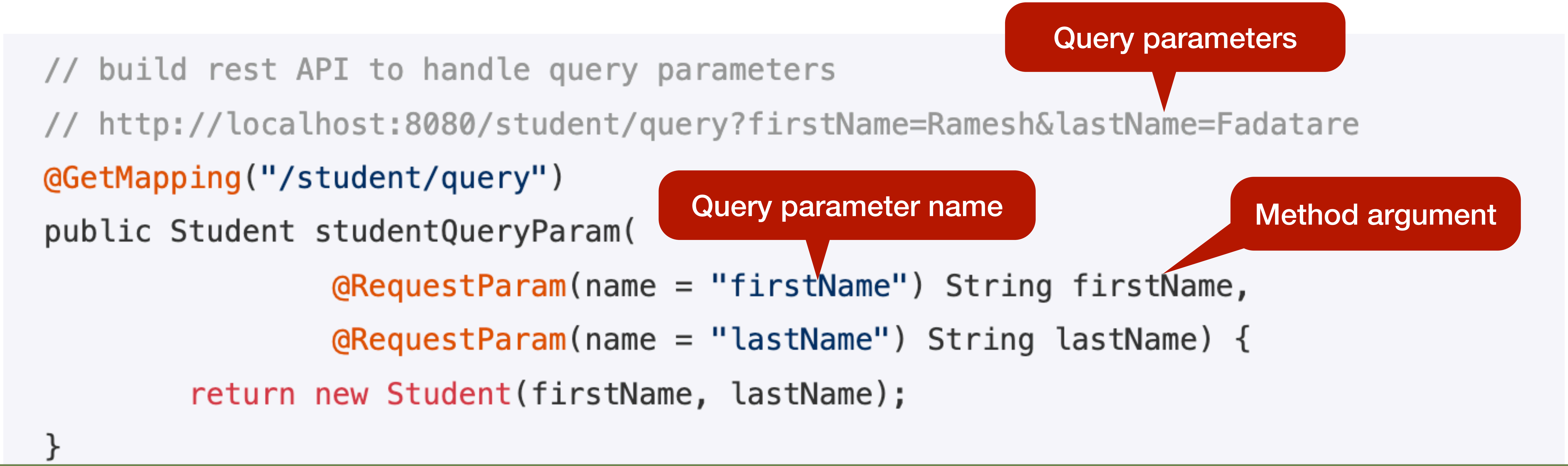
URI template variable

Method argument

```
@GetMapping(path = "/hello-world/{id}/{name}")
public HelloWorldBean helloWorldPathVariable(@PathVariable long id,
    @PathVariable(name = "name") String name) {
    return new HelloWorldBean(id, name);
```

@RequestParam Annotation

1. We can use @RequestParam to extract query parameters, form parameters, and even files from the request.



```
// build rest API to handle query parameters
// http://localhost:8080/student/query?firstName=Ramesh&lastName=Fadatare
@GetMapping("/student/query")
public Student studentQueryParam(
                @RequestParam(name = "firstName") String firstName,
                @RequestParam(name = "lastName") String lastName) {
        return new Student(firstName, lastName);
}
```