

Just Run RESTful Service

4

Java Backend Developer I



Objetivos

Revisar y comprender los conceptos:

- Spring-boot
 - Spring Application
 - Externalized Configuration
 - Profiles
 - Logging
 - JSON
 - Developing Web Applications
 - Caching
 - Calling RESTful w/ RestTemplate



Agenda

Revisión de los siguientes conceptos:

- Spring-boot
 - Spring Application
 - Externalized Configuration
 - Profiles
 - Logging
 - JSON
 - Developing Web Applications
 - Caching
 - Calling RESTful w/ RestTemplate



Spring Boot

Concepto

Es un proyecto de Spring que de acuerdo a la pagina oficial de Spring, se define como:

“... una solución para crear aplicaciones basadas en **Spring** de una manera rápida, autónoma y con características deseables para producción..”

Por lo tanto Spring Boot facilita la creación de aplicaciones independientes basadas en Spring.

Se basa en la plataforma Spring y las bibliotecas de terceros para que podamos empezar con un mínimo esfuerzo. La mayoría de las aplicaciones Spring Boot necesitan una configuración mínima de Spring.



Spring Boot

Características

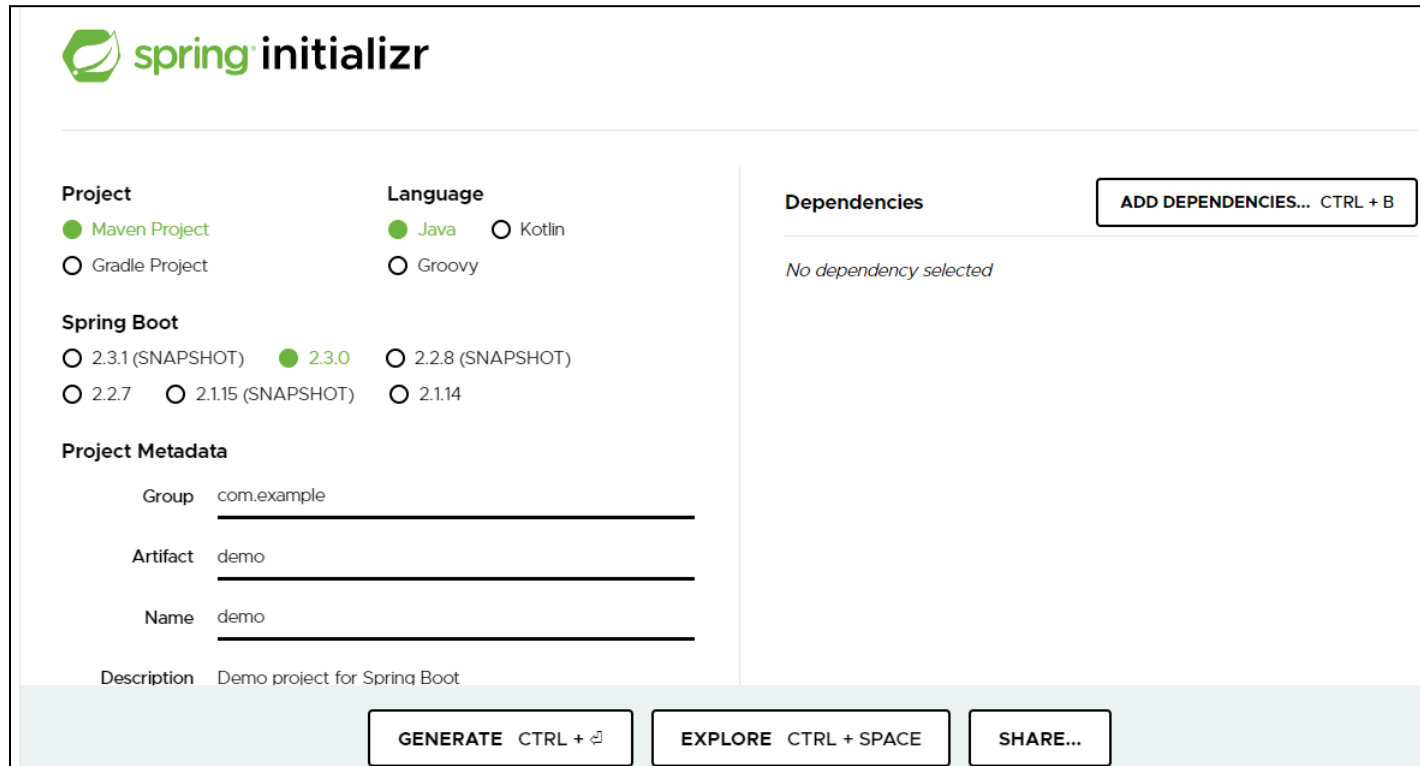
- Contenedores Java embebidos: Tomcat o Jetty
- Soporte para la automatización con Maven y Gradle
- Configuración sugerida para iniciar rápidamente con un proyecto (Starters)
- Configura automáticamente Spring, cuando sea posible
- Características listas para producción: métricas, seguridad, verificación del estatus, externalización de configuración, etc.
- No genera código y no requiere configuración XML
- Crear aplicaciones independientes de Spring.



Spring Boot

Como generar el proyecto

Utiliza el servicio web **Spring** Initializr para establecer la configuración de **Spring Boot** y, después, descárgala como plantilla de proyecto final.



The screenshot displays the Spring Initializr web interface. At the top left is the 'spring initializr' logo. The interface is divided into several sections for configuring a new project:

- Project:** Includes radio buttons for 'Maven Project' (selected) and 'Gradle Project'.
- Language:** Includes radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'.
- Spring Boot:** Includes radio buttons for various versions: '2.3.1 (SNAPSHOT)', '2.3.0' (selected), '2.2.8 (SNAPSHOT)', '2.2.7', '2.1.15 (SNAPSHOT)', and '2.1.14'.
- Project Metadata:** Contains input fields for 'Group' (com.example), 'Artifact' (demo), and 'Name' (demo). A 'Description' field at the bottom contains the text 'Demo project for Spring Boot'.
- Dependencies:** A section on the right with a button 'ADD DEPENDENCIES... CTRL + B' and the text 'No dependency selected'.

At the bottom of the interface are three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'.



Spring Boot

SpringBootApplication

Todo proyecto Spring boot debe tener una clase que contenga el método main el cual al ejecutar se despliega la app.

```
package aplicacion;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

@SpringBootApplication

```
public class SpringBootApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(SpringBootApplication.class, args);  
    }  
}
```



SpringBootApplication

Una salida al ejecutar podría ser

```

:: Spring Boot :: (v2.0.0.BUILD-SNAPSHOT)

2017-09-11 17:07:59.593 INFO 38091 --- [main] i.fiveballoons.springboot.SpringBootApplication : Starting SpringApplication on MacBook-Pro-2.local with PID 38091 (/
2017-09-11 17:07:59.595 INFO 38091 --- [main] i.fiveballoons.springboot.SpringBootApplication : No active profile set, falling back to default profiles: default
2017-09-11 17:07:59.694 INFO 38091 --- [main] ConfigServletWebServerApplicationContext : Refreshing org.springframework.boot.web.servlet.context.Annotation
2017-09-11 17:08:01.587 INFO 38091 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2017-09-11 17:08:01.613 INFO 38091 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2017-09-11 17:08:01.616 INFO 38091 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomcat/8.5.20
2017-09-11 17:08:01.810 INFO 38091 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2017-09-11 17:08:01.810 INFO 38091 --- [ost-startStop-1] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 2121 ms
2017-09-11 17:08:01.987 INFO 38091 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean : Mapping servlet: 'dispatcherServlet' to [/]
2017-09-11 17:08:01.992 INFO 38091 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'characterEncodingFilter' to: [/]
2017-09-11 17:08:01.993 INFO 38091 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'hiddenHttpMethodFilter' to: [/]
2017-09-11 17:08:01.993 INFO 38091 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'httpPutFormContentFilter' to: [/]
2017-09-11 17:08:01.994 INFO 38091 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'requestContextFilter' to: [/]
2017-09-11 17:08:02.419 INFO 38091 --- [main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.web.serv
2017-09-11 17:08:02.559 INFO 38091 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[[/],methods=[GET]]" onto public java.lang.String info
2017-09-11 17:08:02.568 INFO 38091 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[[/error]]" onto public org.springframework.http.Respor
2017-09-11 17:08:02.569 INFO 38091 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[[/error],produces=[text/html]]" onto public org.spring
2017-09-11 17:08:02.605 INFO 38091 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/webjars/**] onto handler of type [class org.spr
2017-09-11 17:08:02.605 INFO 38091 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**] onto handler of type [class org.springfr
2017-09-11 17:08:02.667 INFO 38091 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**/favicon.ico] onto handler of type [class c
2017-09-11 17:08:02.845 INFO 38091 --- [main] o.s.j.e.a.AnnotationMBeanExporter : Registering beans for JMX exposure on startup
2017-09-11 17:08:02.950 INFO 38091 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http)
2017-09-11 17:08:02.956 INFO 38091 --- [main] i.fiveballoons.springboot.SpringBootApplication : Started SpringApplication in 3.868 seconds (JVM running for 4.274)
```



Spring Boot

Externalized Configuration

- Spring Boot permite externalizar su configuración para poder trabajar con el mismo código de aplicación en diferentes entornos.
- Puede usar archivos de propiedades, archivos YAML, variables de entorno y argumentos de línea de comandos para externalizar la configuración.
- Los property Values se pueden inyectar directamente en sus beans utilizando la anotación `@Value`, a la que se accede a través de la abstracción de Spring Environment, o se pueden vincular a objetos estructurados a través de `@ConfigurationProperties`.



Spring Boot

Externalized Configuration

Spring Boot utiliza un orden PropertySource muy particular que está diseñado para permitir una anulación razonable de los valores. Las propiedades se consideran en el siguiente orden:

1. Propiedades de configuración global de Devtools en su directorio de inicio (~ / .spring-boot-devtools.properties cuando devtools está activo).
2. @TestPropertySource anotaciones en sus pruebas.
3. propiedades de atributos en sus pruebas. Disponible en @SpringBootTest y las anotaciones de prueba para probar una porción particular de su aplicación.
4. Argumentos de línea de comando.
5. Propiedades de SPRING_APPLICATION_JSON (JSON en línea incrustado en una variable de entorno o propiedad del sistema).



Spring Boot

Externalized Configuration

Las propiedades se consideran en el siguiente orden:

6. Parámetros de inicio de ServletConfig.
7. Parámetros de inicio de ServletContext.
8. Atributos JNDI de java: comp / env.
9. Propiedades del sistema Java (System.getProperties ()).
10. Variables de entorno del sistema operativo.
11. Un RandomValuePropertySource que tiene propiedades solo al azar. *.
12. Propiedades de aplicación específicas de perfil fuera de su jar empaquetado (application- {profile} .properties y variantes YAML).



Spring Boot

Externalized Configuration

Las propiedades se consideran en el siguiente orden:

13. Propiedades de aplicación específicas de perfil empaquetadas dentro de su jar (application- {profile} .properties y variantes YAML).
14. Propiedades de la aplicación fuera de su jar empaquetado (application.properties y variantes YAML).
15. Propiedades de la aplicación empaquetadas dentro de su jar (application.properties y variantes YAML).
16. Anotaciones de @PropertySource en sus clases de @Configuration.
17. Propiedades predeterminadas (especificadas al establecer SpringApplication.setDefaultProperties).



Spring Boot

Profiles

Concepto

Los perfiles son una característica central del marco, lo que nos permite asignar nuestros beans a diferentes perfiles, por ejemplo, dev, test, prod.

Nosotros podemos activar diferentes perfiles en diferentes entornos para arrancar solo los beans que necesitamos:

@Profile on a Bean

Comencemos de manera simple y veamos cómo podemos hacer que un bean pertenezca a un perfil en particular. Usando la anotación @Profile, estamos asignando el bean a ese perfil en particular; la anotación simplemente toma los nombres de uno (o múltiples) perfiles.



Spring Boot

Profiles

@Profile on a Bean

Considere un escenario básico: tenemos un bean que solo debe estar activo durante el desarrollo, pero no desplegado en la producción.

Por anotaciones Anotamos ese bean con un perfil "dev", y solo estará presente en el contenedor durante el desarrollo; en producción, el dev simplemente no estará activo:

```
@Component
@Profile("dev")
public class DevDatasourceConfig{}
```



Spring Boot

Profiles

@Profile on a Bean

Considere un escenario básico: tenemos un bean que solo debe estar activo durante el desarrollo, pero no desplegado en la producción.

Por XML: Los perfiles también se pueden configurar en XML: la etiqueta <beans> tiene el atributo "profiles" que toma valores separados por comas de los perfiles aplicables:

```
<beans profile="dev">  
  <bean id="devDatasourceConfig"  
    class="org.aplicacion.profiles.DevDatasourceConfig" />  
</beans>
```



Spring Boot

Setear profile

El siguiente paso es activar y configurar los perfiles para que los beans respectivos se registren en el contenedor.

Esto se puede hacer de varias maneras:

1.- Via **WebApplicationInitializer** Interface

@Configuration

```
public class MyWebApplicationInitializer implements WebApplicationInitializer {  
    @Override  
    public void onStartup(ServletContext servletContext) throws ServletException {  
        servletContext.setInitParameter(  
            "spring.profiles.active", "dev");  
    }  
}
```



Spring Boot

Setear profile

2.- Via ConfigurableEnvironment (variable)

```
@Autowired  
private ConfigurableEnvironment env;  
  
...  
env.setActiveProfiles("someProfile");
```

3.- Via Context Parameter in web.xml

```
<context-param>  
  <param-name>contextConfigLocation</param-name>  
  <param-value>/WEB-INF/app-config.xml</param-value>  
</context-param>  
<context-param>  
  <param-name>spring.profiles.active</param-name>  
  <param-value>dev</param-value>  
</context-param>
```



Spring Boot

Setear profile

4.- Via Maven Profile

```
<profiles>
  <profile>
    <id>dev</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
      <spring.profiles.active>dev</spring.profiles.active>
    </properties>
  </profile>
  <profile>
    <id>prod</id>
    <properties>
      <spring.profiles.active>prod</spring.profiles.active>
    </properties>
  </profile>
</profiles>
```



Spring Boot

Setear profile

En el POM.xml

```
<plugins>
  <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
      <profiles>
        <profile>dev</profile>
      </profiles>
    </configuration>
  </plugin>
  ...
</plugins>
```



Spring Boot

Setear profile in spring boot:

Properties

- Pero la característica más importante relacionada con los perfiles que trae Spring Boot son los archivos de propiedades específicos del perfil. Estos deben ser nombrados en el formato aplicaciones- {perfil} .properties.
- Spring Boot cargará automáticamente las propiedades en un archivo application.properties para todos los perfiles, y las que están en archivos .properties específicos del perfil solo para el perfil especificado.



Spring Boot

Setear profile in spring boot:

Properties

Ejemplo: Podemos configurar diferentes fuentes de datos para los perfiles de desarrollo y producción mediante el uso de dos archivos llamados `application-dev.properties` y `application-production.properties`:

- En el archivo `application-production.properties`, podemos configurar una fuente de datos MySQL:

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

```
spring.datasource.url=jdbc:mysql://localhost:3306/db
```

```
spring.datasource.username=root
```

```
spring.datasource.password=root
```



Spring Boot

Setear profile in spring boot: properties

Ejemplo

- Luego, podemos configurar las mismas propiedades para el perfil de desarrollo en el archivo application-dev.properties, para usar una base de datos H2 en memoria:

```
spring.datasource.driver-class-name=org.h2.Driver  
spring.datasource.url=jdbc:h2:mem:db;DB_CLOSE_DELAY=-1  
spring.datasource.username=sa  
spring.datasource.password=sa
```

De esta manera, podemos proporcionar fácilmente diferentes configuraciones para diferentes entornos.



Spring Boot

Logging

- Spring Boot utiliza el registro de Apache Commons para todos los registros internos. Las configuraciones predeterminadas de Spring Boot brindan soporte para el uso de Java Util Logging, Log4j2 y Logback. Con estos, podemos configurar el registro de la consola, así como el registro de archivos.
- Si está utilizando Spring Boot Starters, Logback proporcionará un buen soporte para el registro. Además, Logback también proporciona un buen soporte para Common Logging, Util Logging, Log4J y SLF4J.



Spring Boot

Logging

El default Spring Boot Log format es similar a:

```
gram Files\Java\jdk1.8.0_65\bin\java" ...
3 Mon 23:30:44.435 INFO 14768 --- [ main] ExampleMain : Starting ExampleMain on JoeMchn with PID 147
3 Mon 23:30:44.438 INFO 14768 --- [ main] ExampleMain : No active profile set, falling back to defau
3 Mon 23:30:44.489 INFO 14768 --- [ main] AnnotationConfigApplicationContext : Refreshing org.springframework.context.annot
3 Mon 23:30:45.177 INFO 14768 --- [ main] AnnotationMBeanExporter : Registering beans for JMX exposure on startu
3 Mon 23:30:45.188 INFO 14768 --- [ main] ExampleMain : Started ExampleMain in 0.983 seconds (JVM ru
3 Mon 23:30:45.188 WARN 14768 --- [ main] ExampleMain : warning msg
3 Mon 23:30:45.188 ERROR 14768 --- [ main] ExampleMain : error msg
3 Mon 23:30:45.189 INFO 14768 --- [Thread-2] AnnotationConfigApplicationContext : Closing org.springframework.context.annotati
3 Mon 23:30:45.190 INFO 14768 --- [Thread-2] AnnotationMBeanExporter : Unregistering JMX-exposed beans on shutdown

finished with exit code 0
```

Nos Da la siguiente información:

- **Fecha y hora** que da la fecha y hora del registro
- **El nivel de registro** muestra INFO, ERROR o WARN Identificación de proceso
- **El ---** que es un separador
- **El nombre del hilo** está encerrado entre corchetes [] Nombre del registrador que muestra el nombre de la clase de origen
- **El mensaje de registro**



Spring Boot

Logging En el main:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.logging.LoggingSystem;
import org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication
public class ExampleMain {
    private static final Logger logger = LoggerFactory.getLogger(ExampleMain.class);
    public static void main(String[] args) throws InterruptedException {

        ConfigurableApplicationContext context =
            SpringApplication.run(ExampleMain.class, args);
        logger.warn("warning msg");
        logger.error("error msg");
    }
}
```



Spring Boot

JSON

Cómo crear un servicio REST para consumir y producir contenido JSON con Spring Boot. Se revisa como emplear la semántica HTTP RESTful.

1.- REST Service

Escribir un servicio JSON REST en Spring Boot es simple, ya que esa es su opinión predeterminada cuando Jackson está en el classpath



Spring Boot

JSON

REST Service

```
@RestController
@RequestMapping("/empleados")
public class EmpleadoController {

    @Autowired
    private EmpleadoService service;

    @GetMapping("/{id}")
    public Empleado read(@PathVariable String id) {
        return service.find(id);
    }
}
```

...



Spring Boot y JSON

Rest Service

Explicación

- Al anotar nuestro `EmpleadoController` con `@RestController`, le hemos dicho a Spring Boot que escriba el tipo de retorno del método de lectura en el cuerpo de la respuesta.
- Como también tenemos un `@RequestMapping` a nivel de clase, sería lo mismo para cualquier otro método público que agreguemos.



Spring Boot y JSON

Rest Service

Explicación

Aunque simple, este enfoque carece de semántica HTTP. Por ejemplo, ¿qué debería pasar si no encontramos al empleado solicitado? En lugar de devolver un código de estado 200 o 500, es posible que queramos devolver un 404.

Requerimos obtener más control sobre la respuesta HTTP en sí y, a su vez, agreguemos algunos comportamientos RESTful típicos a nuestro controlador.



Spring Boot y JSON

2. Create

Cuando necesitamos controlar aspectos de la respuesta que no sean el cuerpo, como el código de estado, podemos devolver una `ResponseEntity`:

@PostMapping("/")

```
public ResponseEntity<Empleado> create(@RequestBody Empleado empleado) throws
URISyntaxException {
    Empleado createdEmpleado= service.create(empleado);
    if (createdEmpleado == null) {
        return ResponseEntity.notFound().build();
    } else {
        URI uri = ServletUriComponentsBuilder.fromCurrentRequest()
            .path("/{id}")
            .buildAndExpand(createdEmpleado.getId())
            .toUri();

        return ResponseEntity.created(uri)
            .body(createdEmpleado);
    }
}
```



Spring Boot y JSON

2. Create

Aquí, estamos haciendo mucho más que simplemente devolver al Empleado creado en la respuesta. Además, respondemos con un estado HTTP semánticamente claro y, si la creación se realizó con éxito, un URI para el nuevo recurso.

3. Read

Como se mencionó anteriormente, si queremos leer un solo Empleado, es más semánticamente claro devolver un 404 si no podemos encontrar al estudiante:



Spring Boot y JSON

3. Read

Como se mencionó anteriormente, si queremos leer un solo Empleado, es más semánticamente claro devolver un 404 si no podemos encontrar al estudiante:

```
@GetMapping("/{id}")
public ResponseEntity<Empleado> read(@PathVariable("id") Long id) {
    Empleado foundEmpleado = service.read(id);
    if (foundEmpleado == null) {
        return ResponseEntity.notFound().build();
    } else {
        return ResponseEntity.ok(foundEmpleado);
    }
}
```



Spring Boot y JSON

3. Read

Aquí, podemos ver claramente la diferencia de nuestra implementación inicial de read ().

De esta forma, el objeto Empleado se asignará correctamente al cuerpo de respuesta y se devolverá con un estado adecuado al mismo tiempo.



Spring Boot y JSON

4. Update

La actualización es muy similar a la creación, excepto que está asignada a PUT en lugar de POST, y el URI contiene una identificación del recurso que estamos actualizando:

```
@PutMapping("/{id}")
public ResponseEntity<Empleado> update(@RequestBody Empleado
empleado, @PathVariable Long id) {
    Student updatedEmpleado = service.update(id, empleado);
    if (updatedEmpleado == null) {
        return ResponseEntity.notFound().build();
    } else {
        return ResponseEntity.ok(updatedEmpleado);
    }
}
```



Spring Boot y JSON

5. Delete

La operación de eliminación se asigna al método DELETE. URI también contiene la identificación del recurso:

```
@DeleteMapping("/{id}")  
public ResponseEntity<Object> deleteEmpleado(@PathVariable  
Long id) {  
    service.delete(id);  
    return ResponseEntity.noContent().build();  
}
```

No implementamos un manejo de errores específico, porque el método delete () en realidad falla al lanzar una excepción.



Spring Boot

JSON

Conclusión

Vimos cómo consumir y producir contenido JSON en un servicio CRUD REST típico desarrollado con Spring Boot. Además, demostramos cómo implementar un control de estado de respuesta y manejo de errores adecuados.

Para simplificar las cosas, esta vez no fuimos persistentes, pero Spring Data REST proporciona una forma rápida y eficiente de construir un servicio de datos RESTful.



Spring Boot

RestTemplate

Concepto

El acceso a un servicio REST de terceros dentro de una aplicación Spring gira en torno al uso de la clase Spring RestTemplate.

La clase RestTemplate está diseñada con los mismos principios que las muchas otras clases Spring * Template (por ejemplo, JdbcTemplate, JmsTemplate), proporcionando un enfoque simplificado con comportamientos predeterminados para realizar tareas complejas.



Spring Boot

Clase RestTemplate

Dado que la clase RestTemplate está diseñada para llamar a los servicios REST, no debería sorprendernos que sus métodos principales estén estrechamente relacionados con los fundamentos de REST, que son los métodos del protocolo HTTP: HEAD, GET, POST, PUT, DELETE y OPTIONS.



Spring Boot

Spring RestTemplate

HTTP GET Method

Soporta los siguientes metodos:

getForObject (url, classType): recupera una representación haciendo un GET en la URL. La respuesta (si la hay) se desarma al tipo de clase dado y se devuelve.

getForEntity (url, responseType): recupera una representación como ResponseEntity haciendo un GET en la URL.

exchange (requestEntity, responseType): ejecuta la solicitud especificada y devuelve la respuesta como ResponseEntity.

execute (url, httpMethod, requestCallback, responseExtractor): ejecuta httpMethod en la plantilla de URI dada, prepara la solicitud con RequestCallback y lee la respuesta con ResponseExtractor.



Lecturas adicionales

Para obtener información adicional, puede consultar los siguientes enlaces:

<https://docs.spring.io/spring-boot/docs/2.1.13.RELEASE/reference/html/boot-features-logging.html>

<https://docs.spring.io/spring-boot/docs/2.1.8.RELEASE/reference/html/boot-features-external-config.html>



Resumen

En este capítulo, usted aprendió:

- Spring-boot
- Spring Application
- Externalized Configuration
- Profiles
- Logging
- JSON
- Developing Web Applications
- Caching
- Calling RESTful w/ RestTemplate

