

El concepto de Spring REST Client es muy necesario para la mayor parte de los que trabajamos con Spring Framework. En muchas ocasiones tenemos que construir servicios **REST con @RestController**. Pero en muchos otros casos necesitamos acceder de una forma efectiva a esos servicios creados con Spring Framework . Una de las formas más sencillas de realizar esta tarea es utilizar la clase RestTemplate de Spring que nos simplifica sobre manera la forma de trabajar. Vamos a ver lo sencillo que es, para ello nos construiremos un servicio REST con Spring Boot y la anotación RestController.

```
package com.arquitecturajava.springrest;

import java.util.concurrent.Future;

import org.springframework.scheduling.annotation.Async;
import org.springframework.scheduling.annotation.AsyncResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

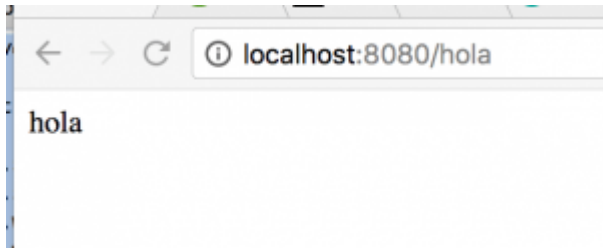
@RestController
public class HolaRest {

    @GetMapping("/hola")
    public String hola() {

        return "hola";
    }

}
```

Podemos arrancar nuestra aplicación de Spring Boot y tendremos a nuestra disposición el servicio REST a través de un navegador.



Spring REST Client

Sin embargo en muchas ocasiones lo que necesitamos es acceder a este servicio a través de Java y de Spring Framework. Una solución sencilla es usar RestTemplates que nos genera una plantilla para tener un acceso muy sencillo al servicio. Vamos a verlo.

```
package com.arquitecturajava.cliente;
```

```
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.client.RestTemplate;
```

```
@SpringBootApplication
```

```
public class ClienteApplication implements CommandLineRunner {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication app = new
SpringApplication(ClienteApplication.class);
        app.setWebEnvironment(false);
        app.run(args);
```

```

    }

    @Override
    public void run(String... args) throws Exception {

        RestTemplate plantilla = new RestTemplate();
        String resultado =
plantilla.getForObject("http://localhost:8080/hola", String.class);
        System.out.println(resultado);

    }
}

```

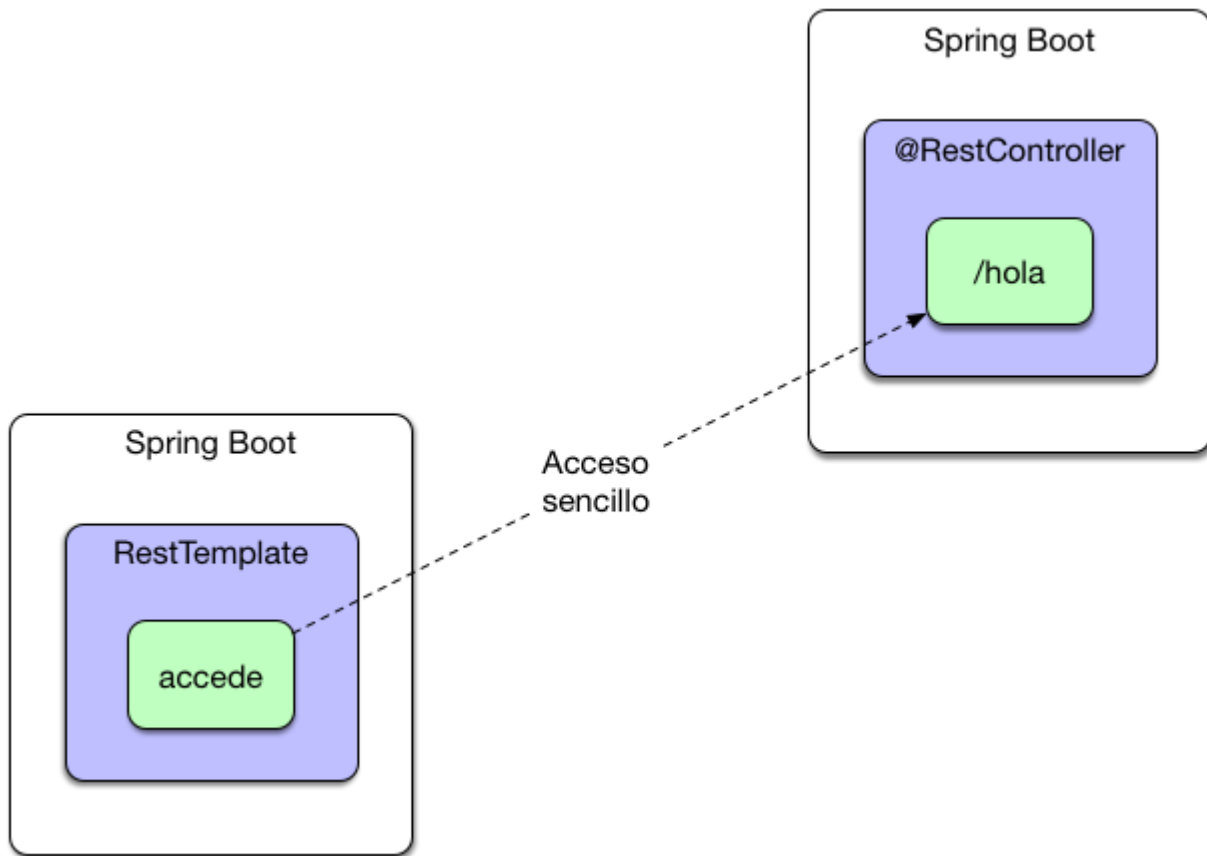
Acabamos de utilizar un objeto de tipo RestTemplate para invocar de forma directa una URL y acceder a los datos que se encuentran en /hola. En este caso hemos usado un tipo básico como es String sin embargo podríamos haber usado cualquier objeto de negocio sin problemas (Persona, Factura etc) y hubieramos obtenido los datos almacenados en esa URL. Si ejecutamos nuestro proyecto de Spring Boot a nivel de consola veremos el resultado:

```

2017-12-19 17:52:04.684 INFO 29208 --- [
2017-12-19 17:52:04.686 INFO 29208 --- [
2017-12-19 17:52:04.753 INFO 29208 --- [
2017-12-19 17:52:05.411 INFO 29208 --- [
hola
2017-12-19 17:52:08.474 INFO 29208 --- [
2017-12-19 17:52:08.476 INFO 29208 --- [
2017-12-19 17:52:08.477 INFO 29208 --- [

```

Acabamos de construir dos proyectos que usan Spring Boot y se conectan entre ellos utilizando Spring RestTemplate para crear un Spring REST Client.



Spring nos permite como siempre simplificar sobremanera la forma de acceder a los recursos más habituales utilizando el patrón template.

Otros artículos relacionados:

1. [Spring Boot WAR sin Microservicios](#)
2. [Spring 5 Hello World](#)
3. [Spring Security Annotation y su configuración](#)
4. [Spring REST](#)
5. [Spring GetMapping ,PostMapping etc](#)