Consuming a RESTful Web Service

This guide walks you through the process of creating an application that consumes a RESTful web service.

What you'll build

You'll build an application that uses Spring's `RestTemplate` to retrieve a random Spring Boot quotation at http://gturnquist-quoters.cfapps.io/api/random.

What you'll need

- About 15 minutes
- A favorite text editor or IDE
- JDK 1.8 or later
- Gradle 2.3+ or Maven 3.0+
- You can also import the code from this guide as well as view the web page directly into Spring Tool Suite (STS) and work your way through it from there.

Get the Code

HTTPS SSH Subversion

https://github.com/spring-guic

DOWNLOAD ZIP

GO TO REPO

Table of contents

How to complete this guide

Like most Spring Getting Started guides, you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to Build with Gradle.

To **skip the basics**, do the following:

- Download and unzip the source repository for this guide, or clone it using Git:
  ```
  git clone https://github.com/spring-guides/gs-consuming-rest.git
  ```
- cd into `gs-consuming-rest/initial`
- Jump ahead to Fetch a REST resource.

When you're finished, you can check your results against the code in `gs-consuming-rest/complete`.

Build with Gradle

Build with Maven

Tags

rest

Concepts and technologies

WAR

Git

Tomcat

Build with Spring Tool Suite

Fetch a REST resource

With project setup complete, you can create a simple application that consumes a RESTful service.

A RESTful service has been stood up at http://gturnquist-quoters.cfapps.io/api/random. It randomly fetches quotes about Spring Boot and returns them as a JSON document.

If you request that URL through your web browser or curl, you’ll receive a JSON document that looks something like this:

```
{
   type: "success",
   value: {
      id: 10,
      quote: "Really loving Spring Boot, makes stand alone Sprin
   }
}
```

Easy enough, but not terribly useful when fetched through a browser or through curl.

A more useful way to consume a REST web service is programmatically. To help you with that task, Spring provides a convenient template class called `RestTemplate`. `RestTemplate` makes interacting with most RESTful services a one-line incantation. And it can even bind that data to custom domain types.

First, create a domain class to contain the data that you need. If all you need to know are Pivotal's name, phone number, website URL, and what the pivotalsoftware page is about, then the following domain class should do fine:

`src/main/java/hello/Quote.java`

```java
package hello;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;

@Data
@JsonIgnoreProperties(ignoreUnknown = true)
public class Quote {

    private String type;
    private Value value;

}
```

As you can see, this is a simple Java class with a handful of properties and matching getter methods. It's annotated with `@JsonIgnoreProperties` from the Jackson JSON processing library to indicate that any properties not bound in this type should be ignored.

It also uses Project Lombok's `@Data` annotation, which provides a getter, a setter, a toString, and other supporting methods.

An additional class is needed to embed the inner quotation itself.

`src/main/java/hello/Value.java`

```java
package hello;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;

@Data
@JsonIgnoreProperties(ignoreUnknown = true)
public class Value {

    private Long id;
    private String quote;

}
```

This uses the same annotations but simply maps onto other data fields.

Make the application executable

Although it is possible to package this service as a traditional WAR file for deployment to an external application server, the simpler approach demonstrated below creates a standalone application. You package everything in a single, executable JAR file, driven by a good old Java `main()` method. Along the way, you use Spring's support for embedding the Tomcat servlet container as the HTTP runtime, instead of deploying to an external instance.

Now you can write the `Application` class that uses `RestTemplate` to fetch the data from our Spring Boot quotation service.

`src/main/java/hello/Application.java`

```
package hello;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplicat
import org.springframework.web.client.RestTemplate;
```

```
@SpringBootApplication
public class Application implements CommandLineRunner {

    private static final Logger log = LoggerFactory.getLogger(Ap

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

    @Override
    public void run(String... strings) throws Exception {
        RestTemplate restTemplate = new RestTemplate();
        Quote quote = restTemplate.getForObject("http://gturnqu
        log.info(quote.toString());
    }
}
```

Because the Jackson JSON processing library is in the classpath, `RestTemplate` will use it (via a message converter) to convert the incoming JSON data into a `Quote` object. From there, the contents of the `Quote` object will be logged to the console.

Here you've only used `RestTemplate` to make an HTTP `GET` request. But `RestTemplate` also supports other HTTP verbs such as `POST`, `PUT`, and `DELETE`.

Build an executable JAR

If you are using Gradle, you can run the application using `./gradlew bootRun`.

You can build a single executable JAR file that contains all the necessary dependencies, classes, and resources. This makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

```
./gradlew build
```

Then you can run the JAR file:

```
java -jar build/libs/gs-consuming-rest-0.1.0.jar
```

If you are using Maven, you can run the application using `mvn spring-boot:run`. Or you can build the JAR file with `mvn clean package` and run the JAR by typing:

```
java -jar target/gs-consuming-rest-0.1.0.jar
```

Note: The procedure above will create a runnable JAR. You can also opt to build a classic WAR file instead.

You should see the following output:

```
2015-06-22 13:55:47.688  INFO 71932 --- [main] hello.Applicatior
```

Summary

Congratulations! You have just developed a simple REST client using Spring.