01

Consumindo e testando um webservice

Transcrição

Serviços web estão por toda a parte, já que hoje a quantidade de aplicações, dos mais diferentes domínios e tecnologias, precisam se comunicar. Por exemplo, uma loja virtual precisa consultar os Correios para saber o valor de determinado frete, ou uma aplicação de compra de passagem aérea que fala com a aplicação de reserva de carros para fazer um pacote turístico.

Existem diferentes maneiras para se distribuir sistemas e fazê-los conversar entre si. As maneiras mais conhecidas são SOAP e REST. Neste curso, lidaremos com serviços REST, já que essa tem sido a preferência de uma boa parte da indústria ultimamente. Aplicações REST fazem uso de requisições e respostas HTTP simples, geralmente trafegando dados em XML ou JSON.

Como em toda nossa formação de testes, continuaremos com a nossa aplicação de leilão. Dessa vez, nossa aplicação possui um conjunto de serviços web para disponibilizar todos os seus recursos, como usuários, leilões e lances dados. Temos um monte de webservices prontos para serem testados. E, da mesma forma que no curso de Selenium, essa aplicação está pronta para você, e tudo que deve fazer é rodá-la. Aqui, focaremos apenas nos testes.

Baixe a aplicação aqui (http://s3.amazonaws.com/caelum-online-public/rest-assured/leiloes-ws.zip). Para rodá-la, basta ir no terminal, e digitar a seguinte instrução:

```
ant jetty.run
```

Pronto! A aplicação está de pé. Vamos agora começar a testá-la. Abra seu browser, e acesse o seguinte endereço: http://localhost:8080/usuarios?_format=xml (http://localhost:8080/usuarios?_format=xml) . A aplicação está preparada para responder em XML e JSON, se você passar o parâmetro `_format` para ela. Mais pra frente, veremos também que ela responderá certo caso passemos o parâmetro `Accept` no header do HTTP.

Veja o retorno. Existem alguns usuários já pré-cadastrados e o XML voltou corretamente:

```
<list>
<usuario>
<id>1</id>
<nome>Mauricio Aniche</nome>
<email>mauricio.aniche@caelum.com.br</email>
</usuario>
<usuario>
<id>2</id>
<nome>Guilherme Silveira</nome>
<email>guilherme.silveira@caelum.com.br</email>
</usuario>
</list>
```

Excelente. É assim que um serviço Web em REST funciona: fazemos uma requisição e ele nos devolve uma resposta. Precisamos agora testar que sempre que invocarmos esse serviço web, ele nos devolverá usuários. Para escrever esse teste, faremos uso do framework chamado **Rest Assured**. Ele, junto com o JUnit, nos ajudará a escrever testes que consomem serviços web. Ele já tem um monte de métodos que nos ajudam a fazer requisições, ler respostas, e etc. O

Rest-Assured pode ser encontrado aqui: https://code.google.com/p/rest-assured/ (https://code.google.com/p/rest-assured/).

Para facilitar nosso trabalho, baixe aqui (https://s3.amazonaws.com/caelum-online-public/rest-assured/leiloes-rest-assured.zip) um projeto que já contém todas as bibliotecas do Rest-Assured incluídos (veja que são várias!), bem como os modelos já escritos. Veja que, como vamos trafegar Usuários, Leilões, e Lances de um lado para o outro, precisamos ter a representação em Java deles. São classes Java convencionais, por exemplo:

```
public class Usuario{

    private Long id;
    private String nome;
    private String email;

        // getters e setters
}
```

Vamos começar nosso primeiro teste. O que ele fará é justamente uma requisição do tipo "GET" para o servidor, e garantirá que a lista com 2 usuários foi recuperada.

Escrever testes já não é mais segredo nessa altura. Usaremos JUnit como sempre. Mas aqui, faremos uso da API do Rest-Assured para fazer essa requisição. A API do framework é fluente, e faz muito uso de métodos estáticos. Vamos importá-los todos desde já:

```
import static com.jayway.restassured.RestAssured.*;
import static com.jayway.restassured.matcher.RestAssuredMatchers.*;
import static org.hamcrest.Matchers.*;
```

O primeiro método que vamos aprender é o método `get(URL)`. Ele faz uma requisição do tipo GET para a URL. Em seguida, precisamos dizer a ele que queremos tratar a resposta como XML. Veja só como fica a linha:

```
public class UsuariosWSTest {

    @Test
    public void deveRetornarListaDeUsuarios() {
        XmlPath path = get("/usuarios?_format=xml").andReturn().xmlPath();
    }
}
```

Com esse objeto `XmlPath`, podemos agora pegar os dados desse XML. Por exemplo, sabemos que essa URL nos devolve 2 usuários. Vamos então recuperá-los. Para isso, usaremos o método `getObject()`, que recebe um caminho dentro do XML, e a classe que ele deve desserializar:

```
    @Test
    public void deveRetornarListaDeUsuarios() {
        XmlPath path = get("/usuarios?_format=xml").andReturn().xmlPath();
        Usuario usuario1 = path.getObject("list.usuario[0]", Usuario.class);
        Usuario usuario2 = path.getObject("list.usuario[1]", Usuario.class);
    }
```

Com esses dois objetos em mãos, basta agora fazermos asserções neles:

```
@Test
public void deveRetornarListaDeUsuarios() {
    XmlPath path = get("/usuarios?_format=xml").andReturn().xmlPath();
    Usuario usuario1 = path.getObject("list.usuario[0]", Usuario.class);
    Usuario usuario2 = path.getObject("list.usuario[1]", Usuario.class);

    Usuario esperado1 = new Usuario(1L, "Mauricio Aniche", "mauricio.aniche@caelum.com.br");
    Usuario esperado2 = new Usuario(2L, "Guilherme Silveira", "guilherme.silveira@caelum.cor

    assertEquals(esperado1, usuario1);
    assertEquals(esperado2, usuario2);

}
```

Pronto. Nosso teste passa! Veja que com o uso do Rest-Assured, não gastamos tempo algum fazendo requisições, ou mesmo parseando as respostas. Nos preocupamos apenas com o comportamento esperado.

Como falamos anteriormente, podemos passar a informação de que esperamos a resposta em XML pelo próprio header HTTP. Isso também é fácil com Rest-Assured. Veja o código abaixo, que faz uso dos métodos `given()` e `header()`:

```
@Test
public void deveRetornarListaDeUsuarios() {
    XmlPath path = given()
            .header("Accept", "application/xml")
            .get("/usuarios")
            .andReturn().xmlPath();

    Usuario usuario1 = path.getObject("list.usuario[0]", Usuario.class);
    Usuario usuario2 = path.getObject("list.usuario[1]", Usuario.class);

    Usuario esperado1 = new Usuario(1L, "Mauricio Aniche", "mauricio.aniche@caelum.com.br");
    Usuario esperado2 = new Usuario(2L, "Guilherme Silveira", "guilherme.silveira@caelum.cor

    assertEquals(esperado1, usuario1);
    assertEquals(esperado2, usuario2);

}
```

Nosso teste continua passando. Esses são os primeiros passos com o Rest-Assured.