Geeks México

BLOG DE PROGRAMACIÓN EN ESPAÑOL SOBRE JAVA, FRAMEWORKS, BASES DE DATOS, CÓMPUTO EN LA NUBE, ETC. EN ESPAÑOL Y EN INGLÉS.

Aprende a consumir servicios REST

vía HTTPS leyendo precios de criptomonedas

HACE 1 HORA DEJA UN COMENTARIO

Una de las tareas más comunes de un desarrollador es consumir servicios REST vía HTTPS, en este post se explicará lo siguiente:

- Generar un certificado HTTPS
- Instalar el certificado en la máquina virtual
- Crear un cliente REST
- Hacer una petición para obtener el valor de criptomonedas

Paso 1 Analizar el api a consumir

En este ejemplo leeremos información sobre criptomonedas, para hacerlo utilizaremos el api REST de **Bitso** (Empresa dedicada a la compra y venta de bitcoins, ripple, litecoin, entre otras criptomonedas). Para más información sobre los endpoints disponibles ver el siguiente link Documentación Bitso. (https://bitso.com/api_info) En este ejemplo se consumirá un servicio para obtener el precio del bitcoin y ripple para esto se utilizará el endpoint ticker (https://bitso.com/api_info#ticker) con los siguientes parámetros:

- **book=xrp_mxn** : Para obtener el precio del ripple(https://api.bitso.com/v3/ticker?book=xrp_mxn)
- **book=btc_mxn :** Para obtener el precio del bitcoin(https://api.bitso.com/v3/ticker?book=btc_mxn)

Es importante mencionar que los precios mostrados en este sitio son en pesos mexicanos.

Paso 2 Configurar el projecto

Una vez que entendemos el api que se va a consumir que conocemos los endpoints que se ejecutarán, el siguiente paso será crear un proyecto, para esto se creará un proyecto Maven e incluiremos las siguientes dependencias:

```
<dependencies>
    <dependency>
        <groupId>org.apache.storm</groupId
        <artifactId>storm-core</artifactId
        <version>1.0.1</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.core
        <artifactId>jersey-client</artifac
        <version>2.17</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.medi
        <artifactId>jersey-media-json-jack
        <version>2.17</version>
    </dependency>
</dependencies>
```

Y el siguiente plugin de Maven:

```
<build>
    <plugins>
        <plugin>
            <artifactId>maven-compiler-plu
            <version>3.2</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
```

```
    </plugins>
</build>
```

Con estas dependencias y este plugin tendremos la versión de Java 8 y las dependencias necesarias para trabajar con Jersey client (El api que se utilizará para consumir servicios).

Paso 3 Creando modelo de la aplicación

Un paso importante al crear un cliente REST es des serializar la respuesta a objetos java, para esto debemos crear una representación del JSON en clases Java.

Json:

```
{
    success: true,
    payload: {
        high: "15.13",
        last: "14.69",
        created_at: "2017-12-18T14:13:36+0
        book: "xrp_mxn",
        volume: "1700513.79486861",
        vwap: "14.14839779",
        low: "14.07",
        ask: "14.69",
        bid: "14.50"
    }
}
```

Clases Java:

PayLoad.java

```
import com.fasterxml.jackson.annotation.J

/**
 * @author raidentrance
 *
 */
public class Payload {

    @JsonProperty("high")
    private double high;

    @JsonProperty("last")
```

```
13      private double last;
14
15      @JsonProperty("created_at")
16      private String createdAt;
17
18      @JsonProperty("book")
19      private String book;
20
21      @JsonProperty("volume")
22      private Double volume;
23
24      @JsonProperty("vwap")
25      private Double vwap;
26
27      @JsonProperty("low")
28      private Double low;
29
30      @JsonProperty("ask")
31      private Double ask;
32
33      @JsonProperty("bid")
34      private Double bid;
35
36      public double getHigh() {
37          return high;
38      }
39
40      public void setHigh(double high) {
41          this.high = high;
42      }
43
44      public double getLast() {
45          return last;
46      }
47
48      public void setLast(double last) {
49          this.last = last;
50      }
51
52      public String getCreatedAt() {
53          return createdAt;
54      }
55
56      public void setCreatedAt(String creat
57          this.createdAt = createdAt;
58      }
59
60      public String getBook() {
61          return book;
62      }
63
64      public void setBook(String book) {
65          this.book = book;
66      }
67
68      public Double getVolume() {
69          return volume;
70      }
71
72      public void setVolume(Double volume)
73          this.volume = volume;
74      }
```

```
75
76      public Double getVwap() {
77          return vwap;
78      }
79
80      public void setVwap(Double vwap) {
81          this.vwap = vwap;
82      }
83
84      public Double getLow() {
85          return low;
86      }
87
88      public void setLow(Double low) {
89          this.low = low;
90      }
91
92      public Double getAsk() {
93          return ask;
94      }
95
96      public void setAsk(Double ask) {
97          this.ask = ask;
98      }
99
100     public Double getBid() {
101         return bid;
102     }
103
104     public void setBid(Double bid) {
105         this.bid = bid;
106     }
107
108     @Override
109     public String toString() {
110         return "Payload [high=" + high +
111                 + volume + ", vwap=" + vw
112     }
113
114 }
```

CoinPrice.java

```
1  /**
2   * @author raidentrance
3   *
4   */
5  public class CoinPrice {
6      private boolean success;
7      private Payload payload;
8
9      public boolean isSuccess() {
10         return success;
11     }
12
13     public void setSuccess(boolean succes
14         this.success = success;
15     }
16
17     public Payload getPayload() {
```

```
18          return payload;
19      }
20
21      public void setPayload(Payload payloa
22          this.payload = payload;
23      }
24
25      @Override
26      public String toString() {
27          return "RipplePrice [success=" + s
28      }
29
30  }
```

Paso 4 Creando el cliente REST

El siguiente paso es empezar a escribir código, lo primero que escribiremos será un AbstractClient, que servirá para dar soporte para escribir multiples clientes REST:

AbstractClient.java

```
1   import java.util.logging.Logger;
2
3   import javax.ws.rs.client.Client;
4   import javax.ws.rs.client.ClientBuilder;
5   import javax.ws.rs.client.WebTarget;
6
7   /**
8    * @author raidentrance
9    *
10   */
11  public class AbstractClient {
12      private String url;
13      private String contextPath;
14
15      private static final Logger log = Logg
16
17      public AbstractClient(String url, Stri
18          this.url = url;
19          this.contextPath = contextPath;
20      }
21
22      protected WebTarget createClient(Strin
23          String assembledPath = assembleEnd
24          Client client = ClientBuilder.newC
25          WebTarget target = client.target(a
26          return target;
27      }
28
29      private String assembleEndpoint(String
30          String endpoint = url.concat(conte
31          log.info(String.format("Calling en
32          return endpoint;
```

```
33      }
34  }
```

La clase AbstractClient será la clase base que se utilizará para todos los clientes que creemos, cuenta con dos métodos principales:

- *assembleEndpoint(String path) : Construye la url a ejecutar*
- *WebTarget createClient(String path) : Crea el cliente HTTP para invocar la petición REST, en caso de requerir autenticar la petición este es el lugar para hacerlo.*

ApplicationEndpoint.java

```
1  /**
2   * @author raidentrance
3   *
4   */
5  public class ApplicationEndpoint {
6      private static String TICKER = "/ticke
7
8      public static String getCoinPrice(Stri
9          return TICKER.concat(String.format
10     }
11 }
```

La clase ApplicationEndpoint se utiliza para definir los endpoints a ejecutar en la aplicación.

ServiceException.java

```
1  /**
2   * @author raidentrance
3   *
4   */
5  public class ServiceException extends Exce
6      private Integer httpStatusCode;
7
8      private static final long serialVersio
9
10     public ServiceException(String message
11         super(message);
12         this.httpStatusCode = httpStatusCo
13     }
14
```

```
    public Integer getHttpStatusCode() {
        return httpStatusCode;
    }

    public void setHttpStatusCode(Integer
        this.httpStatusCode = httpStatusCo
    }
}
```

La clase ServiceException se utilizará para propagar los errores en caso de que existan.

BitsoClient

```
import java.util.logging.Logger;

import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;

import com.raidentrance.client.endpoints.A
import com.raidentrance.client.error.Servi
import com.raidentrance.client.model.CoinP

/**
 * @author raidentrance
 *
 */
public class BitsoClient extends AbstractC
    private static final Logger log = Logg

    public BitsoClient(String url, String
        super(url, contextPath);
    }

    public CoinPrice getRipplePrice() thro
        log.info("Getting ripple price");
        WebTarget client = createClient(Ap
        Response response = client.request
        log.info("Status " + response.getS
        CoinPrice result = null;
        Integer status = response.getStatu
        if (Status.OK.getStatusCode() == s
            result = response.readEntity(C
        } else {
            throw new ServiceException(res
        }
        return result;
    }

    public CoinPrice getBitcoinPrice() thr
        log.info("Getting ripple price");
        WebTarget client = createClient(Ap
        Response response = client.request
        log.info("Status " + response.getS
        CoinPrice result = null;
        Integer status = response.getStatu
        if (Status.OK.getStatusCode() == s
```

```
46              result = response.readEntity(C
47          } else {
48              throw new ServiceException(res
49          }
50          return result;
51      }
52
53  }
```

*Ahora es tiempo de crear el cliente REST en este caso será la clase **BitsoClient** como se puede ver recibe los siguientes parámetros en el constructor:*

- *url : Representa la url a consumir*
- *contextPath : Representa la url base de los servicios*

También cuenta con 2 métodos:

- *getRipplePrice() : Como su nombre lo indica devuelve un objeto con el precio actual de la criptomoneda ripple.*
- *getBitcoinPrice() : Como su nombre lo indica devuelve un objeto con el precio actual de la criptomoneda bitcoin.*

TestClient.java

```
1   import com.raidentrance.client.error.Servi
2   import com.raidentrance.client.model.CoinP
3
4   /**
5    * @author raidentrance
6    *
7    */
8   public class TestClient {
9       public static void main(String[] args)
10          BitsoClient client = new BitsoClie
11          CoinPrice ripplePrice = client.get
12          CoinPrice bitcoin = client.getBitc
13          System.out.println(String.format('
14          System.out.println(String.format('
15      }
16  }
```

La clase TestClient se utilizará para mostrar en consola los valores obtenidos por las API's REST.

Ejecutando la aplicación

Si ejecutamos la aplicación y no contamos con el certificado para ejecutar la petición recibiremos el siguiente error:

```
SunCertPathBuilderException: unable to find
```

Así que utilizaremos una clase llamada **InstallCert.java** creada por el equipo de sun que puedes encontrar aquí (https://github.com/escline/InstallCert/blob/master/InstallCert.java), ejecutarla pasando como parámetro la url de la cual deseas extraer el certificado con el siguiente comando :

```
java InstallCert api.bitso.com
```

El cuál mostrará la siguiente salida:

```
Opening connection to api.bitso.com:443...
Starting SSL handshake...

No errors, certificate is already trusted

Server sent 3 certificate(s):

 1 Subject CN=ssl511101.cloudflaressl.com,
   Issuer  CN=COMODO ECC Domain Validation
   sha1    11 bd 1f 03 0a b4 07 8c d3 f1 4
   md5     33 1e 1d 7f 95 09 de 0b 0b b8 3

 2 Subject CN=COMODO ECC Domain Validation
   Issuer  CN=COMODO ECC Certification Aut
   sha1    75 cf d9 bc 5c ef a1 04 ec c1 0
   md5     5e 0e 41 9b 20 ea 57 54 77 f1 1

 3 Subject CN=COMODO ECC Certification Aut
   Issuer  CN=AddTrust External CA Root, O
   sha1    ae 22 3c bf 20 19 1b 40 d7 ff b
   md5     c7 90 a5 6c 69 cb af 0b f3 f3 0

<strong>Enter certificate to add to truste
```

Oprimiremos la tecla de 1 y enter.

Ejecutaremos de nuevo el comando, seleccionaremos 1 de nuevo y notaremos que la salida es ahora la siguiente:

```
Added certificate to keystore 'jssecacerts'
```

Este comando generará un archivo llamado **jssecacerts**, el último paso será copiar ese archivo al directorio **$JAVA_HOME\jre\lib\security** y listo, tu aplicación podrá hacer peticiones https a el dominio de bitso.

El último paso será ejecutar nuestra aplicación para validar que todo funciona bien y generará la siguiente salida:

```
dic 18, 2017 10:37:04 AM com.raidentrance.
INFORMACIÓN: Getting ripple price
dic 18, 2017 10:37:04 AM com.raidentrance.
INFORMACIÓN: Calling endpoint https://api.
dic 18, 2017 10:37:05 AM com.raidentrance.
INFORMACIÓN: Status 200
dic 18, 2017 10:37:05 AM com.raidentrance.
INFORMACIÓN: Getting ripple price
dic 18, 2017 10:37:05 AM com.raidentrance.
INFORMACIÓN: Calling endpoint https://api.
dic 18, 2017 10:37:06 AM com.raidentrance.
INFORMACIÓN: Status 200
<strong>Ripple price RipplePrice [success=
Bitcoin price RipplePrice [success=true, p
```

Como se puede ver se imprimieron los precios de las criptomonedas de forma exitosa, ya dependerá de ti la aplicación que crees con esto.

Conclusión

En este post se tocaron temas importantes como:

- Configurar un proyecto para hacer peticiones REST con Jersey

- Crear un patrón de diseño para la construcción de clientes REST
- Ejecutar peticiones REST vía HTTPS
- Obtener el valor en pesos de las criptomonedas Bitcoin y Ripple

Si te gusta el contenido y quieres enterarte cuando realicemos un post nuevo síguenos en nuestras redes sociales https://twitter.com/geeks_mx (https://twitter.com/geeks_mx) y https://www.facebook.com/geeksJavaMexico/ (https://www.facebook.com/geeksJavaMexico/).

Autor: Alejandro Agapito Bautista

Twitter: @raidentrance

Contacto:raidentrance@gmail.com