Geeks México

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

Consuming RESTful web services in Java

with Jersey and FailSafe

HACE 1 SEMANA DEJA UN COMENTARIO

One of the most common tasks is to consume RESTful web services in Java, in this post I will explain a design pattern using Jersey client.

Step 1 : Configuration

The project will include the dependencies described in the following pom.xml (https://github.com/raidentrance/rest-jersey-client/blob/master/pom.xml), as you can see there we have the following support:

- Jackson : For serialization and deserialization
- slf4j : For logging
- failsafe: Api for retry logic

And at the end we can see we have the maven-compiler-plugin to define the Java version we are using.

Step 2 : Creating an abstract client

The next step will be to create an AbstractClient, you can use it in any rest client you want, lets see and analyze the code:

```
1 import java.io.IOException;
2 import java.util.Optional;
3 import java.util.concurrent.TimeUnit;
4 import java.util.logging.Logger;
5
6 import javax.ws.rs.client.Client;
7 import javax.ws.rs.client.ClientBuilder;
```

```
 8  import javax.ws.rs.client.Entity;
 9  import javax.ws.rs.client.WebTarget;
10  import javax.ws.rs.core.Response;
11  import javax.ws.rs.core.Response.Status;
12
13  import com.fasterxml.jackson.core.JsonPar
14  import com.fasterxml.jackson.core.type.Ty
15  import com.fasterxml.jackson.databind.Jso
16
17  import net.jodah.failsafe.Failsafe;
18  import net.jodah.failsafe.RetryPolicy;
19
20  /**
21   * @author raidentrance
22   *
23   */
24  public abstract class AbstractClient {
25      private String url;
26      private String contextPath;
27      private RetryPolicy defaultRetryPolic
28
29      private static final Logger log = Log
30
31      public AbstractClient(String url, Str
32          this.url = url;
33          this.contextPath = contextPath;
34      }
35
36      public AbstractClient(String url, Str
37          this.url = url;
38          this.contextPath = contextPath;
39          defaultRetryPolicy = new RetryPol
40              if (resp != null) {
41                  return resp.getStatusInfo
42              } else {
43                  return false;
44              }
45          }).withDelay(delay, unit).withMax
46      }
47
48      protected WebTarget createClient(Stri
49          String assembledPath = assembleEn
50          Client client = ClientBuilder.new
51          return client.target(assembledPat
52      }
53
54      private String assembleEndpoint(Strin
55          String endpoint = url.concat(cont
56          log.info(String.format("Calling e
57          return endpoint;
58      }
59
60      /**
61       * Execute a GET http request over th
62       * type specified
63       *
64       * @param endpoint
65       *            Defines the endpoint th
66       * @param type
67       *            Defines the content typ
68       * @return A response object with the
69       */
```

```
70      protected Response get(String endpoi
71          WebTarget client = createClient(e
72          Optional result = getDefaultRetry
73          if (result.isPresent()) {
74              return Failsafe.with(result.g
75          } else {
76              return client.request(type).g
77          }
78      }
79
80      /**
81       * Execute a PUT request over the enc
82       * specified and sending the object r
83       *
84       * @param endpoint
85       *            Endpoint will be execut
86       * @param type
87       *            Defines the content typ
88       * @param entity
89       *            Object will be sent in
90       * @return A response object with the
91       */
92      protected Response put(String endpoi
93          WebTarget client = createClient(e
94          Optional result = getDefaultRetry
95          if (result.isPresent()) {
96              return Failsafe.with(result.g
97          } else {
98              return client.request(type).p
99          }
100     }
101
102     /**
103      * Execute a POST request over the en
104      * specified and sending the object r
105      *
106      * @param endpoint
107      *            Endpoint will be execut
108      * @param type
109      *            Defines the content typ
110      * @param entity
111      *            Object will be sent in
112      * @return A response object with the
113      */
114     protected Response post(String endpoi
115         WebTarget client = createClient(e
116         Optional result = getDefaultRetry
117         if (result.isPresent()) {
118             return Failsafe.with(result.g
119                     .get((response) -> cl
120         } else {
121             return client.request(type).p
122         }
123     }
124
125     /**
126      * If there is a default retry policy
127      *
128      * @return
129      */
130     public Optional getDefaultRetryPolicy
131         return (defaultRetryPolicy != nul
```

```
132         }
133
134         /**
135          * Modify the current default retry p
136          *
137          * @param maxRetries
138          *            Number of times that it
139          * @param delay
140          *            The time that will wait
141          * @param unit
142          *            Unit of the time of the
143          *            etc.
144          */
145         public void setDefaultRetryPolicy(int
146             defaultRetryPolicy = new RetryPol
147                     .retryIf((Response resp)
148                     .withDelay(delay, unit).w
149         }
150
151         /**
152          * Get a Response and Parse to type T
153          * Exception
154          *
155          * @param response
156          *            the HTTP response
157          * @param entityType
158          *            is a generic type that
159          * @param
160          *            The Generic Type that t
161          * @return T
162          * @throws Exception
163          *             if a problem occurs
164          * @throws IOException
165          * @throws JsonMappingException
166          * @throws JsonParseException
167          */
168         protected abstract  T parseResponse(R
169     }
```

Now lets analyze the class:

- Constructors: There are two ways to construct our object
    - Only with the url and the context path
    - Including also the maxRetries, delay and TimeUnit
- **createClient**(String path) : Will create a Jersey client used to consume the api's.
- **assembleEndpoint**(String path) : Based on the Endpoint will build the url we are going to consume.

- Response **get**(String endpoint, String type) : Generic get method that receives two parameters an endpoint and a type.

- Response **put**(String endpoint, String type, Object entity): Generic put method that receives two parameters an endpoint and a type.

- Response **post**(String endpoint, String type, Object entity): Generic post method that receives two parameters an endpoint and a type.

- Optional **getDefaultRetryPolicy**() : if the parameters maxRetries, delay and TimeUnit were assigned it will return the retry policy to use.

- **setDefaultRetryPolicy**(int maxRetries, int delay, TimeUnit unit): You can use it if you want to change the default policy .

- abstract T **parseResponse**(Response response,TypeReference entityType) throws Exception: Your implementation has to override the parseResponse method to define how it will translate the response.

Step 3 : Define a class to set all the endpoints

Once we have our abstract class, we have to define an endpoints class, in our case it will be **ApplicationEndpoints,** it will centralize all the endpoints used in the client.

```
1  /**
2   * @author raidentrance
3   *
4   */
5  public class ApplicationEndpoints {
6      private static final String TICKER = '
```

```
7
8       private ApplicationEndpoints() {
9       }
10
11      public static String getTickers() {
12          return TICKER;
13      }
14
15      public static String getTickerByBook(S
16          return TICKER.concat("?book=").cor
17      }
18
19  }
```

In this case we are using the api of bitso to get prices of crypto currencies, you can see the documentation here (https://bitso.com/api_info?l=es#ticker) and query the api in the url https://api.bitso.com/v3/ticker/ (https://api.bitso.com/v3/ticker/)

Step 4 : Defining the error model

Now we have to handle errors, to do it we will analyze the api we are querying, lets see the error model:

```
1  {
2      "success": false,
3      "error": {
4          "code": "0301",
5          "message": "Unknown OrderBook xrp_m
6      }
7  }
```

Now we have to translate it to java classes.
ErrorCode.java

```
1  import java.io.Serializable;
2
3  /**
4   * @author raidentrance
5   *
6   */
7  public class ErrorCode implements Seriali
8
9      private String code;
10     private String message;
11
12     private static final long serialVersio
13
14     public ErrorCode() {
15
```

```
    }

    public ErrorCode(String code, String m
        super();
        this.code = code;
        this.message = message;
    }

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message)
        this.message = message;
    }

}
```

ErrorMessage.java

```
import java.io.Serializable;

/**
 * @author raidentrance
 *
 */
public class ErrorMessage implements Seria
    private boolean success;
    private ErrorCode error;

    private static final long serialVersio

    public ErrorMessage() {

    }

    public ErrorMessage(boolean success, E
        this.success = success;
        this.error = error;
    }

    public boolean isSuccess() {
        return success;
    }

    public void setSuccess(boolean success
        this.success = success;
    }

    public ErrorCode getError() {
        return error;
    }
```

```
33
34      public void setError(ErrorCode error)
35          this.error = error;
36      }
37
38  }
```

Once we have both classes we are able to deserialize the errors to java objects, now we just need to create an exception to propagate the errors.

```
1   import com.raidentrance.rest.error.model.E
2
3   /**
4    * @author raidentrance
5    *
6    */
7   public class ServiceException extends Exce
8
9       private ErrorMessage errorMessage;
10
11      private static final long serialVersio
12
13      public ServiceException(ErrorMessage e
14          this.errorMessage = errorMessage;
15      }
16
17      public ErrorMessage getErrorMessage()
18          return errorMessage;
19      }
20
21  }
```

The ServiceException will be thrown when an error happens and it will contain the error message we receive from the api.

Step 5 : Creating the model

We defined the model for the errors, but now we have to define the model for our api's, in this case we will be reading tickers with the following structure:

Payload.java

```
1   import com.fasterxml.jackson.annotation.J
2
3   /**
4    * @author raidentrance
```

```
 *
 */
public class Payload {
    @JsonProperty("high")
    private String high;

    @JsonProperty("last")
    private String last;

    @JsonProperty("created_at")
    private String createdAt;

    @JsonProperty("book")
    private String book;

    @JsonProperty("volume")
    private String volume;

    @JsonProperty("vwap")
    private String vwap;

    @JsonProperty("low")
    private String low;

    @JsonProperty("ask")
    private String ask;

    @JsonProperty("bid")
    private String bid;

    public String getHigh() {
        return high;
    }

    public void setHigh(String high) {
        this.high = high;
    }

    public String getLast() {
        return last;
    }

    public void setLast(String last) {
        this.last = last;
    }

    public String getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(String creat
        this.createdAt = createdAt;
    }

    public String getBook() {
        return book;
    }

    public void setBook(String book) {
        this.book = book;
    }

```

```
67      public String getVolume() {
68          return volume;
69      }
70
71      public void setVolume(String volume)
72          this.volume = volume;
73      }
74
75      public String getVwap() {
76          return vwap;
77      }
78
79      public void setVwap(String vwap) {
80          this.vwap = vwap;
81      }
82
83      public String getLow() {
84          return low;
85      }
86
87      public void setLow(String low) {
88          this.low = low;
89      }
90
91      public String getAsk() {
92          return ask;
93      }
94
95      public void setAsk(String ask) {
96          this.ask = ask;
97      }
98
99      public String getBid() {
100         return bid;
101     }
102
103     public void setBid(String bid) {
104         this.bid = bid;
105     }
106
107     @Override
108     public String toString() {
109         return "Payload [high=" + high +
110                 + volume + ", vwap=" + vw
111     }
112
113 }
```

Ticker.java

```
1  import com.fasterxml.jackson.annotation.Js
2
3  /**
4   * @author raidentrance
5   *
6   */
7  public class Ticker {
8      @JsonProperty("success")
9      private boolean success;
10
```

```
    @JsonProperty("payload")
    private Payload payload;

    public boolean isSuccess() {
        return success;
    }

    public void setSuccess(boolean succes
        this.success = success;
    }

    public Payload getPayload() {
        return payload;
    }

    public void setPayload(Payload payloa
        this.payload = payload;
    }

    @Override
    public String toString() {
        return "Ticker [success=" + succe
    }

}
```

TickerList.java

```
import java.util.List;

import com.fasterxml.jackson.annotation.Js

/**
 * @author maagapi
 *
 */
public class TickerList {
    @JsonProperty("success")
    private boolean success;

    @JsonProperty("payload")
    private List payload;

    public boolean isSuccess() {
        return success;
    }

    public void setSuccess(boolean succes
        this.success = success;
    }

    public List getPayload() {
        return payload;
    }

    public void setPayload(List payload) {
        this.payload = payload;
    }

    @Override
```

```
33      public String toString() {
34          return "TickerList [success=" + su
35      }
36  }
```

Once we defined the model, we can call the api's.

Step 6 : Creating the RestClient

The last step will be create the RestClient this class will be the responsible to join all the pieces:

```
1   import java.io.IOException;
2   import java.io.StringReader;
3   import java.util.concurrent.TimeUnit;
4
5   import javax.ws.rs.core.MediaType;
6   import javax.ws.rs.core.Response;
7   import javax.ws.rs.core.Response.Status;
8
9   import org.slf4j.Logger;
10  import org.slf4j.LoggerFactory;
11
12  import com.fasterxml.jackson.core.type.Typ
13  import com.fasterxml.jackson.databind.Obje
14  import com.raidentrance.rest.commons.Abstr
15  import com.raidentrance.rest.endpoints.App
16  import com.raidentrance.rest.error.excepti
17  import com.raidentrance.rest.error.model.E
18  import com.raidentrance.rest.error.model.E
19  import com.raidentrance.rest.model.Ticker;
20  import com.raidentrance.rest.model.TickerL
21
22  /**
23   * @author raidentrance
24   *
25   */
26  public class RestClient extends AbstractCl
27
28      private static final Logger log = Logg
29
30      public RestClient(String url, String c
31          super(url, contextPath);
32      }
33
34      public RestClient(String url, String c
35          super(url, contextPath, maxRetries
36      }
37
38      public TickerList getTickers() throws
39          return parseResponse(get(Applicati
40                  new TypeReference() {
41                  });
42      }
43
44      public Ticker getTickerByBook(String b
```

```
45          return parseResponse(get(Applicati
46                  new TypeReference() {
47                  });
48      }
49
50      @Override
51      protected  T parseResponse(Response re
52          int status = response.getStatus();
53          log.info("Status {}", status);
54          if (response.getStatusInfo().getFa
55              try {
56                  return new ObjectMapper().
57              } catch (IOException e) {
58                  throw new ServiceExceptior
59                          new ErrorMessage(t
60              }
61          } else {
62              throw new ServiceException(res
63          }
64      }
65
66  }
```

As you can see now just with one line of code we make an http request, handle the retries and parse the response.

Step 7 : Test it

In other posts we will show how to write unit tests for this kind of components, now lets just create a main method to test it:

```
1   public static void main(String[] args) thr
2           RestClient client = new RestClient
3           TickerList tickers = client.getTic
4           log.info("Getting tickers ");
5           for (Payload payload : tickers.get
6               log.info(payload.toString());
7           }
8           log.info("Getting ripple ticker");
9           Ticker ripple = client.getTickerBy
10          log.info(ripple.toString());
11          log.info("Not existing ticker");
12          Ticker alex = client.getTickerByBo
13          log.info(alex.toString());
14
15      }
```

If we execute the code we will see the following output (It can be different according with the ripple price :P):

```
mar 12, 2018 5:09:02 PM com.raidentrance.r
INFORMACIÓN: Calling endpoint https://api.
[main] INFO com.raidentrance.rest.RestClie
[main] INFO com.raidentrance.rest.RestClie
[main] INFO com.raidentrance.rest.RestClie
[main] INFO com.raidentrance.rest.RestClie
[main] INFO com.raidentrance.rest.RestClie
[main] INFO com.raidentrance.rest.RestClie
[main] INFO com.raidentrance.rest.RestClie
[main] INFO com.raidentrance.rest.RestClie
[main] INFO com.raidentrance.rest.RestClie
[main] INFO com.raidentrance.rest.RestClie
mar 12, 2018 5:09:03 PM com.raidentrance.r
INFORMACIÓN: Calling endpoint https://api.
[main] INFO com.raidentrance.rest.RestClie
[main] INFO com.raidentrance.rest.RestClie
[main] INFO com.raidentrance.rest.RestClie
mar 12, 2018 5:09:03 PM com.raidentrance.r
INFORMACIÓN: Calling endpoint https://api.
[main] INFO com.raidentrance.rest.RestClie
Exception in thread "main" com.raidentranc
    at com.raidentrance.rest.RestClient.pa
    at com.raidentrance.rest.RestClient.ge
    at com.raidentrance.rest.RestClient.ma
```

As you can see we are testing the following cases:

- Get the price of all the crypto currencies
- Get the prices of one crypto currency
- Ask for a price that doesn't exist and throw an exception with the message.

You can find all the code of this post in the following repository https://github.com/raidentrance/rest-jersey-client (https://github.com/raidentrance/rest-jersey-client).

If you get an **SunCertPathBuilderException: unable to find valid certification path to requested target** remember that you have to install the certificate to do https requests you can see a guide about how to do it here (https://www.mkyong.com/webservices/jax-ws/suncertpathbuilderexception-unable-to-find-valid-certification-path-to-requested-target/).

If you like our posts follow us in our social networks 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

Contact: raidentrance@gmail.com