

Accès à une API Rest en java

-- Ressources

Nous avons besoin d'une librairie pour lire du JSON et le convertir en objet Java, et inversement :

- [Google GSON](#)

Nous avons également besoin d'effectuer des requêtes (GET, POST, PUT, DELETE...) vers le server Http exposant l'api Rest :

- [Apache Http components](#)

-- JSON

-- Modèle

Créer une classe **Model** qui nous servira de classe métier :

- Générer
 - le constructeur par défaut
 - les accesseurs
 - la méthode toString

```
public class Model {
    private int id;
    private String name;
    private boolean access;
    private Date date;
    public Model() {
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public boolean isAccess() {
        return access;
    }
    public void setAccess(boolean access) {
        this.access = access;
    }
    public Date getDate() {
```

```

        return date;
    }
    public void setDate(Date date) {
        this.date = date;
    }

    @Override
    public String toString() {
        return "Model [id=" + id + ", name=" + name + ", access=" + access
            + ", date=" + date + " ]";
    }
}

```

-- Classe de test

Créer une classe de test qui va nous permettre de tester GSON :

L'objet **Gson** instancié dans le constructeur nous permettra d'effectuer la conversion dans les 2 sens :

```

public class TestJSON {
    private Gson gson;
    public TestJSON() {
        gson = new GsonBuilder()
            .setDateFormat("yyyy-MM-dd HH:mm:ss")
            .create();
    }
}

```

-- De JSON au model

Créer la méthode suivante retournant une instance de Model construite à partir d'une chaîne JSON :

```

...
public Model jsonToModel(String jsonString){
    return gson.fromJson(jsonString, Model.class);
}
...

```

Ajouter la méthode Main dans la classe pour la tester :

```

...
public static void main(String args[]){
    TestJSON jsonTest=new TestJSON();
    String jsonStr="{id:1,name:'nom',access:true,date:'2015-03-15 19:22:00'}";
    Model m=jsonTest.jsonToModel(jsonStr);
    System.out.println(m);
}
...

```

L'exécution doit retourner :

```
Model [id=1, name=nom, access=true, date=Sun Mar 15 19:22:00 CET 2015]
```

-- Du model à JSON

Ajouter la méthode suivante retournant une chaîne JSON construite à partir d'une instance de la classe **Model** :

```
...
public String modelToJson(Model m){
    return gson.toJson(m).toString();
}
...
```

Ajouter dans la méthode Main le code suivant :

```
...
public static void main(String args[]){
    TestJSON jsonTest=new TestJSON();
    String jsonStr="{id:1,name:'nom',access:true,date:'2015-03-15 19:22:00'}";
    Model m=jsonTest.jsonToModel(jsonStr);
    System.out.println(m);
    m.setName("Autre nom");
    System.out.println(jsonTest.modelToJson(m));
}
...
```

L'exécution doit retourner :

```
Model [id=1, name=nom, access=true, date=Sun Mar 15 19:22:00 CET 2015]
{"id":1,"name":"Autre nom","access":true,"date":"2015-03-15 19:22:00"}
```

-- Http requests

Pour mettre en oeuvre les tests, vous devez disposer d'un serveur HTTP hébergeant un service Rest.

-- GET

Créer une classe TestHttp, instanciant un objet Gson qui nous servira pour les conversions JSON⇌Objet Java :

```
public class TestHttp {
    private Gson gson;
    public TestHttp() {
        gson = new GsonBuilder()
            .setDateFormat("yyyy-MM-dd HH:mm:ss")
```

```

        .create();
    }
}

```

Implémenter la méthode `getHttp` :

```

...
public String getHTML(String urlToRead) throws ClientProtocolException,
IOException {
    String result="";
    CloseableHttpClient httpClient = HttpClients.createDefault();
    try {
        HttpGet getRequest = new HttpGet(urlToRead);
        ResponseHandler<String> responseHandler = new BasicResponseHandler();
        result = httpClient.execute(getRequest, responseHandler);
    }finally {
        httpClient.close();;
    }
    return result;
}
...

```

Ajouter la méthode **main** dans la classe pour tester le Get, n'oubliez pas de démarrer le serveur :

```

...
public static void main(String args[]) {
    TestHttp test = new TestHttp();

    try {
        String result = test.getHTML("http://127.0.0.1/[restServer]");
        System.out.println(result);

    } catch (IOException e) {
        e.printStackTrace();
    }
}
...

```

-- POST

Implémenter la méthode `restPostJSON` :

```

...
public String restPostJSON(String urlToRead, Object o) throws
ClientProtocolException, IOException {
    String result = "";
    CloseableHttpClient httpClient = HttpClients.createDefault();
    try {
        HttpPost postRequest = new HttpPost(urlToRead);
        postRequest.setHeader("content-type", "application/json");
    }
}

```

```
        postRequest.setHeader("Accept", "application/json");
        String jsonString = gson.toJson(o);
        StringEntity params = new StringEntity(jsonString);
        params.setContentType("application/json");
        params.setContentEncoding("UTF-8");
        postRequest.setEntity(params);
        ResponseHandler<String> responseHandler = new BasicResponseHandler();
        result = httpClient.execute(postRequest, responseHandler);
    } finally {
        httpClient.close();
    }
    return result;
}
...
```

Modifier la méthode **main** de la classe pour tester le restPostJSON :

```
...
public static void main(String args[]) {
    TestHttp test = new TestHttp();

    try {
        ...
        System.out.println(test.restPostJSON(
            "http://127.0.0.1/[restServer]/mondes", new Monde("Nouveau")));
    } catch (IOException e) {
        e.printStackTrace();
    }
}
...
```

-- POST Classique

Le post classique est légèrement plus complexe, puisqu'il nécessite :

- La conversion en JsonObject de l'objet à poster
- L'envoi dans l'en-tête HTTP des couples nomDeMembre/valeur de l'objet
- La définition du content-type de la requête : **"application/x-www-form-urlencoded"**

Implémenter la méthode postJSON :

```
...
public String postJSON(String urlToRead, Object o)
    throws ClientProtocolException, IOException {
    String result = "";
    CloseableHttpClient httpClient = HttpClients.createDefault();
    try {
        HttpPost postRequest = new HttpPost(urlToRead);
        postRequest.setHeader("content-type", "application/x-www-form-
urlencoded");
    }
}
```

```
List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>();
JsonElement elm = gson.toJsonTree(o);
JsonObject jsonObj = elm.getAsJsonObject();
for (Map.Entry<String, JsonElement> entry : jsonObj.entrySet()) {
    nameValuePairs.add(new BasicNameValuePair(entry.getKey(), entry
        .getValue().getAsString()));
}
postRequest.setEntity(new UrlEncodedFormEntity(nameValuePairs));
ResponseHandler<String> responseHandler = new BasicResponseHandler();
result = httpClient.execute(postRequest, responseHandler);
} finally {
    httpClient.close();
}
return result;
}
...
```

Modifier la méthode **main** de la classe pour tester le POST classique, il s'agit ici d'un exemple avec une classe User :

```
...
public static void main(String args[]) {
    TestHttp test = new TestHttp();

    try {
        String result = test.getHTML("http://127.0.0.1/[restServer]");
        System.out.println(result);

        System.out.println(test.postJSON(
            "http://127.0.0.1/[restServer]/user/connect", new User(
                "admin@local.fr", "0000")));
    } catch (IOException e) {
        e.printStackTrace();
    }
}
...
```

From:
<http://slamwiki2.kobject.net/> - **SlamWiki 2.1**

Permanent link:
<http://slamwiki2.kobject.net/slam4/gui/rest?rev=1426533627>

Last update: **2019/08/31 14:39**

