eBook Gratuit

APPRENEZ gson

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#gson

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: gson

It is an unofficial and free gson ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official gson.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec gson

Remarques

Gson est une bibliothèque Java Open Source pouvant être utilisée pour convertir des objets Java en leur représentation JSON. Il peut également être utilisé pour convertir une chaîne JSON en un objet Java équivalent.

Buts pour Gson

- Fournit des mécanismes faciles à utiliser comme toString () et constructeur (méthode factory) pour convertir Java en JSON et vice-versa
- Autoriser les objets non modifiables préexistants à être convertis vers et depuis JSON
- Autoriser les représentations personnalisées pour les objets
- Soutenir des objets arbitrairement complexes
- Génère une sortie JSON compacte et lisible

Le code source de Gson est disponible sur Github .

Mode d'emploi

Examples

Installation

Pour utiliser Gson, vous devez l'inclure dans votre projet. Vous pouvez le faire en ajoutant la dépendance suivante de la version de Gson disponible dans Maven Central:

Maven

Ajouter à pom.xml

```
<dependencies>
    <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.8.0</version>
      <scope>compile</scope>
    </dependency>
</dependencies>
```

Gradle :

Ajouter à build.gradle

```
compile 'com.google.code.gson:gson:2.8.0'
```

Sérialisation et désérialisation

```
Gson gson = new Gson(); //Create a Gson object
MyType target = new MyType(); //This is the object you want to convert to JSON
String json = gson.toJson(target); // serializes target to Json
MyType target2 = gson.fromJson(json, MyType.class); // deserializes json into target2
```

Tableaux

JSON:

```
[
  {
    "id": 8484,
    "name": "David",
    "height": 173.2,
    "weight": 75.42
  },
  {
    "id": 8485,
    "name": "Ronald",
    "height": 183.73,
    "weight": 83.1
  }
]
```

Person.java

```
public class Person {
    public int id;
    public String name;
    public double height;
    public double weight;

    @Override
    public String toString() {
        return "[ id: " + String.valueOf(id) + ", name: " + name + ", height: " +
String.valueOf(height) + ", weight: " + String.valueOf(weight) + " ]";
    }
}
```

Usage:

```
Gson gson = new Gson();
Person[] persons = gson.fromJson(json, Person[].class);
for(Person person : persons)
    System.out.println(person.toString());
```

Sortie:

```
[ id: 8484, name: David, height: 173.2, weight: 75.42 ]
```

```
[ id: 8485, name: Ronald, height: 183.73, weight: 83.1 ]
```

Exemple simple

La bibliothèque Gson fournit `Gson.class` qui gère toutes les conversions entre les objets Java et JSON. Une instance de cette classe peut être créée en appelant le constructeur par défaut. Vous souhaitez généralement avoir une instance de Gson pour la plupart des opérations de votre programme.

```
Gson gson = new Gson();
```

Tout d'abord, nous devons créer une classe de notre objet avec lequel nous travaillerons

```
class Person {
   public String name;
   public int age;

   public Person(String name, int age){
       this.name = name;
       this.age = age;
   }
}
```

La classe Gson fournit des méthodes à `toJson` et `fromJson` qui sont les principaux points d'entrée pour les objets JSON et Java

Essayons de convertir un objet java en JSON et de nouveau en objet java

```
Person person = new Person("Jason", 29);
//using gson object which we created earlier
String json = gson.toJson(person);
System.out.println(json);
//Outputs: {"name": "Jason", "age": 29}
```

Et maintenant de retour

```
String json = "{\"name\": \"Jason\", \"age\": 29}";
Person person = gson.fromJson(json, Person.class);
System.out.println(person.age + "yo " + person.name + " walks into a bar");
//Outputs "29 yo Jason walks into a bar"
```

Convertir une chaîne en JsonObject sans POJO

```
String jsonStr = "{\"name\" : \"Abcd\", \"greeting\": \"Hello\", }"; //Sample Json String

Gson gson = new Gson(); // Creates new instance of Gson
JsonElement element = gson.fromJson (jsonStr, JsonElement.class); //Converts the json string
to JsonElement without POJO
JsonObject jsonObj = element.getAsJsonObject(); //Converting JsonElement to JsonObject

String name = jsonObj.get("name").getAsString(); //To fetch the values from json object
```

```
String greeting = jsonObj.get("greeting").getAsString();
```

Utiliser GSON avec héritage

GSON ne supporte pas l'héritage de notre boîte. Disons que nous avons la hiérarchie de classes suivante:

```
public class BaseClass {
    int a;

    public int getInt() {
        return a;
   }
}

public class DerivedClass1 extends BaseClass {
     int b;

     @Override
     public int getInt() {
         return b;
     }
 }

public class DerivedClass2 extends BaseClass {
    int c;

    @Override
    public int getInt() {
        return c;
    }
}
```

Et maintenant, nous voulons sérialiser une instance de `DerivedClass1` en une chaîne json

```
DerivedClass1 derivedClass1 = new DerivedClass1();
derivedClass1.b = 5;
derivedClass1.a = 10;

Gson gson = new Gson();
String derivedClass1Json = gson.toJson(derivedClass1);
```

Maintenant, à un autre endroit, nous recevons cette chaîne json et souhaitons la désérialiser - mais au moment de la compilation, nous savons seulement qu'elle est supposée être une instance de `BaseClass` :

```
BaseClass maybeDerivedClass1 = gson.fromJson(derivedClass1Json, BaseClass.class);
System.out.println(maybeDerivedClass1.getInt());
```

Mais GSON ne sait pas que: `derivedClass1Json` était à l'origine une instance de `DerivedClass1` , donc cela imprimera 10.

Comment résoudre ce problème?

Vous devez créer votre propre `JsonDeserializer` , qui gère de tels cas. La solution n'est pas parfaitement propre, mais je n'ai pas pu en trouver une meilleure.

Tout d'abord, ajoutez le champ suivant à votre classe de base

```
@SerializedName("type")
private String typeName;
```

Et l'initialiser dans le constructeur de la classe de base

```
public BaseClass() {
    typeName = getClass().getName();
}
```

Ajoutez maintenant la classe suivante:

```
public class JsonDeserializerWithInheritance<T> implements JsonDeserializer<T> {

 @Override
 public T deserialize(
     JsonElement json, Type typeOfT, JsonDeserializationContext context)
     throws JsonParseException {
     JsonObject jsonObject = json.getAsJsonObject();
     JsonPrimitive classNamePrimitive = (JsonPrimitive) jsonObject.get("type");

     String className = classNamePrimitive.getAsString();

     Class<?> clazz;
     try {
     clazz = Class.forName(className);
     } catch (ClassNotFoundException e) {
     throw new JsonParseException(e.getMessage());
     }
     return context.deserialize(jsonObject, clazz);
 }
}
```

Tout ce qui reste à faire est de tout brancher -

```
GsonBuilder builder = new GsonBuilder();
 builder
 .registerTypeAdapter(BaseClass.class, new JsonDeserializerWithInheritance<BaseClass>());
 Gson gson = builder.create();
```

Et maintenant, en exécutant le code suivant

```
 DerivedClass1 derivedClass1 = new DerivedClass1();
 derivedClass1.b = 5;
 derivedClass1.a = 10;
 String derivedClass1Json = gson.toJson(derivedClass1);

 BaseClass maybeDerivedClass1 = gson.fromJson(derivedClass1Json, BaseClass.class);
 System.out.println(maybeDerivedClass1.getInt());
```

Imprimera 5.

Lire Démarrer avec gson en ligne: https://riptutorial.com/fr/gson/topic/4804/demarrer-avec-gson

Chapitre 2: Utilisation de Gson avec JAX-RS (services Web RESTful)

Examples

Fournisseur JAX-RS pour utiliser Gson

Ceci est un JAX-RS `@Provider` pour utiliser Gson comme analyseur JSON. L'exemple montre également comment utiliser des convertisseurs de date / heure Java 8 personnalisés.

```
@Provider
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class JerseyServerGson
        implements MessageBodyWriter<Object>, MessageBodyReader<Object>
{
  @Override
  public boolean isReadable(Class<?> type,
                            Type genericType,
                            Annotation[] annotations,
                            MediaType mediaType)
  {
    return true;
  }

  @Override
  public Object readFrom(Class<Object> type,
                         Type genericType,
                         Annotation[] annotations,
                         MediaType mediaType,
                         MultivaluedMap<String, String> httpHeaders,
                         InputStream entityStream)
        throws IOException, WebApplicationException
  {
    try ( InputStreamReader input =
                new InputStreamReader(entityStream, "UTF-8") ) {
      Gson gson = getGson();
      return gson.fromJson(input, genericType);
    }
  }

  @NotNull
  private Gson getGson() {
    return new GsonBuilder()
            .registerTypeAdapter(LocalDateTime.class,
                                 new AdapterLocalDateTime().nullSafe())
            .registerTypeAdapter(LocalDate.class,
                                 new AdapterLocalDate().nullSafe())
            .setPrettyPrinting()
            .serializeNulls()
            .create();
  }

  @Override
```

```
  public boolean isWriteable(Class<?> type,
                             Type genericType,
                             Annotation[] annotations,
                             MediaType mediaType)
  {
    return true;
  }

  @Override
  public long getSize(Object o,
                      Class<?> type,
                      Type genericType,
                      Annotation[] annotations,
                      MediaType mediaType)
  {
    // Deprecated and ignored in Jersey 2
    return -1;
  }

  @Override
  public void writeTo(Object o,
                      Class<?> type,
                      Type genericType,
                      Annotation[] annotations,
                      MediaType mediaType,
                      MultivaluedMap<String, Object> httpHeaders,
                      OutputStream entityStream)
          throws IOException, WebApplicationException
  {
    try ( OutputStreamWriter writer =
                  new OutputStreamWriter(entityStream, "UTF-8") ) {
      getGson().toJson(o, genericType, writer);
    }
  }
}
```

Lire Utilisation de Gson avec JAX-RS (services Web RESTful) en ligne: https://riptutorial.com/fr/gson/topic/4893/utilisation-de-gson-avec-jax-rs--services-web-restful-

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec gson	Community, Daniil Dubrovsky, Derlin, Egor Neliuba, Ginandi, James, Maverick, pr0gramist, Uttam
2	Utilisation de Gson avec JAX-RS (services Web RESTful)	Derlin, sargue