1.简单的处理 list 和 map

Java 代码

```
1. Gson gson = new Gson();
2. List testList = new ArrayList();
3. testList.add("first");
4. testList.add("second");
5. String listToJson = gson.toJson(testList);
6. System.out.println(listToJson);
7. //prints ["first","second"]
8.
9. Map testMap = new HashMap();
10. testMap.put("id", "id.first");
11. testMap.put("name","name.second");
12. String mapToJson = gson.toJson(testMap);
13. System.out.println(mapToJson);
14. //prints {"id":"id.first","name":"name.second"}
```

```
Gson gson = new Gson();
List testList = new ArrayList();
testList.add("first");
testList.add("second");
String listToJson = gson.toJson(testList);
System.out.println(listToJson);
//prints ["first","second"]

Map testMap = new HashMap();
testMap.put("id", "id.first");
testMap.put("name","name.second");
String mapToJson = gson.toJson(testMap);
System.out.println(mapToJson);
//prints {"id":"id.first","name":"name.second"}
```

2.处理带泛型的集合

Java 代码

```
1. List<TestBean> testBeanList = new ArrayList<TestBean>();
2. TestBean testBean = new TestBean();
3. testBean.setId("id");
4. testBean.setName("name");
5. testBeanList.add(testBean);
```

```
List<TestBean> testBeanList = new ArrayList<TestBean>();
TestBean testBean = new TestBean();
testBean.setId("id");
testBean.setName("name");
testBeanList.add(testBean);
```

Java 代码

```
1. java.lang.reflect.Type type = new com.google.gson.reflect.TypeToken<List<T
   estBean>>() {
2. }.getType();
3. String beanListToJson = gson.toJson(testBeanList,type);
4. System.out.println(beanListToJson);
5. //prints [{"id":"id","name":"name"}]
6.
7. List<TestBean> testBeanListFromJson = gson.fromJson(beanListToJson, type);

8. System.out.println(testBeanListFromJson);
9. //prints [TestBean@1ea5671[id=id,name=name,birthday=<null>]]
```

```
java.lang.reflect.Type type = new
com.google.gson.reflect.TypeToken<List<TestBean>>() {
}.getType();
String beanListToJson = gson.toJson(testBeanList,type);
System.out.println(beanListToJson);
//prints [{"id":"id","name":"name"}]

List<TestBean> testBeanListFromJson = gson.fromJson(beanListToJson, type);
System.out.println(testBeanListFromJson);
//prints [TestBean@1ea5671[id=id,name=name,birthday=<null>]]
```

map 等其他集合类型同上

3.Date 类型转化

先写工具类

Java 代码

```
1. import java.lang.reflect.Type;
2.
3. import com.google.gson.JsonDeserializationContext;
4. import com.google.gson.JsonDeserializer;
5. import com.google.gson.JsonElement;
6. import com.google.gson.JsonParseException;
7.
8. public class UtilDateDeserializer implements JsonDeserializer<java.util.Da
   te> {
9.
10.     @Override
11.     public java.util.Date deserialize(JsonElement json, Type typeOfT, Jso
```

```
      nDeserializationContext context)
    12.             throws JsonParseException {
    13.         return new java.util.Date(json.getAsJsonPrimitive().getAsLong());

    14.     }
    15. }
```

```
import java.lang.reflect.Type;

import com.google.gson.JsonDeserializationContext;
import com.google.gson.JsonDeserializer;
import com.google.gson.JsonElement;
import com.google.gson.JsonParseException;

public class UtilDateDeserializer implements JsonDeserializer<java.util.Date> {

        @Override
        public java.util.Date deserialize(JsonElement json, Type typeOfT,
JsonDeserializationContext context)
                        throws JsonParseException {
                return new java.util.Date(json.getAsJsonPrimitive().getAsLong());
        }
}
```

Java 代码

```
    1. import java.lang.reflect.Type;
    2.
    3. import com.google.gson.JsonElement;
    4. import com.google.gson.JsonPrimitive;
    5. import com.google.gson.JsonSerializationContext;
    6. import com.google.gson.JsonSerializer;
    7.
    8. public class UtilDateSerializer implements JsonSerializer<java.util.Date>
       {
    9.
    10.     @Override
    11.     public JsonElement serialize(java.util.Date src, Type typeOfSrc,
    12.             JsonSerializationContext context) {
    13.         return new JsonPrimitive(src.getTime());
    14.     }
    15.
    16. }
```

```
import java.lang.reflect.Type;

import com.google.gson.JsonElement;
import com.google.gson.JsonPrimitive;
import com.google.gson.JsonSerializationContext;
import com.google.gson.JsonSerializer;
```

```
public class UtilDateSerializer implements JsonSerializer<java.util.Date> {

    @Override
    public JsonElement serialize(java.util.Date src, Type typeOfSrc,
            JsonSerializationContext context) {
        return new JsonPrimitive(src.getTime());
    }

}
```

Java 代码

```
1. /**
2.      * 序列化方法
3.      * @param bean
4.      * @param type
5.      * @return
6.      */
7.     public static String bean2json(Object bean, Type type) {
8.         Gson gson = new GsonBuilder().registerTypeAdapter(java.util.Date.c
   lass, new UtilDateSerializer())
9.                 .setDateFormat(DateFormat.LONG).create();
10.         return gson.toJson(bean);
11.     }
12.
13.     /**
14.      * 反序列化方法
15.      * @param json
16.      * @param type
17.      * @return
18.      */
19.     public static <T> T json2bean(String json, Type type) {
20.         Gson gson = new GsonBuilder().registerTypeAdapter(java.util.Date.
   class, new UtilDateDeserializer())
21.                 .setDateFormat(DateFormat.LONG).create();
22.         return gson.fromJson(json, type);
23.     }
```

```
/**
        * 序列化方法
        * @param bean
        * @param type
        * @return
        */
       public static String bean2json(Object bean, Type type) {
              Gson gson = new GsonBuilder().registerTypeAdapter(java.util.Date.class, new
```

```
UtilDateSerializer())
                                .setDateFormat(DateFormat.LONG).create();
                return gson.toJson(bean);
        }

        /**
         * 反序列化方法
         * @param json
         * @param type
         * @return
         */
        public static <T> T json2bean(String json, Type type) {
                Gson gson = new GsonBuilder().registerTypeAdapter(java.util.Date.class, new
UtilDateDeserializer())
                                .setDateFormat(DateFormat.LONG).create();
                return gson.fromJson(json, type);
        }
```

现在开始测试

Java 代码

```
1. List<TestBean> testBeanList = new ArrayList<TestBean>();
2. TestBean testBean = new TestBean();
3. testBean.setId("id");
4. testBean.setName("name");
5. testBean.setBirthday(new java.util.Date());
6. testBeanList.add(testBean);
7.
8. java.lang.reflect.Type type = new com.google.gson.reflect.TypeToken<List<T
   estBean>>() {
9. }.getType();
10. String beanListToJson = bean2json(testBeanList, type);
11. System.out.println("beanListToJson:" + beanListToJson);
12. //prints [{"id":"id","name":"name","birthday":1256531559390}]
13.
14. List<TestBean> testBeanListFromJson = json2bean(beanListToJson, type);
15. System.out.println(testBeanListFromJson);
16. //prints [TestBean@77a7f9[id=id,name=name,birthday=Mon Oct 26 12:39:05 CS
   T 2009]]
```

```
        List<TestBean> testBeanList = new ArrayList<TestBean>();
        TestBean testBean = new TestBean();
        testBean.setId("id");
        testBean.setName("name");
        testBean.setBirthday(new java.util.Date());
```

```
                testBeanList.add(testBean);

                java.lang.reflect.Type type = new
com.google.gson.reflect.TypeToken<List<TestBean>>() {
                }.getType();
                String beanListToJson = bean2json(testBeanList, type);
                System.out.println("beanListToJson:" + beanListToJson);
                //prints [{"id":"id","name":"name","birthday":1256531559390}]

                List<TestBean> testBeanListFromJson = json2bean(beanListToJson, type);
                System.out.println(testBeanListFromJson);
                //prints [TestBean@77a7f9[id=id,name=name,birthday=Mon Oct 26 12:39:05 CST 2009]]
```

后记：对于 java.sql.Date 的转化同上类似，写两个类用于其序列化和反序列化即可

SQLDateDeserializer **implements** JsonDeserializer<java.sql.Date>

SQLDateSerializer **implements** JsonSerializer<java.sql.Date>

GsonBuilder api ：

com.google.gson

Class GsonBuilder

java.lang.Object
  └ **com.google.gson.GsonBuilder**

---

public final class **GsonBuilder** extends Object

Use this builder to construct a Gson instance when you need to set configuration options other than the default. For Gson with default configuration, it is simpler to use `new Gson()`. `GsonBuilder` is best used by creating it, and then invoking its various configuration methods, and finally calling create.

The following is an example shows how to use the `GsonBuilder` to construct a Gson instance:

```
 Gson gson = new GsonBuilder()
     .registerTypeAdapter(Id.class, new IdTypeAdapter())
     .serializeNulls()
     .setDateFormat(DateFormat.LONG)
     .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
     .setPrettyPrinting()
     .setVersion(1.0)
     .create();
```

NOTE: the order of invocation of configuration methods does not matter.

Author:
Inderjeet Singh, Joel Leitch

Constructor Summary

GsonBuilder() Creates a GsonBuilder instance that can be used to build Gson with various configuration settings.

Method Summary

Gson	create() Creates a Gson instance based on the current configuration.
GsonBuilder	excludeFieldsWithModifiers(int... modifiers) Configures Gson to excludes all class fields that have the specified modifiers.
GsonBuilder	excludeFieldsWithoutExposeAnnotation() Configures Gson to exclude all fields from consideration for serialization or deserialization that do not have the Expose annotation.
GsonBuilder	registerTypeAdapter(Type type, Object typeAdapter) Configures Gson for custom serialization or deserialization.
GsonBuilder	serializeNulls() Configure Gson to serialize null fields.
GsonBuilder	setDateFormat(int style) Configures Gson to to serialize Date objects according to the style value provided.
GsonBuilder	setDateFormat(int dateStyle, int timeStyle) Configures Gson to to serialize Date objects according to the style value provided.
GsonBuilder	setDateFormat(String pattern) Configures Gson to serialize Date objects according to the pattern provided.
GsonBuilder	setFieldNamingPolicy(FieldNamingPolicy namingConvention) Configures Gson to apply a specific naming policy to an object's field during serialization and deserialization.
GsonBuilder	setPrettyPrinting() Configures Gson to output Json that fits in a page for

批注 [1]: ======== CONSTRUCTOR SUMMARY ========

批注 [2]:

批注 [3]: ========== METHOD SUMMARY ==========

批注 [4]:

	pretty printing.
GsonBuilder	setVersion(double ignoreVersionsAfter) Configures Gson to enable versioning support.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

批注 [5]:

批注 [6]: ========= CONSTRUCTOR DETAIL ========

批注 [7]:

Constructor Detail

批注 [8]:

GsonBuilder

public **GsonBuilder**()

Creates a GsonBuilder instance that can be used to build Gson with various configuration settings. GsonBuilder follows the builder pattern, and it is typically used by first invoking various configuration methods to set desired options, and finally calling create().

批注 [9]: ========== METHOD DETAIL ==========

批注 [10]:

Method Detail

批注 [11]:

setVersion

public GsonBuilder **setVersion**(double ignoreVersionsAfter)

Configures Gson to enable versioning support.

Parameters:

ignoreVersionsAfter - any field or type marked with a version higher than this value are ignored during serialization or deserialization.

Returns:

a reference to this GsonBuilder object to fulfill the "Builder" pattern

---

批注 [12]:

excludeFieldsWithModifiers

public GsonBuilder **excludeFieldsWithModifiers**(int... modifiers)

Configures Gson to excludes all class fields that have the specified modifiers. By default, Gson will exclude all fields marked transient or static. This method will override that behavior.

Parameters:

modifiers - the field modifiers. You must use the modifiers specified in the Modifier class. For example, Modifier.TRANSIENT, Modifier.STATIC.

Returns:

a reference to this GsonBuilder object to fulfill the "Builder" pattern

---

excludeFieldsWithoutExposeAnnotation

批注 [13]:

public GsonBuilder **excludeFieldsWithoutExposeAnnotation**()

Configures Gson to exclude all fields from consideration for serialization or deserialization that do not have the Expose annotation.

Returns:

a reference to this GsonBuilder object to fulfill the "Builder" pattern

---

serializeNulls

批注 [14]:

public GsonBuilder **serializeNulls**()

Configure Gson to serialize null fields. By default, Gson omits all fields that are null during serialization.

Returns:

a reference to this GsonBuilder object to fulfill the "Builder" pattern

Since:

1.2

---

setFieldNamingPolicy

批注 [15]:

public GsonBuilder **setFieldNamingPolicy**(FieldNamingPolicy namingConvention)

Configures Gson to apply a specific naming policy to an object's field during serialization and deserialization.

Parameters:

namingConvention - the JSON field naming convention to use for serialization and deserialization.

Returns:
a reference to this GsonBuilder object to fulfill the "Builder" pattern

---

setPrettyPrinting

批注 [16]:

public GsonBuilder **setPrettyPrinting**()

Configures Gson to output Json that fits in a page for pretty printing. This option only affects Json serialization.

Returns:
a reference to this GsonBuilder object to fulfill the "Builder" pattern

---

setDateFormat

批注 [17]:

public GsonBuilder **setDateFormat**(String pattern)

Configures Gson to serialize Date objects according to the pattern provided. You can call this method or setDateFormat(int) multiple times, but only the last invocation will be used to decide the serialization format.

Note that this pattern must abide by the convention provided by SimpleDateFormat class. See the documentation in SimpleDateFormat for more information on valid date and time patterns.

Parameters:
pattern - the pattern that dates will be serialized/deserialized to/from
Returns:
a reference to this GsonBuilder object to fulfill the "Builder" pattern
Since:
1.2

---

setDateFormat

批注 [18]:

public GsonBuilder **setDateFormat**(int style)

Configures Gson to to serialize Date objects according to the style value provided. You can call this method or setDateFormat(String) multiple times,

but only the last invocation will be used to decide the serialization format.

Note that this style value should be one of the predefined constants in the DateFormat class. See the documentation in DateFormat for more information on the valid style constants.

Parameters:
style - the predefined date style that date objects will be serialized/deserialized to/from
Returns:
a reference to this GsonBuilder object to fulfill the "Builder" pattern
Since:
1.2

---

setDateFormat

批注 [19]:

```
public GsonBuilder setDateFormat(int dateStyle,
                                 int timeStyle)
```

Configures Gson to to serialize Date objects according to the style value provided. You can call this method or setDateFormat(String) multiple times, but only the last invocation will be used to decide the serialization format.

Note that this style value should be one of the predefined constants in the DateFormat class. See the documentation in DateFormat for more information on the valid style constants.

Parameters:
dateStyle - the predefined date style that date objects will be serialized/deserialized to/from
timeStyle - the predefined style for the time portion of the date objects
Returns:
a reference to this GsonBuilder object to fulfill the "Builder" pattern
Since:
1.2

---

registerTypeAdapter

批注 [20]:

```
public GsonBuilder registerTypeAdapter(Type type,
```

Object typeAdapter)

Configures Gson for custom serialization or deserialization. This method combines the registration of an InstanceCreator, JsonSerializer, and a JsonDeserializer. It is best used when a single object **typeAdapter** implements all the required interfaces for custom serialization with Gson. If an instance creator, serializer or deserializer was previously registered for the specified **type**, it is overwritten.

Parameters:

type - the type definition for the type adapter being registered

typeAdapter - This object must implement at least one of the InstanceCreator, JsonSerializer, and a JsonDeserializer interfaces.

Returns:

a reference to this **GsonBuilder** object to fulfill the "Builder" pattern

---

批注 [21]:

create

public Gson **create**()

Creates a Gson instance based on the current configuration. This method is free of side-effects to this **GsonBuilder** instance and hence can be called multiple times.

Returns:

an instance of Gson configured with the options currently set in this builder