# Json转换利器Gson之实例二-Gson注解和GsonBuilder

标签： Gson GsonBuilder json JSON Json 实例

2012-06-22 21:23 44583人阅读 评论(7) 收藏 举报

分类：

Json（5）



有时候我们不需要把实体的所有属性都导出,只想把一部分属性导出为Json.

有时候我们的实体类会随着版本的升级而修改.

有时候我们想对输出的json默认排好格式.

... ...

请看下面的例子吧:

实体类:

```java
import java.util.Date;

import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;

public class Student {
    private int id;

    @Expose
    private String name;

    @Expose
    @SerializedName("bir")
    private Date birthDay;

    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;
    }
```

```
31.
32.     public Date getBirthDay() {
33.         return birthDay;
34.     }
35.
36.     public void setBirthDay(Date birthDay) {
37.         this.birthDay = birthDay;
38.     }
39.
40.     @Override
41.     public String toString() {
42.         return "Student [birthDay=" + birthDay + ", id=" + id + ", name="
43.                 + name + "]";
44.     }
45.
46. }
```

测试类:

```
[java]
01. import java.util.ArrayList;
02. import java.util.Date;
03. import java.util.List;
04.
05. import com.google.gson.FieldNamingPolicy;
06. import com.google.gson.Gson;
07. import com.google.gson.GsonBuilder;
08. import com.google.gson.reflect.TypeToken;
09.
10. public class GsonTest2 {
11.
12.     public static void main(String[] args) {
13.         //注意这里的Gson的构建方式为GsonBuilder,区别于test1中的Gson gson = new Gson();
14.         Gson gson = new GsonBuilder()                                    载
15.         .excludeFieldsWithoutExposeAnnotation() //不导出实体中没有用@Expose注解的属性
16.         .enableComplexMapKeySerialization() //支持Map的key为复杂对象的形式
17.         .serializeNulls().setDateFormat("yyyy-MM-dd HH:mm:ss:SSS")//时间转化为特定格式
18.         .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)//会把字段首字母大写,注:对于实体上使用了@SerializedName注解的不会生效.
19.         .setPrettyPrinting() //对json结果格式化.
20.         .setVersion(1.0)    //有的字段不是一开始就有的,会随着版本的升级添加进来,那么在进行序列化和返序列化的时候就会根据版本号来选择是否要序列化.
21.                             //@Since(版本号)能完美地实现这个功能.还的字段可能,随着版本的升级而删除,那么
22.                             //@Until(版本号)也能实现这个功能,GsonBuilder.setVersion(double)方法需要调用.
23.         .create();
24.
25.
26.
27.         Student student1 = new Student();
28.         student1.setId(1);
29.         student1.setName("李坤");
30.         student1.setBirthDay(new Date());
31.
32.         // ////////////////////////////////////////////////////
```

```
33.         System.out.println("----------简单对象之间的转化-------------");
34.         // 简单的bean转为json
35.         String s1 = gson.toJson(student1);
36.         System.out.println("简单Bean转化为Json===" + s1);
37.
38.         // json转为简单Bean
39.         Student student = gson.fromJson(s1, Student.class);
40.         System.out.println("Json转为简单Bean===" + student);
41.         // ////////////////////////////////////////////////////
42.
43.         Student student2 = new Student();
44.         student2.setId(2);
45.         student2.setName("曹贵生");
46.         student2.setBirthDay(new Date());
47.
48.         Student student3 = new Student();
49.         student3.setId(3);
50.         student3.setName("柳波");
51.         student3.setBirthDay(new Date());
52.
53.         List<Student> list = new ArrayList<Student>();
54.         list.add(student1);
55.         list.add(student2);
56.         list.add(student3);
57.
58.         System.out.println("----------带泛型的List之间的转化-------------");
59.         // 带泛型的list转化为json
60.         String s2 = gson.toJson(list);
61.         System.out.println("带泛型的list转化为json==" + s2);
62.
63.         // json转为带泛型的list
64.         List<Student> retList = gson.fromJson(s2,
65.                 new TypeToken<List<Student>>() {
66.                 }.getType());
67.         for (Student stu : retList) {
68.             System.out.println(stu);
69.         }
70.
71.     }
72. }
```

输出结果:

```
[plain]
01. ----------简单对象之间的转化-------------
02. 简单Bean转化为Json==={
03.   "Name": "李坤",
04.   "bir": "2012-06-22 21:26:40:592"
05. }
06. Json转为简单Bean===Student [birthDay=Fri Jun 22 21:26:40 CST 2012, id=0, name=李坤]
07. ----------带泛型的List之间的转化-------------
08. 带泛型的list转化为json==[
09.   {
10.     "Name": "李坤",
```

```
    "bir": "2012-06-22 21:26:40:592"
  },
  {
    "Name": "曹贵生",
    "bir": "2012-06-22 21:26:40:625"
  },
  {
    "Name": "柳波",
    "bir": "2012-06-22 21:26:40:625"
  }
]
Student [birthDay=Fri Jun 22 21:26:40 CST 2012, id=0, name=李坤]
Student [birthDay=Fri Jun 22 21:26:40 CST 2012, id=0, name=曹贵生]
Student [birthDay=Fri Jun 22 21:26:40 CST 2012, id=0, name=柳波]
```

Json转换利器Gson之实例一-简单对象转化和带泛型的List转化
(http://blog.csdn.net/lk_blog/article/details/7685169)
Json转换利器Gson之实例二-Gson注解和GsonBuilder
(http://blog.csdn.net/lk_blog/article/details/7685190)
Json转换利器Gson之实例三-Map处理(上)
(http://blog.csdn.net/lk_blog/article/details/7685210)
Json转换利器Gson之实例四-Map处理(下)
(http://blog.csdn.net/lk_blog/article/details/7685224)
Json转换利器Gson之实例五-实际开发中的特殊需求处理
(http://blog.csdn.net/lk_blog/article/details/7685237)
Json转换利器Gson之实例六-注册TypeAdapter及处理Enum类型
(http://blog.csdn.net/lk_blog/article/details/7685347)

实例代码下载: http://download.csdn.net/detail/lk_blog/4387822