



河北师范大学软件学院  
Software College of Hebei Normal University

# Gson实战



Android教研室



Gson简介



Gson的目标



Gson的使用





## Gson简介

- Gson是一个Java库，可以用来将Java对象转换成JSON字符串表示，或者将JSON字符串转换为等效的Java对象。
- **JSON**(JavaScript Object Notation) 是一种轻量级的数据交换格式，广泛应用于不同平台间数据的传递，尤其是服务器与客户端的交互。

```
{  
    "name": "zhangsan",  
    "sex": "man",  
    "age": 18  
}
```



## Gson的目标

- 提供一种简单的、易于使用的机制实现Java对象与JSON字符串的转换。
- 支持任意复杂的对象的表示。
- 生成可读的JSON字符串输出。



## Gson处理对象的几个重要点

- 推荐把成员变量都声明为private
- 所有包含在当前类（包括父类）中的字段都默认被序列化或者反序列化
- 如果某个字段被 transient、static关键词修饰，就不会被序列化或者反序列化



# Gson的使用

## ➤ 添加依赖

```
dependencies {  
    .....  
    //Gson  
    implementation 'com.google.code.gson:gson:2.8.5'  
}
```



## Gson的使用

- 使用的主要类是Gson，您可以通过调用new Gson()来创建它。还有一个类GsonBuilder可用来创建一个Gson实例，它具有各种设置，如版本控制等。
- Gson实例在调用Json操作时不维护任何状态。因此，多个序列化和反序列化操作中可以使用相同的Gson对象。



# Gson的使用

## ➤ 基本类型

*// Serialization*

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

```
gson.toJson(1);
```

*// ==> 1*

```
gson.toJson("abcd");
```

*// ==> "abcd"*

```
gson.toJson(new Long(10));
```

*// ==> 10*

```
int[] values = { 1 };
```

```
gson.toJson(values);
```

*// ==> [1]*





# Gson的使用

## ➤ 基本类型

```
// Deserialization
```

```
int num1 = gson.fromJson("1", int.class);  
Integer num2 = gson.fromJson("1", Integer.class);  
Long num3 = gson.fromJson("1", Long.class);  
Boolean bool = gson.fromJson("false", Boolean.class);  
String str = gson.fromJson("\"abc\"", String.class);  
String[] str1 = gson.fromJson("[\"abc\"]",  
                                String[].class);
```



## Gson的使用

### ➤ POJO(Plain Ordinary Java Object)的序列化和反序列化

```
public class User {  
    private String name;  
    private int age;  
    private String sex;  
    //省略getter和setter方法  
    .....  
}
```



# Gson的使用

## ➤ POJO的序列化

*// Serialization*

```
User user = new User( "张三", 20, "男 ");
```

```
String jsonObj = gson.toJson(user);
```

*//json is : {"name":"张三", "age":20, "sex":"男"}*



## Gson使用

### ➤ POJO的反序列化

```
// POJO Deserialization
```

```
String jsonString =
```

```
“{\\\"name\\\":\\\"lily\\\", \\\"age\\\":24, \\\"sex\\\":\\\"男\\\"}”;
```

```
User user1 = gson.fromJson(jsonString, User.class);
```



## Gson的使用

- 当序列化的时候，如果对象的某个字段为null，是不会输出到Json字符串中的。
- 当反序列化的时候，某个字段在Json字符串中找不到对应的值，就会被赋值为null。



## Gson的使用

- 使用GsonBuilder导出null值、格式化输出、格式化日期

*//GsonBuilder的用法*

```
Gson gson = new GsonBuilder()
```

```
    //各种配置
```

```
    .create(); //生成配置好的Gson
```



## Gson的使用

### ➤ 使用GsonBuilder导出null值、格式化输出、格式化日期

```
Gson gson = new GsonBuilder()  
    //各种配置  
    .serializeNulls() //允许导出null值  
    .setPrettyPrinting() //格式化输出  
    .setDateFormat("yyyy-MM-dd") //设置日期输出格式  
    .create(); //生成配置好的Gson
```



## Gson的使用

### ➤ 嵌套的Json数据解析

- Json数据：

```
{"name":"John",  
  "age":20,"grade":{"course":"English","score":100,"level":"A"}}
```
- 定义含有内部类的类
  - Student、Grade





## Gson的使用

### ➤ 嵌套的Json数据解析

- 序列化

```
Grade grade = new Grade();  
grade.setCourse("Math");
```

.....

```
Student john = new Student();  
john.setName("John");
```

.....

```
Gson gson = new Gson();  
String jsonString = gson.toJson(john);
```



## Gson的使用

### ➤ 嵌套的Json数据解析

- 反序列化

```
String jsonString = "{ 'name' : ' John' ,  
    'age' :20,' grade' :{ 'course' : ' English' , scor  
e' :100,' level' : ' A' } }";  
Gson gson = new Gson();  
Student lily =  
gson.fromJson(jsonString, Student.class);
```



## Gson使用

### ➤ 数组的序列化

```
Gson gson = new Gson();  
int[] ints = {1, 2, 3, 4, 5};  
String[] strings = {"abc", "def", "ghi"};  
  
// Serialization  
gson.toJson(ints);    // ==> [1,2,3,4,5]  
gson.toJson(strings); // ==> ["abc", "def", "ghi"]
```



### ➤ 数组的反序列化

```
// Deserialization  
int[] ints2 = gson.fromJson("[1,2,3,4,5]", int[].class);  
// ==> ints2 will be same as ints
```



## Gson使用

### ➤ 集合序列化

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

```
List<Integer> ints 中有元素 : 1,2,3,4,5;
```

```
// Serialization
```

```
String json = gson.toJson(ints); // ==> json is  
[1,2,3,4,5]
```



### ➤ 集合反序列化

```
// Deserialization
Type collectionType = new
TypeToken<Collection<Integer>>().getType();
Collection<Integer> ints2 = gson.fromJson(json,
collectionType);
// ==> ints2 is same as ints
```



## Gson的使用

### ➤ TypeToken

- GSON 提供的 用来捕获像 List 这样的泛型信息的类
- Java编译器把捕获到的泛型信息编译到这个匿名内部类里，在运行时被 `getType()` 方法用反射的 API 提取到具体的类型信息



## Gson的使用

### ➤ 复杂的嵌套的Json数据解析——对象集合

- Json数据：

```
[{"name":"John","age":18,"grade":{"gradeName":"English","gradeScore":100}}, {"name":"Tom","age":20,"grade":{"gradeName":"English","gradeScore":86}}]
```





## Gson的使用

### ➤ 嵌套的Json数据解析

- 序列化

```
//TODO : 构造对象集合List<Person> persons  
.....  
Gson gson = new Gson();  
String jsonString = gson.toJson(persons);
```



## Gson的使用

### ➤ 嵌套的Json数据解析

- 反序列化

```
String jsonString =  
"[{'name':'John','age':18,'grade':{'gradeName':'English','gradeScore':100}},{'name':'Tom','age':20,'grade':{'gradeName':'English','gradeScore':86}}]";  
Gson gson = new Gson();  
List<Person> persons = gson.fromJson(jsonData,  
new TypeToken<List<Person>>(){}.getType());
```



## Gson的使用

### ➤ 复杂的嵌套的Json数据解析——对象集合

- 思考？

- 1. 如果外部类的某个属性是类的对象的集合呢？
- 2. 如果Json串中对象的数组有一个名称呢？



**Thank You!**

