**[Gson基本操作，JsonObject，JsonArray，String，JavaBean，List互转](http://www.cnblogs.com/robbinluobo/p/7217387.html)**

http://www.cnblogs.com/jianyungsun/p/6647203.html

http://blog.csdn.net/pg\_guo/article/details/51083502

String、JsonObject、JavaBean 互相转换

User user = new Gson().fromJson(jsonObject, User.class);

User user = new Gson().fromJson(string, User.class);

String string = new Gson().toJson(user);

JsonObject jsonObject = new Gson().toJsonTree(user).getAsJsonObject();

JsonObject jsonObject = new JsonParser().parse(string).getAsJsonObject();

String、JsonArray、List互相转换

List<User> userList = gson.fromJson(string, new TypeToken<List<User>>() {}.getType());

List<User> userList = gson.fromJson(jsonArray, new TypeToken<List<User>>() {}.getType());

String string = new Gson().toJson(userList);

JsonArray jsonArray = new Gson().toJsonTree(userList, new TypeToken<List<User>>() {}.getType()).getAsJsonArray();

JsonArray jsonArray = new JsonParser().parse(string).getAsJsonArray();

* 1

关于GSON的入门级使用，这里就不提了，如有需要可以看这篇博文 [《Google Gson的使用方法,实现Json结构的相互转换》](http://blog.csdn.net/qxs965266509/article/details/42774691) ，写的很好，通俗易懂。

我为什么写这篇文章呢？因为前几晚跟好友 [xiasuhuei321](http://www.jianshu.com/users/e1fed0fb341b/latest_articles) 探讨了一下GSON解析复杂的JSON的时候，能不能只解析源数据中的数组，甚至只解析数组的某一部分。探讨了二十分钟，得出结论：没用过，不知道。

所以今天特地研究了一下，发现真的So Easy！之前想复杂了，学习的过程中，发现有五种方式分别搞定不同情况的JSON数组，也就是今天说的五大招！

在介绍之前先来个约定，比如下面的这个JSON：

"muser": [

{

"name": "zhangsan",

"age": "10",

"phone": "11111",

"email": "11111@11.com"

},

...

]

* 这里的 “muser” ，也就是数组的名称，称它为数据头，防止跟里面的 字段 有歧义；
* 如果没有数据头，那就叫它纯数据，或者纯数组数据；
* 代码中用到的 JsonArray/JsonObject 等熟悉的类全部来自 GSON 。

开始过招吧！

**第一招 A**

**没有数据头的纯数组JSON如何解析？**

根据约定，也就是这个 JSON 里面只有一个数组（JsonArray），而且这个数组没有名字，比如像下面这样的：

[

{

"name": "zhangsan",

"age": "10",

"phone": "11111",

"email": "11111@11.com"

},

{

"name": "lisi",

"age": "20",

"phone": "22222",

"email": "22222@22.com"

},

...

]

这里其实是最简单的一种 JSON 数组格式，强大的 GSON 可以直接解析成一个 List 。但在这里我先不直接解析，就用比较老实的方法去解析，因为需要引出两个东西。

首先我们需要建立一个Bean对象，注意变量名要跟字段名称一致，没什么好说的：

public class UserBean {

//变量名跟JSON数据的字段名需要一致

private String name ;

private String age;

private String phone;

private String email;

...

}

下面这是解析过程，先看代码：

/\*\*

\* 解析没有数据头的纯数组

\*/

private void parseNoHeaderJArray() {

//拿到本地JSON 并转成String

String strByJson = JsonToStringUtil.getStringByJson(this, R.raw.juser\_1);

//Json的解析类对象

JsonParser parser = new JsonParser();

//将JSON的String 转成一个JsonArray对象

JsonArray jsonArray = parser.parse(strByJson).getAsJsonArray();

Gson gson = new Gson();

ArrayList<UserBean> userBeanList = new ArrayList<>();

//加强for循环遍历JsonArray

for (JsonElement user : jsonArray) {

//使用GSON，直接转成Bean对象

UserBean userBean = gson.fromJson(user, UserBean.class);

userBeanList.add(userBean);

}

mainLView.setAdapter(new UserAdapter(this, userBeanList));

}

从代码中可以看出解析的步骤如下：

* 无论 JSON 来自本地还是网络获取，都要先将 JSON 转成 String ；
* 需要一个 JSON 解析类对象将JSON的字符串转成 JsonArray ，前提是我们知道 JSON 中只有纯数组；
* 循环遍历 JsonArray ，并用 GSON 解析成相应的对象。

代码本身不难，容易看懂，但前面说到，这里我故意这样写，因为需要说两个东西：

1、JsonParse

从名称我们就可以看出，这是一个解析类。没错，它可以把 JSON 数据分别通过 getAsJsonObject 和 getAsJsonArray 解析成 JsonObject 和 JsonArray 。这跟普通的解析 JSON 差不多，不展开说。

2、JsonElement

这个类我是第一次见，它是一个抽象类，代表 JSON 串中的某一个元素，可以是 JsonObject/JsonArray/JsonPrimitive/… 中的任何一种元素。

所以在上面的代码中，我们可以看到它能把 JsonArray 中的每一个元素转成 JsonObject ，甚至说它本身就是 JsonObject 。

好了，就为了说这两个东西。记住，后面将会用到。

来看一下运行的图吧，很简单的东西，后面的二三都是这样的效果，就不重复贴图了：

**第二招 Q**

**有数据头的纯数组数据该怎么解析？**

内容跟上面的 JSON 一模一样，只不过加了一个名称 “muser” ，也就是约定好的 数据头 ：

{

"muser": [

{

"name": "zhangsan",

"age": "10",

"phone": "11111",

"email": "11111@11.com"

},

{

"name": "lisi",

"age": "20",

"phone": "22222",

"email": "22222@22.com"

},

...

]

}

有人说，这还不简单，在第一招中的 getAsJsonArray 加一个字符串就是咯，就像这样：

JsonArray jsonArray = parser.parse(strByJson).getAsJsonArray("muser");

思路是对的，但是不要忘了，数组装在一个 { } 括起来的 JsonObject 里。还记得上面的 JsonParse 么，它的 getAsJsonObject 可以做到这点，所以代码就是这样啦，很简单就不再解释了：

/\*\*

\* 解析有数据头的纯数组

\*/

private void parseHaveHeaderJArray() {

//拿到本地JSON 并转成String

String strByJson = JsonToStringUtil.getStringByJson(this, R.raw.juser\_2);

//先转JsonObject

JsonObject jsonObject = new JsonParser().parse(strByJson).getAsJsonObject();

//再转JsonArray 加上数据头

JsonArray jsonArray = jsonObject.getAsJsonArray("muser");

Gson gson = new Gson();

ArrayList<UserBean> userBeanList = new ArrayList<>();

//循环遍历

for (JsonElement user : jsonArray) {

//通过反射 得到UserBean.class

UserBean userBean = gson.fromJson(user, new TypeToken<UserBean>() {}.getType());

userBeanList.add(userBean);

}

mainLView.setAdapter(new UserAdapter(this, userBeanList));

}

注意，这里又引出了一个东西： TypeToken ，它是什么呢？

3、TypeToken

这个东西很有意思，本来我不知道到是干嘛的，看了看源码，看不懂。后来无意发现它所在的包：

import com.google.gson.reflect.TypeToken;

哎哟我去， reflect 这不是反射么，一下子就明白了。没错，它其实是一个匿名内部类，看一下官方解释：

GSON 提供了 TypeToken 这个类来帮助我们捕获（capture）像 List 这样的泛型信息。Java编译器会把捕获到的泛型信息编译到这个匿名内部类里，然后在运行时就可以被 getType() 方法用反射的 API 提取到。

解释的很官方，实际上就是一句 通俗但不严谨 的话，它将泛型 T 转成 .class 。比如上面的 TypeToken 经过 getType() 后就是 UserBean.class 。

好了，说到这里基本铺垫就完成了，再次强调一下：

对于上面的 JSON 完全可以直接通过 GSON 转成 List ，不用这么麻烦，我只是为了引出3个小知识。

**第三招 W**

**有数据头的复杂数据该如何解析呢？**

简单的说完了，铺垫也铺完了，来看一看复杂的吧：

{

"code": 200,

"msg": "OK",

"muser": [

{

"name": "zhangsan",

"age": "10",

"phone": "11111",

"email": "11111@11.com"

},

{

"name": "lisi",

"age": "20",

"phone": "22222",

"email": "22222@22.com"

},

...

]

}

这里就不再是纯数组数据了，还有两个凑数的不知道干嘛用的字段，这里也有数据头，之前用的是笨方法，现在来真正见识一下GSON的威力吧。

第一步根据 JSON 建立 Bean ，注意这里的 Bean 是返回所有字段，因为 GSON 能直接解析成 List ，所以 Bean 是下面这样的，同样把占地方的 get/set 省略：

/\*\*

\* Created by xiarui on 2016/8/30.

\* 返回所有结果的Bean

\*/

public class ResultBean {

//注意变量名与字段名一致

private int code;

private String msg;

private List<UserBean> muser;

public class UserBean{

private String name ;

private String age;

private String phone;

private String email;

...

}

...

}

注意，这个 ResultBean 里面有一个 UserBean 。 它虽然跟上面第一第二招虽然内容一样，但是作用不一样，这是作为 JsonArray 解析后存入 List 中的对象。

算了，有点拗口，直接上代码吧：

/\*\*

\* 有消息头 复杂数据 常规方式

\*/

private void parseComplexJArrayByCommon() {

//拿到Json字符串

String strByJson = JsonToStringUtil.getStringByJson(this, R.raw.juser\_3);

//GSON直接解析成对象

ResultBean resultBean = new Gson().fromJson(strByJson,ResultBean.class);

//对象中拿到集合

List<ResultBean.UserBean> userBeanList = resultBean.getMuser();

//展示到UI中

mainLView.setAdapter(new ResultAdapter(this, userBeanList));

}

没错，就是这么四句话搞定第一二招的内容。看出GSON的强大了吧，当然如果有人想不开只写一句话的话：

mainLView.setAdapter(new ResultAdapter(this,new Gson().fromJson(JsonToStringUtil.getStringByJson(this,R.raw.juser\_3),ResultBean.class).getMuser()));

我也是没意见的，不过请对自己好一点，谢谢。

**第四招 E**

**只想解析复杂JSON中的数组或数组中的某部分内容怎么办？**

好了，来到重点了，这也是跟好友 [xiasuhuei321](http://www.jianshu.com/users/e1fed0fb341b/latest_articles) 没有讨论出来的情况。

还是上面的JSON数据，这里为了篇幅就不贴重复代码了，假如我只想取 “muser” 这个数组中的年龄（age）大于30岁的怎么办？

OK，当然可以先全部解析，再从 List 中取。那假如我有一万条数据呢？全部解析不是很麻烦呢？

所以一个思路就是第一二招中说的： 遍历！

OK，你会问先遍历还不是要读一万条，是的，还是要读一万条，但是假如我要把这些存入数据库呢？假如一万条数据中只有一条符合条件，难道我先存一万条，再从数据库中查询么？

当然这种情况是极端情况，但也说明了一个问题，不能所有情况下都先全部解析，假如有一万个字段，Bean还得写多长…可怕。

现在来说一下完整的思路，也是我学习中思考的过程：

* 第一点肯定就是刚才提到的遍历，这个很好理解，所以我们先要取这一个数组（JsonArray），那么如何取呢？还记得之前提到的 JsonParse 么，它的 getAsJsonArray() 可以传入 数据头 拿到数组，当然不要忘了最外面一层是个 JsonObject 。
* //最外层
* JsonObject jsonObject = new JsonParser().parse(strByJson).getAsJsonObject();
* //需要遍历的数组

JsonArray jsonArray = jsonObject.getAsJsonArray("muser");

* 拿到数组以后，我们就可以遍历了，经过第一二招的洗礼，相信在遍历上，应该没什么问题了，使用的还是之前提到的 JsonElement 。
* //循环遍历数组
* for (JsonElement user : jsonArray) {
* UserBean userBean = new Gson().fromJson(user, new TypeToken<UserBean>() {}.getType());
* //根据条件过滤
* if (Integer.parseInt(userBean.getAge()) > 30) {
* userBeanList.add(userBean);
* }

}

* 上面的代码很简单，也用到了之前提到的 TypeToken ，什么意思就不用解释了吧。

好了，完整的代码如下：

/\*\*

\* 有数据头 复杂数据 截取方式

\*/

private void parseComplexJArrayByDirect() {

//拿到JSON字符串

String strByJson = JsonToStringUtil.getStringByJson(this, R.raw.juser\_3);

List<UserBean> userBeanList = new ArrayList<>();

//拿到数组

JsonObject jsonObject = new JsonParser().parse(strByJson).getAsJsonObject();

JsonArray jsonArray = jsonObject.getAsJsonArray("muser");

//循环遍历数组

for (JsonElement user : jsonArray) {

UserBean userBean = new Gson().fromJson(user, new TypeToken<UserBean>() {

}.getType());

//根据条件过滤

if (Integer.parseInt(userBean.getAge()) > 30) {

userBeanList.add(userBean);

}

}

mainLView.setAdapter(new UserAdapter(this, userBeanList));

}

运行的结果图如下：

可以看到，现在我们做到了只取 JSON 数据中数组中某一部分了。那么扩展一下，只取 JSON 数据中的某一个数组中的某一个字段呢？当然可以实现，不过还是留给大家自己思考吧，当然下面反人类的第五招也是可以解决这个问题的。

**第五招 R**

**如果一个 JSON 数据很很很复杂怎么解析？**

什么叫做复杂，这里我简单写了个比较复杂的，有数据头，一层嵌套一层，我还没有写数组呢：

{

"group": {

"user": {

"name": "张三",

"age": "10",

"phone": "11111",

"email": "11111@11.com"

},

"info": {

"address": "北京",

"work": "Android Dev",

"pay": "10K",

"motto": "先定一个小目标，比如我先赚一个亿"

}

}

}

三种方式解析：

* 第三招，全部解析出来；
* 第四招，要什么解析什么；
* 第五招，反人类的 JsonReader 。

至于为什么反人类，不好说。大家看代码就知道了，代码很简单，跟 XML 的解析差不多，是根据节点来的，至于怎么用，还是那句话直接看代码吧，确实处理起来逻辑清晰，但是代码量上，真的不敢恭维。

只贴代码不作解释，如想详细了解，看文末链接。

/\*\*

\* 通过JsonReader的方式去解析

\*/

private void parseComplexJArrayByReader() throws IOException {

String strByJson = JsonToStringUtil.getStringByJson(this, R.raw.juser\_4);

JsonReader reader = new JsonReader(new StringReader(strByJson));

try {

reader.beginObject();

String tagName = reader.nextName();

if (tagName.equals("group")) {

//读group这个节点

readGroup(reader);

}

reader.endObject();

} finally {

reader.close();

}

}

/\*\*

\* 读group这个节点

\*

\* @param reader JsonReader

\*/

private void readGroup(JsonReader reader) throws IOException {

reader.beginObject();

while (reader.hasNext()) {

String tagName = reader.nextName();

if (tagName.equals("user")) {

readUser(reader);

} else if (tagName.equals("info")) {

readInfo(reader);

}

}

reader.endObject();

}

/\*\*

\* 读用户基本消息 user节点

\*

\* @param reader JsonReader

\*/

private void readUser(JsonReader reader) throws IOException {

reader.beginObject();

while (reader.hasNext()) {

String tag = reader.nextName();

if (tag.equals("name")) {

String name = reader.nextString();

nameText.setText(name);

} else if (tag.equals("age")) {

String age = reader.nextString();

ageText.setText(age);

}

...

else {

reader.skipValue();//忽略

}

}

reader.endObject();

}

/\*\*

\* 读用户其他消息 info节点

\*

\* @param reader JsonReader

\*/

private void readInfo(JsonReader reader) throws IOException {

reader.beginObject();

while (reader.hasNext()) {

String tag = reader.nextName();

if (tag.equals("address")) {

String address = reader.nextString();

addressText.setText(address);

} else if (tag.equals("work")) {

String work = reader.nextString();

workText.setText(work);

}

...

else {

reader.skipValue();//忽略

}

}

reader.endObject();

}

上面代码有省略，因为好长…运行图如下：

五招过完，多谢指教！

**总结**

以上几乎就是 JSO N数组的所有情况了，这五招也几乎能全部搞定！不得不说，GSON 确实比较强大，强大在于可以将 JSON 直接解析成对象，比以前的手动去解析方便太多，当然 fastJson 也能实现这点，但是这东西还是官方的用的顺手。

在学习的过程中，也是一步一步来的，所以文章也是学习的过程，从简单的例子学到关键内容，再解决复杂情况。由于文章写得仓促，如有疑问或错误，欢迎交流与指正，谢谢！

**参考资料**

[灵活组装Json的数据使用Gson的JsonParser和JsonReader解析Json详解例子](http://www.tuicool.com/articles/FrI3uq)

[使用Gson解析复杂的json数据 – tkwxty](http://blog.csdn.net/tkwxty/article/details/34474501/)

[JsonElement的简单说明 – chunqiuwei](http://blog.csdn.net/chunqiuwei/article/details/49160321)

[Java进阶(四)Java反射TypeToken解决泛型运行时类型擦除的有关问题解决](http://www.myexception.cn/program/1909359.html)

**项目源码**

[GsonArrayDemo – IamXiaRui – Github](https://github.com/IamXiaRui/Android_5.0_ViewDemo/tree/master/GsonArrayDemo)

String、JsonObject、JavaBean 互相转换

User user = new Gson().fromJson(jsonObject, User.class);

User user = new Gson().fromJson(string, User.class);

String string = new Gson().toJson(user);

JsonObject jsonObject = new Gson().toJsonTree(user).getAsJsonObject();

JsonObject jsonObject = new JsonParser().parse(string).getAsJsonObject();

String、JsonArray、List互相转换

List<User> userList = gson.fromJson(string, new TypeToken<List<User>>() {}.getType());

List<User> userList = gson.fromJson(jsonArray, new TypeToken<List<User>>() {}.getType());

String string = new Gson().toJson(userList);

JsonArray jsonArray = new Gson().toJsonTree(userList, new TypeToken<List<User>>() {}.getType()).getAsJsonArray();

JsonArray jsonArray = new JsonParser().parse(string).getAsJsonArray();

* 7

标签: [Gson JsonObject JsonArray String JavaBean List Map](http://www.cnblogs.com/robbinluobo/tag/Gson%20JsonObject%20JsonArray%20String%20JavaBean%20List%20Map/)