# JSON格式及FastJson使用详解

在进行数据处理或前后端交互的时候，会不可避免的碰到json格式的数据。Json是一种轻量级的数据交换格式，采用一种“键：值”对的文本格式来存储和表示数据，是一种理想的数据交换语言。本文对son的使用以及fastjson包的使用进行阐述，通过本文的学习，可以解决部分常见的JSON数据问题，内容包括以下：

一、JSON形式与语法

1.1 JSON对象

1.2 JSON对象数组

1.3：JSON字符串

二 Fastjson介绍

2.1 Fastjson简介

2.2 Fastjson使用

2.2.1 JSONObject类使用

2.2.2 JSONArray类使用

2.2.3 JSON类使用

三 JSON案例

3.1 json字符串—》JSONObject

3.2 JSONObject—》json字符串

3.3 JSON字符串数组—》JSONArray

3.4 JSONArray—》json字符串

3.5 复杂JSON格式字符串—》JSONObject

3.6 复杂JSONObject—》json字符串

3.7 json字符串—》JavaBean

3.8 JavaBean —》json字符串

3.9 json字符串数组—》JavaBean-List

3.10 JavaBean-List —》json字符串数组

3.11 复杂嵌套json格式字符串—》JavaBean\_obj

3.12 JavaBean\_obj —》复杂json格式字符串

### 一、JSON形式与语法

### 1.1 JSON对象

{

​

"ID": 1001,

​

"name": "张三",

​

"age": 24

​

}

​

这个数据就是一个Json对象，Json对象的特点如下：

1. 数据在花括号中
2. 数据以"键：值"对的形式出现（其中键多以字符串形式出现，值可取字符串，数值，甚至其他json对象）
3. 每两个"键：值"对以逗号分隔（最后一个"键：值"对省略逗号）

### 1.2 JSON对象数组

[

​

{"ID": 1001, "name": "张三", "age": 24},

​

{"ID": 1002, "name": "李四", "age": 25},

​

{"ID": 1003, "name": "王五", "age": 22}

​

]

​

这个数据就是一个Json对象数组，Json对象数组的特点如下：

1. 数据在方括号中（可理解为数组）
2. 方括号中每个数据以json对象形式出现
3. 每两个数据以逗号分隔（最后一个无需逗号）

上面两个是Json的基本形式，结合在一起就可以得出其他的数据形式，例如这个：

{

​

"部门名称":"研发部",

​

"部门成员":[

​

{"ID": 1001, "name": "张三", "age": 24},

​

{"ID": 1002, "name": "李四", "age": 25},

​

{"ID": 1003, "name": "王五", "age": 22}],

​

"部门位置":"xx楼21号"

​

}

​

通过这种变形，使得数据的封装具有很大的灵活性。

### 1.3：JSON字符串

Json字符串应满足以下条件：

1. 1它必须是一个字符串，支持字符串的各种操作
2. 里面的数据格式应该要满足其中一个格式，可以是json对象，也可以是json对象数组或者是两种基本形式的组合变形。

总结：json可以简单的分为基本形式：json对象，json对象数组。两种基本格式组合变形出其他的形式，但其本质还是json对象或者json对象数组中的一种。json对象或对象数组可以转化为json字符串，使用于不同的场合。

### 二 Fastjson介绍

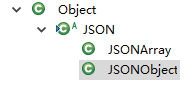
### 2.1 Fastjson简介

fastjson是阿里巴巴开发的一款专门用于Java开发的包，可以方便的实现json对象与JavaBean对象的转换，实现JavaBean对象与json字符串的转换，实现json对象与json字符串的转换。除了这个fastjson以外，还有Google开发的Gson包，其他形式的如net.sf.json包，都可以实现json的转换。

### 2.2 Fastjson使用

在fastjson包中主要有3个类，JSON,JSONArray，JSONObject

三者之间的关系如下，JSONObject和JSONArray继承JSON



联系上面讲到的json基础知识并对应这三个类，可以发现，JSONObject代表json对象，JSONArray代表json对象数组，JSON代表JSONObject和JSONArray的转化。

### 2.2.1 JSONObject类使用

JSONObject实现了Map接口，而json对象中的数据都是以"键：值"对形式出现， JSONObject底层操作是由Map实现的。类中主要是get()方法。JSONObject相当于json对象，该类中主要封装了各种get方法，通过"键：值"对中的键来获取其对应的值。

### 2.2.2 JSONArray类使用

JSONArray的内部是通过List接口中的方法来完成操作的。JSONArray代表json对象数组，json数组对象中存储的是一个个json对象，所以类中的方法主要用于直接操作json对象。比如其中的add(),remove()，containsAll()方法，对应于json对象的添加，删除与判断。 其内部主要由List接口中的对应方法来实现。

跟JSONObject一样，JSONArray里面也有一些get()方法，不过不常用，最有用的应该是getJSONObject(int index)方法，该方法用于获取json对象数组中指定位置的JSONObject对象，配合size()方法，可用于遍历json对象数组中的各个对象。 通过以上两个方法，在配合for循环，即可实现json对象数组的遍历。此外JSONArray中也实现了迭代器方法来遍历。 通过遍历得到JSONObject对象，然后再利用JSONObject类中的get()方法，即可实现最终json数据的获取。

### 2.2.3 JSON类使用

JSON类主要是实现转化用的，最后的数据获取，还是要通过JSONObject和JSONArray来实现。类中的主要是实现json对象，json对象数组，javabean对象，json字符串之间的相互转化。

总结一下fastjson中三个类的用途和方法：

* JSONObject：解析Json对象，获取对象中的值，通常是使用类中的get()方法
* JSONArray：JSON对象数组，通常是通过迭代器取得其中的JSONObject，再利用JSONObeject的get()方法进行取值。
* JSON：主要是实现json对象，json对象数组，javabean对象，json字符串之间的相互转化。 转换之后取值还是按各自的方法进行。

### 三 JSON案例

### 3.1 json字符串—》JSONObject

用**JSON.parseObject()**方法即可将JSon字符串转化为JSON对象，利用JSONObject中的get()方法来获取JSONObject中的相对应的键对应的值

**import** com.alibaba.fastjson.JSON**;**

**import** com.alibaba.fastjson.JSONArray**;**

**import** com.alibaba.fastjson.JSONObject**;**

**import** com.alibaba.fastjson.TypeReference**;**

**import** com.jiyong.config.Student**;**

**import** com.jiyong.config.Teacher**;**

**import** org.junit.Test**;**

​

**import** java.util.ArrayList**;**

**import** java.util.Iterator**;**

**import** java.util.List**;**

​

*/\*\**

*\* Fastjson用法*

*\*/*

​

**public** **class** **FastJsonOper** **{**

*//json字符串-简单对象型,加\转义*

**private** **static** **final** String JSON\_OBJ\_STR **=** "{\"studentName\":\"lily\",\"studentAge\":12}"**;**

​

*//json字符串-数组类型*

**private** **static** **final** String JSON\_ARRAY\_STR **=** " [{\"studentName\":\"lily\",\"studentAge\":12}," **+**

"{\"studentName\":\"lucy\",\"studentAge\":15}]"**;**

​

*//复杂格式json字符串*

**private** **static** **final** String COMPLEX\_JSON\_STR **=** "{\"teacherName\":\"crystall\"," **+**

"\"teacherAge\":27,\"course\":{\"courseName\":\"english\",\"code\":1270},\"students\":[{\"studentName\":\"lily\",\"studentAge\":12},{\"studentName\":\"lucy\",\"studentAge\":15}]}"**;**

​

*// json字符串与JSONObject之间的转换*

@Test

**public** **void** **JsonStrToJSONObject(){**

JSONObject jsonObject **=** JSON**.**parseObject**(**JSON\_OBJ\_STR**);**

System**.**out**.**println**(**"StudentName: " **+** jsonObject**.**getString**(**"studentName"**)** **+** "," **+** "StudentAge: " **+** jsonObject**.**getInteger**(**"studentAge"**));**

​

**}**

​

**}**

### 3.2 JSONObject—》json字符串

用JSON.toJSONString()方法即可将JSON对象转化为JSON字符串

*/\*\**

*\* 将JSONObject转换为JSON字符串,用JSON.toJSONString()方法即可将JSON字符串转化为JSON对象*

*\*/*

@Test

**public** **void** **JSONObjectToJSONString(){**

JSONObject jsonObject **=** JSON**.**parseObject**(**JSON\_OBJ\_STR**);**

String s **=** JSON**.**toJSONString**(**jsonObject**);**

System**.**out**.**println**(**s**);**

**}**

### 3.3 JSON字符串数组—》JSONArray

将JSON字符串数组转化为JSONArray，通过JSON的parseArray()方法。JSONArray本质上还是一个数组，对其进行遍历取得其中的JSONObject，然后再利用JSONObject的get()方法取得其中的值。有两种方式进行遍历

* 方式一：通过jsonArray.size()获取JSONArray中元素的个数，再通过getJSONObject(index)获取相应位置的JSONObject，循环变量取得JSONArray中的JSONObject,再利用JSONObject的get()进行取值。
* 方式二：通过jsonArray.iterator()获取迭代器

*/\*\**

*\* 将JSON字符串数组转化为JSONArray，通过JSON的parseArray()方法*

*\*/*

@Test

**public** **void** **JSONArrayToJSONStr(){**

JSONArray jsonArray **=** JSON**.**parseArray**(**JSON\_ARRAY\_STR**);**

*/\*\**

*\* JSONArray本质上还是一个数组，对其进行遍历取得其中的JSONObject，然后再利用JSONObject的get()方法取得其中的值*

*\* 方式一是通过jsonArray.size()获取JSONArray中元素的个数，*

*\* 再通过getJSONObject(index)获取相应位置的JSONObject，在利用JSONObject的get()进行取值*

*\* 方式二是通过jsonArray.iterator()获取迭代器*

*\**

*\*/*

*// 遍历方式一*

*// int size = jsonArray.size();*

*// for(int i = 0;i < size;i++){*

*// JSONObject jsonObject = jsonArray.getJSONObject(i);*

*// System.out.println("studentName: " + jsonObject.getString("studentName") + ",StudentAge: " + jsonObject.getInteger("studentAge"));*

*// }*

*// 遍历方式二*

Iterator**<**Object**>** iterator **=** jsonArray**.**iterator**();**

**while** **(**iterator**.**hasNext**()){**

JSONObject jsonObject **=** **(**JSONObject**)** iterator**.**next**();**

System**.**out**.**println**(**"studentName: " **+** jsonObject**.**getString**(**"studentName"**)** **+** ",StudentAge: " **+** jsonObject**.**getInteger**(**"studentAge"**));**

**}**

​

**}**

### 3.4 JSONArray—》json字符串

用JSON.toJSONString()方法即可将JSONArray转化为JSON字符串

*/\*\**

*\* JSONArray到json字符串-数组类型的转换*

*\*/*

@Test

**public** **void** **JSONArrayToJSONString(){**

JSONArray jsonArray **=** JSON**.**parseArray**(**JSON\_ARRAY\_STR**);**

String s **=** JSON**.**toJSONString**(**jsonArray**);**

System**.**out**.**println**(**s**);**

**}**

### 3.5 复杂JSON格式字符串—》JSONObject

将复杂JSON格式字符串转换为JSONObject,也是通过JSON.parseObject()

*/\*\**

*\* 将复杂JSON格式字符串转换为JSONObject,也是通过JSON.parseObject()，可以取其中的部分*

*\*/*

@Test

**public** **void** **JSONStringTOJSONObject(){**

JSONObject jsonObject **=** JSON**.**parseObject**(**COMPLEX\_JSON\_STR**);**

*// 获取简单对象*

String teacherName **=** jsonObject**.**getString**(**"teacherName"**);**

Integer teacherAge **=** jsonObject**.**getInteger**(**"teacherAge"**);**

System**.**out**.**println**(**"teacherName: " **+** teacherName **+** ",teacherAge " **+** teacherAge**);**

*// 获取JSONObject对象*

JSONObject course **=** jsonObject**.**getJSONObject**(**"course"**);**

*// 获取JSONObject中的数据*

String courseName **=** course**.**getString**(**"courseName"**);**

Integer code **=** course**.**getInteger**(**"code"**);**

System**.**out**.**println**(**"courseName: " **+** courseName **+** " code: " **+** code**);**

*// 获取JSONArray对象*

JSONArray students **=** jsonObject**.**getJSONArray**(**"students"**);**

*// 获取JSONArray的中的数据*

Iterator**<**Object**>** iterator **=** students**.**iterator**();**

**while** **(**iterator**.**hasNext**()){**

JSONObject jsonObject1 **=** **(**JSONObject**)** iterator**.**next**();**

System**.**out**.**println**(**"studentName: " **+** jsonObject1**.**getString**(**"studentName"**)** **+** ",StudentAge: "

**+** jsonObject1**.**getInteger**(**"studentAge"**));**

**}**

​

**}**

### 3.6 复杂JSONObject—》json字符串

用JSON.toJSONString()方法即可将复杂JSONObject转化为JSON字符串

*/\*\**

*\* 复杂JSONObject到json字符串的转换*

*\*/*

@Test

**public** **void** **JSONObjectTOJSON(){**

JSONObject jsonObject **=** JSON**.**parseObject**(**COMPLEX\_JSON\_STR**);**

String s **=** JSON**.**toJSONString**(**jsonObject**);**

System**.**out**.**println**(**s**);**

**}**

### 3.7 json字符串—》JavaBean

定义JavaBean类

**package** com.fastjson**;**

​

​

**public** **class** **Student** **{**

**private** String studentName**;**

**private** **int** studentAge**;**

​

**public** **Student()** **{**

**}**

​

**public** **Student(**String studentName**,** **int** studentAge**)** **{**

**this.**studentName **=** studentName**;**

**this.**studentAge **=** studentAge**;**

**}**

​

**public** String **getStudentName()** **{**

**return** studentName**;**

**}**

​

**public** **void** **setStudentName(**String studentName**)** **{**

**this.**studentName **=** studentName**;**

**}**

​

**public** **int** **getStudentAge()** **{**

**return** studentAge**;**

**}**

​

**public** **void** **setStudentAge(int** studentAge**)** **{**

**this.**studentAge **=** studentAge**;**

**}**

​

@Override

**public** String **toString()** **{**

**return** "Student{" **+**

"studentName='" **+** studentName **+** '\'' **+**

", studentAge=" **+** studentAge **+**

'}'**;**

**}**

**}**

​

**package** com.jiyong.config**;**

​

*/\*\**

*\* 对于复杂嵌套的JSON格式，利用JavaBean进行转换的时候要注意*

*\* 1、有几个JSONObject就定义几个JavaBean*

*\* 2、内层的JSONObject对应的JavaBean作为外层JSONObject对应的JavaBean的一个属性*

*\* 3、解析方法有两种*

*\* 第一种方式,使用TypeReference<T>类*

*\* Teacher teacher = JSONObject.parseObject(COMPLEX\_JSON\_STR, new TypeReference<Teacher>() {})*

*\* 第二种方式,使用Gson思想*

*\* Teacher teacher = JSONObject.parseObject(COMPLEX\_JSON\_STR, Teacher.class);*

*\*/*

​

**import** java.util.List**;**

​

**public** **class** **Teacher** **{**

**private** String teacherName**;**

**private** **int** teacherAge**;**

**private** Course course**;**

**private** List**<**Student**>** students**;**

​

**public** **Teacher()** **{**

**}**

​

**public** **Teacher(**String teacherName**,** **int** teacherAge**,** Course course**,** List**<**Student**>** students**)** **{**

**this.**teacherName **=** teacherName**;**

**this.**teacherAge **=** teacherAge**;**

**this.**course **=** course**;**

**this.**students **=** students**;**

**}**

​

**public** String **getTeacherName()** **{**

**return** teacherName**;**

**}**

​

**public** **void** **setTeacherName(**String teacherName**)** **{**

**this.**teacherName **=** teacherName**;**

**}**

​

**public** **int** **getTeacherAge()** **{**

**return** teacherAge**;**

**}**

​

**public** **void** **setTeacherAge(int** teacherAge**)** **{**

**this.**teacherAge **=** teacherAge**;**

**}**

​

**public** Course **getCourse()** **{**

**return** course**;**

**}**

​

**public** **void** **setCourse(**Course course**)** **{**

**this.**course **=** course**;**

**}**

​

**public** List**<**Student**>** **getStudents()** **{**

**return** students**;**

**}**

​

**public** **void** **setStudents(**List**<**Student**>** students**)** **{**

**this.**students **=** students**;**

**}**

​

@Override

**public** String **toString()** **{**

**return** "Teacher{" **+**

"teacherName='" **+** teacherName **+** '\'' **+**

", teacherAge=" **+** teacherAge **+**

", course=" **+** course **+**

", students=" **+** students **+**

'}'**;**

**}**

**}**

**package** com.jiyong.config**;**

​

**public** **class** **Course** **{**

**private** String courseName**;**

**private** **int** code**;**

​

**public** **Course()** **{**

**}**

​

**public** **Course(**String courseName**,** **int** code**)** **{**

**this.**courseName **=** courseName**;**

**this.**code **=** code**;**

**}**

​

**public** String **getCourseName()** **{**

**return** courseName**;**

**}**

​

**public** **void** **setCourseName(**String courseName**)** **{**

**this.**courseName **=** courseName**;**

**}**

​

**public** **int** **getCode()** **{**

**return** code**;**

**}**

​

**public** **void** **setCode(int** code**)** **{**

**this.**code **=** code**;**

**}**

​

@Override

**public** String **toString()** **{**

**return** "Course{" **+**

"courseName='" **+** courseName **+** '\'' **+**

", code=" **+** code **+**

'}'**;**

**}**

**}**

Jason字符串转换为JavaBean有三种方式，推荐通过反射的方式。

*/\*\**

*\* json字符串-简单对象到JavaBean之间的转换*

*\* 1、定义JavaBean对象*

*\* 2、Jason字符串转换为JavaBean有三种方式，推荐通过反射的方式*

*\*/*

@Test

**public** **void** **JSONStringToJavaBeanObj(){**

*// 第一种方式*

JSONObject jsonObject **=** JSON**.**parseObject**(**JSON\_OBJ\_STR**);**

String studentName **=** jsonObject**.**getString**(**"studentName"**);**

Integer studentAge **=** jsonObject**.**getInteger**(**"studentAge"**);**

Student student **=** **new** Student**(**studentName**,** studentAge**);**

*// 第二种方式，//第二种方式,使用TypeReference<T>类,由于其构造方法使用protected进行修饰,故创建其子类*

Student student1 **=** JSON**.**parseObject**(**JSON\_OBJ\_STR**,** **new** TypeReference**<**Student**>()** **{});**

*// 第三种方式，通过反射，建议这种方式*

Student student2 **=** JSON**.**parseObject**(**JSON\_OBJ\_STR**,** Student**.**class**);**

​

**}**

### 3.8 JavaBean —》json字符串

也是通过JSON的toJSONString，不管是JSONObject、JSONArray还是JavaBean转为为JSON字符串都是通过JSON的toJSONString方法。

​

*/\*\**

*\* JavaBean转换为Json字符串,也是通过JSON的toJSONString，不管是JSONObject、JSONArray还是JavaBean转为为JSON字符串都是通过JSON的toJSONString方法*

*\*/*

@Test

**public** **void** **JavaBeanToJsonString(){**

Student lily **=** **new** Student**(**"lily"**,** 12**);**

String s **=** JSON**.**toJSONString**(**lily**);**

System**.**out**.**println**(**s**);**

**}**

### 3.8 json字符串数组—》JavaBean-List

​

*/\*\**

*\* json字符串-数组类型到JavaBean\_List的转换*

*\*/*

@Test

**public** **void** **JSONStrToJavaBeanList(){**

*// 方式一：*

JSONArray jsonArray **=** JSON**.**parseArray**(**JSON\_ARRAY\_STR**);**

*//遍历JSONArray*

List**<**Student**>** students **=** **new** ArrayList**<**Student**>();**

Iterator**<**Object**>** iterator **=** jsonArray**.**iterator**();**

**while** **(**iterator**.**hasNext**()){**

JSONObject next **=** **(**JSONObject**)** iterator**.**next**();**

String studentName **=** next**.**getString**(**"studentName"**);**

Integer studentAge **=** next**.**getInteger**(**"studentAge"**);**

Student student **=** **new** Student**(**studentName**,** studentAge**);**

students**.**add**(**student**);**

**}**

*// 方式二,使用TypeReference<T>类,由于其构造方法使用protected进行修饰,故创建其子类*

List**<**Student**>** studentList **=** JSON**.**parseObject**(**JSON\_ARRAY\_STR**,new** TypeReference**<**ArrayList**<**Student**>>()** **{});**

*// 方式三，使用反射*

List**<**Student**>** students1 **=** JSON**.**parseArray**(**JSON\_ARRAY\_STR**,** Student**.**class**);**

System**.**out**.**println**(**students1**);**

​

**}**

### 3.10 JavaBean-List —》json字符串数组

*/\*\**

*​*

*\* JavaBean\_List到json字符串-数组类型的转换,直接调用JSON.toJSONString()方法即可*

*​*

*\*/*

​

@Test

​

**public** **void** **JavaBeanListToJSONStr(){**

​

Student student **=** **new** Student**(**"lily"**,** 12**);**

​

Student student1 **=** **new** Student**(**"lucy"**,** 13**);**

​

List**<**Student**>** students **=** **new** ArrayList**<**Student**>();**

​

students**.**add**(**student**);**

​

students**.**add**(**student1**);**

​

String s **=** JSON**.**toJSONString**(**student**);**

​

System**.**out**.**println**(**s**);**

​

**}**

​

### 3.11 复杂嵌套json格式字符串—》JavaBean\_obj

对于复杂嵌套的JSON格式，利用JavaBean进行转换的时候要注意：

1. 有几个JSONObject就定义几个JavaBean
2. 内层的JSONObject对应的JavaBean作为外层JSONObject对应的JavaBean的一个属性

*/\*\**

*\* 复杂json格式字符串到JavaBean\_obj的转换*

*\*/*

@Test

**public** **void** **ComplexJsonStrToJavaBean(){**

*//第一种方式,使用TypeReference<T>类,由于其构造方法使用protected进行修饰,故创建其子类*

Teacher teacher **=** JSON**.**parseObject**(**COMPLEX\_JSON\_STR**,** **new** TypeReference**<**Teacher**>()** **{});**

*// 第二种方式，使用反射*

Teacher teacher1 **=** JSON**.**parseObject**(**COMPLEX\_JSON\_STR**,** Teacher**.**class**);**

**}**

### 3.12 JavaBean\_obj —》复杂json格式字符串

​

*/\*\**

*\* 复杂JavaBean\_obj到json格式字符串的转换*

*\*/*

@Test

**public** **void** **JavaBeanToComplexJSONStr(){**

Teacher teacher **=** JSON**.**parseObject**(**COMPLEX\_JSON\_STR**,** Teacher**.**class**);**

String s **=** JSON**.**toJSONString**(**teacher**);**

System**.**out**.**println**(**s**);**

**}**

# TypeReference

**1. 基础使用**

在fastjson中提供了一个用于处理泛型反序列化的类TypeReference。

import com.alibaba.fastjson.TypeReference;

List<VO> list = JSON.parseObject("...", new TypeReference<List<VO>>() {});

如下写法有更好的性能

import com.alibaba.fastjson.TypeReference;

final static Type type = new TypeReference<List<VO>>() {}.getType();

List<VO> list = JSON.parseObject("...", type);

在这里例子中，通过TypeReference能够解决List中T的类型问题。

**2. 带参数使用**

在1.2.9 & 1.1.49.android版本中，TypeReference支持泛型参数，方便一些框架实现通用的反序列化类。用法如下：

**2.1. 单参数例子**

public class Response<T> {

public T data;

}

public static <T> Response<T> parseToMap(String json, Class<T> type) {

return JSON.parseObject(json,

new TypeReference<Response<T>>(type) {});

}

**2.2. 双参数例子**

public static <K, V> Map<K, V> parseToMap(String json,

Class<K> keyType,

Class<V> valueType) {

return JSON.parseObject(json,

new TypeReference<Map<K, V>>(keyType, valueType) {

});

}

// 可以这样使用

String json = "{1:{name:\"ddd\"},2:{name:\"zzz\"}}";

Map<Integer, Model> map = parseToMap(json, Integer.class, Model.class);

assertEquals("ddd", map.get(1).name);

assertEquals("zzz", map.get(2).name);