# 一、JSON简介

## 1、什么是JSON

JSON是行业使用广泛的一种数据格式。

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式。简单地说，JSON可以将 JavaScript 对象中表示的一组数据转换为字符串，然后就可以在函数之间轻松地传递这个字符串，或者在异步应用程序中将字符串从 Web 客户机传递给服务器端程序。这个字符串看起来有点儿古怪，但是JavaScript很容易解释它，而且 JSON 可以表示比 "名称 / 值对" 更复杂的结构。

总结什么是 JSON ？

JSON 指的是 JavaScript 对象表示法（JavaScript Object Notation）；

JSON 是**轻量级**的文本**数据交换格式；**

JSON **独立于语言** \*；

JSON 具有自我描述性，更易理解。

\* JSON 使用 JavaScript 语法来描述数据对象（JSON 语法是 JavaScript 对象表示法语法的子集），但是 JSON 仍然独立于语言和平台。JSON 解析器和 JSON 库支持许多不同的编程语言。

JSON的优点：易于人的阅读和编写，方便程序解析与生成。

## 2、JSON语法结构

标准的JSON数据表示

数据结构

Object：使用花括号 {} 包含的键值对结构，**key必须是string类型，value为任何基本类型或数据结构**，逗号分隔数据。

Array：使用中括号 [] 来表示，并用逗号来分隔元素。

基本类型：string、number、true、false、null。

简单实例（first.json）

{

"name" : "张三",

"age" : 25,

"birthday" : "1994-12-12",

"school" : "西安电子科技大学",

"major" : ["Java", "C++"],

"has\_house" : false,

"house" : null,

"car" : {

"sign" : "奔驰",

"model" : "SUV"

},

"comment" : "标准JSON里面没有注释"

}

# 二、使用Java操作JSON

使用Java操作JSON可以使用org.json包（Android SDK中使用的JSON官方库），在JSON的官网：http://www.json.org/中可以找到。使用Maven项目则添加如下依赖即可：

<dependency>

<groupId>org.json</groupId>

<artifactId>json</artifactId>

<version>20180813</version>

</dependency>

## 1、创建JSON对象的几种方式

**import** java.util.HashMap;

**import** java.util.Map;

**import** org.json.JSONObject;

**import** com.zxt.demo.Car;

**import** com.zxt.demo.Person;

**public** **class** JSONObjectTest {

**public** **static** **void** main(String[] args) {

*jSONObject*();

*creatJsonByMap*();

*creatJsonByBean*();

}

/\*\*

\* **@Description**：使用JSONObject来构建JSON对象

\*/

**private** **static** **void** jSONObject() {

// 创建一个JSONObject对象

JSONObject zhangsan = **new** JSONObject();

Object nullObj = **null**;

zhangsan.put("name", "张三");

zhangsan.put("age", 25);

zhangsan.put("birthday", "1994-12-12");

zhangsan.put("school", "西安电子科技大学");

zhangsan.put("major", **new** String[] {"Java", "C++"});

zhangsan.put("has\_house", **false**);

zhangsan.put("house", nullObj);

JSONObject car = **new** JSONObject();

car.put("sign", "奔驰");

car.put("model", "SUV");

zhangsan.put("car", car);

zhangsan.put("comment", "标准JSON里面没有注释");

// print

System.***out***.println(zhangsan.toString());

}

/\*\*

\* **@Description**：使用Map来构建JSON对象

\*/

**private** **static** **void** creatJsonByMap() {

Map<String, Object> zhangsan = **new** HashMap<String, Object>();

zhangsan.put("name", "张三");

zhangsan.put("age", 25);

zhangsan.put("birthday", "1994-12-12");

zhangsan.put("school", "西安电子科技大学");

zhangsan.put("major", **new** String[] {"Java", "C++"});

zhangsan.put("has\_house", **false**);

zhangsan.put("house", **null**);

Map<String, Object> car = **new** HashMap<String, Object>();

car.put("sign", "奔驰");

car.put("model", "SUV");

zhangsan.put("car", car);

zhangsan.put("comment", "标准JSON里面没有注释");

// JSONObject支持使用Map对象直接构造JSON对象

System.***out***.println(**new** JSONObject(zhangsan));

}

/\*\*

\* **@Description**：使用Java Bean对象来构建JSON对象 （推荐使用的方式）

\*/

**private** **static** **void** creatJsonByBean() {

Person zhangsan = **new** Person();

zhangsan.setName("张三");

zhangsan.setAge(25);

zhangsan.setBirthday("1994-12-12");

zhangsan.setSchool("西安电子科技大学");

zhangsan.setMajor(**new** String[] {"Java", "C++"});

zhangsan.setHas\_house(**false**);

zhangsan.setHouse(**null**);

zhangsan.setCar(**new** Car("奔驰", "SUV"));

zhangsan.setComment("标准JSON里面没有注释");

// JSONObject支持使用Java Bean对象直接构造JSON对象

System.***out***.println(**new** JSONObject(zhangsan));

}

}

## 2、从json文件中读取JSON对象

1）、首先引入commons-io的依赖，以便方便读取文件

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.4</version>

</dependency>

2）、读取json文件，并操作JSON对象

**import** java.io.File;

**import** java.io.IOException;

**import** org.apache.commons.io.FileUtils;

**import** org.json.JSONArray;

**import** org.json.JSONObject;

**public** **class** ReadJSONTest {

**public** **static** **void** main(String[] args) {

File file = **new** File(ReadJSONTest.**class**.getResource("/zhangsan.json").getFile());

**try** {

String content = FileUtils.*readFileToString*(file);

JSONObject jsonObject = **new** JSONObject(content);

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

// 一般来说需要先对要获取的属性进行判空操作，防止异常发生

**if** (!jsonObject.isNull("name")) {

System.***out***.println("姓名是：" + jsonObject.getString("name"));

}

System.***out***.println("年龄是：" + jsonObject.getInt("age"));

System.***out***.println("是否有房子：" + jsonObject.getBoolean("has\_house"));

System.***out***.println("汽车：" + jsonObject.getJSONObject("car"));

JSONArray jsonArray = jsonObject.getJSONArray("major");

**for** (Object str : jsonArray) {

System.***out***.println("专业：" + str);

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

# 三、GSON介绍

org.json包进行JSON对象的创建与操作（JSONObject），具有两个缺点：

1、没有日期类型；

2、可以由Java Bean对象生成JSON对象，但是无法通过JSON对象反解析出Java Bean对象。

这里介绍一种更强大的Java 处理JSON数据的包，gson（Gson）。

<dependency>

<groupId>com.google.code.gson</groupId>

<artifactId>gson</artifactId>

<version>2.8.5</version>

</dependency>

## 1、创建JSON对象

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

**public** **class** Person {

@SerializedName("myname")

**private** String name;

**private** Integer age;

**private** String birthday;

**private** String school;

// 使用Gson进行解析时，这里可以直接换成Java的集合类，list、set等，Gson可以直接解析出来

**private** String[] major;

**private** **boolean** has\_house;

**private** String house;

**private** Car car;

**private** String comment;

// 有些属性不需要转换为JSON的key，则声明变量时添加 transient

**private** **transient** String ignore;

// 此处省略了一系列set和get方法

@Override

**public** String toString() {

**return** "Person [name=" + name + ", age=" + age + ", birthday=" + birthday + ", school=" + school + ", major="

+ Arrays.*toString*(major) + ", has\_house=" + has\_house + ", house=" + house + ", car=" + car

+ ", comment=" + comment + "]";

}

}

**import** java.lang.reflect.Field;

**import** com.google.gson.FieldNamingStrategy;

**import** com.google.gson.Gson;

**import** com.google.gson.GsonBuilder;

**import** com.zxt.demo.Car;

**import** com.zxt.demo.Person;

**public** **class** GsonTest {

**public** **static** **void** main(String[] args) {

Person zhangsan = **new** Person();

zhangsan.setName("张三");

zhangsan.setAge(25);

zhangsan.setBirthday("1994-12-12");

zhangsan.setSchool("西安电子科技大学");

zhangsan.setMajor(**new** String[] {"Java", "C++"});

zhangsan.setHas\_house(**false**);

zhangsan.setHouse(**null**);

zhangsan.setCar(**new** Car("奔驰", "SUV"));

zhangsan.setComment("标准JSON里面没有注释");

/\*\*

\* Java Bean中的有些属性不需要转换为JSON的key，则声明变量时添加 transient

\*/

zhangsan.setIgnore("不要看见我");

/\*\*

\* 若不做处理，则生成的JSON对象的key就是Java Bean对象的属性名

\* Gson可以通过在Java Bean对象的属性上面使用 **@SerializedName**("")注解， 来设置生成的JSON的key名称

\* org.json包中的JSONObject则没有这个功能，无法修改生成的JSON对象key名称

\*/

// Gson gson = new Gson();

/\*\*

\* 使用GsonBuilder来生成JSON对象，可以拥有更多操作

\*/

GsonBuilder gsonBuilder = **new** GsonBuilder();

// 格式化输出

gsonBuilder.setPrettyPrinting();

// 更改JSON对象key的名称

gsonBuilder.setFieldNamingStrategy(**new** FieldNamingStrategy() {

@Override

**public** String translateName(Field field) {

**if**(field.getName().equals("age")) {

**return** "myage";

}

**return** field.getName();

}

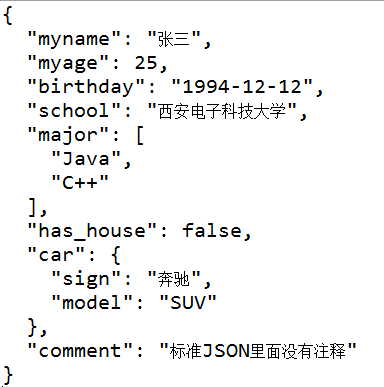
});

Gson gson = gsonBuilder.create();

System.***out***.println(gson.toJson(zhangsan));

}

}



## 2、解析JSON对象

使用GSON解析JSON对象时，可以把JSON对象直接转换成Java Bean对象，不仅如此GSON还支持Java Bean的Date属性的数据的解析。并且数组类型的属性可以直接使用Java的集合类表示，GSON也可以方便地解析出来。

**JSON解析成Java Bean对象**

**import** java.io.File;

**import** java.io.IOException;

**import** org.apache.commons.io.FileUtils;

**import** com.google.gson.Gson;

**import** com.zxt.demo.Person;

**import** com.zxt.json.ReadJSONTest;

**public** **class** ReadGsonTest {

**public** **static** **void** main(String[] args) {

File file = **new** File(ReadJSONTest.**class**.getResource("/zhangsan.json").getFile());

**try** {

String content = FileUtils.*readFileToString*(file);

Gson gson = **new** Gson();

// Gson可以直接由JSON字符串解析为Java Bena对象，JSONObject没有这个功能

Person person = gson.fromJson(content, Person.**class**);

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

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**GSON解析成带有日期格式的Java Bean对象**

**import** java.io.File;

**import** java.io.IOException;

**import** org.apache.commons.io.FileUtils;

**import** com.google.gson.Gson;

**import** com.google.gson.GsonBuilder;

**import** com.zxt.demo.PersonWithBirthday;

**import** com.zxt.json.ReadJSONTest;

**public** **class** ReadGsonWithDataTest {

**public** **static** **void** main(String[] args) {

File file = **new** File(ReadJSONTest.**class**.getResource("/zhangsan.json").getFile());

**try** {

String content = FileUtils.*readFileToString*(file);

GsonBuilder gsonBuilder = **new** GsonBuilder();

// 设置日期解析格式

gsonBuilder.setDateFormat("yyyy-MM-dd");

Gson gson = gsonBuilder.create();

PersonWithBirthday person = gson.fromJson(content, PersonWithBirthday.**class**);

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

} **catch** (IOException e) {

e.printStackTrace();

}

}

}