# JSON

JSON无论是在Web开发还是服务器中开发中，都是相当重要的数据传输格式。一般情况下，我们不需要关心JSON的解析和构造的性能，除非系统对性能要求很高的情况下。

在使用JSON库解析时，最好为JavaBean提供默认的构造方法，否则可能导致解析失败。因为将Json字符串解析为JavaBean时是通过默认的构造方法来获取实例的。

目前Java开源的JSON库有很多种，其中最常用的有以下四种：Gson，FastJson，Jackson，Json-lib。

1. Fastjson是由阿里巴巴使用Java语言编写的一个高性能的JSON处理器。

优点：

它没有依赖外部Jar包，因此使用时直接导入Fastjson就可以使用，无需再添加其他jar包。

Fastjson使用独创的算法，将解析速度提升到极致，超过所有的Json库。

缺点：

Fastjson在复杂Bean转换为Json时可能会出现引用类型的问题，导致json转换报错。需要制定引用。

1. 和Json-lib相比，Jackson依赖的jar包较少，简单易用并且性能也高，尤其是数据量较大时。

优点：

在处理数据量较大时，性能在几种Json库中最好。

Jackson社区相对比较活跃，更新速度也相对较快。

缺点：

Jackson在将一些比较复杂的Json转换为Bean时可能会出现问题，比如一些集合Map和List等。

Jackson在将复杂类型的Bean转换为Json时也不是标准的Json格式。

1. Gson是目前功能最全的Json解析神器。是谷歌公司自行研发的。

优点：

不需要添加额外的Jar包，可以直接在JDK上运行。

只要类里面有属性的get和set方法，Gson可以完成复杂类型的Json到Bean或Bean到Json的转换工作。

缺点：

在将Json字符串转换为对象之前，需要创建好对象，并设置好它的成员，才能成功转换。

1. Json-lib是最早也是曾经应用最广泛的json解析工具。但是Json-lib确实依赖了太多第三方的Jar包，包括commons-beanutils.jar，commons-collections-3.2.jar，commons-lang-2.6.jar，commons-logging-1.1.1.jar，ezmorph-1.0.6.jar。

优点：

使用简单，最常用的只有JSONObject、JSONArray和JsonConfig对象。

缺点：

依赖了太多第三方的Jar包。

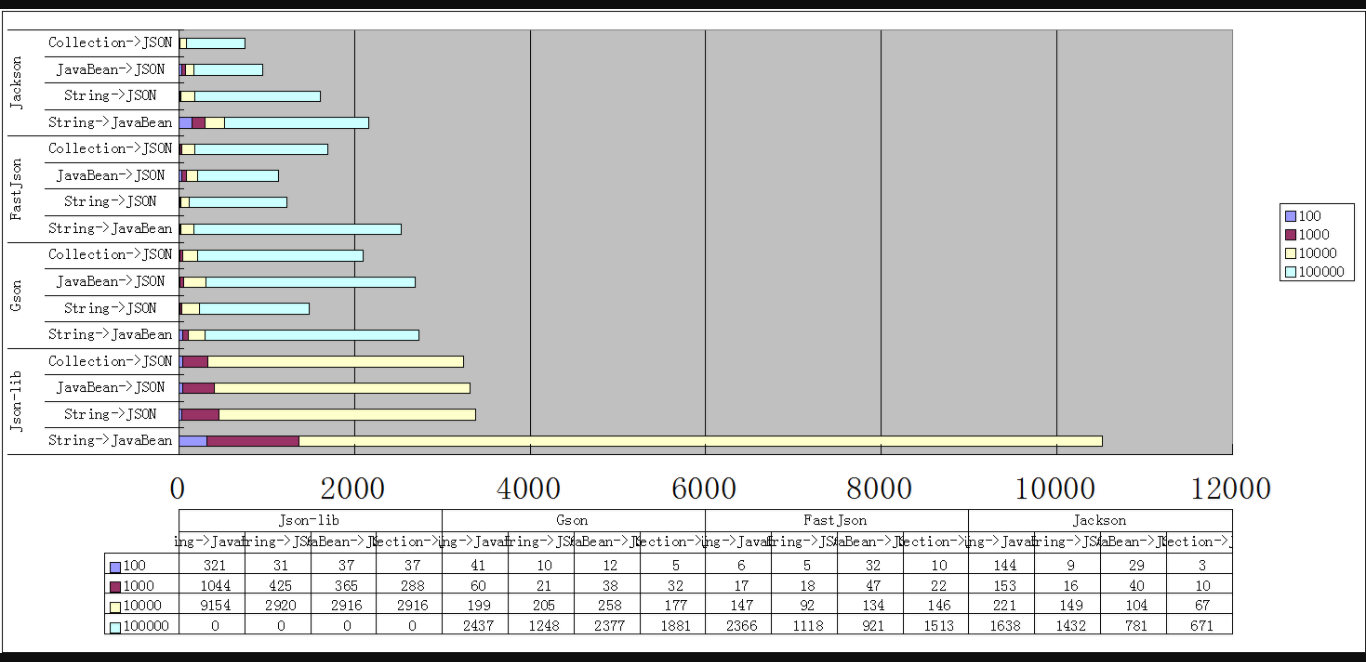
对于复杂类型的Json，比如一个对象里面包含另一个对象的List集合，在将Json字符串转换为Bean时，会丢失对象集合。也就是转换存在问题。

性能较差，尤其是当数据量增大的时候。因此，Json-lib无论是性能还是功能上都不能满足现在的互联网需求。

通过比较我们大概了解了几种Json库的优劣问题。但在项目中如何选择一个合适的JSON库需要从下面几个方面考虑：

1. 字符串解析成JSON性能；
2. 字符串解析成JavaBean性能；
3. JavaBean构造JSON性能；
4. 集合构造JSON性能；
5. 易用性。

前面四条主要是JSON库的解析构造成本，最后一条则是开发成本，也就是学习成本。如果开发难度较大，也是不可取的。毕竟除了Json-lib外，其余三种JSON库的性能差异并不大。



从上到下，分别是：Jackson、Fastjson、Gson和Json-lib的性能展示。

从易用性角度来分析的话，FastJson的API设计的最简单，最方便使用，直接使用JSON的两个静态方法即可完成四种操作；而Gson和Jackson都需要new一个对象，虽然这个对象可以复用，但是在实际使用过程中还需要用一个全局变量来保存改变量，同时API设计的也不是很好理解，对于FastJson来说复杂的API是因为他支持流式解析，适合对JSON进行大量且复杂的操作，但是实际应用中对于JSON的操作都是简单的解析成JavaBean，然后JavaBean序列化成JSON字符串即可，复杂的操作很少。

## FastJson

FastJson被誉为解析速度最快的JSON库。并且在进行一般的JSON和Bean相互转换时，使用非常简单。

FastJson中最常使用的是JSON、JSONObject和JSONArray对象。

### JSON

JSON中主要提供了两种方法，一种是parse方法，用于将Json字符串转换为JSONObject、JavaBean或Java对象；另一种是toJSON方法，主要用于将Java对象转换为Json字符串形式。

Parse系列方法：

JSON提供了大约20个parse方法，用于适应不同的场景。一般我们使用parseObject(String text, Class<T> clazz)或parse(String text)方法即可。

**public static** Object parse(String text)

**public static** Object parse(String text, **int** features)

…

**public static** JSONObject parseObject(String text)

**public static** <T> T parseObject(String text, Class<T> clazz)

**public static** <T> T parseObject(String input, Type clazz, Feature... features)

即使一个JavaBean对象中有另一个JavaBean对象的list集合，通过parseObject(String text, Class<T> clazz)方法，也可以轻松进行解析。

User user = **new** User(1L, **"用户"**, **"测试"**);  
Student student = **new** Student(1L, **"阿里巴巴"**, 23);  
user.setStudentList(Arrays.*asList*(student));  
User user1 = JSON.*parseObject*(jsonString, User.**class**);  
System.***out***.println(user1);

// User{userId=1, userName='用户', sex='测试', studentList=[Student{studentId=1, name='阿里巴巴', age=23}]}

toJSON系列方法：

JSON提供了10个以上的toJSON方法，用于适用不同的场景。

**public static** String toJSONString(Object object)

**public static** String toJSONString(Object object, **boolean** prettyFormat)

**public static** String toJSONStringWithDateFormat(Object object, String dateFormat, SerializerFeature... features)

…

**public static** String toJSONString(Object object, SerializeConfig config, SerializerFeature... features)

**public static byte**[] toJSONBytes(Object object, SerializeConfig config, SerializerFeature... features)a

toJSON方法使用起来更简单，一般可以直接将Java对象JSON格式化。当调用toJSONString(Object object, boolean prettyFormat)方法，并给第二个boolean型传参true时，则可以输出标准化的Json字符串。

**public static** <T> T toJavaObject(JSON json, Class<T> clazz)

**private static void** setFilter(JSONSerializer serializer, SerializeFilter... filters)

**private static void** setFilter(JSONSerializer serializer, SerializeFilter filter)

### JSONObject

由于FastJson提供了JSON类，由JSON负责解析构造过程，所以JSONObject和JSONArray对象的作用相较于Json-lib中的就小了很多。当然，也支持基本的JSONObject对象属性的增加删除获取。

### JSONArray

## Jackson

内容挺多的，怪不得效率高，没占主导地位。学习成本稍高。

菜鸟教程：http://www.yiibai.com/jackson/jackson\_environment.html

博文：<http://blog.csdn.net/sdyy321/article/details/40298081>

<http://blog.csdn.net/a123demi/article/details/43112547>

## Gson

## Json-lib

Json-lib，即net.sf.json。可以将Java对象转成json格式的字符串，也可以将Java对象转换成xml格式的文档，同样可以将json字符串转换成Java对象或是将xml字符串转换成Java对象。

net.sf.json中使用的类主要是以下几个：JSONObject、JSONArray和JsonConfig。

在使用Json-lib时，除了需要要导入Json-lib的jar包外，还额外依赖如下几个jar包：

jakarta commons-lang 2.5

jakarta commons-beanutils 1.8.0

jakarta commons-collections 3.2.1

jakarta commons-logging 1.1.1

ezmorph 1.0.6

如果是Maven项目，需要额外添加如下依赖（注意：需要使用classifier标签修饰json-lib，同时要注意依赖版本之间不要冲突）：

<**dependency**>  
 <**groupId**>net.sf.json-lib</**groupId**>  
 <**artifactId**>json-lib</**artifactId**>  
 <**version**>2.4</**version**>

<**classifier**>jdk15</**classifier**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>commons-beanutils</**groupId**>  
 <**artifactId**>commons-beanutils</**artifactId**>  
 <**version**>1.8.0</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>commons-collections</**groupId**>  
 <**artifactId**>commons-collections</**artifactId**>  
 <**version**>3.2.2</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>commons-lang</**groupId**>  
 <**artifactId**>commons-lang</**artifactId**>  
 <**version**>2.6</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>commons-logging</**groupId**>  
 <**artifactId**>commons-logging</**artifactId**>  
 <**version**>1.2</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>net.sf.ezmorph</**groupId**>  
 <**artifactId**>ezmorph</**artifactId**>  
 <**version**>1.0.6</**version**>  
</**dependency**>

### JSONObject

JSONObject是一个Json对象。在JavaBean与Json格式转换之间发挥着重要作用。

http://blog.csdn.net/itlwc/article/details/38442667

<http://blog.csdn.net/z69183787/article/details/18655143>

<https://www.cnblogs.com/handsomeye/p/5107398.html>

http://blog.csdn.net/lishuangzhe7047/article/details/28880009

#### 获取JSONObject

构造方法：

JSONObject提供了两种构造方法，一种是默认的无参构造方法；一种是构造JSONObject时传递一个boolean型参数。

**public** JSONObject()

**public** JSONObject(**boolean** isNull)

也就是如果我们在new一个Json对象时，传递一个true，则默认构造一个空的JSONObject实例，打印结果为null，但并不会抛出空指针异常。

JSONObject jsonObject = **new** JSONObject();  
System.***out***.println(jsonObject); //{}  
JSONObject jsonObject1 = **new** JSONObject(**true**);  
System.***out***.println(jsonObject1); //null

fromObject方法：

fromObject方法会返回一个JSONObject对象，它主要用于将一个对象转换为一个JSONObject对象，实现Json格式转换。

**public static** JSONObject fromObject(Object object)

**public static** JSONObject fromObject(Object object, JsonConfig jsonConfig)

JSONObject提供了如上形式的fromObject方法。我们一般在进行对象转换时，调用第一种方法即可，它会自动帮我们创建一个jsonConfig对象，然后调用第二个fromObject方法。如果我们在转换时，有特殊需要，可以自定义一个JsonConfig对象，然后直接调用第二种方式获取JSONObject实例。

String jsonData = **"{\"name\":\"阿里云\"}"**;  
JSONObject jsonObject = JSONObject.*fromObject*(jsonData);  
System.***out***.println(jsonObject); //"name":"阿里云"}  
  
User user = **new** User(1L, **"阿里云"**, **"网络"**);  
JSONObject jsonObject1 = JSONObject.*fromObject*(user);  
System.***out***.println(jsonObject1); // {"sex":"网络","userId":1,"userName":"阿里云"}

#### 2 json对象转换为Java或JavaBean对象

JSONObject中提供了5中toBean方法，用于将一个JSONObject对象转换为一个普通的Java对象或者一个JavaBean。但其实几种方式只是方法传参不一样。一般我们使用的是第二种方式将json对象转换为Java对象，它会在方法内部自动创建一个空的JsonConfig，然后调用第四种声明方式获取一个Object实例。最后将得到的Object实例进行类型转换即可。

**public static** Object toBean(JSONObject jsonObject)

**public static** Object toBean(JSONObject jsonObject, Class beanClass)

**public static** Object toBean(JSONObject jsonObject, Class beanClass, Map classMap)

**public static** Object toBean(JSONObject jsonObject, JsonConfig jsonConfig)

**public static** Object toBean(JSONObject jsonObject, Object root, JsonConfig jsonConfig)

如果有特殊需要，我们可以自定义一个JsonConfig，然后直接调用第4种或第5种方式进行类型转换。

String jsonData = **"{\"name\":\"阿里云\"}"**;  
JSONObject jsonObject = JSONObject.*fromObject*(jsonData);  
String jsonData1 = JSONObject.*toBean*(jsonObject).toString();  
System.***out***.println(jsonData1);  
  
User user = **new** User(1L, **"阿里云"**, **"网络"**);  
JSONObject jsonObject1 = JSONObject.*fromObject*(user);  
User user1 = (User) JSONObject.*toBean*(jsonObject1, User.**class**);  
System.***out***.println(user1);

#### 3 设值方法

JSONObject提供了3种方法，用于json对象的设值问题。三种方式都可以将value保存到JSONObject对应的key下，但仍有一些细微的差别。

1. put方法

将value映射到JSONObject对应的key下。如果此时，key已经有了一个对应的value值，当前的value值会替换掉旧的value值。也就是值覆盖。

**public** Object put(Object key, Object value)

**public void** putAll(Map map)

**public void** putAll(Map map, JsonConfig jsonConfig)

putAll方法中传递的是一个键值对Map，它会通过迭代器对Map进行遍历，循环将Map中的键值对设置如JSONObject中。

1. element方法

将键值对设置到JSONObject对象中。如果当前value为空（null），如果key存在的话，会将这个key从JSONObject中移除。如果这个key有对应的value值，那么新值会将旧值覆盖。

element方法一般接收两个参数，也就是键和值。其中键是String类型的，而值可以是boolean、int、long、double、Collection、Map和Object型。

**public** JSONObject element(String key, **boolean** value)

…

**public** JSONObject element(String key, Map value)

**public** JSONObject element(String key, Map value, JsonConfig jsonConfig)

**public** JSONObject element(String key, Object value)

**public** JSONObject element(String key, Object value, JsonConfig jsonConfig)

注意：element方法中的第二个参数value不能直接传个null，但可以传一个为null的对象。而accumulate方法可以直接传null。

JSONObject jsonObject = **new** JSONObject();  
String nullStr = **null**;  
jsonObject.element(**"1"**, **"一"**);  
jsonObject.element(**"2"**, **"二"**);  
jsonObject.element(**"3"**, **"三"**);  
System.***out***.println(jsonObject); // {"1":"一","2":"二","3":"三"}  
jsonObject.element(**"3"**, **"交集"**);  
System.***out***.println(jsonObject); // {"1":"一","2":"二","3":"交集"}  
jsonObject.element(**"3"**,nullStr );  
System.***out***.println(jsonObject); // {"1":"一","2":"二"}

1. accumulate方法

accumulate方法和element方法类似。不同的是，将键值对设置到JSONObject中时，如果这个key已经有对应的value，那么会新建一个JSONArray对象，然后将两个value值都放入JSONArray中，并将这个JSONarray作为key对应的value值，设置到JSONObject中。如果JSONObject对应key的值已经是一个JSONArray，则直接将新的value值添加进对应key的JSONArray中。达到值累加的目的。

Accumulate方法也提供了多种方式用于接收不同类型的参数，它的value值可以为boolean、int、double、long和Object。另外，它的accumulateAll方法接收Map类型的参数。

**public** JSONObject accumulate(String key, **boolean** value)

…

**public** JSONObject accumulate(String key, Object value)

**public** JSONObject accumulate(String key, Object value, JsonConfig jsonConfig)

**public void** accumulateAll(Map map)

**public void** accumulateAll(Map map, JsonConfig jsonConfig)

element方法当key存在，value传null时候，会将key从JSONObject中移除；但null也属于Object类型，所以accumulate方法会将它也累加到value中。

JSONObject jsonObject = **new** JSONObject();  
jsonObject.accumulate(**"1"**, **"一"**);  
jsonObject.accumulate(**"2"**, **"二"**);  
jsonObject.accumulate(**"3"**, **"三"**);  
System.***out***.println(jsonObject); //{"1":"一","2":"二","3":"三"}  
jsonObject.accumulate(**"3"**, **"交集"**);  
System.***out***.println(jsonObject); //{"1":"一","2":"二","3":["三","交集"]}  
jsonObject.accumulate(**"2"**, **null**);  
System.***out***.println(jsonObject); //{"1":"一","2":["二",null],"3":["三","交集"]}

#### 根据key获取值

JSONObject中存储的既然是键值对，自然也提供了根据key获取对应value值的方法。并且提供了不止一种，除了通过get方法获取一个Object类型的值外，我们还可以通过opt系列的方法直接获取指定类型的值。当然，它只适用于我们明确知道值类型的情况。

**public** Object get(Object key)

**public** Object get(String key)

使用opt时，opt方法本质上仍是通过get获取key对应的值，只是当我们调用optBoolean等方法时，它会自动对获取到的value值进行类型转换。另外，opt方法会设置一个默认值，所以，即使key不存在，也会返回一个对应类型的默认值。

**public** Object opt(String key)

**public boolean** optBoolean(String key)

…

**public** JSONArray optJSONArray(String key)

当key存在时，使用opt和get获取对应的value值是一样的。

JSONObject jsonObject = **new** JSONObject();  
jsonObject.element(**"1"**, **"一"**);  
jsonObject.element(**"2"**, **"二"**);  
System.***out***.println(jsonObject.get(**"3"**)); //null  
System.***out***.println(jsonObject.optString(**"3"**)); //返回空

判断JSONObject中存储的键值对是否包含制定的key或是否包含制定的value。

**public boolean** containsKey(Object key)

**public boolean** containsValue(Object value)

将JSONObject转换为Set形式，并返回。输出类型为[key=value]：[2=二, test1=一]

**public** Set entrySet()

keys方法可以获取到JSONObject中所有key的迭代器对象，通过迭代器对象，可以对key进行遍历。keySet方法可以将所有的key以set形式返回。Names方法是将所有的key组成的JSONArray对象返回。

**public** Iterator keys()

**public** Set keySet()

**public** JSONArray names()

values方法和keys方法类似，将所有的value值以集合形式返回。

**public** Collection values()

#### 移除key

移除key可以直接通过JSONObject提供的remove或discard方法，从JSONObject中删除指定的键值对。当然，我们也可以利用element中key对应value为null时移除键值对的特性删除key。

JSONObject jsonObject = **new** JSONObject();  
jsonObject.element(**"1"**, **"一"**);  
jsonObject.element(**"2"**, **"二"**);  
System.***out***.println(jsonObject); // {"1":"一","2":"二"}  
jsonObject.remove(**"2"**);  
System.***out***.println(jsonObject); // {"1":"一"}  
jsonObject.discard(**"1"**);  
System.***out***.println(jsonObject); //{}

目前没看到两个方法之间有什么区别。

Clear方法可以移除JSONObject中的所有键值对，也就是清空JSONObject对象。

**public void** clear()

#### 其他方法

**public** Writer write(Writer writer)

**public int** size()

返回JSONObject中键值对的数量。

### JSONArray

### JsonConfig

需要代码测试：

<http://blog.csdn.net/chao2263263364/article/details/46801561>

<http://blog.csdn.net/u013456370/article/details/53503958>

<https://www.cnblogs.com/handsomeye/p/5107398.html>

#### 防止自包含

#### 自定义格式转换(date)

#### 字段过滤

#### 解决延迟加载产生异常的问题

# 哈希码

文件哈希码，防止文件重复上传:

*/\*\*  
 \* 获取Sha256  
 \*  
 \** ***@param fileBytes*** *\*  
 \** ***@return*** *String  
 \*/***private** String getHash256(**byte**[] fileBytes) {  
  
 String value = **null**;  
 *// 获取hash值* **try** {  
 **byte**[] buffer = **new byte**[1024];  
 **int** numRead;  
  
 InputStream fis = **new** ByteArrayInputStream(fileBytes);  
 *// 如果想使用SHA-1或SHA-256，则传入SHA-1,SHA-256* MessageDigest complete = MessageDigest.*getInstance*(**"SHA-256"**);  
 **do** {  
 *// 从文件读到buffer* numRead = fis.read(buffer);  
 **if** (numRead > 0) {  
 *// 用读到的字节进行MD5的计算，第二个参数是偏移量* complete.update(buffer, 0, numRead);  
 }  
 } **while** (numRead != -1);  
  
 fis.close();  
 value = **new** String(Base64.*encodeBase64*(complete.digest()));  
 } **catch** (NoSuchAlgorithmException e1) {  
 e1.printStackTrace();  
 } **catch** (FileNotFoundException e2) {  
 e2.printStackTrace();  
 } **catch** (IOException e3) {  
 e3.printStackTrace();  
 }  
 **return** value;  
}