Java 的 JSON 开源类库选择比较

有效选择七个关于Java的JSON开源类库

April 4, 2014 By Constantin Marian Alin

翻译：无若

（英语原文：http://www.developer.com/lang/jscript/top-7-open-source-json-binding-providers-available-today.html）

简介

JSON是JavaScript Object Notation的缩写，是一种轻量级的数据交换形式，是一种XML的替代方案，而且比XML更小，更快而且更易于解析。因为JSON描述对象的时候使用的是JavaScript语法，它是语言和平台独立的，并且这些年许多JSON的解析器和类库被开发出来。在这篇文章中，我们将会展示7种Java JSON类库。基本上，我们将会试着把Java对象转换JSON格式并且存储到文件，并且反向操作，读JSON文件转换成一个对象。为了让文章更有意义，我们将会测量每一种JSON类库在不同情况下的处理速度。

（一）类库介绍及其使用

（1）使用Jackson类库

第一个介绍的是Jackson类库，Jackson库是一个“旨在为开发者提供更快，更正确，更轻量级，更符合人性思维” 的类库。Jackson为处理JSON格式提供了三种模型的处理方法。

1、流式API或者增量解析/产生（ incremental parsing/generation）：读写JSON内容被作为离散的事件。

2、树模型：提供一个可变内存树表示JSON文档。

3、数据绑定（Data binding）：实现JSON与POJO（简单的Java对象（Plain Old Java Object））的转换

我们感兴趣的是Java对象与JSON的转换，因此，我们将集中于第三种处理方法。首先我们需要下载Jackson。Jackson的核心功能使用三个类库，分别是jackson-core-2.3.1, jackson-databind-2.3.1和jackson-annotations-2.3.1; 三个类库的下载都来自于Maven仓库，给出地

址：

（译者注：在http://repo1.maven.org/maven2/com/fasterxml/jackson/core/中，正好是三个类库的文件夹）

现在，让我们来工作吧，为了从Java对象中获得一个一个复杂的JSON对象，我们将会使用下面的类去构造一个对象。同样的Java对象将会被用于这篇文章的所有的类库中。

```
public class JsonThirdObject {



          private int age = 81;

          private String name = "Michael Caine";

          private List<String> messages;



          public JsonThirdObject() {

            this.messages = new ArrayList<String>() {

                    {

                              add("You wouldn't hit a man with no
 trousers..");

                              add("At this point, I'd set you up
with a..");

                              add("You know, your bobby dangler,
giggle stick,..");

                              }

                    };
```

```
        }

        // Getter and setter

}


public class JsonSecondObject {


        private int age = 83;

        private String name = "Clint Eastwood";

        private JsonThirdObject jsnTO = new JsonThirdObject();

        private List<String> messages;


        public JsonSecondObject() {

          this.messages = new ArrayList<String>() {

                  {

                          add("This is the AK-47 assault..");

                          add("Are you feeling lucky..");

                          add("When a naked man's chasing a..
");

                          }
```

```
                    };

          }

          // Getter and setter

}



public class JsonFirstObject {



          private int age = 76;

          private String name = "Morgan Freeman";

          private JsonSecondObject jsnSO = new JsonSecondObject()
;

          private List<String> messages;



          public JsonFirstObject() {

            this.messages = new ArrayList<String>() {

                    {

                              add("I once heard a wise man say.."
);

                              add("Well, what is it today? More..
");

                              add("Bruce... I'm God. Circumstance
```

```
s have..");
                                }
                    };
          }
          // Getter and setter
}


public class Json {


          private int age = 52;
          private String name = "Jim Carrey";
          private JsonFirstObject jsnFO = new JsonFirstObject();
          private List<String> messages;


          public Json() {
            this.messages = new ArrayList<String>() {
                    {
                                add("Hey, maybe I will give you..")
;
                                add("Excuse me, I'd like to..");
```

```
                                add("Brain freeze. Alrighty Then I
just..");

                                }

                    };

          }

          // Getter and setter

}
```

上面的Java对象转换成JSON格式是下面这样的。

```
{

   "age":52,

   "name":"Jim Carrey",

   "jsnFO":{

      "age":76,

      "name":"Morgan Freeman",

      "jsnSO":{

         "age":83,

         "name":"Clint Eastwood",

         "jsnTO":{

            "age":81,
```

```
            "name":"Michael Caine",
            "messages":[
               "You wouldn't hit a man..",
               "At this point, I'd set you..",
               "You know, your bobby dangler.."
            ]
         },
         "messages":[
            "This is the AK-47 assault..",
            "Are you feeling lucky..",
            "When a naked man's chasing a.."
         ]
      },
      "messages":[
         "I once heard a wise man..",
         "Well, what is it today? More..",
         "Bruce... I'm God. Circumstances have.."
      ]
   },
```

```
   "messages":[

      "Hey, maybe I will give you a call..",

      "Excuse me, I'd like to ask you a few..",

      "Brain freeze. Alrighty Then I just heard.."

   ]

}
```

现在，让我们来看看怎么样把Java对象转换成JSON并且写入文件。Jackson使用一个ObjectMapper功能，我们第一步要做的是：

```
Json jsonObj = new Json();

ObjectMapper mapper = new ObjectMapper();
```

然后，我们将会使用这个ObjectMapper直接写入值到文件。

```
System.out.println("Convert Java object to JSON format and save t
o file");

try {

     mapper.writeValue(new File("c:\\jackson.json"), jsonObj);

} catch (JsonGenerationException e) {

} catch (JsonMappingException e) {

} catch (IOException e) {

}
```

现在，我们有了一个JSON文件，但是，怎么样转回Java对象呢？我们可以这样做：

```
System.out.println("Read JSON from file, convert JSON back to obj
ect");

try {

     jsonObj = mapper.readValue(new File("c:\\jackson.json"), Jso
n.class);

} catch (JsonGenerationException e) {

} catch (JsonMappingException e) {

} catch (IOException e) {

}
```

从上面的例子我们知道了JSON和Java对象的相互转换，在try-catch中总共也就两行，看起来不错是吧，但是它快么？我们将会在后面的文章中揭晓。

（2）使用 Google-Gson类库

第二种就是 Google-Gson，我们立即开始讨论 Gson，你或许更喜欢他的全名Google-Gson。Gson能实现Java对象和JSON之间的相互转换。甚至都不需要注释。Gson的特点：

1）提供简单的toJson()方法和fromJson()去实现相互转换。

2）可以从JSON中转换出之前存在的不可改变的对象。

3）扩展提供了Java泛型。

4）支持任意复杂的对象。

Gson就需要一个.jar文件，gson-2.2.4.jar，可以通过http://code.google.com/p/google-gson/downloads/list下载。下面是例子，把Java对象转换成JSON。

```
Json jsonObj = new Json();

Gson gson = new Gson();
```

```
System.out.println("Convert Java object to JSON format and save t
o file");

try (FileWriter writer = new FileWriter("c:\\gson.json")) {

     writer.write(gson.toJson(jsonObj));

} catch (IOException e) {

}
```

JSON转换成Java对象：

```
System.out.println("Read JSON from file, convert JSON string back
 to object");

try (BufferedReader reader = new BufferedReader(new FileReader("c
:\\gson.json"))) {

   jsonObj = gson.fromJson(reader, Json.class);

} catch (FileNotFoundException e) {

} catch (IOException e) {

}
```

上面就是我们所有需要做的，接下来我们可以对 jsonObj 作进一步处理。当调用JSON操作的时候，因为Gson的实例化对象没有维持任何状态，我们可以重复使用一个对象为多个JSON序列化和反序列化操作。

（3）使用JSON-lib类库

JSON-lib类库是基于Douglas Crockford的工作成果，能转换bean，map，集合（collection），java数组和XML转换成JSON并能反向转换成beans和动态bean（DynaBean）。JSON-lib类库的下载地址：http://sourceforge.net/projects/json-

lib/files/ 下面这些是依赖文件

(译者注：Douglas Crockford是Web开发领域最知名的技术权威之一，ECMA JavaScript2.0标准化委员会委员。被JavaScript之父Brendan Eich称为JavaScript的大宗师（Yoda）。曾任Yahoo!资深JavaScript架构师，现任PayPal高级JavaScript架构师。他是JSON、JSLint、JSMin和ADSafe的创造者，也是名著《JavaScript: The Good Parts》（中文版《JavaScript语言精粹》）的作者。撰写了许多广为流传、影响深远的技术文章，包括"JavaScript:世界上最被误解的语言"。Douglas Crockford曾在著名的Lucasfilm电影公司任技术总监；在Paramount（派拉蒙）公司任新媒体高级总监；communities社区创始人兼CEO；State软件公司CTO。2012.05.14，Paypal宣布Douglas Crockford加入Paypal。)

同样，让我们来把Java对象转成JSON。

```
Json jsonObj = new Json();

JSONObject json;

System.out.println("Convert Java object to JSON format and save t
o file");

try (FileWriter writer = new FileWriter("c:\\json-lib.json")) {

          json = JSONObject.fromObject(jsonObj);

          json.write(writer);

} catch (IOException e) {

}
```

JSON转Java对象

```
System.out.println("Read JSON from file, convert JSON string back
 to object");

try (BufferedReader reader = new BufferedReader(new FileReader("c
:\\json-lib.json"))) {
```

```
          jsonObj = (Json) JSONObject.toBean(JSONObject.fromObjec
t(reader), Json.class);

} catch (IOException ex) {

}
```

这里有个问题，这些依赖关系会影响到性能吗？文章在下面揭晓。

（4）使用Flexjson类库

Flexjson是一个轻量级的库，能序列化和反序列化Java对象和JSON，允许深层和浅层对象的拷贝。深度拷贝意味着一个被Flexjson序列化的对象，它能让对象做到类似于延迟加载（lazy-loading）的技术，能让我们在对对象有需要时才提取。当我们想把整个对象写入到文件时，这不是一个好的情况，但是它知道需要才去做时，这是很好的。

Flexjson下载地址：http://sourceforge.net/projects/flexjson/files/ 它不需要其他库就可以工作。下面是例子：Java对象转JSON。

```
Json jsonObj = new Json();

JSONSerializer serializer = new JSONSerializer();

System.out.println("Convert Java object to JSON format and save t
o file");

try (FileWriter writer = new FileWriter("c:\\flexjson.json")) {

          serializer.deepSerialize(jsonObj, writer);

} catch (IOException e) {

}
```

JSON转Java对象

```
System.out.println("Read JSON from file, convert JSON string back
 to object");
```

```
try (BufferedReader reader = new BufferedReader(new FileReader("c
:\\flexjson.json"))){

    jsonObj = new JSONDeserializer<Json>().deserialize(reader);

} catch (FileNotFoundException e) {

} catch (IOException e) {

}
```

简单有效是吧！

（5）使用Json-io类库

json-io有两个主要的类，一个读和一个写，排除了使用ObjectInputStream 和 ObjectOutputStream两个类去读写。Json-io能序列化任意的Java对象图（graph）转变成JSON，并且能记忆完整的语义图（graph semantics）和对象类型。下载地址： Maven Central Repository

它不需要其他依赖。

例子：Java对象转JSON

```
Json jsonObj = new Json();

System.out.println("Convert Java object to JSON format and save t
o file");

try (JsonWriter writer = new JsonWriter(new FileOutputStream("c:\
\json-io.json"))){

          writer.write(jsonObj);

} catch (IOException e) {

}
```

JSON转Java对象

```
System.out.println("Read JSON from file, convert JSON string back
 to object");

try (JsonReader reader = new JsonReader(new FileInputStream(new F
ile("c:\\json-io.json")))) {

          jsonObj = (Json) reader.readObject();

} catch (FileNotFoundException e) {

} catch (IOException e) {

}
```

它的文档上说，Json-io比JDK的ObjectInputStream 和ObjectOutputStream的序列化操作要快，我们将会在后面的文章中说明。

（6）使用Genson类库

Genson是一个可扩展的，可伸缩的，易于使用的开源库。除此之外，Genson完整支持了泛型，支持JSON在JAX-RS的实现，支持JAXB的注释（annotation）和类型（types），并且允许序列化和反序列化拥有复杂关键字的map。

下载地址：http://code.google.com/p/genson/downloads/list ，它没有任何依赖。

例子：Java对象转JSON

```
Json jsonObj = new Json();

Genson genson = new Genson();

System.out.println("Convert Java object to JSON format and save t
o file");

try (FileWriter writer = new FileWriter("c:\\genson.json")) {


```

```
          writer.write(genson.serialize(jsonObj));

} catch (IOException | TransformationException e) {

}
```

JSON转Java对象

```
System.out.println("Read JSON from file, convert JSON string back
 to object");

try (BufferedReader reader = new BufferedReader(new FileReader("c
:\\genson.json"))) {

          jsonObj = genson.deserialize(reader, Json.class);

} catch (FileNotFoundException e) {

} catch (IOException | TransformationException e) {

}
```

（7）使用JSONiJ类库

最后一个讨论的是JSONiJ。JSONiJ是JSON的解析器，一个JPath和Marshaller的实现，能实现Java对象和JSON的相互转换。下载地址：https://bitbucket.org/jmarsden/jsonij/downloads

它不需要任何依赖。

例子：Java对象转JSON

```
Json jsonObj = new Json();

System.out.println("Convert Java object to JSON format and save t
o file");

try (FileWriter writer = new FileWriter("c:\\jsonij.json")) {
```

```
          writer.write(JSONMarshaler.marshalObject(jsonObj).toJSO
N());

} catch (IOException | JSONMarshalerException e) {

}
```

JSON转Java 对象

```
System.out.println("Read JSON from file, convert JSON string back
 to object");

try (BufferedReader reader = new BufferedReader(new FileReader("c
:\\jsonij.json"))) {

          JSON json = JSON.parse(reader);



          // Now we need to parse the JSONObject object and put v
alues back

          // to our Json object

          for (Field field : jsonObj.getClass().getDeclaredFields
()) {

                    try {

                              field.setAccessible(true);

                              field.set(field.getName(), json.get
(field.getName()));

                    } catch (IllegalArgumentException | IllegalAc
cessException e) {
```

```
                    }

            }

} catch (FileNotFoundException e) {

} catch (IOException | ParserException e) {

}
```

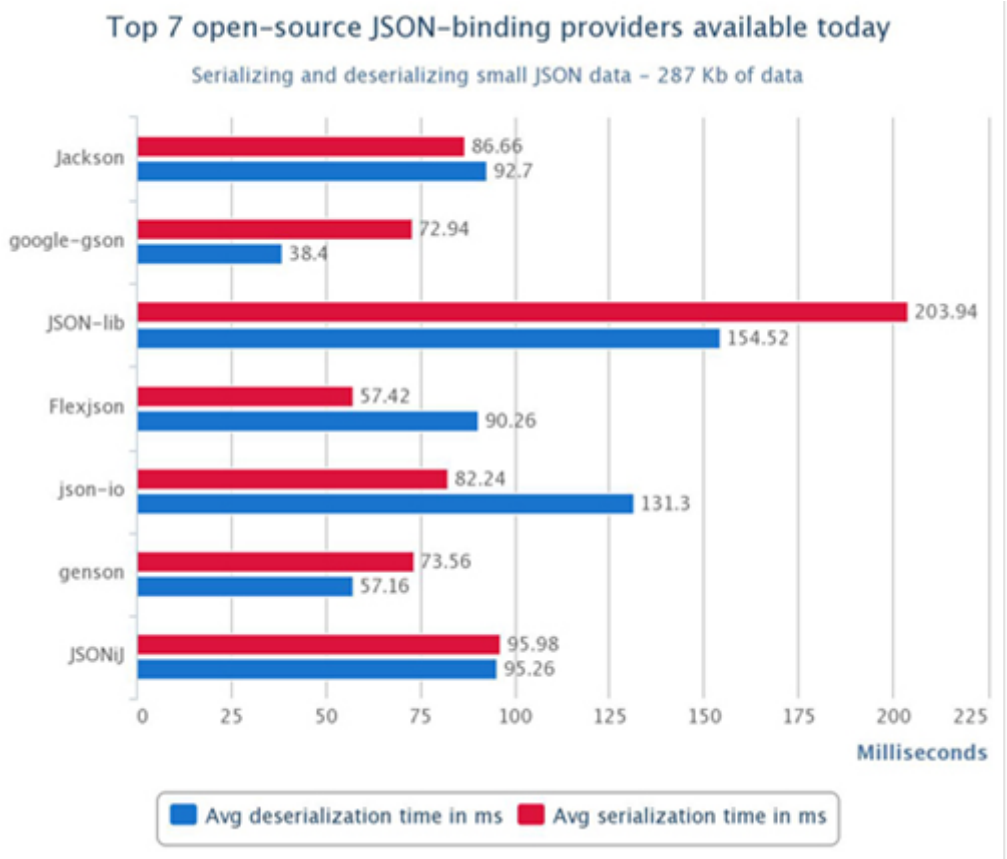
看起来JSONiJ需要的代码多些，性能怎么样，我们看下面。

（二）基准测试

现在我们要来看看性能了，测试硬件配置：Intel Core i5 laptop with 2.50GHz 单通道 DDR3 RAM 4G，软件配置：Windows 7 Ultimate 64-bit SP1

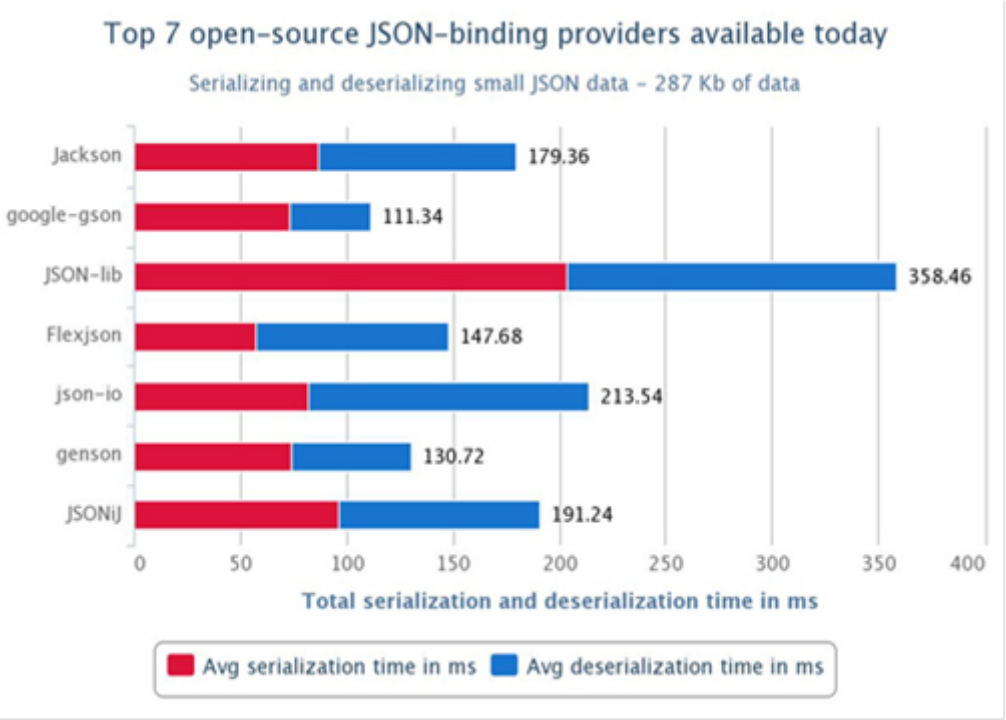
基准测试运行一样的虚拟机（JVM），在测试之前，每一个类库都有一个热身，去限制内存使用的造成的影响，用一个显式调用垃圾收集器。下面的图表代表的是序列化和反序列化JSON数据以毫秒级使用50次迭代和10次热身（warm-up）迭代的平均的时间。

（译者注：红色为序列化（Java对象转JSON），蓝色为反序列化（JSON转Java对象））



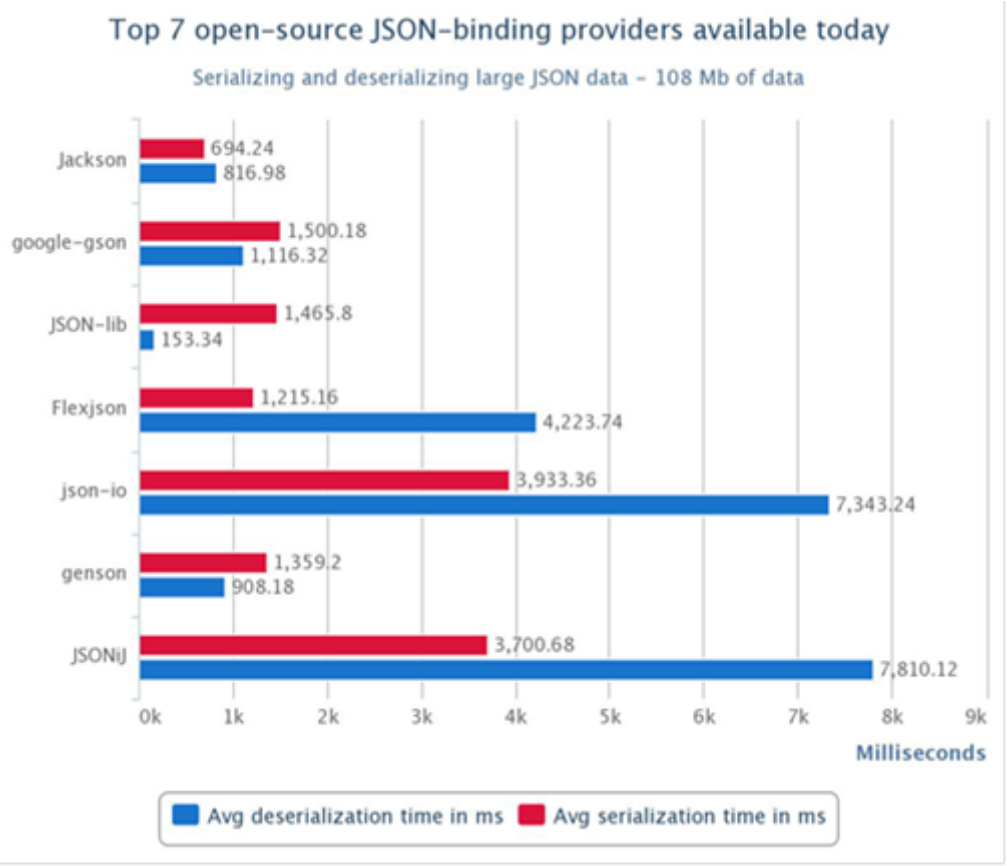
上面的图表显示，Flexjson序列化小数据时是最快的，而JSON-lib是最慢的。反序列化的时候，Gson最快，JSON-lib还是最慢的。

下面的图表代表的是我们的数据在287kb时，序列化和反序列化所花费的平均时间。

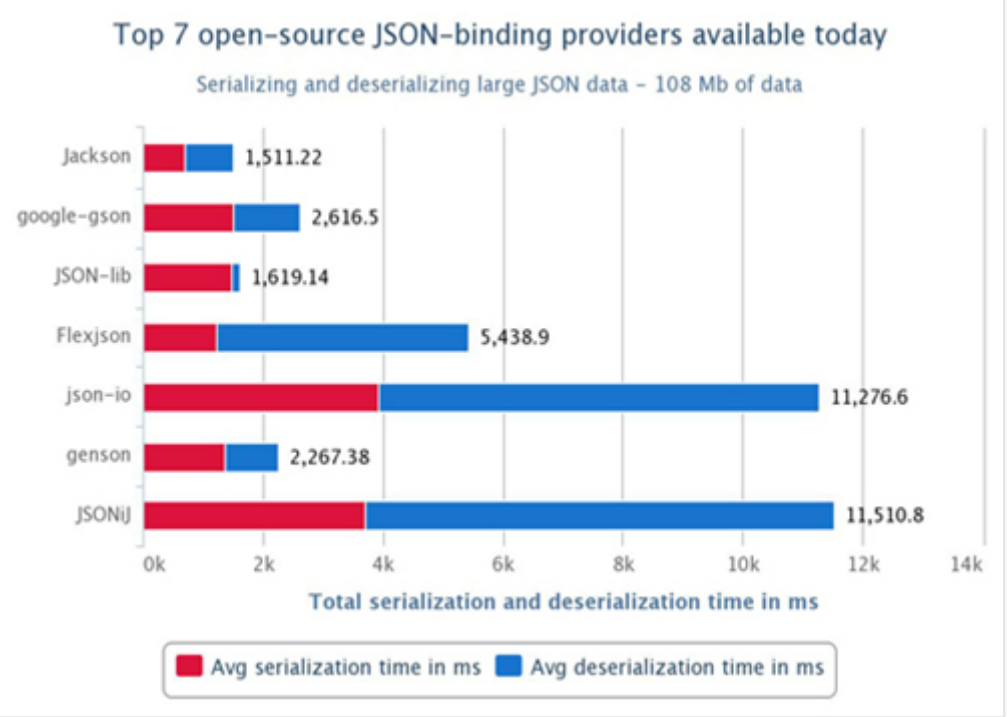


这张图显示，我们对少量的数据操作时，最快的是Gson ,之后的是 Genson和Flexjson。

当变成大数据时，结果变得很不一样。在下面的图表中，使用的是108Mb的数据，在序列化的时候，Jackson变成了最快的，Flexjson变成第二快。在反序列化的时候，JSON-lib变成了最快的，之前在处理小数据时，它是最慢的，第二快的是Jackson。



下面的图表，显示的是处理更大一点的数据时，我们应该使用Jackson和JSON-lib。



另外一个重要的测试是关于.jar包的大小。这对于移动端的开发很重要，我们从下图中看到，json-io最小，之后依次是Flexjson和JSONiJ:

（三）结论

在这篇文章中，我们知道了七种方式来实现Java对象和JSON之间的互相转换。以及哪一个

类库更快，哪一个更慢，在什么情况下使用等。作为结论，如果你在你的应用中是想使用小一点的数据量，你应该使用Flexjson或者Gson，如果你需要大的数据量你应该考虑Jackson 和JSON-lib。