本篇通过JMH来测试一下Java中几种常见的JSON解析库的性能。每次都在网上看到别人说什么某某库性能是如何如何的好，碾压其他的库。但是百闻不如一见，只有自己亲手测试过的才是最值得相信的。

JSON不管是在Web开发还是服务器开发中是相当常见的数据传输格式，一般情况我们对于JSON解析构造的性能并不需要过于关心，除非是在性能要求比较高的系统。

目前对于Java开源的JSON类库有很多种，下面我们取4个常用的JSON库进行性能测试对比， 同时根据测试结果分析如果根据实际应用场景选择最合适的JSON库。

这4个JSON类库分别为：Gson，FastJson，Jackson，Json-lib。

简单介绍

选择一个合适的JSON库要从多个方面进行考虑：

- 字符串解析成JSON性能
- 字符串解析成JavaBean性能
- JavaBean构造JSON性能
- 集合构造JSON性能
- 易用性

先简单介绍下四个类库的身份背景。

Gson

> 项目地址：https://github.com/google/gson

Gson是目前功能最全的Json解析神器，Gson当初是为因应Google公司内部需求而由Google自行研发而来，但自从在2008年五月公开发布第一版后已被许多公司或用户应用。Gson的应用主要为toJson与fromJson两个转换函数，无依赖，不需要例外额外的jar，能够直接跑在JDK上。在使用这种对象转换之前，需先创建好对象的类型以及其成员才能成功的将JSON字符串成功转换成相对应的对象。类里面只要有get和set方法，Gson完全可以实现复杂类型的json到bean或bean到json的转换，是JSON解析的神器。

FastJson

> 项目地址：https://github.com/alibaba/fastjson

Fastjson是一个Java语言编写的高性能的JSON处理器,由阿里巴巴公司开发。无依赖，不需要例外额外的jar，能够直接跑在JDK上。FastJson在复杂类型的Bean转换Json上会出现一些问题，可能会出现引用的类型，导致Json转换出错，需要制定引用。FastJson采用独创的算法，将parse的速度提升到极致，超过所有json库。

Jackson

> 项目地址：https://github.com/FasterXML/jackson

Jackson是当前用的比较广泛的，用来序列化和反序列化json的Java开源框架。Jackson社区相对比较活跃，更新速度也比较快， 从Github中的统计来看，Jackson是最流行的json解析器之一，Spring MVC的默认json解析器便是Jackson。

Jackson优点很多：

- Jackson 所依赖的jar包较少，简单易用。
- 与其他 Java 的 json 的框架 Gson 等相比，Jackson 解析大的 json 文件速度比较快。
- Jackson 运行时占用内存比较低，性能比较好
- Jackson 有灵活的 API，可以很容易进行扩展和定制。

目前最新版本是2.9.4，Jackson 的核心模块由三部分组成：

- jackson-core 核心包，提供基于"流模式"解析的相关 API，它包括 JsonPaser 和 JsonGenerator。Jackson 内部实现正是通过高性能的流模式 API 的 JsonGenerator 和 JsonParser 来生成和解析 json。
- jackson-annotations 注解包，提供标准注解功能；
- jackson-databind 数据绑定包，提供基于"对象绑定" 解析的相关 API（ ObjectMapper ）和"树模型" 解析的相关 API（JsonNode）；基于"对象绑定" 解析的 API 和"树模型"解析的 API 依赖基于"流模式"解析的 API。

Json-lib

> 项目地址：http://json-lib.sourceforge.net/index.html

json-lib最开始的也是应用最广泛的json解析工具，json-lib 不好的地方确实是依赖于很多第三方包，对于复杂类型的转换，json-lib对于json转换成bean还有缺陷， 比如一个类里面会出现另一个类的list或者map集合，json-lib从json到bean的转换就会出现问题。json-lib在功能和性能上面都不能满足现在互联网化的需求。

编写性能测试

接下来开始编写这四个库的性能测试代码。

添加maven依赖

当然首先是添加四个库的maven依赖，公平起见，我全部使用它们最新的版本：

```
<!-- Json libs-->
<dependency>
    <groupId>net.sf.json-lib</groupId>
    <artifactId>json-lib</artifactId>
    <version>2.4</version>
    <classifier>jdk15</classifier>
</dependency>
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.2</version>
</dependency>
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.46</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.4</version>
</dependency>
<dependency>
```

```
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.9.4</version>
</dependency>
```

四个库的工具类

FastJsonUtil.java

```
public class FastJsonUtil {
    public static String bean2Json(Object obj) {
        return JSON.toJSONString(obj);
    }

    public static <T> T json2Bean(String jsonStr, Class<T> objClass) {
        return JSON.parseObject(jsonStr, objClass);
    }
}
```

GsonUtil.java

```
public class GsonUtil {
    private static Gson gson = new GsonBuilder().create();

    public static String bean2Json(Object obj) {
        return gson.toJson(obj);
    }

    public static <T> T json2Bean(String jsonStr, Class<T> objClass) {
        return gson.fromJson(jsonStr, objClass);
    }

    public static String jsonFormatter(String uglyJsonStr) {
        Gson gson = new GsonBuilder().setPrettyPrinting().create();
        JsonParser jp = new JsonParser();
        JsonElement je = jp.parse(uglyJsonStr);
        return gson.toJson(je);
    }
}
```

JacksonUtil.java

```
public class JacksonUtil {
    private static ObjectMapper mapper = new ObjectMapper();

    public static String bean2Json(Object obj) {
        try {
            return mapper.writeValueAsString(obj);
        } catch (JsonProcessingException e) {
            e.printStackTrace();
            return null;
        }
    }

    public static <T> T json2Bean(String jsonStr, Class<T> objClass) {
        try {
            return mapper.readValue(jsonStr, objClass);
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

JsonLibUtil.java

```
public class JsonLibUtil {

    public static String bean2Json(Object obj) {
        JSONObject jsonObject = JSONObject.fromObject(obj);
        return jsonObject.toString();
    }

    @SuppressWarnings("unchecked")
    public static <T> T json2Bean(String jsonStr, Class<T> objClass) {
        return (T) JSONObject.toBean(JSONObject.fromObject(jsonStr), objClass);
    }
}
```

准备Model类

这里我写一个简单的Person类，同时属性有Date、List、Map和自定义的类FullName，最大程度模拟真实场景。

```
public class Person {
    private String name;
    private FullName fullName;
    private int age;
    private Date birthday;
    private List<String> hobbies;
    private Map<String, String> clothes;
    private List<Person> friends;
    // getter/setter省略
    @Override
    public String toString() {
        StringBuilder str = new StringBuilder("Person [name=" + name + ", fullName=" + fullName + ", a
                + age + ", birthday=" + birthday + ", hobbies=" + hobbies
                + ", clothes=" + clothes + "]");
        if (friends != null) {
            str.append("Friends:");
            for (Person f : friends) {
                str.append("  ").append(f);
            }
        }
        return str.toString();
    }

}
```

```
public class FullName {
    private String firstName;
    private String middleName;
    private String lastName;

    public FullName() {
    }
```

```
    public FullName(String firstName, String middleName, String lastName) {
        this.firstName = firstName;
        this.middleName = middleName;
        this.lastName = lastName;
    }
    // 省略getter和setter
    @Override
    public String toString() {
        return "[firstName=" + firstName + ", middleName="
                + middleName + ", lastName=" + lastName + "]";
    }
}
```

JSON序列化性能基准测试

```
@BenchmarkMode(Mode.SingleShotTime)
@OutputTimeUnit(TimeUnit.SECONDS)
@State(Scope.Benchmark)
public class JsonSerializeBenchmark {
    /**
     * 序列化次数参数
     */
    @Param({"1000", "10000", "100000"})
    private int count;

    private Person p;

    public static void main(String[] args) throws Exception {
        Options opt = new OptionsBuilder()
                .include(JsonSerializeBenchmark.class.getSimpleName())
                .forks(1)
                .warmupIterations(0)
                .build();
        Collection<RunResult> results =  new Runner(opt).run();
        ResultExporter.exportResult("JSON序列化性能", results, "count", "秒");
    }

    @Benchmark
    public void JsonLib() {
```

```
        for (int i = 0; i < count; i++) {
            JsonLibUtil.bean2Json(p);
        }
    }

    @Benchmark
    public void Gson() {
        for (int i = 0; i < count; i++) {
            GsonUtil.bean2Json(p);
        }
    }

    @Benchmark
    public void FastJson() {
        for (int i = 0; i < count; i++) {
            FastJsonUtil.bean2Json(p);
        }
    }

    @Benchmark
    public void Jackson() {
        for (int i = 0; i < count; i++) {
            JacksonUtil.bean2Json(p);
        }
    }

    @Setup
    public void prepare() {
        List<Person> friends=new ArrayList<Person>();
        friends.add(createAPerson("小明",null));
        friends.add(createAPerson("Tony",null));
        friends.add(createAPerson("陈小二",null));
        p=createAPerson("邵同学",friends);
    }

    @TearDown
    public void shutdown() {
    }

    private Person createAPerson(String name,List<Person> friends) {
```

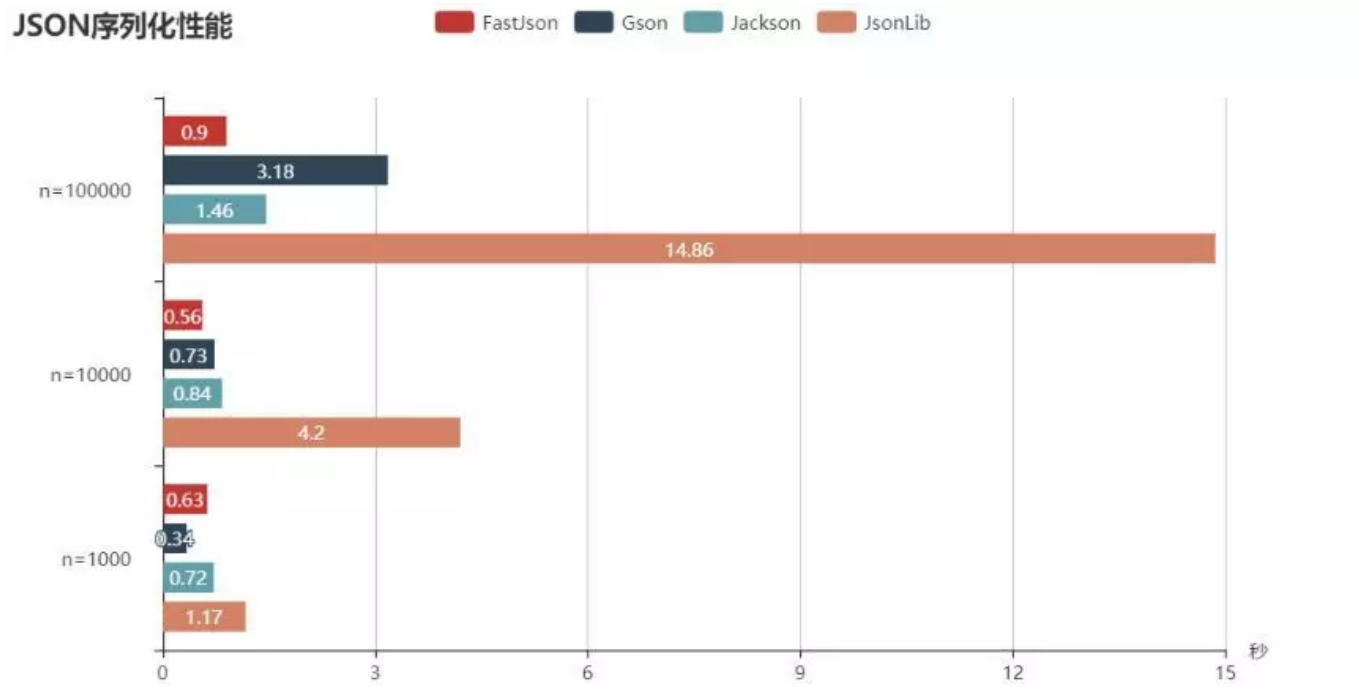
```
        Person newPerson=new Person();
        newPerson.setName(name);
        newPerson.setFullName(new FullName("zjj_first", "zjj_middle", "zjj_last"));
        newPerson.setAge(24);
        List<String> hobbies=new ArrayList<String>();
        hobbies.add("篮球");
        hobbies.add("游泳");
        hobbies.add("coding");
        newPerson.setHobbies(hobbies);
        Map<String,String> clothes=new HashMap<String, String>();
        clothes.put("coat", "Nike");
        clothes.put("trousers", "adidas");
        clothes.put("shoes", "安踏");
        newPerson.setClothes(clothes);
        newPerson.setFriends(friends);
        return newPerson;
    }
}
```

说明一下，上面的代码中

```
ResultExporter.exportResult("JSON序列化性能", results, "count", "秒");
```

这个是我自己编写的将性能测试报告数据填充至Echarts图，然后导出png图片的方法。

执行后的结果图：



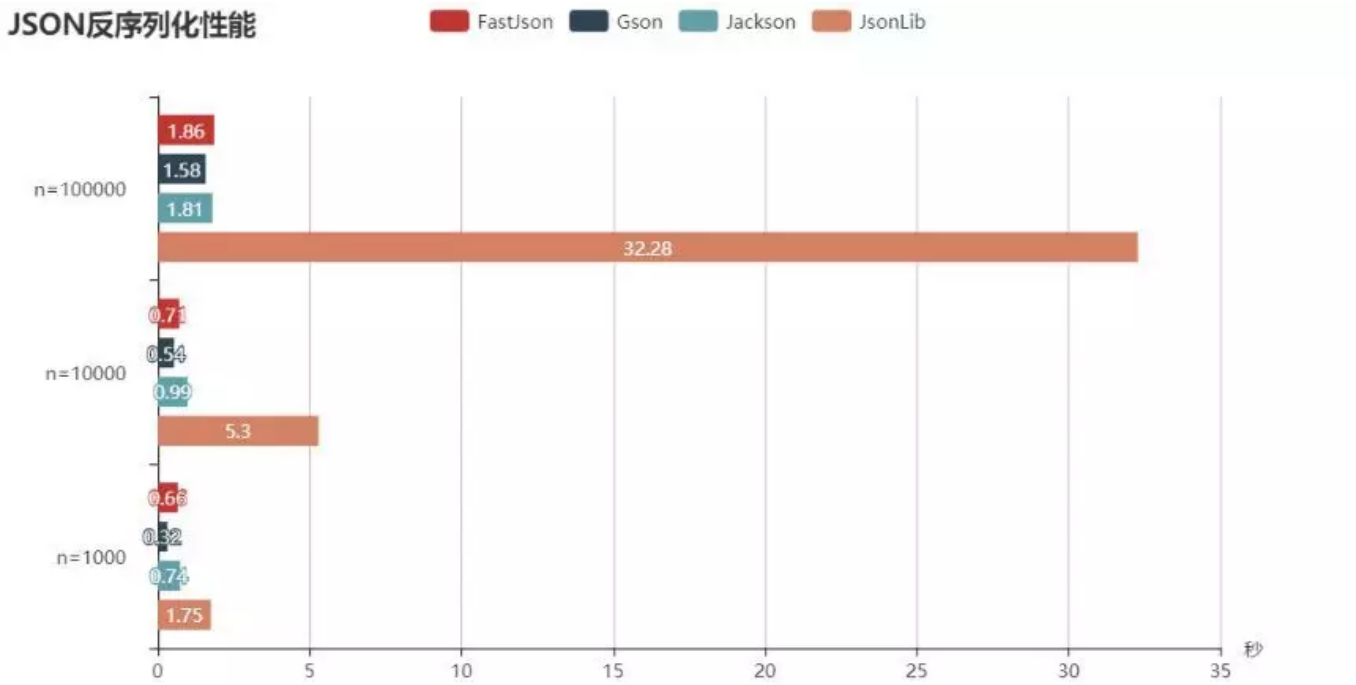
从上面的测试结果可以看出，序列化次数比较小的时候，Gson性能最好，当不断增加的时候到了100000，Gson明细弱于Jackson和FastJson， 这时候FastJson性能是真的牛，另外还可以看到不管数量少还是多，Jackson一直表现优异。而那个Json-lib简直就是来搞笑的。^_^

JSON反序列化性能基准测试

```
@BenchmarkMode(Mode.SingleShotTime)
@OutputTimeUnit(TimeUnit.SECONDS)
@State(Scope.Benchmark)
public class JsonDeserializeBenchmark {
    /**
     * 反序列化次数参数
     */
    @Param({"1000", "10000", "100000"})
    private int count;

    private String jsonStr;

    public static void main(String[] args) throws Exception {
        Options opt = new OptionsBuilder()
                .include(JsonDeserializeBenchmark.class.getSimpleName())
                .forks(1)
                .warmupIterations(0)
                .build();
        Collection<RunResult> results =  new Runner(opt).run();
        ResultExporter.exportResult("JSON反序列化性能", results, "count", "秒");
```

```
    }

    @Benchmark
    public void JsonLib() {
        for (int i = 0; i < count; i++) {
            JsonLibUtil.json2Bean(jsonStr, Person.class);
        }
    }

    @Benchmark
    public void Gson() {
        for (int i = 0; i < count; i++) {
            GsonUtil.json2Bean(jsonStr, Person.class);
        }
    }

    @Benchmark
    public void FastJson() {
        for (int i = 0; i < count; i++) {
            FastJsonUtil.json2Bean(jsonStr, Person.class);
        }
    }

    @Benchmark
    public void Jackson() {
        for (int i = 0; i < count; i++) {
            JacksonUtil.json2Bean(jsonStr, Person.class);
        }
    }

    @Setup
    public void prepare() {
        jsonStr="{"name":"邵同学","fullName":{"firstName":"zjj_first","middleName":"zjj_middle","lastNa
    }

    @TearDown
    public void shutdown() {
    }
}
```

执行后的结果图：



从上面的测试结果可以看出，反序列化的时候，Gson、Jackson和FastJson区别不大，性能都很优异，而那个Json-lib还是来继续搞笑的。