Fastjson 源码分析

1 Fastjson 简介

1.1 JSON 介绍

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，易于人阅读和编写，同时也易于机器解析和生成[1]。在 JS 语言中，一切都是对象。因此，任何支持的类型都可以通过 JSON 来表示，例如字符串、数字、对象、数组等

JSON 可以将 JavaScript 对象中表示的一组数据转换为字符串，以键值对({"name": "json"})的方式来保存和表示数据，即可在网络或者程序之间方便地传递这个字符串，并在需要的时候将它还原为各编程语言所支持的数据格式。

```
'use strict';

var xiaoming = {
    name: '小明',
    age: 14,
    gender: true,
    height: 1.65,
    grade: null,
    'middle-school': '\"W3C\" Middle School',
    skills: ['JavaScript', 'Java', 'Python', 'Lisp']
};
```

JavaScript 对象

```
{
  "name": "小明",
  "age": 14,
  "gender": true,
  "height": 1.65,
  "grade": null,
  "middle-school": "\"W3C\" Middle School",
  "skills": [
    "JavaScript",
    "Java",
    "Python",
    "Lisp"
  ]
}
```

JSON 表达

[1] 引用自 http://www.json.org/json-zh.html

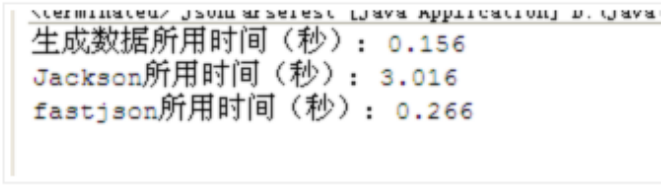
1.2 fastjson

在日志解析,前后端数据传输交互中,经常会遇到字符串(String)与 json,XML 等格式相互转换与解析，其中 json 以跨语言，跨前后端的优点在开发中被频繁使用，是一种标准的数据交换格式[2]。fastjson 是阿里巴巴的开源 JSON 解析库，它可以解析 JSON 格式的字符串，支持将 Java Bean 序列化为 JSON 字符串，也可以从 JSON 字符串反序列化到 JavaBean.

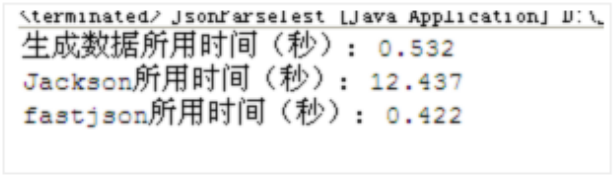
fastjson 是一个 java 语言编写的高性能且功能完善的 JSON 库，运行速度快，不依赖于其他类库，功能强大，已经被广泛使用在缓存序列化，协议交互，Web 输出等各种应用场景中。它采用一种“假定有序快速匹配”的算法,把 JSON Parse 的性能提升到了极致。

下图为开源中国社区一博主对 Jackson 和 fastjson 进行的性能对比，在处理 1000 条、5000 条和 10000 条数据的量级上,fastjson 的解析速度均远远大于 Jackson。

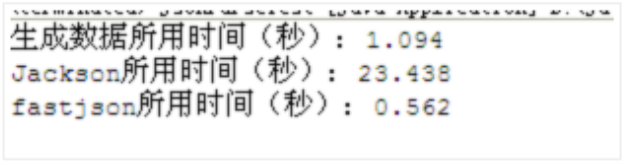
4. [图片] **1000.png**



5. [图片] **5000.png**



6. [图片] **10000.png**



fastjson 和 Jackson 对比[3]

[2] 引用自 https://www.jianshu.com/p/b9794f3d9862

[3] 引用自开源中国社区 https://www.oschina.net/code/snippet_1156226_26432

1.3 fastjson 优点[4]

（1）速度快。fastjson 相对其他 JSON 库的特点是快，从 2011 年 fastjson 发布 1.1.x 版本之后，其性能从未被其他 Java 实现的 JSON 库超越。

（2)使用广泛。fastjson 在阿里巴巴大规模使用，在数万台服务器上部署，fastjson 在业界被广泛接受。在 2012 年被开源中国评选为最受欢迎的国产开源软件之一。

（3）测试完备。fastjson 有非常多的 testcase，在 1.2.11 版本中，testcase 超过 3321 个。每次发布都会进行回归测试，保证质量稳定。

（4）使用简单。fastjson 的 API 十分简洁。

```
String text = JSON.toJSONString(obj); //序列化
VO vo = JSON.parseObject("{...}", VO.class); //反序列化
```

（5）功能完备。支持泛型，支持流处理超大文本，支持枚举，支持序列化和反序列化扩展。

1.4 fastjson 功能分析

（1）在服务段和安卓用户端提供最佳体验

（2）提供简单的 toJSONString() 和 parseObject()进行 Java 对象和 JSON 表达之间的转换。

（3）允许将预先存在的不可修改对象转换为 JSON，以及从 JSON 转换出。

（4）为 Java 范性提供额外支持。

（5）对对象进行习惯性表达

（6）支持任意复杂的对象（具有深层继承层次结构和泛型类型的广泛使用）

1.5 简单示例

定义下列 Person 类：

```
public class Person {
    private int age;
    private String Name;
    public Person(int age, String fullName) {
        super();
```

[4] 引用自 fastjson 的 github 页面 https://github.com/alibaba/fastjson/wiki/Quick-Start-CN

```
        this.age = age;
        this.Name= Name;
    }
}
```

使用 fastjson 将 java 对象转换成 JSON 对象：

```
private List<Person> listOfPersons = new ArrayList<Person>();
public void SetUp() {
    listOfPersons.add(new Person(17, "Jack"));
}
public void ConvertToJson () {
    String json= JSON.toJSONString(listOfPersons);
}
```

输出为：

```
[
    {
    "AGE":17,
    "NAME":"Jack"
    }
    ]
```

2 序列化

fastjson 的核心功能包括序列化和反序列化，本文中仅涉及序列化。fastjson 序列化的主要使用入口在 JSON.java 中，它提供了简便的 api 将 java 对象转换成 json 字符串。

2.1 toJSONstring/Serializer

```java
public static String toJSONString(Object object) {
    return toJSONString(object, emptyFilters);
}

public static String toJSONString(Object object, SerializerFeature... features) {
    return toJSONString(object, DEFAULT_GENERATE_FEATURE, features);
}
```

toJSONstring 是一个便捷接口，可以将 java 对象转为 json 字符串，内部调用 `toJSONString(Object object, SerializerFeature... features)`

继续跟踪该方法：

```java
public static String toJSONString(Object object, //
                                  SerializeConfig config, //
                                  SerializeFilter[] filters, //
                                  String dateFormat, //
                                  int defaultFeatures, //
                                  SerializerFeature... features) {
    SerializeWriter out = new SerializeWriter(null, defaultFeatures, features);

    try {
        JSONSerializer serializer = new JSONSerializer(out, config);

        if (dateFormat != null && dateFormat.length() != 0) {
            serializer.setDateFormat(dateFormat);
            serializer.config(SerializerFeature.WriteDateUseDateFormat, true);
        }

        if (filters != null) {
            for (SerializeFilter filter : filters) {
                serializer.addFilter(filter);
            }
        }

        serializer.write(object);

        return out.toString();
    } finally {
        out.close();
    }
}
```

该序列化方法首先做了全局序列化配置，并追加了序列化拦截器，并未真正执

行序列化操作，查找具体序列化实例委托给了 config 对象。

接着进入真正完成序列化对象操作的 ObjectSerializer 实例：

```
    ObjectSerializer serializer = config.getObjectWriter(clazz);
    if (serializer instanceof JavaBeanSerializer) {
        JavaBeanSerializer javaBeanSerializer = (JavaBeanSerializer) serializer;

        JSONObject json = new JSONObject();
        try {
            Map<String, Object> values = javaBeanSerializer.getFieldValuesMap(javaObject);
            for (Map.Entry<String, Object> entry : values.entrySet()) {
                json.put(entry.getKey(), toJSON(entry.getValue()));
            }
        } catch (Exception e) {
            throw new JSONException("toJSON error", e);
        }
        return json;
    }

    String text = JSON.toJSONString(javaObject);
    return JSON.parse(text);
}
```

需要注意的是代码 config.getObjectWriter(clazz)，方法内部调用 getObjectWriter(clazz)查找序列化实例。接下来，我们可以看到具体每一类数据的序列化：

```
if (Map.class.isAssignableFrom(clazz)) {
    put(clazz, writer = MapSerializer.instance);
    put(clazz, writer = ListSerializer.instance);
    ...//相似内容，不做赘述
```

当 class 实现类 map 接口时，用 MapSerializer 实现序列化；当 class 实现类 List 接口时，用 ListSerializer 实现序列化……等等，基本思想为根据 class 的类型或接口类型实行查找。以 MapSerializer 为例，我们进入 MapSerializer 进行查看，重点关注以下几行代码：

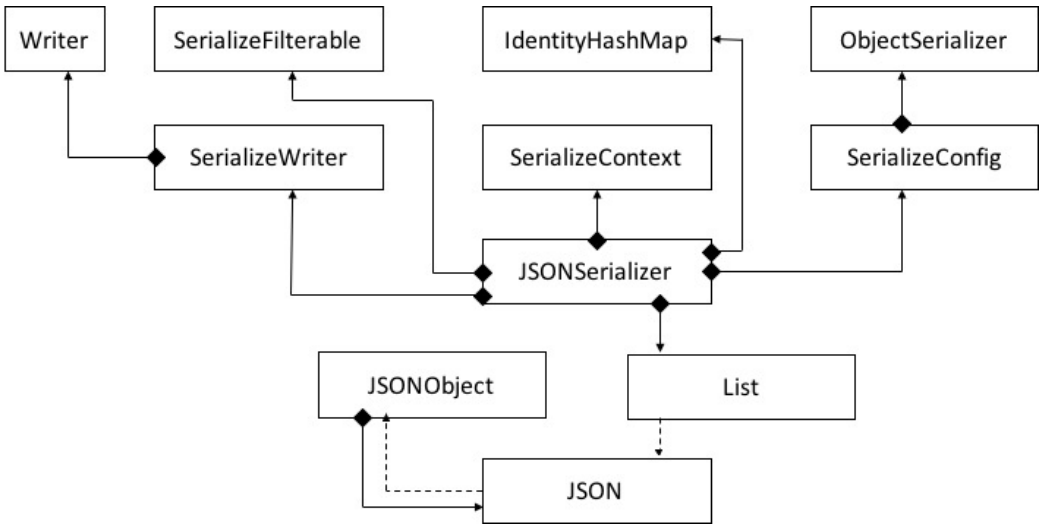
```
for (Map.Entry entry : map.entrySet()) {
Object value = entry.getValue();
Object entryKey = entry.getKey();
...
```

```
List<PropertyPreFilter> preFilters = serializer.propertyPreFilters;
if(preFilters !=null && PreFilters.size() > 0 ){
    if (entryKey == null || entryKey instanceof String) {
        if (!this.applyName(serializer, object, (String) entryKey)){
            continue;
            }
    } else if (entryKey.getClass().isPrimitive() || entryKey
instanceof Number){
        String strKey = JSON.toJSONString(entryKey);
        if (!this.applyName(serializer, object, strKey)){
            continue;
        }
    }
}
…//无关部分，先忽略
```

可以看到，MapSerializer 遍历 PropertyPreFilter 拦截器，针对 map 的 key 和 value 执行拦截操作。在 map 的序列化中，还处理了对象引用，使用 IdentityHashMap 类判断对象严格相等。

2.2 UML 图

下面是序列化的类和类图：



图中可以看出上文所述的类之间的关系。fastjson 序列化的 api 入口在 JSON，它提供了许多静态接口，通过其中 toJSONstring（）静态方法，可以将 java 对象转换成 json 字符串。而 toJSONstring（）的实现，需要将序列化过程中产生的数据临时填充在 SerialzeWriter 中，查找具体序列化实例委托给了 SerializeConfig 对象，通过一个个 ObjectSerializer 实例真正完成了序列化对象操作。

3 结语

作为物理系的本科生，在本科最后一个选课学期选了面向对象程序设计这门课，初衷是希望丰富自己的知识，却没想到迎来了从 0 开始学 java、阅读源码这么大的挑战。非常感谢唐助教和王老师一学期的辛勤工作。我的报告一定充满了生疏的漏洞，还望谅解，并请老师不吝赐教。