# fastjson 分析

## 一、json 及 fastjson 背景介绍

fastjson 是一个高性能功能完善的 JSON 库,可用于将 Java 对象转换为 JSON 序列，也可以将 JSON 反序列化为 Java 对象。fastjson 是目前解析 JSON 最快的 Java 库。fastjson 是 github 上的开源项目，它的接口简单易用,已经被广泛使用在缓存序列化、协议交互、Web 输出、Android 客户端等多种应用场景。本文将对 fastjson 的序列化和反序列化功能做分析。

既然 fastjson 是解析 JSON，首先认识一下 JSON。JSON 是 JavaScript Object Notation 的缩写，意为 JavaScript 对象表示法，它是一种数据交换格式。在 JSON 出现之前，大家一直用 XML 来传递数据。因为 XML 是一种纯文本格式，所以它适合在网络上交换数据。但 XML 的规则过于复杂，使人很不愉悦。后来在 2002 年的一天，道格拉斯·克罗克福特（Douglas Crockford）为了拯救深陷水深火热同时又被某几个巨型软件企业长期愚弄的软件工程师，发明了 JSON 这种超轻量级的数据交换格式。道格拉斯长期担任雅虎的高级架构师，自然钟情于 JavaScript。他设计的 JSON 实际上是 JavaScript 的一个子集。JSON 的数据类型与 JavaScript 的相差无几。JSON 本质上是一个字符串，要把任何 JavaScript 对象变成 JSON，就是把这个对象序列化成一个 JSON 格式的字符串，这样才能够通过网络传递给其他计算机。如果我们收到一个 JSON 格式的字符串，只需要把它反序列化成一个 JavaScript 对象，就可以在 JavaScript 中直接使用这个对象了。fastjson 即可高效的完成这些工作。

## 二、fastjson 代码分析

### （一）概述

Fastjson 的基本功能是序列化和反序列化。序列化指的是将 Java 对象转换为 JSON 字符串。反序列就是相反的过程，即将 JSON 字符串转换为 Java 对象。需要注意的是 JSON 是一种数据交换格式，因此 JSON 一般不会用来传递方法，所谓将 Java 对象转换为 JSON 字符串，是将对象的属性以 JSON 格式保存。而将 JSON 字符串转换为 Java 对象，一般指将其转换为一种称为 JavaBean 的特殊的类。JavaBean 类的定义是，提供一个默认的无参构造函数，需要被序列化并且实现了 Serializable 接口，可能有一系列 private 的可读写属性，可能有一系列的"getter"或"setter"方法，也就是一种十分简单的类。

使用 fastjson 的 API 包含在 com.alibaba.fastjson.JSON 类中，该类提供了有关序列化和反序列化的静态方法，可以直接调用，主要包括以下几个：

1. public static JSONObject parseObject(String text)

//将 JSON 文本转换为 JSONObject 对象

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

//将 JSON 文本转换为指定的类的示例

3. public static JSONArray parseArray(String text)

//将 JSON 文本转换为 JSONArray 对象

4. public static <T> List<T> parseArray(String text, Class<T> clazz)

//将 JSON 文本转换为指定类的对象列表

5. public static String toJSONString(Object object)

//将 Java 对象转换为 JSON 文本

使用 fastjson 时还需要注意，fastjson 并不支持所有类的序列化和反序列化，实际上 fastjson 支持的可序列化和可反序列的类如下：

| 注册的类型 | 序列化实例 | 是否支持序列化 | 是否支持反序列化 |
| --- | --- | --- | --- |
| Boolean | BooleanCodec | 是 | 是 |
| Character | CharacterCodec | 是 | 是 |
| Byte | IntegerCodec | 是 | 是 |
| Short | IntegerCodec | 是 | 是 |
| Integer | IntegerCodec | 是 | 是 |
| Long | LongCodec | 是 | 是 |
| Float | FloatCodec | 是 | 是 |
| Double | DoubleSerializer | 是 | - |
| BigDecimal | BigDecimalCodec | 是 | 是 |
| BigInteger | BigIntegerCodec | 是 | 是 |
| String | StringCodec | 是 | 是 |
| byte[] | PrimitiveArraySerializer | 是 | - |
| short[] | PrimitiveArraySerializer | 是 | - |
| int[] | PrimitiveArraySerializer | 是 | - |
| long[] | PrimitiveArraySerializer | 是 | - |
| float[] | PrimitiveArraySerializer | 是 | - |
| double[] | PrimitiveArraySerializer | 是 | - |
| boolean[] | PrimitiveArraySerializer | 是 | - |
| char[] | PrimitiveArraySerializer | 是 | - |
| Object[] | ObjectArrayCodec | 是 | 是 |
| Class | MiscCodec | 是 | 是 |
| SimpleDateFormat | MiscCodec | 是 | 是 |
| Currency | MiscCodec | 是 | 是 |
| TimeZone | MiscCodec | 是 | 是 |
| InetAddress | MiscCodec | 是 | 是 |
| Inet4Address | MiscCodec | 是 | 是 |
| Inet6Address | MiscCodec | 是 | 是 |
| InetSocketAddress | MiscCodec | 是 | 是 |
| File | MiscCodec | 是 | 是 |
| Appendable | AppendableSerializer | 是 | - |
| StringBuffer | AppendableSerializer | 是 | - |
| StringBuilder | AppendableSerializer | 是 | - |
| Charset | ToStringSerializer | 是 | - |
| Pattern | ToStringSerializer | 是 | - |
| Locale | ToStringSerializer | 是 | - |

| | | | |
|---|---|---|---|
| URI | ToStringSerializer | 是 | - |
| URL | ToStringSerializer | 是 | - |
| UUID | ToStringSerializer | 是 | - |
| AtomicBoolean | AtomicCodec | 是 | 是 |
| AtomicInteger | AtomicCodec | 是 | 是 |
| AtomicLong | AtomicCodec | 是 | 是 |
| AtomicReference | ReferenceCodec | 是 | 是 |
| AtomicIntegerArray | AtomicCodec | 是 | 是 |
| AtomicLongArray | AtomicCodec | 是 | 是 |
| WeakReference | ReferenceCodec | 是 | 是 |
| SoftReference | ReferenceCodec | 是 | 是 |
| LinkedList | CollectionCodec | 是 | 是 |

注：转载自 reference_1。

在实现具体的序列化和反序列化的时候会使用第二列的序列化示例，详细过程见下文。

下面给出使用 fastjson 的示例，不失一般性，示例定义了几个被称为 JavaBean 的特殊的类。

```
public class TestFastJson {
    static class Person{
        private String name;
        private int age;

        public Person(){
        }
        public Person(String name,int age){
            this.name=name;
            this.age=age;
        }
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public int getAge() {
            return age;
        }
        public void setAge(int age) {
            this.age = age;
        }
    }
    public static void main(String[] args) {
        method1();
        method2();
    }

```

```
30      static void method1(){
31          System.out.println("javabean转化示例开始----------");
32          Person person = new Person("Jack",18);
33
34          //这里将javabean转化成json字符串
35          String jsonString = JSON.toJSONString(person);
36          System.out.println(jsonString);
37          //这里将json字符串转化成javabean对象，
38          person =JSON.parseObject(jsonString,Person.class);
39          System.out.println(person.toString());
40
41          System.out.println("javabean转化示例结束----------");
42      }
43
44      static void method2(){
45          System.out.println("List<javabean>转化示例开始----------");
46
47          Person person1 = new Person("Mike",19);
48          Person person2 = new Person("James",20);
49          List<Person> persons = new ArrayList<Person>();
50          persons.add(person1);
51          persons.add(person2);
52          //将对象列表转换为JSON字符串
53          String jsonString = JSON.toJSONString(persons);
54          System.out.println("json字符串:"+jsonString);
55
56          //json字符串转换为Person类的对象列表
57          List<Person> persons_t = JSON.parseArray(jsonString,Person.class);
58          //输出解析后的person对象
59          System.out.println("person1对象: "+persons_t.get(0).toString());
60          System.out.println("person2对象: "+persons_t.get(1).toString());
61
62          System.out.println("List<javabean>转化示例结束----------");
63      }
64  }
65
```

接下对 fastjson 的序列化和反序列化功能做进一步的分析。

（二）fastjson 的序列化过程

将 Java 对象转换为 JSON 文本需要使用的类主要有 JSON 类，JSONSerializer 类，SerializeWriter 类。

使用 fastjson 的 API 包含在 JSON 类中，这是一个抽象类。在《Java 语言程序设计》一书中说，“在继承的层次结构中，随着每个新子类的出现，类会变得越来越明确和具体。如果从一个子类型追溯到父类，类就会变得更通用、更加不明确。类的设计应该确保父类包含它的子类的共同特征。有时候，一个父类设计得非常抽象，以至于它都没有任何具体的实例。这样的类称为抽象类。”也就是说使用 JSON 类时不需要为其构造具体的实例。而且该类提供了有关序列化和反序列化的静态方法，可以直接调用。

JSON 类开始定义了几个成员变量，之后是几个由 static 修饰的代码块，说明 JSON 类会首先执行这几段代码。代码主要是为成员变量 feature 赋值，feature 是与 fastjson 进行序列化和反序列化功能的配置相关的变量。这些配置信息包括了一些输出格式等等，这里是设置了一些默认的配置。

之前提到将 Java 对象转换为 JSON 文本调用的方法为 public static String toJSONString(Object object)，这里重点关注该方法。其实在 JSON 类中还有许多关于 toJSONString 的重载方法，其余的这些方法增添了关于各种配置信息的的参数，这些配置信息对分析 fastjson 序列化功能关系不大。跟踪 toJSONString(Object object)方法最终会调用重载方法 public static String toJSONString(Object object, int defaultFeatures, SerializerFeature... features)，该方法源码如下图所示：

```
617         public static String toJSONString(Object object, int defaultFeatures, SerializerFeature... features) {
618             SerializeWriter out = new SerializeWriter((Writer) null, defaultFeatures, features);
619 
620             try {
621                 JSONSerializer serializer = new JSONSerializer(out);
622                 serializer.write(object);
623                 return out.toString();
624             } finally {
625                 out.close();
626             }
627         }
```

可以看到重载方法多出来的参数是一些配置信息，对我们分析方法功能关系不大。这个方法里构造了一个 SerializeWriter 实例，接下来我们先看看 SerializeWriter 这个类的作用是什么。

SerializeWriter 类定义在 com.alibaba.fastjson.serializer 中，这个类继承了 Writer 类。Writer 是 Java 自带的一个类，是处理与输出流相关的一个类。SerializeWriter 类的主体部分定义了一个字符数组 buf[]，这个数组用来存放转换成功的 JSON 字符串。SerializeWriter 类也定义了一些关于配置信息的成员变量，这些变量和 buf[]数组都是 protected 的，因此只能被该类的子类所访问。阅读源码后发现这个类实现了一些将特定类型变量写入输出流，也就是 buf[]中的方法。以 public void writeInt(int i)方法示意如下：

```
516         public void writeInt(int i) {
517             if (i == Integer.MIN_VALUE) {
518                 write("-2147483648");
519                 return;
520             }
521 
522             int size = (i < 0) ? IOUtils.stringSize(-i) + 1 : IOUtils.stringSize(i);
523 
524             int newcount = count + size;
525             if (newcount > buf.length) {
526                 if (writer == null) {
527                     expandCapacity(newcount);
528                 } else {
529                     char[] chars = new char[size];
530                     IOUtils.getChars(i, size, chars);
531                     write(chars, 0, chars.length);
532                     return;
533                 }
534             }
535 
536             IOUtils.getChars(i, newcount, buf);
537 
538             count = newcount;
539         }
```

这个方法是将 int 型的值转换为字符串写到了输出流中。

现在我们知道了 SerializeWriter 类主要是实现了一些将不同类型变量写到输出流中的方法。对于不同的类型变量，对应的方法有其具体的实现，有一些方法还需要根据配置信息决定往输出流写入的格式。具体实现不做介绍，现在只需要理解这个类的功能就行了。

继续回到之前 JSON 类里的 public static String toJSONString(Object object, int defaultFeatures, SerializerFeature... features)方法：

```
617     public static String toJSONString(Object object, int defaultFeatures, SerializerFeature... features) {
618         SerializeWriter out = new SerializeWriter((Writer) null, defaultFeatures, features);
619 
620         try {
621             JSONSerializer serializer = new JSONSerializer(out);
622             serializer.write(object);
623             return out.toString();
624         } finally {
625             out.close();
626         }
627     }
```

再实例化了 SerializerWriter 类得到对象 out 之后，又将对象 out 作为参数传入 JSONSerializer 类的构造方法，构造了一个 JSONSerializer 实例 serializer，之后回调用 serializer 的 writer(Object object)方法，最后返回 out.toString()。由之前对 SerializerWriter 类的分析可知 toString()方法就是将输出流缓存 buf[]字符数组输出，于是就得到了 Java 对象 Object object 转换为 JSON 字符串的结果。现在来看看 JSONSerializer 类的实现。

JSONSerialize 类定义在 com.alibaba.fastjson.serializer 中，这个类继承了 SerializeFilterable 类。SerializeFilterable 类是序列化拦截器，主要用于过滤一些无用的信息，这里暂不做讨论。这个类同样有一些配置信息的成员变量，但我们主要关注它的 write(Object object)方法。该方法的源码如下：

```
271     public final void write(Object object) {
272         if (object == null) {
273             out.writeNull();
274             return;
275         }
276 
277         Class<?> clazz = object.getClass();
278         ObjectSerializer writer = getObjectWriter(clazz);
279 
280         try {
281             writer.write(this, object, null, null, 0);
282         } catch (IOException e) {
283             throw new JSONException(e.getMessage(), e);
284         }
285     }
```

如果传入参数是 NULL 就输出 out.writeNull()。接下来得到对象 object 的类，这里存在一个泛型的概念。得到 object 的类 clazz 之后，调用 getObjectWriter(clazz)方法返回接口 ObjectSerializer 的示例 writer。接口 ObjectSerializer 源码如下：

```
70     void write(JSONSerializer serializer, //
71                Object object, //
72                Object fieldName, //
73                Type fieldType, //
74                int features) throws IOException;
75 }
```

那么 getObjectWriter(clazz)方法返回的是什么呢？这个方法同样是 JSONSerializer 类里的，源码如下：

```
381     public ObjectSerializer getObjectWriter(Class<?> clazz) {
382         return config.getObjectWriter(clazz);
383     }
```

config 是 SerializeConfig 的实例，调用其 getObjectWriter(Class<?> clazz)方法，之后会再调用重载方法 getObjectWriter(Class<?> clazz, boolean create)，这个方法的实现较长，这里不再放出来了。但知道它的功能是找到传入参数类 clazz 的序列化实例，这些系列化实例在之前的图中已经列举过了，它们也都在 com.alibaba.fastjson.serializer 中。现在我们假设类 clazz 是 Integer 类，以此举例。于是 getObjectWriter(clazz, true)会找到 Integer 类对应的序列化实例：IntegerCodec，显然这些序列化实例都应该实现 ObjectSerializer 接口。于是原来 JSON 类中返回的对象 writer 就是 IntegerCodec 类的实例，这个类实现了 ObjectSerializer 接口的 write 方法，源码如下：

```
40      public void write(JSONSerializer serializer, Object object, Object fieldName, Type fieldType, int features) throws IOException {
41          SerializeWriter out = serializer.out;
42
43          Number value = (Number) object;
44
45          if (value == null) {
46              out.writeNull(SerializerFeature.WriteNullNumberAsZero);
47              return;
48          }
49
50          if (object instanceof Long) {
51              out.writeLong(value.longValue());
52          } else {
53              out.writeInt(value.intValue());
54          }
55
56          if (out.isEnabled(SerializerFeature.WriteClassName)) {
57              Class<?> clazz = value.getClass();
58              if (clazz == Byte.class) {
59                  out.write('B');
60              } else if (clazz == Short.class) {
61                  out.write('S');
62              }
63          }
64      }
```

可见这里调用了之前分析的 SerializerWriter 的 writeLong 和 writeInt 方法，将 Integer 类的变量写到 SerializerWriter 实例 out 的输出流 buf[]中。也就是说在代码

```
617     public static String toJSONString(Object object, int defaultFeatures, SerializerFeature... features) {
618         SerializeWriter out = new SerializeWriter((Writer) null, defaultFeatures, features);
619
620         try {
621             JSONSerializer serializer = new JSONSerializer(out);
622             serializer.write(object);
623             return out.toString();
624         } finally {
625             out.close();
626         }
627     }
```

第 622 行执行后 out 的 buf[]存储着准 JSON 字符串，执行 623 行后这个就得到了转换完成的 JSON 字符串。然后返回该字符串，Java 对象的序列化到此结束。

（二）反序列化

Reference1：

文章作者 ： 诣极(商宗海)

框架作者 ： 高铁

文章地址 ： https://zonghaishang.gitbooks.io/fastjson-source-code-analysis/content/

代码地址 ： https://github.com/zonghaishang/fastjson

框架地址 ： https://github.com/alibaba/fastjson