Fastjson 项目报告

LiHanxuan Student number: 2017K8009908009
lihanxaun17@mails.ucas.ac.cn

January 2020

一、Fastjson简介及反序列化功能的建模

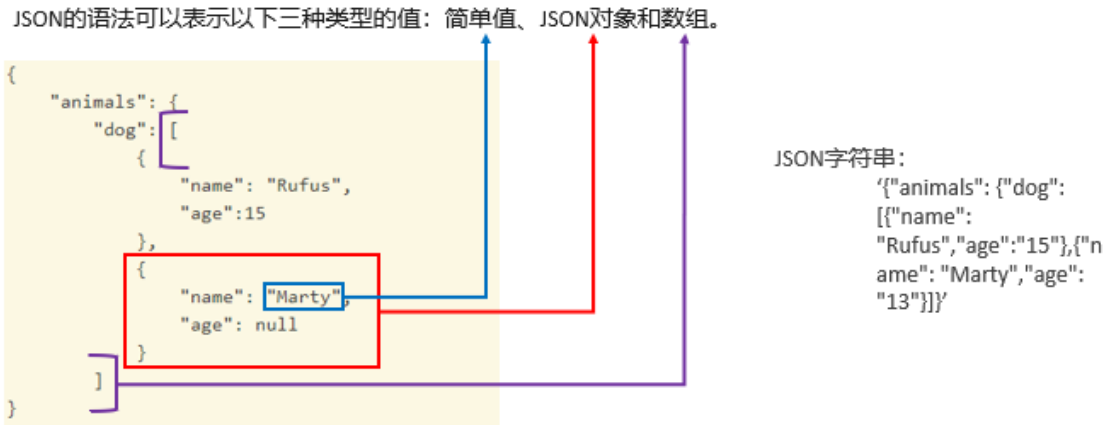
1. JSON介绍

1.1 JSON背景

JSON是JavaScript 的一个严格的子集，利用了JavaScript 中的一些模式来表示结构化数据，用来作为一种轻量级的数据交换格式，作用类似于XML。它不是一种编程语言，仅用来描述数据结构。Douglas Crockford在2001年开始推广JSON数据格式，在2005年-2006年正式成为主流的数据格式，雅虎和谷歌就在那时候开始广泛地使用JSON格式。

Figure 1: Douglas Crockford

1.2 JSON语法



1.2 JSON优点

JSON的作用类似于XML，实际上JSON也就是为了取代XML而诞生的，因此再也没有比用XML与其做对比更好的方式来说明JSON的优越性了。下面简单地举一个例子来说明JSON更加简洁、轻量、可读的优点。

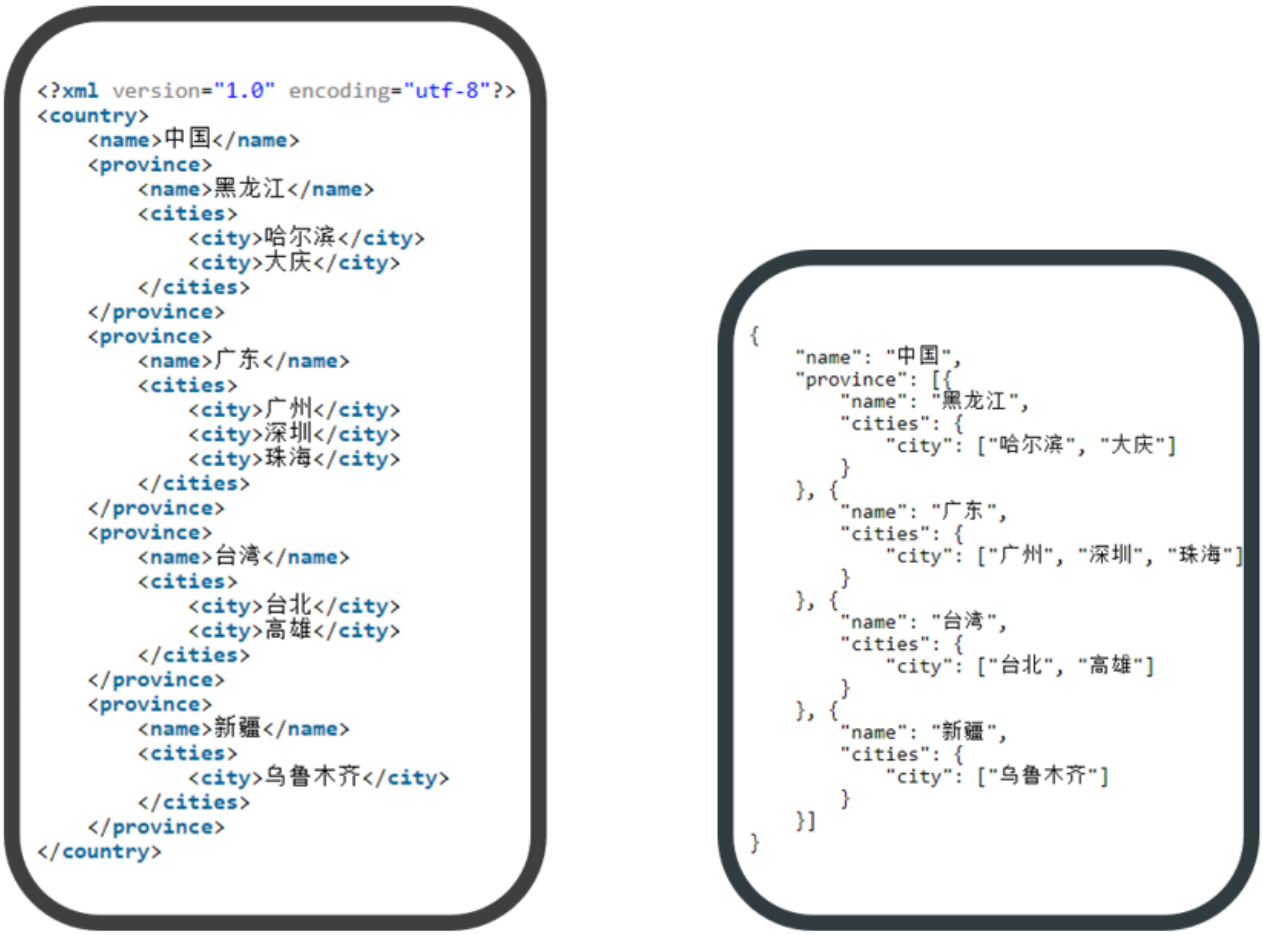


Figure 2: XML vs JSON

以上从左到右分别为具有相同含义的XML和JSON代码，可以明显地看出JSON的语法要更加简洁轻量、且更接近人类阅读的习惯。

2. Fastjson介绍

2.1 Fastjson功能及优点

对于Fastjsoon，阿里官方给的定义是，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。其核心功能如下图所示：

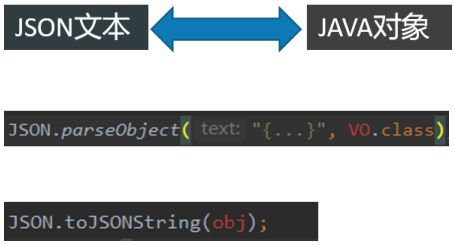


Figure 3: Core Function of Fastjson

Fastjson的优点为：

- 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。
- 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。
- 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。
- 使用简单：fastjson的API 十分简洁。
- 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。

2.2 Fastjson调用

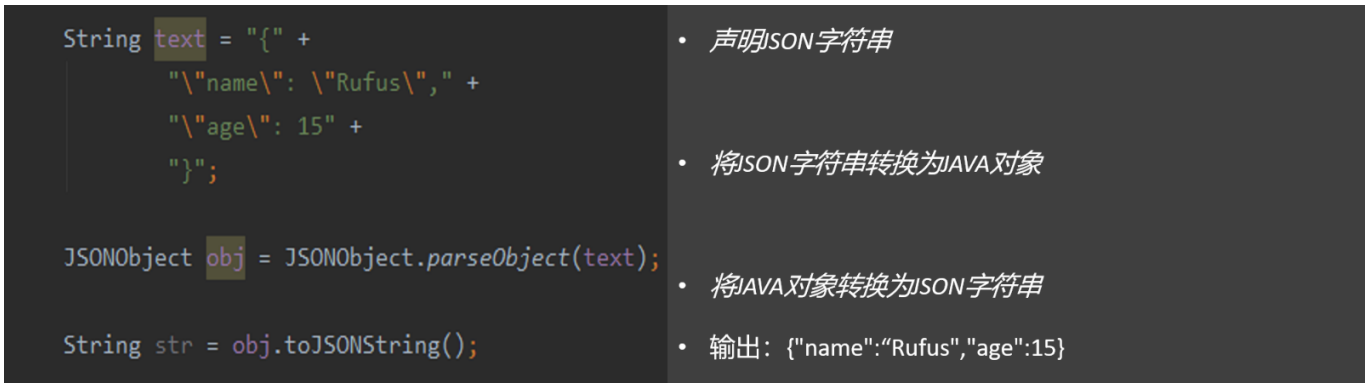


Figure 4: An Example of Fastjson

2.3 Fastjson应用场景

- Web框架处理JSON参数返回JSON结果
- 远程方法调用RPC
- Android/阿里云手机处理JSON
- MessageQueue 传输对象

- 配置文件代替XML
- 保存数据到磁盘、数据库、Hbase

2.4 Fastjson开发者

Figure 5: 温绍锦, 阿里巴巴集团高级专家，Druid和Fastjson开源项目的主要开发者

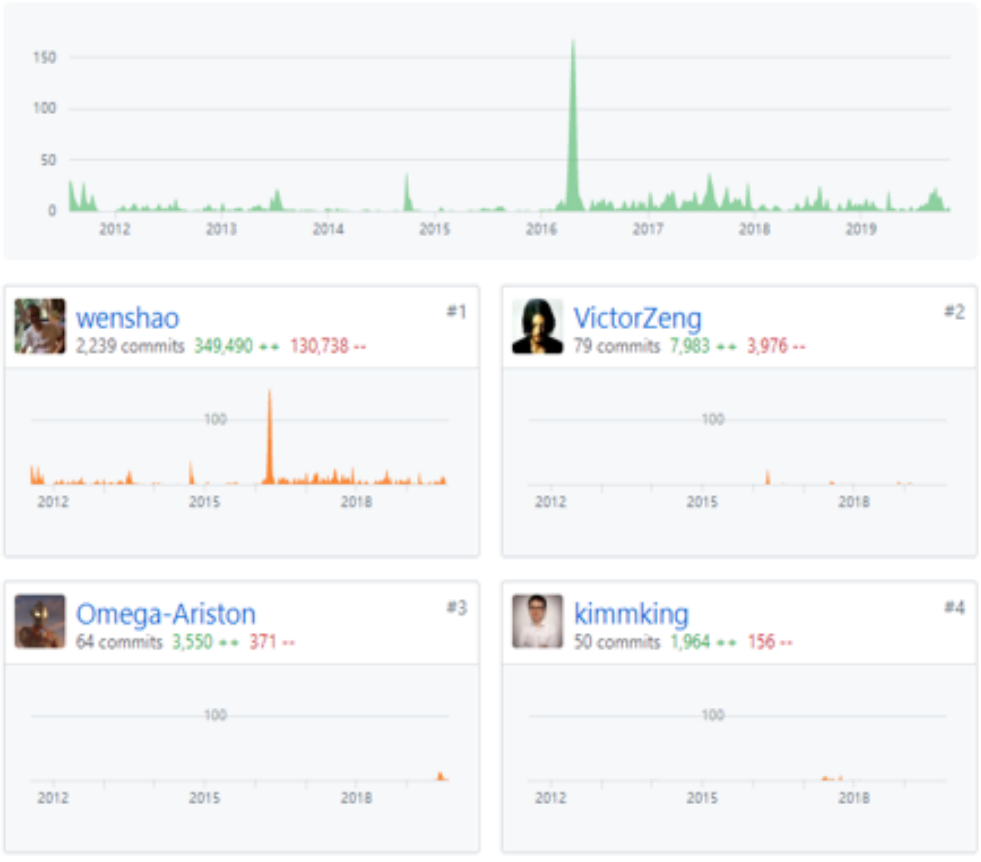


Figure 6: Fastjson贡献排行

3. 反序列化功能的需求分析

下面对反序列化过程面向需求建模，反序列化过程的目的是将JSON文本转化为JAVA对象。

- 输入：JSON文本
- 调用解析器进行处理
- 返回：JAVA对象

反序列化过程的约束与限制是：

- 能够自行判断输入是否合理

按照上述需求，建立三个类：解析器Parser、扫描分析器Scanner、JAVA对象

Figure 7: Class Diagram

按照该建模对整个反序列化场景进行描述如下图所示：

Figure 8: Flow Chart

- Step1:调用接口函数，输入JSON格式的文本
- Step2:解析器调用扫描分析器分析词法、寻找键值对
- Step3:过程中如果发现输入串非法就报错

- Step4:解析器将结果返回给JAVA对象
- Step5:接口函数返回该JAVA对象

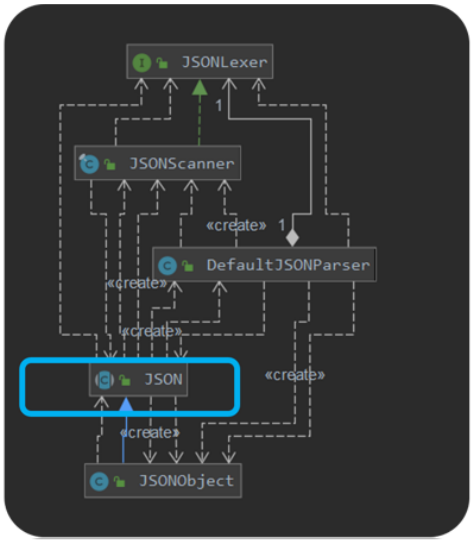
二、核心流程的设计分析

1. 核心流程解析

回顾上节描述的反序列化流程，在DEBUG模式中对该流程进行追踪，并依据各类依赖性关系图进行定位：

- 解析器Parser：DefaultJSONParser
- 扫描分析器Scanner: JSONScanner
- JAVA对象: JSONObject

Step1: 调用接口函数，输入JSON格式的文本



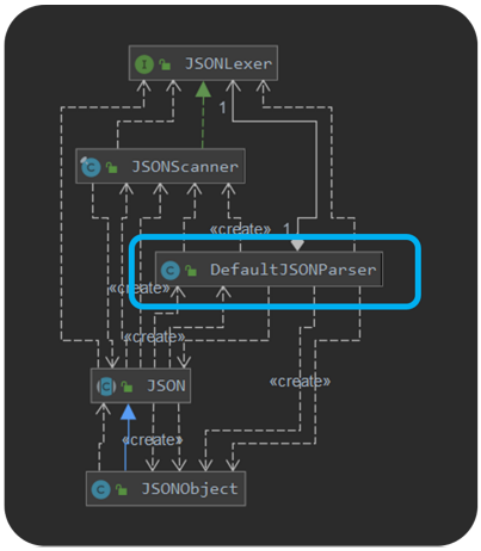

```
public static JSONObject parseObject(String text, Feature... features) {
    return (JSONObject) parse(text, features);
}

public static JSONObject parseObject(String text) {
    Object obj = parse(text);
    if (obj instanceof JSONObject) {
        return (JSONObject) obj;
    }

    try {
        return (JSONObject) JSON.toJSON(obj);
    } catch (RuntimeException e) {
        throw new JSONException("can not cast to JSONObject.", e);
    }
}
```

Figure 9: Step 1

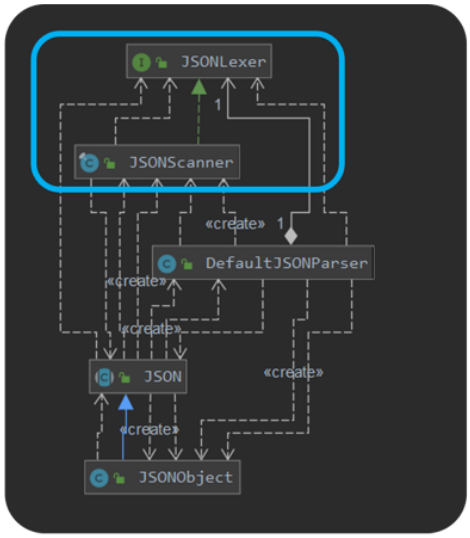
Step2:解析器调用扫描分析器分析词法、寻找键值对




```
public Object parse(Object fieldName) {
    final JSONLexer lexer = this.lexer;
    switch (lexer.token()) {
        case SET:
            lexer.nextToken();
            HashSet<Object> set = new HashSet<~>();
            parseArray(set, fieldName);
            return set;
        case TREE_SET:
```

Figure 10: Step 2

Step3:过程中如果发现输入串非法就报错




```
public final class JSONScanner extends JSONLexerBase {

    private final String text;
    private final int    len;

    public JSONScanner(String input) { this(input, JSON.DEFAULT_PARSER_FEATURE); }

    public JSONScanner(String input, int features){
        super(features);

        text = input;
```

Figure 11: Step 3

Step4:解析器将结果返回给JAVA对象

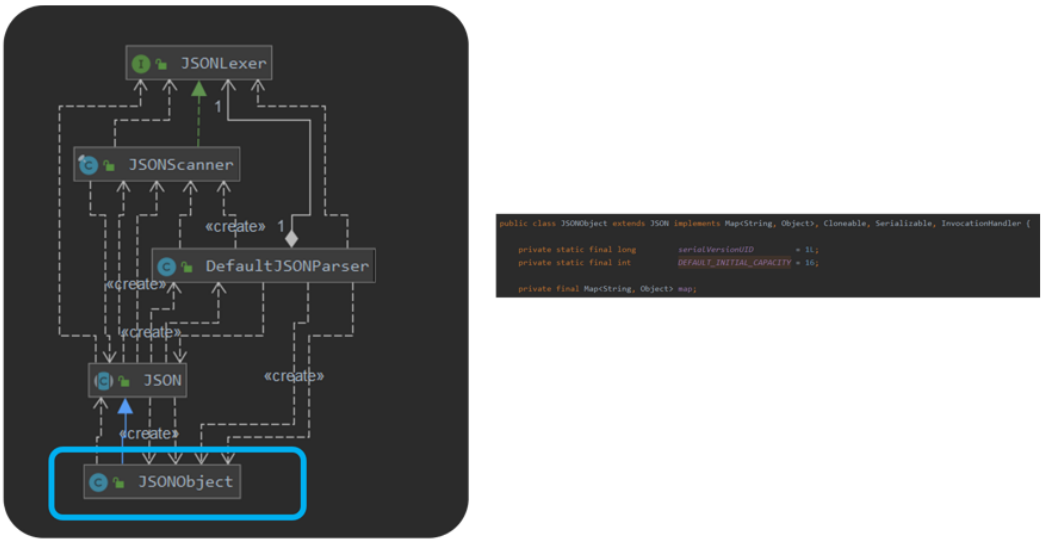


Figure 12: Step 4

Step5:接口函数返回该JAVA对象: 从JAVA接口中返回，与输入类似，不再附图。

2. UML类图分析

选取关键模块进行分析：

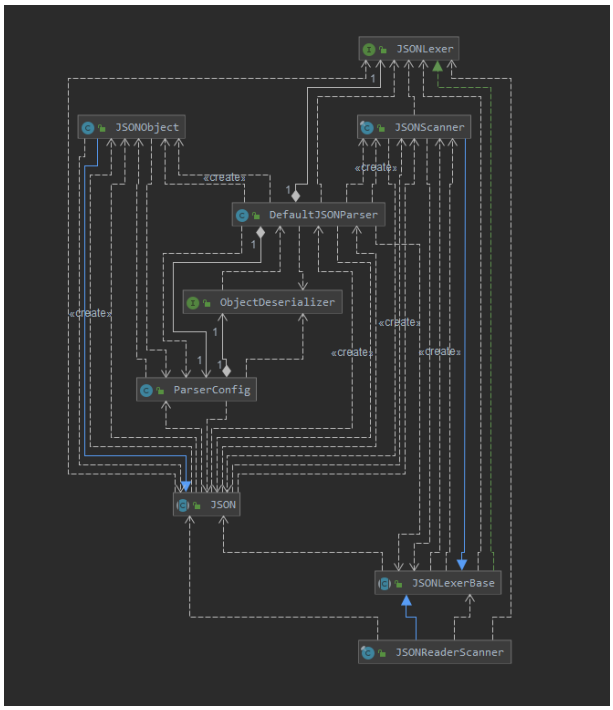


Figure 13: UML Class Diagram

在我们建模的3个类的基础上，Fastjson额外实现了一些其他的接口和类。下面将对上图中的类一一介绍:

- JSON抽象类为入口类，它提供了大量的静态方法。
- JSONObject类继承自JSON抽象类，与我们建模中的用于存储的JAVA对象相对应，存储返回对象。
- DefaultJSONParser与我们建模中的解析器相对应，依赖于ObjectDeserializer接口、ParserConfig类、JSONObject类、JOSNLexerBase抽象类和JSONLexer接口，并组合到了JSON类上。
- ObjectDeserializer接口用于提供泛型的支持；ParserConfig类用于处理配置信息。
- JSONSCanner类和JSONReaderScanner类实现了JSONLexerBase抽象类，该流程中真正被使用的是JSONScanner类，它组合到DefaultJSONParser 类上。
- 除此之外，JSONException类也相当重要，对应了建模约束中的异常处理，但加入后会使依赖图变得相当混乱，故未标出。

3. UML序列图分析

参照IntelliJ自动生成的UML序列图（简化前相当复杂）进行简化（只选取建模中所建立的3个类），得到下图：

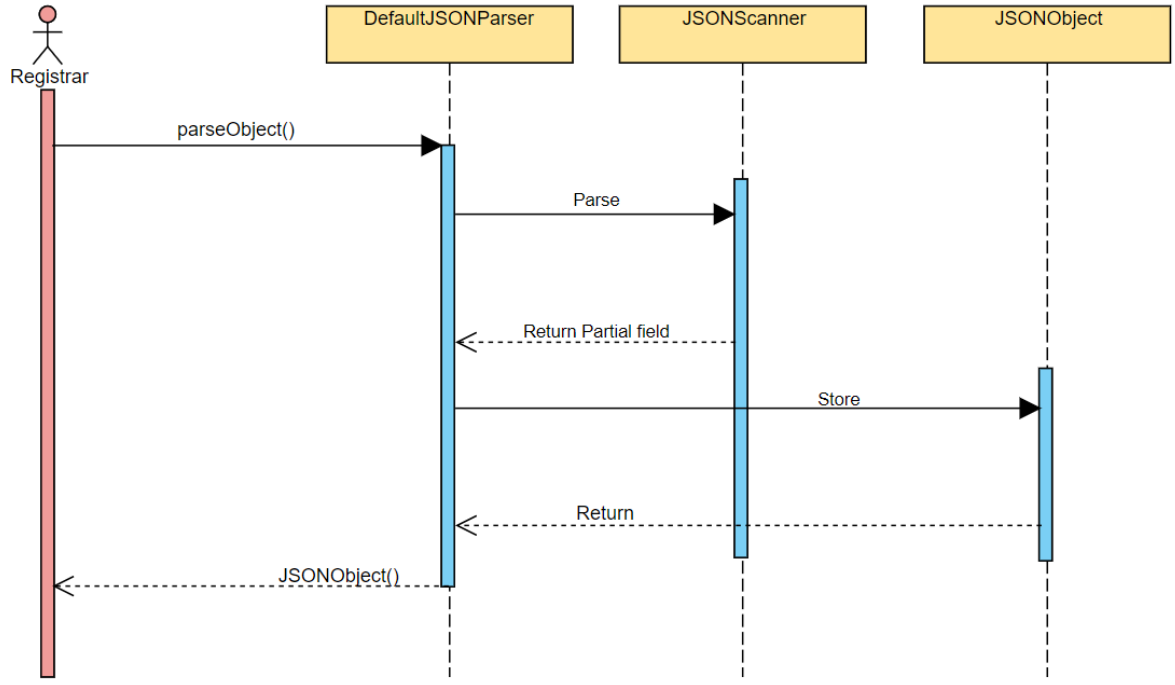


Figure 14: UML Sequence Diagram

该序列图直接体现了上述建模得到的反序列化流程。

4. DSM分析

使用IntelliJ的DSM分析工具对这几个关键的类进行分析，得到下图（位置（m,n）上的数字代表第n列的类多少次依赖于第m行的类）：

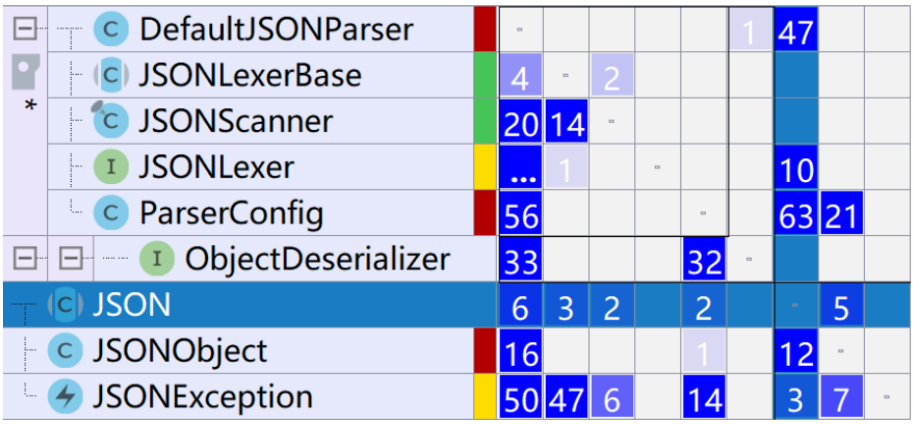


Figure 15: Dependency Structure Matrix

分析DSM图，可以发现顶层类被底层类大量地依赖，即蓝色区域明显未集中在左下方三角区，可以推断出该项目的类的设置并不是十分合理，有违开闭原则。最明显的体现例如右上角的突兀的47。

三、高级设计意图分析

1. 设计原则

回顾上一节对重要类的UML类图的分析，与SOLID五大原则对照，能够发现该架构能够体现下述原则：

1.1.开闭原则

- 开闭原则（OCP）：软件中的对象（类、模块、函数等）应该对于扩展是开放的，但是对于修改是封闭的。

JSONLexer接口和JSONLexerBase抽象类的引入使该项目得能够更好地扩展词法分析器，这符合开闭原则。

1.2. 合成复用原则

- 合成复用原则（CRP）：尽量使用对象组合，而不是继承来达到复用的目的。

当一个类要使用另一个类时，代码编写者都尽可能地使用了关联的关系而非继承关系，这是合成复用原则的一个体现。

1.3.开闭原则

- 单一职责原则（SRP）：就一个类而言，应该仅有一个引起它变化的原因。即一个类中是一组相关性和高的函数，一个类尽量只实现一个功能。

该项目在一些地方明显地将复杂的功能拆成小块，从而使得每块都专注于做一件事情。例如：由于词法解析的过程比较复杂，该工能被拆成了词法分析器和解析器两个部分，这体现了单一职责原则。

2. 设计模式

2.1. 策略模式

策略模式

- 意图：定义一系列的算法,把它们一个个封装起来, 并且使它们可相互替换。
- 主要解决：在有多种算法相似的情况下，使用if...else 所带来的复杂和难以维护。
- 何时使用：一个系统有许多许多类，而区分它们的只是他们直接的行为。
- 如何解决：将这些算法封装成一个一个的类，任意地替换。

下图为ObjectDeserializer接口类及其相关部分的实现：

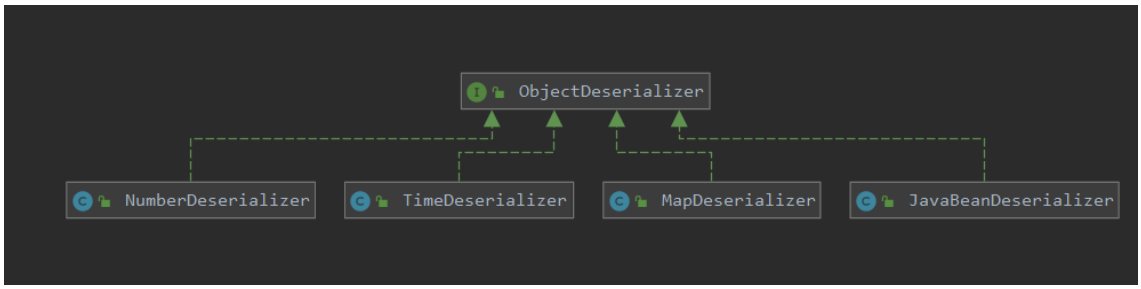


Figure 16: About ObjectDeserializer(Class)

下图为ObjectDeserializer接口类相关部分的代码：

```java
public Object createInstance(DefaultJSONParser parser, Type type) {
    if (type instanceof Class) {
        if (clazz.isInterface()) {
            Class<?> clazz = (Class<?>) type;
            ClassLoader loader = Thread.currentThread().getContextClassLoader();
            final JSONObject obj = new JSONObject();
            Object proxy = Proxy.newProxyInstance(loader, new Class<?>[] { clazz }, obj);
            return proxy;
        }
    }
```

Figure 17: About ObjectDeserializer(Code)

对照可以发现，ObjectDeserializer接口类中只定义了2个方法，它的子类重写了这两个方法，这就构成了策略模式的一个典型的应用场景。这种方式使得该项目更易增加新的特定类型的反序列化器。

2.2. 单例模式

单例模式

- 意图：保证一个类仅有一个实例，并提供一个访问它的全局访问点。
- 主要解决：一个全局使用的类频繁地创建与销毁。
- 何时使用：当您想控制实例数目，节省系统资源的时候。
- 如何解决：判断系统是否已经有这个单例，如果有则返回，如果没有则创建。

下图为MapDeserializer类的相关代码：

```java
public class MapDeserializer implements ObjectDeserializer {
    public static MapDeserializer instance = new MapDeserializer();
```

Figure 18: About MapDeserializer(Code)

该类实现了在类加载时就完成初始化，所以类加载较慢，但获取对象的速度快。这种方式基于类加载机制避免了多线程的同步问题，这是典型的饿汉式的单例模式。这么设计的原因是：当大量反序列化请求同时到达时，Deserilizer会被频繁地创建和销毁，这样会浪费大量的系统开销。采用单例模式便可以解决该问题，明显降低重复创建和校会的系统开销。

四、结语

终于完成了这份报告，在阅读源码的过程中我学到了很多，自学了java的语法，从实际项目的角度理解了课上讲的一些理论知识，学会了使用IntelliJ来分析项目（真心好用）等等。但由

于java水平有限，很多源码都还没有进行仔细分析，毕竟刚接触面向对象编程，要实践的东西还有很多。

在查找资料的过程中，我也发现了一些有趣的事情，发现fastjson虽然速度很快，但使用者的数量似乎依旧比不上gson和jackson，同时可以在各种论坛上找到吐槽fastjson的帖子。在进一步调查后我发现，网友的吐槽大多针对几个方面：其一是fastjson为了提速用了好多“黑科技”，但这些“黑科技”牺牲了整个项目的稳定性和扩展性，而在业界，保证质量过关显然要比提上一点速度重要得多。其二是fastjson的文档和代码比较糟糕，为了求证这一点，我专门对比了gson的英文文档和源码，发现这一点确实无话可说。其三是fastjosn所谓的“快”似乎并没有得到所有人的认同。

这些信息给了我很大的启发，在面向需求建模和编程时，一定要明确各需求间的重要性。比如fastjson宣传的核心就是它的快，可对于企业来说，稳定性可能是更重要的一个部分，至少是不能为了提速牺牲掉的一个部分。以及一定要用心编写文档。

本学期的面向对象程序设计使我受益匪浅，王老师的课别具风格，上课会比较幽默，会结合工业界或者截自己朋友圈的具体的例子来讲解枯燥的概念，所以别具风格。助教哥哥也相当相当友善，QQ上问问题基本都是秒回，实在是太走心了。总的来说，干货很多，相当有诚意。