Presentation

JSON with Java

Presenter: Team17 Time: Nov.2020

Team Members:
1853802 Jintao Xie
1851202 Hong Lei
1853806 Jiaqi Wu
1851906 Yusen Li

CONTENTS

1 • PART 01 Overview

A brief Overview of the Json with Java Technology

2 • PART 02 GSON for Java

An Open-source Library Developed By Google。

3 • PART 03 Jackson for Java

An Open-source JSON Library Developed By FasterXML

4 • PART 04 Comparison

PART ONE

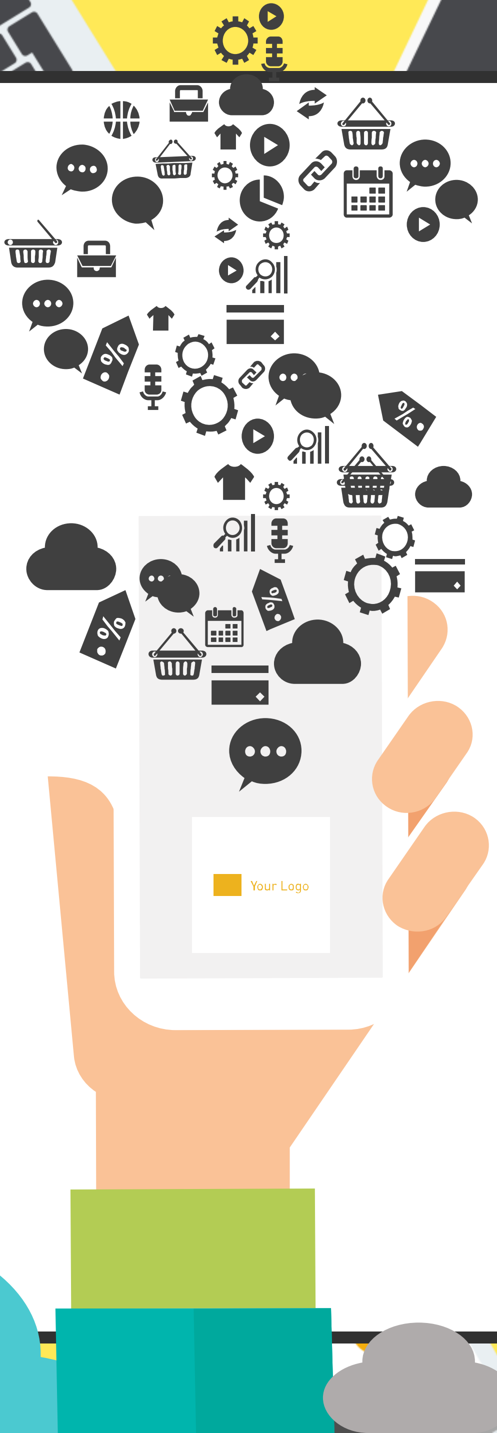
Overview of Json With Java

Ⅰ Definition

Ⅱ Reasons

Ⅲ Implement

Learn It Now



I . Definition

json to java

java to json

Learn It Now

Ⅱ. Reasons

Two format Characters → Serialized and Deserialized → Easy to Delivery Easy to Code

Ⅲ. Implement

Gson

Jackson

Other Libraries (FastJSON etc.)

PART TWO

GSON for Java

——An Open-source Library Developed By Google

Overview

• Gson is a Java library that can be used to convert Java Objects into their JSON representation.

• It can also be used to convert a JSON string to an equivalent Java object.

• Gson can work with arbitrary Java objects including pre-existing objects that you do not have source code of.

• Considering other open-source projects, most of them:

①require that users place Java annotations in your classes

②also do not fully support the use of Java Generics.

• Gson considers both of these as very important design goals.

Use It Now

Why we should be using the library~

5 Advantages:

- **Standardized**

Gson is a standardized library that is managed by Google.

- **Efficient**

Gson is a reliable, fast, and efficient extension to the Java standard library.

- **Optimized**

The library is highly optimized.

- **Support Generics**

It provides extensive support for generics.

- **Supports complex inner classes**

Gson supports complex objects with deep inheritance hierarchies.

Features of Gson

01 **Easy to use -**
Gson API provides a high-level facade to simplify commonly used use-cases.

02 **No need to create mapping –**
Gson API provides default mapping for most of the objects to be serialized.

03 **Performance –**
Gson is quite fast and is of low memory footprint. It is suitable for large object graphs or systems.

04 **Clean JSON –**
Gson creates a clean and compact JSON result which is easy to read.

05 **No Dependency –**
Gson library does not require any other library apart from JDK.

06 **Open Source –**
Gson library is open source; it is freely available.

Two significant methods of GSON

toJson()

Used to convert Java objects to JSON, mainly in the following forms:

(1) String toJson(JsonElement jsonElement);

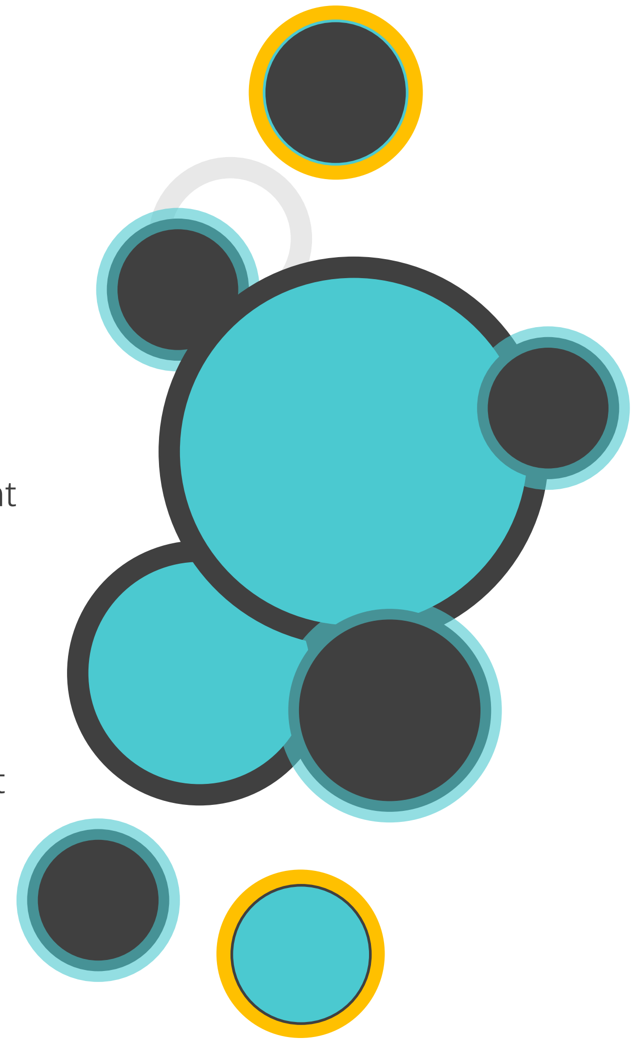
(2) String toJson(Object src);

(3) String toJson(Object src, Type typeOfSrc);

Besides, method (1) is used to convert JsonElement object which can be JsonObject, JsonArray, etc..) into Json data;

method (2) serializes the specified objects into the corresponding JSON data;

method (3) is used to serialize the specified Object which can include generic types into the corresponding JSON data.



fromJson()

Used to convert JSON to Java objects, mainly in the following forms:

(1) <T> T fromJson(JsonElement json, Class<T> classOfT);

(2) <T> T fromJson(JsonElement json, Type typeOfT);

(3) <T> T fromJson(JsonReader reader, Type typeOfT);

(4) <T> T fromJson(Reader reader, Class<T> classOfT);

(5) <T> T fromJson(Reader reader, Type typeOfT);

(6) <T> T fromJson(String json, Class<T> classOfT);

(7) <T> T fromJson(String json, Type typeOfT);

All of them are used to parse the different forms of JSON data into Java objects

Experimentation —— Using Gson

IDE: eclipse

Create a maven project

Import Gson

Primitives Examples

Object Examples

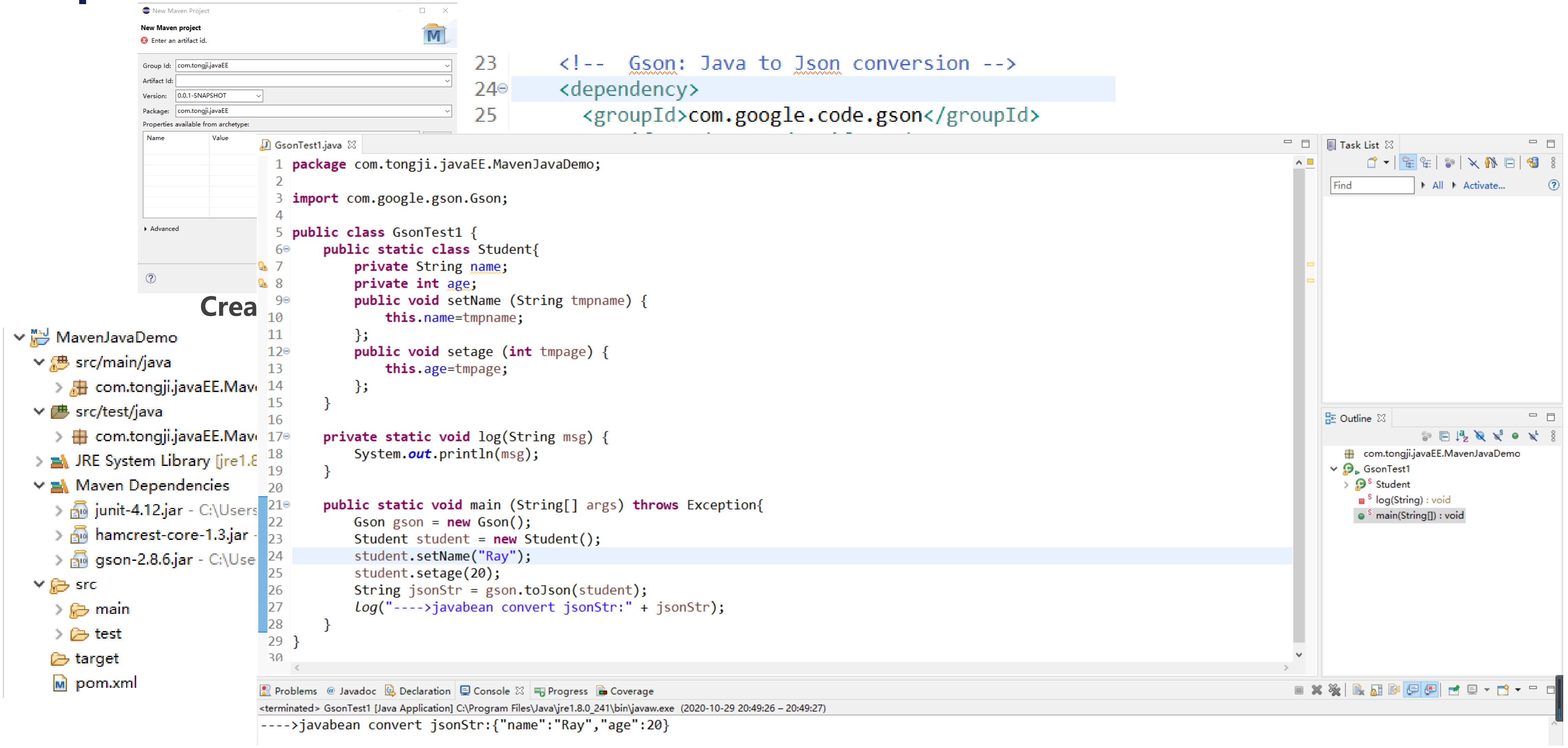
Array Examples

Collections Examples

Serializing and Deserializing

Others

Preparation



New Maven Project

New Maven project

Enter an artifact id.

Group Id: com.tongji.javaEE

Artifact Id:

Version: 0.0.1-SNAPSHOT

Package: com.tongji.javaEE

Properties available from archetype:

Name	Value

Advanced

Crea

```
<!--  Gson: Java to Json conversion -->
<dependency>
  <groupId>com.google.code.gson</groupId>
```

```
MavenJavaDemo
  src/main/java
    com.tongji.javaEE.Mav
  src/test/java
    com.tongji.javaEE.Mav
  JRE System Library [jre1.8
  Maven Dependencies
    junit-4.12.jar - C:\Users
    hamcrest-core-1.3.jar -
    gson-2.8.6.jar - C:\Use
  src
    main
    test
  target
  pom.xml
```

GsonTest1.java

```java
package com.tongji.javaEE.MavenJavaDemo;

import com.google.gson.Gson;

public class GsonTest1 {
    public static class Student{
        private String name;
        private int age;
        public void setName (String tmpname) {
            this.name=tmpname;
        };
        public void setage (int tmpage) {
            this.age=tmpage;
        };
    }

    private static void log(String msg) {
        System.out.println(msg);
    }

    public static void main (String[] args) throws Exception{
        Gson gson = new Gson();
        Student student = new Student();
        student.setName("Ray");
        student.setage(20);
        String jsonStr = gson.toJson(student);
        log("---->javabean convert jsonStr:" + jsonStr);
    }
}
```

```
<terminated> GsonTest1 [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (2020-10-29 20:49:26 – 20:49:27)
---->javabean convert jsonStr:{"name":"Ray","age":20}
```

PART THREE

Jackson for Java

—— An Open-source JSON Library Developed By FasterXML

Features

1

High performance and stability

Low memory usage, excellent performance in convertion between large or small JSON strings and POJOs

2

High Popularity

Default choice for many popular frameworks, complete and standard English documents

3

Easy to Use

Provide flexible APIs, which are extensible

Encapsulate some common modules like new time and date type in Java8

4

Various mapping relations

Jackson covers mapping relations between most string types like JSON or XML and Java objects

5

Spring Frameworks support

Jackson is the default JSON/XML parser in Spring Frameworks

Three Core Modules

com.fasterxml.jackson.core
Define APIs for underlying data processing stream, like JsonParser.class and JsonGenerator.class

com.fasterxml.jackson.annotations
Contains standard Jackson annotations

com.fasterxml.jackson.databind
Implement data binding and object serialization on “com.fasterxml.jackson.core” module, this module depends on th above two modules, it also provides high level APIs that we can directly use

Three key operations

JSON<->OBJECT

(1)ObjectMapper.readValue(JsonElement,Object.class); *// JSON->OBJECT*
(2)ObjectMapper.writeValueAsString(Object); *// OBJECT->JSON*

JSON<->NODE

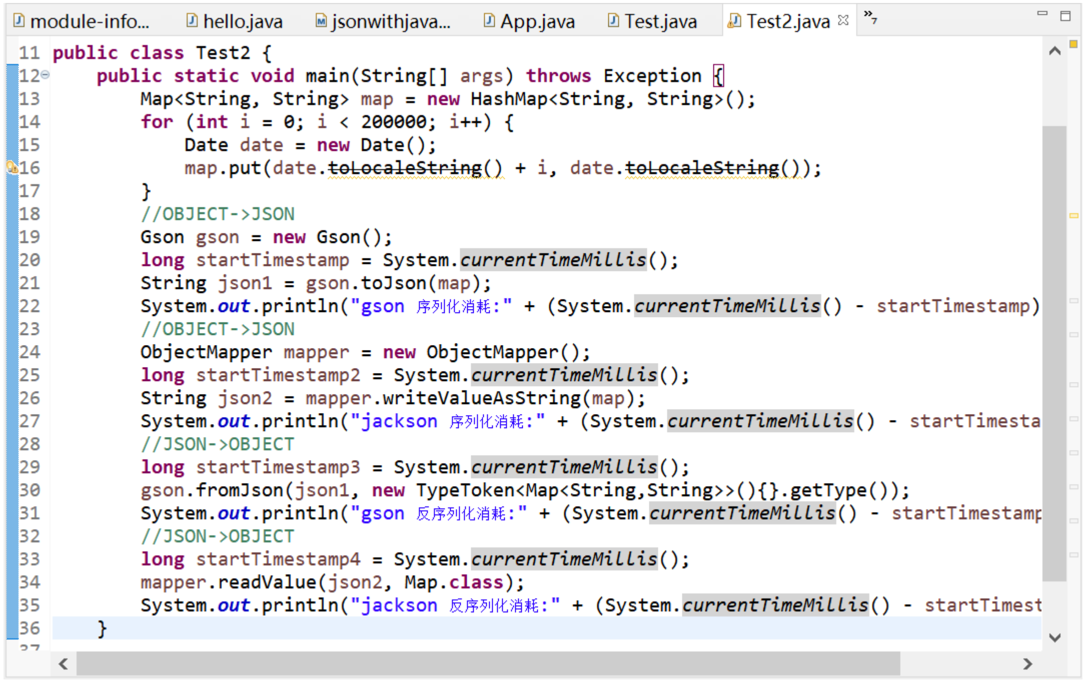
(1)ObjectMapper.readTree(JsonElement); *// JSON->NODE*
(2)ObjectMapper.writeValueAsString(Node); *// NODE->JSON*

OBJECT<->NODE

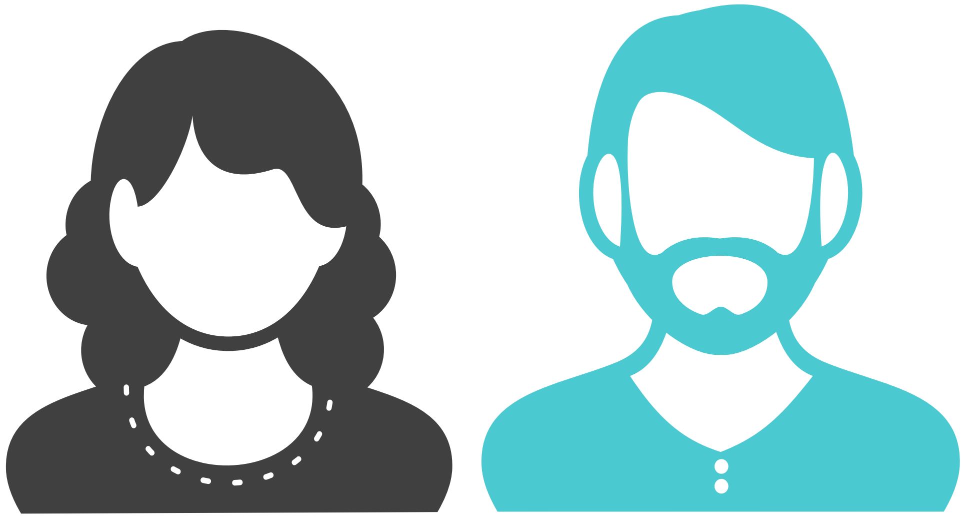
(1)ObjectMapper.treeToValue(Node,Object.class); *// NODE->OBJECT*
(2)ObjectMapper.valueToTree(Object); *// OBJECT->NODE*

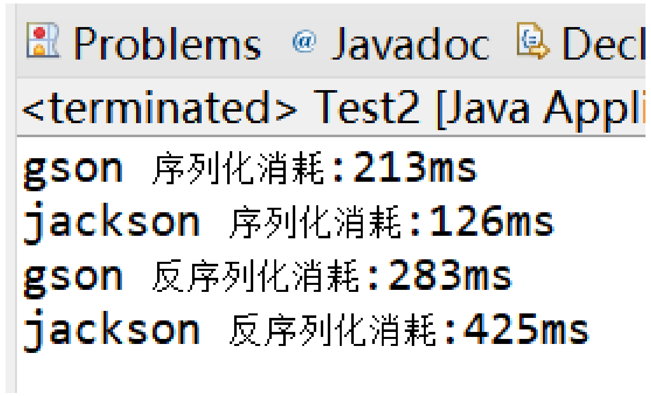
A main interface in Jackson — — ObjectMapper

TEST



```java
public class Test2 {
    public static void main(String[] args) throws Exception {
        Map<String, String> map = new HashMap<String, String>();
        for (int i = 0; i < 200000; i++) {
            Date date = new Date();
            map.put(date.toLocaleString() + i, date.toLocaleString());
        }
        //OBJECT->JSON
        Gson gson = new Gson();
        long startTimestamp = System.currentTimeMillis();
        String json1 = gson.toJson(map);
        System.out.println("gson 序列化消耗:" + (System.currentTimeMillis() - startTimestamp)
        //OBJECT->JSON
        ObjectMapper mapper = new ObjectMapper();
        long startTimestamp2 = System.currentTimeMillis();
        String json2 = mapper.writeValueAsString(map);
        System.out.println("jackson 序列化消耗:" + (System.currentTimeMillis() - startTimesta
        //JSON->OBJECT
        long startTimestamp3 = System.currentTimeMillis();
        gson.fromJson(json1, new TypeToken<Map<String,String>>(){}.getType());
        System.out.println("gson 反序列化消耗:" + (System.currentTimeMillis() - startTimestamp
        //JSON->OBJECT
        long startTimestamp4 = System.currentTimeMillis();
        mapper.readValue(json2, Map.class);
        System.out.println("jackson 反序列化消耗:" + (System.currentTimeMillis() - startTimest
    }
```

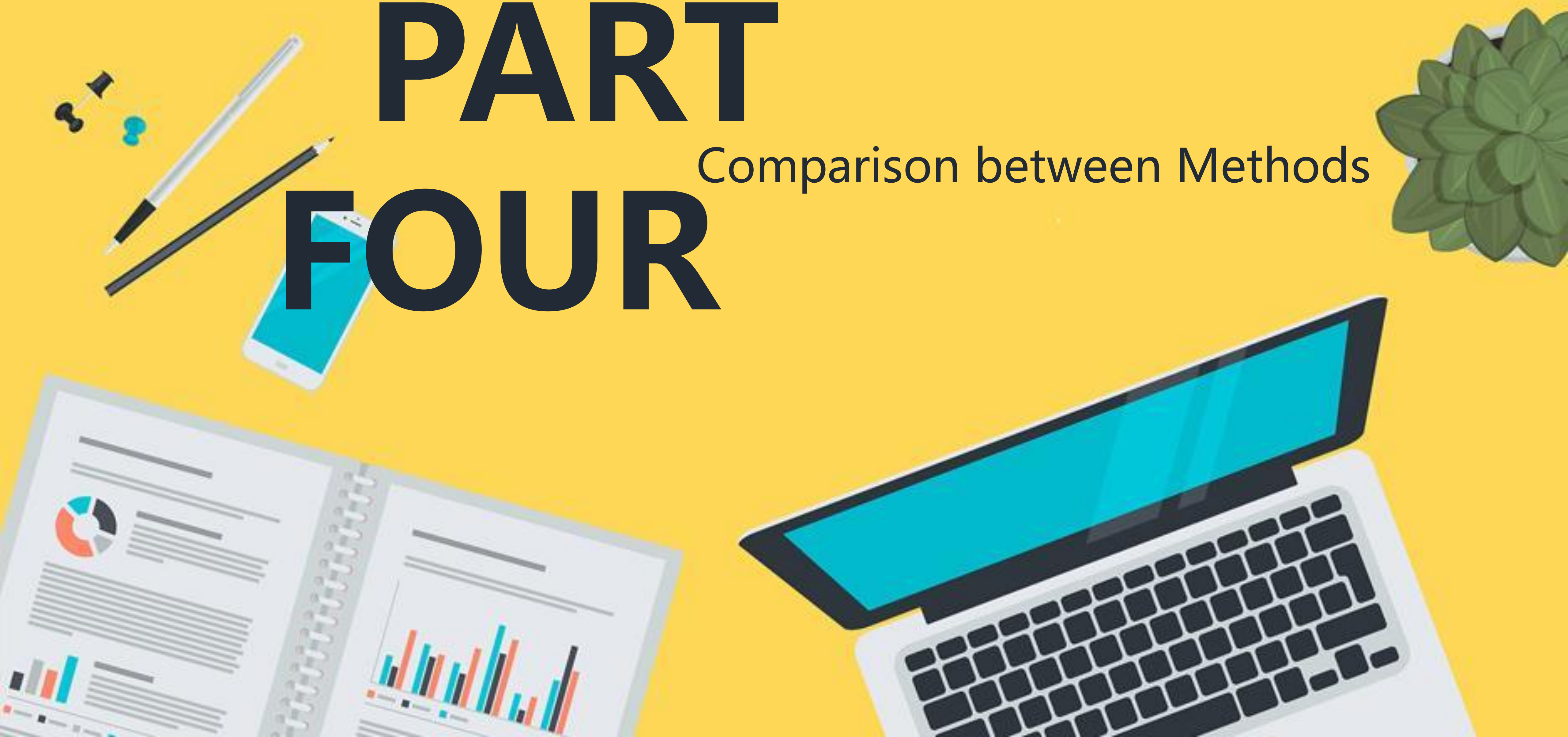




```
<terminated> Test2 [Java Appli
gson 序列化消耗:213ms
jackson 序列化消耗:126ms
gson 反序列化消耗:283ms
jackson 反序列化消耗:425ms
```

PART FOUR

Comparison between Methods



Performance

Serialized (Obj=>Json)

Library	Size	Times	Max T	Min T	Avg T
Jackson	100000	10	1980 (ms)	841 (ms)	880 (ms)
Gson	100000	10	2383 (ms)	1469 (ms)	1520 (ms)

Deserialized (Json=>Obj)

Library	Size	Times	Max T	Min T	Avg T
Jackson	100000	10	7957 (ms)	6632 (ms)	6815 (ms)
Gson	100000	10	8235 (ms)	7006 (ms)	7364 (ms)

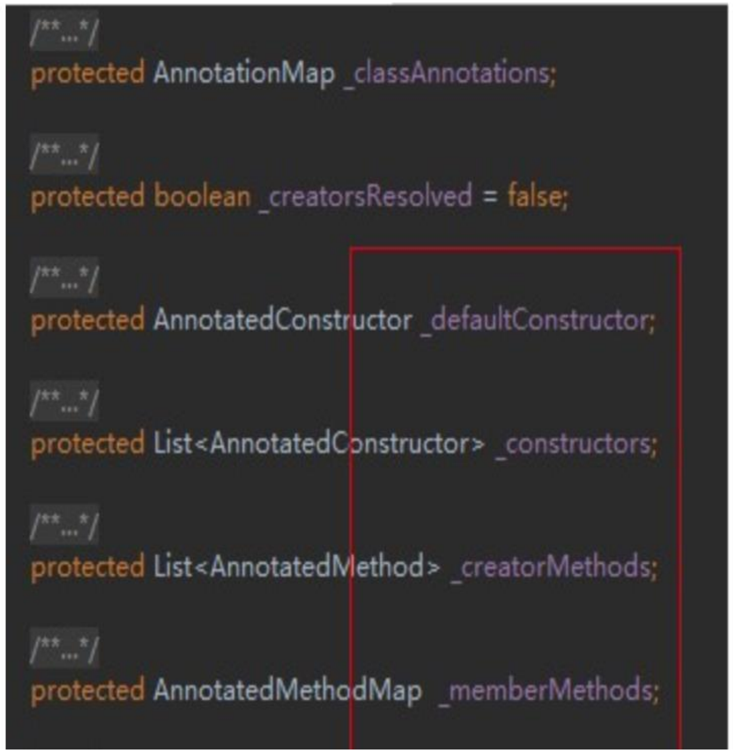
Character

Jackson

Jackson is Powerful：support stream、 databind、path

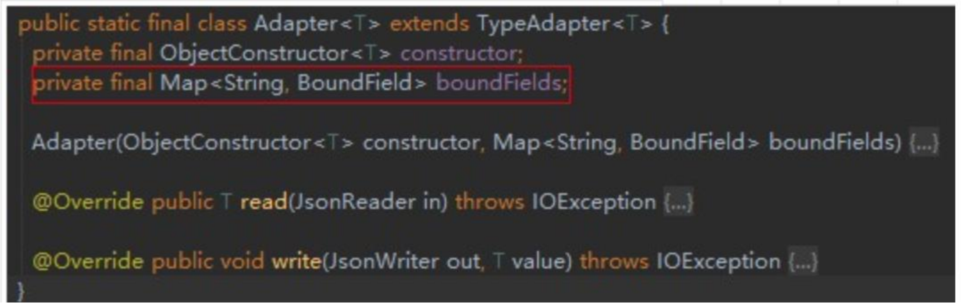
Package:1.2M ,
Several Hundred KB Memory Needed,
Effective Memory Management

Complex Reflection Cache



```
/**...*/
protected AnnotationMap _classAnnotations;

/**...*/
protected boolean _creatorsResolved = false;

/**...*/
protected AnnotatedConstructor _defaultConstructor;

/**...*/
protected List<AnnotatedConstructor> _constructors;

/**...*/
protected List<AnnotatedMethod> _creatorMethods;

/**...*/
protected AnnotatedMethodMap _memberMethods;
```

Gson



```
public static final class Adapter<T> extends TypeAdapter<T> {
  private final ObjectConstructor<T> constructor;
  private final Map<String, BoundField> boundFields;

  Adapter(ObjectConstructor<T> constructor, Map<String, BoundField> boundFields) {...}

  @Override public T read(JsonReader in) throws IOException {...}

  @Override public void write(JsonWriter out, T value) throws IOException {...}
}
```

Direct and Easy Reflection Cache

Package:143 K ,
Less than 100 KB Memory Needed,
Not Such good Memory Management

Thanks for Listening

Presenter: Team17

Time: Nov.2020