GSON: JSON PROCESSING

JSON Processing Libraries

- GSON
- Jackson
- JSON.simple
- JSON-P (JSR 353)

Primitives

```
// Serialization
Gson gson = new Gson();
gson.toJson(1);             // ==> 1
gson.toJson("abcd");        // ==> "abcd"
gson.toJson(new Long(10)); // ==> 10
int[] values = { 1 };
gson.toJson(values);        // ==> [1]

// Deserialization
int one = gson.fromJson("1", int.class);
Integer one = gson.fromJson("1", Integer.class);
Long one = gson.fromJson("1", Long.class);
Boolean false = gson.fromJson("false", Boolean.class);
String str = gson.fromJson("\"abc\"", String.class);
String[] anotherStr = gson.fromJson("[\"abc\"]",
                                    String[].class);
```


Objects

```
class BagOfPrimitives {
  private int value1 = 1;
  private String value2 = "abc";
  private transient int value3 = 3;
  BagOfPrimitives() {
    // no-args constructor
  }
}

// Serialization
BagOfPrimitives obj = new BagOfPrimitives();
Gson gson = new Gson();
String json = gson.toJson(obj);

// ==> json is {"value1":1,"value2":"abc"}
```

Arrays

```
Gson gson = new Gson();
int[] ints = {1, 2, 3, 4, 5};
String[] strings = {"abc", "def", "ghi"};

// Serialization
gson.toJson(ints);     // ==> [1,2,3,4,5]
gson.toJson(strings);  // ==> ["abc", "def", "ghi"]

// Deserialization
int[] ints2 = gson.fromJson("[1,2,3,4,5]", int[].class);
// ==> ints2 will be same as ints
```

101

101

Collections

```
Gson gson = new Gson();
Collection<Integer> ints = Arrays.asList(1,2,3,4,5);

// Serialization
String json = gson.toJson(ints);  // ==> json is [1,2,3,4,5]

// Deserialization
Type collectionType =
    new TypeToken<Collection<Integer>>(){}.getType();

Collection<Integer> ints2 =
    gson.fromJson(json, collectionType);

// ==> ints2 is same as ints
```

102

102

Generics

```
class Foo<T> {
  T value;
}

Gson gson = new Gson();
Foo<Bar> foo = new Foo<Bar>();

gson.toJson(foo);
// May not serialize foo.value correctly

gson.fromJson(json, foo.getClass());
// Raw class, fails to deserialize foo.value as Bar
```

103

103

Generics

```
class Foo<T> {
  T value;
}

import java.lang.reflect.Type;

Gson gson = new Gson();
Foo<Bar> foo = new Foo<Bar>();

Type fooType = new TypeToken<Foo<Bar>>() {}.getType();

gson.toJson(foo, fooType);
// Type token has fully parameterized type

gson.fromJson(json, fooType);
// Type token has fully parameterized type
```

104

104

Custom Date Formats

```
public String DATE_FORMAT = "yyyy-MM-dd";
public String ISO_FORMAT =
     "yyyy-MM-dd'T'HH:mm:ssZ";

GsonBuilder gsonBuilder = new GsonBuilder();
gsonBuilder.setDateFormat(DATE_FORMAT);
gsonBuilder.setDateFormat(ISO_FORMAT);
Gson gson = gsonBuilder.create();
```


Custom Java Types

```
public class TimestampSerializer
        implements JsonSerializer<Date>, JsonDeserializer<Date> {

    public JsonElement serialize(Date src, Type typeOfSrc,
                                 JsonSerializationContext context)
    {
        return new JsonPrimitive(src.getTime());
    }

    public Date deserialize(JsonElement json, Type typeOfSrc,
                            JsonDeserializationContext context)
            throws JsonParseException
    {
        return new Date(json.getAsJsonPrimitive()
                            .getAsLong());
    }
}
```

Custom Java Types

```
public class TimestampSerializer
        implements JsonSerializer<Date>, JsonDeserializer<Date> {

    public JsonElement serialize(Date src, Type typeOfSrc,
                                 JsonSerializationContext context)
    { ... }
    public Date deserialize(JsonElement json, Type typeOfSrc,
                            JsonDeserializationContext context)
    { ... }
}

GsonBuilder gsonBuilder = new GsonBuilder();
gsonBuilder.registerTypeAdapter(Timestamp.class,
                                new TimestampDeserializer());
Gson Gson = gsonBuilder.create();
```

107

107

Excluding Fields

```
public class User {

    private long id;

    @Exclude
    private UUID appId;

    private String name;

    @Exclude
    private boolean active;

    // Last time we heard from this user.
    private Date timestamp;

}
```

108

108

Excluding Fields

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface Exclude {}


public class ExcludeStrategy implements ExclusionStrategy {
    public boolean shouldSkipClass(Class<?> clazz) {
        return false;
    }
    public boolean shouldSkipField(FieldAttributes field) {
        return field.getAnnotation(Exclude.class) != null;
    }
}


GsonBuilder gsonBuilder = new GsonBuilder();
gsonBuilder.setExclusionStrategies(new ExcludeStrategy());
this.gson = gsonBuilder.create();
```

109

109

Streaming

```
InputStream in;
try (JsonReader rd = gson.newJsonReader(
       new BufferedReader(new InputStreamReader(in, "UTF-8")))) {

    rd.beginObject();

    label = rd.nextName();
    if (!"messages".equals(label)) ... // throw exception

    // Upload messages from client.
    rd.beginArray();
    while (rd.hasNext()) {
        Message message = gson.fromJson(rd, Message.class);
        ...
    }
    rd.endArray();

    rd.endObject();
}
```

110

110

Streaming

```
OutputStream out;
List<Message> messages;

try (JsonWriter wr = gson.newJsonWriter(
       new BufferedWriter(new OutputStreamWriter(out, "UTF-8")))) {
    wr.beginObject();

    // Download list of messages to client
    wr.name("messages");
    wr.beginArray();
    for (Message message : messages) {
        gson.toJson(message, Message.class, wr);
    }
    wr.endArray();

    wr.endObject();
    wr.flush();
}
```

111

111