Atbash JSON Smart

Rudy De Busscher

Version 0.9.0, ??/??/2018

Table of Contents

User manual for Atbash configuration.

Release Notes

0.9.0

1. Initial release, based on the code of JSON smart v2.
2. Support for @MappedBy for easy customized mapping logic.
3. Support for *top-level* Parameterized Types
4. Optimization for use within Octopus JWT

Introduction

JSON Smart v2, is a small fast library for converting POJO to and from JSON.

When creating the JSON and JWT support within Octopus, I found out it was difficult to have custom logic for a few cases.

That was the reason to start from the original code and tailor it to my needs (since the library itself is no longer maintained).

Standard usage

```
  T JSONValue.parse(String, Class<T>);
```

Converts the JSON to instance of T.

```
  List<Bean> data = (List<Bean>) JSONValue.parse("<json>", new
TypeReference<List<Bean>>(){});
```

Converts the <json> to instance of List containing Bean instances.

Without the TypeReference information, Atbash JSON is only capable of creating a List of JSONObject.

```
  List<JSONObject> data = JSONValue.parse("<json>", ArrayList.class);
```

```
  String JSONValue.toJSONString(T);
```

Converts the POJO to a JSON String

Customizations

Custom JSON creation

The creation of the JSON can be customized in 2 ways. You can implement the interface *be.atbash.json.JSONAware* or define a custom *Writer* with the *@MappedBy* annotation.

```
public interface JSONAware {

    /**
     * @return JSON text
     */
    String toJSONString();

}
```

The result of the *toJSONString()* method will be added to the JSON output. One can make use of the *JSONObject* class to help in the creation of JSON like,

```
    public String toJSONString() {
        JSONObject result = new JSONObject();
        result.put("key1", key1);
        result.put("key2", key2);
        result.put("key3", key3);
        for (Map.Entry<String, String> entry : additional.entrySet()) {
            result.put(entry.getKey(), entry.getValue());
        }
        return result.toJSONString();
    }
```

You need to make sure that you serialize the complete object tree to JSON.

Another option, but very similar, is to use an annotation to indicate the code which needs to be called when the Object needs to be Serialized to JSON. This way, the code to create the JSON can be kept out of the class itself.

Annotate the Object with *be.atbash.json.parser.MappedBy* and specify the Writer within the *writer()* member.

```
@MappedBy(writer = PriceJSONWriter.class)
```

and

```
public class PriceJSONWriter implements JSONWriter<PriceWithWriter> {

    @Override
    public <E extends PriceWithWriter> void writeJSONString(E value, Appendable out)
throws IOException {
        out.append(String.format("\"%s%s\"", value.getValue(),
value.getCurrency().toJSONString()));
    }
}
```

In this example, the Currency object implements the *JSONAware* interface.

Custom reading of JSON

The conversion from JSON to an object instance can be customized by encoders which can be defined with *@MappedBy*.

The most generic way is to use an implementation of *be.atbash.json.parser.CustomJSONEncoder*

```
public interface CustomJSONEncoder<T> {

    T parse(Object data);

}
```

The data parameter is most of the time an instance of String, but can be any primitive, JSONArray or JSONObject in case the JSON is malformed or has wrong contents (other contents then expected).

An implementation of this interface needs a no argument constructor.

There is a special encoder available, *be.atbash.json.writer.CustomBeanJSONEncoder*, which tries to use the setters if they are available, or call the *setCustomValue()* method otherwise. An example can be seen at the test class *be.atbash.json.testclasses.Token* and *be.atbash.json.testclasses.TokenJSONEncoder*.

An instance of this class needs a constructor which takes an *be.atbash.json.parser.reader.JSONReader* parameter.

Both classes needs to be specified by a @MappedBy annotation, *encoder()* member for the simple CustomJSONEncoder implementation, *beanEncoder()* member for CustomBeanJSONEncoder class.

@JsonIgnore

TODO