SAJ

Version 1.2.0

Developer Guide

retro della copertina

Table of Contents

1 Preface....2
1.1 Introduction....2
1.2 Overview....2
1.3Numeric consideration....2
1.4Event consideration....2
2 API Overview....3
2.1Consts....3
2.2ContentHandler....3
2.3ErrorHandler....4
2.4JSONReader....4
2.5SAJReaderFactory....4

1 Preface

1.1 Introduction

SAJ “Simple Api for Json” is an API to read a json stream. Enable to read massive json stream without overload the memory. This library is similar to SAX for XML.

Other json parser transform the json stream in java classes that represent JSON immediately overloading system resources.

1.2 Overview

SAJ supply an API to read json stream in an event system. Using callback interfaces advices for event in the json stream.

Exist an API to generalize the requisite of the reading and callback function and an SPI to implement a default json parser.

The implementation can be loaded using factory classes to make transparent the implementation.

1.3 Numeric consideration

SAJ would map a json number with java.math.BigDecimal. Being that json specification see only number idea (and not int, float, long, ecc...) saj use the best java implementation of java.lang.Number to avoid error in round or truncate.

1.4 Event consideration

In SAJ a json event is :

Event	note
Start object	When a { occur
End Object	When a } occur
Start array	When a [occur
End array	When a] occur
Attribute	When the parser finds a attribute name

Start json document	When a parser starts to read a json stream
End json document	When a parser ends to read a document
String	When a string value is found how attribute value
Boolean	When a boolean value is found how attribute value
Number	When a number value is found how attribute value
Null	When a null value is found how attribute value

The parser implementation must advise for event using an implementation of it.dangelo.saj.ContentHandler

2 API Overview

This chapter show API classes and there responsability. Below a class diagram that show classes and dependencies:

TODO : Draw class diagram

The API package is it.dangelo.saj and below the description of classes.

2.1 Consts

Contains JSON constants. This class can be used by a parser implementation.

2.2 ContentHandler

it.dangelo.saj.ContentHandler is the callback interface that a parser implementation must call when a json event occours.

Below a mapping of events with method:

Event	Method call
Start object	`void startObject() throws SAJException;`

End Object	`void endObject() throws SAJException;`
Start array	`void startArray() throws SAJException;`
End array	`void endArray() throws SAJException;`
Attribute	`void attribute(String name) throws SAJException;`
Start json document	`void startJSON() throws SAJException;`
End json document	`endJSON()void endJSON() throws SAJException;`
String	`void _string(String value) throws SAJException;`
Boolean	`void _boolean(boolean value) throws SAJException;`
Number	`void _number(BigDecimal value) throws SAJException;`
Null	`void _null() throws SAJException;`

2.3 ErrorHandler

This interface is used when an parser error occours. Tree methods are called:

Method name	note
error	Called when a recoverable error occurs. This occur when a number or a string can't be decoded
fatalError	Called when a parser error occurs.
warning	Not used now.

2.4 JSONReader

This interface represents the json parser. An implementation “vendor” must be implements this interface to supply parser services.

2.5 SAJReaderFactory

Factory class to retrieve and config an implementation of SAJReaderFactory and JSONReader.

This abstract class retrieve an implementation factory that must extend SAJReaderFactory using the FactoryFinder subproject (please see the FactoryFinder documentation to know how the system find a factory)

3 Schema support

SAJ supports schema validation. The json schema is in proposal state so the schema support implementation (in this time) doesn't comply directly to json schema proposal (http://www.json.com/json-schema-proposal/).

Below there are implementation's differences about proposal.

schema attribute names	To remove collision between schema attributes and object attributes has been added a namespace (js => json schema)to the schema attribute. Example js:type. All attribute in json schema that starts with js: are considered how json schema attributes.
removed format attribute	Format attribute is not used in the system. All format are managed with type attribute
transient, and readonly attributes don't used	These attributes doesn't used for validation but will use when the schema will be used to generate java classes (other project)
Replace length attribute with minLength and maxLength	Only length attribute is limiting. The length attribute is used with minLength and maxLength with same values.

Modified the management of array type	To manage array type must be added the type attribute to “array”.
	Items attribute has been replaced with itemsType and are added maxItems and minItems to check the array size.
	Example:
	`"myarray" : {`
	`  "js:type" : "array",`
	`  "js:itemsType" : {`
	`  "name" : {"js:type" : "string"}`
	`  },`
	`  "js:minItems" : 3,`
	`  "js:maxItems" : 5`
	`}`

Added maxDecimal attribute	Added max decimal check for float attribute type
Added date and datetype types	Added date (yyyy-mm-dd) and datetime (yyyy-mm-ddThh:min:ssZ[zone]) types.

Added the referenceSchema attribute	This attribute is the schema name. The schema name is used (not in these release) to be referenced in the json stream using the $ref attribute. Given that SAJ can be used in “not web” environment this name can be (for now) an unique string. Using the interfaces, the developer, can register more schemas that will be used by validator. A schema that don't have the schema reference will be register how default schema used in json stream that don't have the schema directive. Schemas with the same name will be overwritten.

3.1 Using the schema validation

In this release the validation can be done only with external schema.

To use a schema the developer must be compile the schema using the SchemaBuilderFactor class. A schema file can contain an array of schemas and the system will return a Schema[] when parser is called. If only a schema is present in the file (not how array) the system return a Schema[] with one element.

```
InputStream stream =
this.getClass().getClassLoader().getResourceAsStream("schema/schema1.json");
SchemaBuilderFactory factory = SchemaBuilderFactory.newInstance();
Schema[] schemas = factory.parse(stream);
```

This schemas are compiled in memory and can be used for any stream and json parser.

Now, schemas must be register in the json parser:

```
SAJReaderFactory readerFactory = SAJReaderFactory.newInstance();
1) readerFactory.setValidating(true);
2) readerFactory.registerSchema(schemas[0]);
JSONReader reader = readerFactory.newJSONReader();
reader.parse(stream);
```

with 1) row the validation directive will be to true. If this directive is not setted to true the schema validation don't start.

With 2) one o more schemas are register in the parser. In these release only a default schema is supported. If validating directive is true and no schema are registered an error occurs.

Example of json schema file:

```
[
{
          "name": {"js:type":"string"},
          "surname": {"js:type":"string"},
          "data" : {
                    "js:type" : "array",
                    "js:maxItems": 3,
                    "js:minItems":3,
                    "js:itemsType": { "js:type" : "integer" }
          },
          "obj2" : {
                    "js:type" : {
                              "list" : {
                                        "js:type" : "array",
                                        "js:minItems":1,
                                        "js:itemsType": { "js:type" : "float"
}
                              }
                    }
          },
          "array" : {
                    "js:type" : "array",
                    "js:itemsType" : {
                                        "nome" : {"js:type" : "string"}
                              },
                    "js:minItems" : 3,
                    "js:maxItems" : 15
          },
          "js:final" : true
}
,
{
"js:schemaReference" : "persona",
"nome" : {"js:type" : "string"}
}
]
```

3.2 Schema support limitation

This release is a irst release and below the limit of the release are listed:

Schema reference	The schema reference in json stream isn't supporting. Only schema validation with default schema is supported how written in 3.1Using the schema validation
unique attribute	The unique function is not supported

type	New types are in development status (time, gYearDay, gMonthDay , gDay,gMonth, gYear, uri, base64 and more).
datetime format	The format of date and datetime will be correct.