

2003/2 JKVPF

Holger Zahnleiter
holger@zahnleiter.org

Date: 07.04.03

Abstract

Java Key/Value Parsing Framework is a class library allowing to declare a list of key/value pairs one expects to be passed as an argument. Then, the parser parses the input and verifies that no unallowd keys are passed. Additionally the values are getting passed to mapper objects. These validate the values and transform them to an appropriate object. In most cases this will be a string but might also be an integer or date object.

1 Project characteristics

Project code	2003/2
Project name	JKVPF
Started	01.04.03
Ended	03.04.03
Version/release	1/1
Used tools and libraries	JDK 1.4.1 (sould also work with older versions of JDK); JUtil (my Java library, available on my homepage);
Runs on platform	Any system with a JVM

2 Release notes

- **Version 1, release 1**

- Checks for uniqueness of successors in key tracker network.
- Checks for cycles in key tracker network. Cycles are not allowed.
- Checks for uniqueness of key per path. Each key must appear only once along the path from the root to the leafes.
- Provides basic mappers for string, integer, date, Java identifier and string enumerations.
- Parser can return a BNF definition for the key/value expressions it will accept (depending on the current tracking network).

3 Preface

This document will only explain the functional principle behind the framework. For more details refer to the Javadoc available on my homepage. Also read the Javaworld article "Mapping XML to Java".

4 The parser

Purpose of the parser is to consume the symbols from the input (key/value expressions) and use the network to make sure that the expression is compliant with the grammar defined by that network. Key/value expressions look like this: `key1=value1; key2=value2; ...; keyN=valueN`.

The parser is represented by the class `KeyValueParser`. Basically, it is an engine which gets controlled by the tracking network, which was passed to the parser at construction time. Additionally, the parser uses mapper objects to transform the values (string) into objects. At the same time these mappers also check the syntax of the values. Parsing stops when all symbols are consumed or in case of error. The result is a map object (`HashMap`) which contains all the value objects. These are accessible by their name (key).

There is another function of the parser, which shall not stay unmentioned. The method `getBNF()` returns a BNF definition for the key/value expressions accepted by the parser, depending on the underlying tracking network. This is meant for automated generation of usage messages.

5 The trackers

A tracker (`KeyValueTracker`) basically represents a key, therefore it contains a string to hold the key's name. Additionally it contains a list of keys which might follow this key in the key/value expression. Any of the keys in this successor list is allowed to appear in the input after this key (options). Keys can also be optional, they might or might not appear in the input. Third, it contains a mapper, which maps the value (string) of the key to whatever object.

To find out if the current key, taken from the input, is valid, the parser just needs to compare it with the key held by the tracker. If both are identical, then the key is valid. An "unexpected key" exception will be thrown otherwise.

After a key was found to be valid, the parser consumes the value. Then the value is passed to the mapper. The input is always a string of characters. But the mappers introduce a chance to create objects from these values. A key might, for example, represent a date. Then, you would choose the `DateValueMapper`, which creates `java.util.Date` objects. But the mapper can also check the values for validity. Invalid dates, integer values etc. would then throw a parsing exception.

6 The mappers

All mappers have to implement the interface `ValueMapper`. The method `map()` has to be implemented in order to create an `java.lang.Object` from the given value `java.lang.String`. A number of basic mappers is shipped with the framework:

Name	Function
<code>BooleanValueMapper</code>	Accepts the strings 'true' and 'false' and creates a <code>java.lang.Boolean</code> object with corresponding value.
<code>DateValueMapper</code>	Accepts date strings and creates a <code>java.util.Date</code> object.
<code>EnumerationValueMapper</code>	At construction time a list of strings gets set. It accepts all strings from the input, which are in this list and returns a corresponding <code>java.lang.String</code> . This is meant for cases where you explicitly want to allow a key to have some particular values only.
<code>IntegerMapper</code>	Accepts strings representing an integer value. Returns a <code>java.lang.Integer</code> .
<code>JavaIdentifierValueMapper</code>	Accepts and returns all strings, which are compliant with the rules for Java identifier names.
<code>NonEmptyStringValueMapper</code>	Accepts and returns only none-empty strings.
<code>StringValueMapper</code>	Accepts and returns any string.

Table 2: Basic mappers

You can, of course, derive new mappers, which, for example, accept only (syntactically) valid IP addresses or URLs and which return `java.net.URL` objects.

7 Example: simple list of key/value pairs

A very simple case would be this. Imagine you want to pass some parameters to one of your programs. This could be a string like this: `name=<a string>; password=<a string>`. Set up a tracking network like shown below:

```
KeyValueTracker pwd = new KeyValueTracker(
    "password", //name of the password key
    true, //this key is mandatory
    new NonEmptyStringValueMapper(), //don't allow empty values
); //this key doesn't have any successors

KeyValueTracker usr = new KeyValueTracker(
    "user", //name of the user key
    true, //this key is mandatory
    new NonEmptyStringValueMapper(), //don't allow empty values
    pwd //this key is followed by "password"
);

KeyValueTracker root = new KeyValueTracker(
    null, //root must not have a name
    true, //don't care
    null, //root must not have a mapper
    usr //the first key is "user"
);
```

Now that the network is set up you can continue and pass it to the parser and parse an expression:

```
KeyValueParser parser = new KeyValueParser( root );
HashMap valuePairs = parser.parse( "user=guest; password=xyz123" );
```

This piece of code

```
System.out.println( valuePairs.get( "user" ) );
System.out.println( valuePairs.get( "password" ) );
```

will print the following two lines

```
guest
xyz123
```

8 Example: options

You might want to have a choice, whether to take this key or another key. Therefore a key must have more than one successors. Imagine, you would like to accept either `age=<an integer>` or `birthday=<a date>`. Therefore set up a tracker network like this:

```
KeyValueTracker age = new KeyValueTracker(
    "age", //name of the key
    true, //this key is mandatory
    new IntegerValueMapper(), //accept integers only
); //this key doesn't have any successors

KeyValueTracker birth = new KeyValueTracker(
    "birthday", //name of the key
    true, //this key is mandatory
    new DateValueMapper(), //accept dates only
); //this key doesn't have any successors

KeyValueTracker root = new KeyValueTracker(
    null, //root must not have a name
    true, //don't care
    null, //root must not have a mapper
    new KeyValueTracker[]{age, birth} //the first and only key is "age" or "birth"
);
```

Now that the network is set up you can continue and pass it to the parser and parse an expression:

```
KeyValueParser parser = new KeyValueParser( root );
HashMap valuePairs = parser.parse( "age=32" );
```

This piece of code

```
System.out.println( valuePairs.get( "age" ).toString() ); //It's a java.lang.Integer!!!
```

will print the following line

```
32
```

Please note: since both keys are marked as compulsory, an exception will be thrown in case none of the keys are contained in the input.

9 Example: enumerations

This example demonstrates how you can restrict a key to a limited set of possible values by using the `EnumerationValueMapper`.

```
KeyValueTracker md = new KeyValueTracker(  
    "mode", //name of the key  
    true, //this key is mandatory  
    new EnumerationValueMapper(  
        new String[]{"verbose", "diagnostic", "silent"}  
    ), //accept these values only  
); //this key doesn't have any successors
```

```
KeyValueTracker root = new KeyValueTracker(  
    null, //root must not have a name  
    true, //don't care  
    null, //root must not have a mapper  
    md //the first and only key is "mode"  
);
```

Make use of the network and parse a expression:

```
KeyValueParser parser = new KeyValueParser( root );  
HashMap valuePairs = parser.parse( "mode=verbose" );
```

This

```
System.out.println( valuePairs.get( "mode" ) ); //It's a java.lang.String
```

prints the following

```
verbose
```

A List of project files

File Name	Comments
bin\jkrpf.jar	Java archive.
build.xml	Ant build file.
src\org\zahnleiter\jkrpf\BooleanValueMapper.java	Maps string values to Integer objects.
src\org\zahnleiter\jkrpf\DateValueMapper.java	Maps string values to Date objects.
...\zahnleiter\jkrpf\EnumerationValueMapper.java	Maps string values to String objects.
src\org\zahnleiter\jkrpf\IntegerMapper.java	Maps string values to Integer objects.
...hnleiter\jkrpf\JavaIdentifierValueMapper.java	Maps string values to String objects.
src\org\zahnleiter\jkrpf\KeyValueParser.java	Parses key/value expressions.
src\org\zahnleiter\jkrpf\KeyValueTracker.java	Used for creating key tracking networks (declarative style).
...nleiter\jkrpf\NoneEmptyStringValueMapper.java	Maps string values to String objects.
src\org\zahnleiter\jkrpf\StringValueMapper.java	Maps string values to String objects.
src\org\zahnleiter\jkrpf\ValueMapper.java	Abstract mapper (interface).

Table 3: Project files

B Related web pages

[W1] **Mapping XML to Java - Part 1**
www.javaworld.com/javaworld/jw-08-2000/jw-0804-sax.html

Excelent article about SAX-parsing an XML document and mapping it to Java objects by using a tag tracking network.

[W2] **Mapping XML to Java - Part 2**
www.javaworld.com/javaworld/jw-10-2000/jw-1006-sax.html

Excelent article about SAX-parsing an XML document and mapping it to Java objects by using a tag tracking network.

Disclaimer

The use of my page's content (programs, wiring diagrams, pictures, documents) is free for non-commercial purposes only.

The information in this document has been carefully reviewed and is believed to be reliable, but I do not assume any liability arising out of the application or use of any documents, programs or circuit described herein.

Furthermore I want to declare that I'm not responsible in any way for the content of other web pages, books and other sources I'm referring to.