Java SE utils

Rudy De Busscher

Version 0.9.0, 04/02/2018

Table of Contents

Release notes

0.9.0

- Initial version with classes from various sources usable in any environment.

Setup

Add the following artifact to your maven project file

```
<dependency>
    <groupId>be.atbash.utils</groupId>
    <artifactId>utils-se</artifactId>
    <version>0.9.0</version>
</dependency>
```

Requirements

Java 7+

Features

Utilities compiled for Java 7 and no dependencies.

The most important methods are described here. For the full list of methods, see the Javadoc or the source code.

BASE64

Can be used to do encoding and decoding to BASE64 for Java 7 (Only Java 8 has a JVM native implementation)

```
be.atbash.util.base64.Base64Codec#encodeToString(byte[], boolean)
```

Converts a byte array to a BASE64 encoded String.

```
be.atbash.util.base64.Base64Codec#encodeToString(byte[], boolean)
```

Converts a byte array to a BASE64 encoded String. The boolean defines if urlSafe characters need to be used or not.

```
byte[] be.atbash.util.base64.Base64Codec#decode(String)
```

Decodes a BASE64 String into a byte array.

```
be.atbash.util.base64.Base64Codec#isBase64Encoded(String)
```

Verifies if the String is a valid Base64 value.

Instantiations

The dynamic instantiation of classes is important when you define the class name within configuration values.

With the **ClassUtils** utility you can verify if the class name effectively exists and instantiate it with some arguments.

The classes, but also the resources, are searched in the following order

1. context classloader attached to the current thread
2. classloader who has loaded the ClassUtils class
3. system class loader

```
be.atbash.util.reflection.ClassUtils#isAvailable(String)
```

Verifies if the class defined by its FQCN (a fully qualified class name which is package name and class name) is found by one of the 3 class loaders.

```
be.atbash.util.reflection.ClassUtils#newInstance(String)
be.atbash.util.reflection.ClassUtils#newInstance(Class)
```

Creates an instance of the class (specified by the FQCN or the class instance) using the no-args constructor. When such a constructor is not available or there was an *Exception* thrown during the instantiation of the class, an **be.atbash.util.reflection.InstantiationException** is thrown.

```
be.atbash.util.reflection.ClassUtils#newInstance(String, Object...)
be.atbash.util.reflection.ClassUtils#newInstance(Class, Object...)
```

Creates an instance of the class (specified by the FQCN or the class instance) using a Constructor which matches the arguments.

The *Constructor* which will be used to instantiate the class is not determined by the *Class.getConstructor(argTypes)* method as it doesn't work when one of the arguments is *null*. The following algorithm is used to find the *Constructor*.

1. Loop over all *Constructor*s
2. Consider a *Constructor* when it has the same number of arguments
3. Check if the argument types have the same class (using *equals*) as the parameter type. When the argument is *null*, it is considered as a match.
4. When no *Constructor* is found, all *Constructor*s with the correct number of arguments is verified again but now a less strict match is used (using *isAssignableFrom* to allow subtypes)
5. When there is not exactly 1 Constructor found, an **be.atbash.util.reflection.NoConstructorFoundException** is thrown.

When an *Exception* is thrown during the instantiation of the class, an **be.atbash.util.reflection.InstantiationException** is thrown.

```
be.atbash.util.reflection.ClassUtils#getResourceAsStream(String)
```

Returns the resource using the 3 class loaders as described above.

Reading version

With the class **be.atbash.util.version.VersionReader**, you can read the version information stored within the *META-INF/MANIFEST.MF* file.

Define the version information by configuring the *maven-jar-plugin* or *maven-war-plugin* in the maven build section.

```
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>2.5</version>
        <executions>
            <execution>
                <id>manifest</id>
                <goals>
                    <goal>jar</goal>
                </goals>
            </execution>
        </executions>
        <configuration>
            <archive>
                <manifestEntries>
                    <Release-Version>${project.parent.version}</Release-Version>
                    <buildTime>${maven.build.timestamp}</buildTime>
                </manifestEntries>
            </archive>
        </configuration>
    </plugin>
```

This information can be read by using the following snippet

```
VersionReader versionReader = new versionReader("atbash-config");
versionReader.getReleaseVersion();
versionReader.getBuildTime();
```

The constructor argument is the artifact from which we want to read this information (actually it is the first part of the name of the jar file but these are in most cases the same).

Base exceptions

There are 3 exception classes defined which can be handy in all applications.

- be.atbash.util.exception.AtbashException

This is a *RuntimeException* used as a parent class for all Atbash defined exceptions. It makes it possible to define a generic Exception handler (within JSF or JAX-RS) to handle all the Exceptions uniformly (logging, showing info to end user, ...)

- be.atbash.util.exception.AtbashIllegalActionException

This exception is thrown when the Atbash code detects a wrong usage of the framework by the developer. An example is a usage of a non-existing URL filter name in the Octopus framework (maybe a typo).

It is recommended that the error message starts with a code (like *(OCT_DEV_001)*) and the documentation describes then the situation and what actually is done wrong and how it can be fixed.

- be.atbash.util.exception.AtbashUnexpectedException

Can be used to convert a checked exception (like an IOException) into an *AtbashException* so that it can be handled by the general exception handler. Most checked exceptions never occur during the execution of the application, but they need to be caught or thrown.

String utils

```
be.atbash.util.StringUtils.hasText(String)
be.atbash.util.StringUtils.isEmpty(String)
```

Verifies if the String contains something meaning full (something different then whitespace) or not.

When the argument is *null*, empty String (“”) or contains only whitespace (” ”) it is considered as empty.

```
be.atbash.util.StringUtils.hasLength(String)
```

Verifies if the String contains characters or not but handles null as the empty String. Whitespace characters are counted as a real character.

```
be.atbash.util.StringUtils.clean(String)
```

Cleans the argument, this are the rules

Argument	**Result**
null	null
""	null
other cases	.trim()

```
be.atbash.util.StringUtils.startsWithIgnoreCase(String, String)
```

Verifies if the String starts with a certain prefix, case insensitive. Method handles correctly the situation where one or both arguments are *null*.

```
be.atbash.util.StringUtils.split(String)
```

Break down the String within items, delimited by , by default (there exist an overloaded method to define also the delimiter. You can use " to define the start and end of an item. The following example has thus only 2 items

```
key , "value1,value2"
```

The quotes are removed and the item is trimmed before the placed in the return array.

Collection utils

```
be.atbash.util.CollectionUtils.asSet(E...)
be.atbash.util.CollectionUtils.asList(E...)
```

Returns the items specified in the argument as *Set* or *List* respectively.

```
be.atbash.util.CollectionUtils.isEmpty(Collection)
be.atbash.util.CollectionUtils.isEmpty(Map)
```

Verifies if the argument is null or contains no elements.

```
be.atbash.util.CollectionUtils.size(Collection)
be.atbash.util.CollectionUtils.size(Map)
```

Returns the size of the *Collection* or *Map* but handles null argument correctly.