Table of Contents

Im Laufe jeder Software Entwicklung, egal ob es nun eine Applikation oder ein Service ist, werden Funktionen benötigt, die dem Entwickler die Arbeit erleichtern. Damit die Funktionen wiederverwendet werden können lagert man diese üblicherweise in Utility oder Helper Klassen aus.

Auch in JVx wird die eine oder andere Utility Klasse verwendet. Diese wollen wir niemanden vorenthalten. Sämtliche Utility Klassen sind im Package com.sibvisions.util zu finden.

XML Bearbeitung

Um XML Dateien einzulesen ist ein XML Parser erforderlich. Es existieren bereits einige Frameworks mit denen XML Dateien gelesen und auch wieder gespeichert werden können. Doch der Zugriff auf die einzelnen Elemente ist meist mühsam über Objektbäume gelöst.

In JVx sind die Klassen XmlWorker und XmlNode enthalten, mit denen direkt auf Tags zugegriffen werden kann. Außerdem wird auf Attribute wie auf Sub Tags zugegriffen. Die Performance und der Speicherverbrauch ist ebenfalls sehr gering.

Anhand eines kurzen Beispiels lesen wir eine XML Datei ein, verändern einen Sub Tag und speichern das Ergebnis als neue XML Datei:

```
XmlWorker xmw = new XmlWorker();

XmlNode xmnRead = xmw.read(new FileInputStream("example.xml"));

xmnRead.setNode("/archive/element(1)/user", "xml");

xmw.write(new FileOutputStream("example_v2.xml"), xmnRead);
```

Weitere Details über den Umgang mit XML Daten.

Arbeiten mit Arrays

Array Operationen sind immer mühsame Codier Arbeit und jeder Entwickler könnte gut und gerne darauf verzichten. Ein einzelnes Element aus einem Array zu entfernen ist zwar keine Hexerei aber eine Standard Funktion ist dafür nicht vorhanden. Und wer würde nicht gerne ein Array nach einem Element durchsuchen ohne eine Schleife zu codieren?

Mit dem ArrayUtil wurden wesentliche Array Operationen implementiert und außerdem implementiert es das List Interface vollständig. Einen Performance Gewinn gegenüber Standard Listen erhält man als kleines Extra.

Der Objekt Cache

Ab und zu müssen Objekte zwischengespeichert und zwischen verschiedenen Instanzen ausgetauscht werden. Dazu bieten sich Parameter und public Methoden nahezu an. Doch nicht immer möchte man ein Objekt über unzählige Instanzen zum Zielobjekt durchschleusen. Und nicht immer befinden sich die benötigten Objekte im selben Scope.

Dafür existiert der ObjectCache. Dieser verwaltet Objekte statisch, für bestimmte Zeiträume oder eben dauerhaft, innerhalb einer VM.

Zum Vergleich stellt man sich eine statische Hashtable vor, ergänzt mit einer Ablaufzeit.

Geordnete Hashtable

Die Standard Hashtable verwaltet ihre Keys in einer undefinierten Reihenfolge. Manchmal wäre es sehr nützlich wenn die Reihenfolge des hinzufügens erhalten bleiben würde.

Das übernimmt die OrderedHashtable.

Hashtable mit automatischer Listenverwaltung

Die Standard Hashtable verwaltet pro Key, genau einen Value. Häufig kommt es vor, daß pro Key mehrere Values verwaltet werden müssen. Also muss man eine Liste als Value verwenden. Darin werden dann die tatsächlichen Values gespeichert. Das ist zwar ebenfalls keine Hexerei, aber eine sehr langweilige Aufgabe.

Die KeyValueList übernimmt die Listen Verwaltung und der Entwickler hat Zeit für wichtigeres.

Reflective

Mit der Reflective Klasse können Sie Instanzen erstellen oder Methoden aufrufen mittels Reflection, aber direkt `java.lang.reflect` zu verwenden.

Weitere details, siehe hier.

Weitere Klassen

Es gibt noch viele weitere interessante Utility Klassen wie z.B.: StringUtil, BeanUtil, DateUtil, FileUtil, ImageUtil, ...

Genauere Information darüber sind in der javadoc zu finden.