Introduction to Utils

Maik Schmidt

23. September 2009

Inhaltsverzeichnis

Dokumenthistorie

Dieses Dokument wird fortlaufend gepflegt. Die nachfolgende Tabelle gibt eine Übersicht über die Änderungen in einzelnen Versionen.

Datum	Änderungen
07.09.09	Gliederung
08.09.09	Inhalt
14.09.09	Angleichen v. Konzept und Realität
22.09.09	Angleichen v. Konzept und Realität, komplett überarbeitet

Geplante Änderungen

Folgende Änderungen sind bereits absehbar, wurden im aktuellen Dokument berücksichtigt aber noch nicht umgesetzt:

- Teilweise Überarbeitung und weitere Dokumentation!

1 Einführung

Die Bibliothek „org.sidiff.common“ fasst allgemeine, in vielen Kontexten wiederverwendbare Funktionen zusammen. Sie bildet daher die Grundlage aller SiDiff-Module und Applikationen. Die Bibliothek ist hybrid ausgelegt und kann als OSGI-Bundle oder als Bibliothek im Classpath der nutzenden Applikation eingebunden werden. [1] Die OSGI spezifischen Teile werden durch den Aktivator *org.sidiff.common.Activator* aktiviert.

1.1 Klassen von Utilities

Diese Bibliothek adressiert verschiedene Anwendungsgebiete. Die Hilfsfunktionen verteilen sich dementsprechend auf die folgenden Klassen/Pakete.

- org.sidiff.common.app (Bau von Applikationen)
- org.sidiff.common.collections (Ver-/Bearbeiten von Java-Collections)
- org.sidiff.common.crypto (Kryptographie und Lizenzierung)
- org.sidiff.common.exceptions (Einheitliche Ausnahmebehandlung)
- org.sidiff.common.io (Daten Ein-/Aus-/Übergabe)
- org.sidiff.common.lcs (Allgemeine lcs-Implementierung)
- org.sidiff.common.logging (Einheitlicher Protokoll-Mechanismus)
- org.sidiff.common.xml (XML Verarbeitung und Transformation)
- org.sidiff.common.util (Enthält nicht weiter gruppierbare Hilfsfunktionen)

Die Pakete enthalten zum Teil Klassen, die lediglich statische Funktionen bereitstellen. Diese sind mit dem Suffix „Util“ gekennzeichnet. Instanziierbare Klassen besitzen diesen Suffix nicht.

1.2 Abhängigkeiten

Die Apache Bibliothek XERCES muss zur Laufzeit als Bibliothek im Classpath oder aber als entsprechendes Bundle (org.apache.xerces) verfügbar sein. Optional werden org.osgi.service.log (org.eclipse.osgi.services) und org.osgi.framework (org.eclipse.osgi) genutzt.

[1] Je nach Kontext werden beispielsweise Meldungen an einen ggf. vorhandenen OSGI Logger oder aber an die Konsole übergeben

2 Application

In diesem Paket werden Hilfsfunktionen, für den Bau von Applikationen angeboten.

2.1 CommandLine

Bitte in der Javadoc nachlesen.

3 Crypto

Dieses Paket umfasst einzelne Klassen und Applikationen zur

- Verschlüsselung von Konfigurationsdaten
- (Teil)Signierung von Konfigurationsdaten
  (Erlaubt gezielte Eingriffe in die Konfiguration in Abhängigkeit der Lizenz)
- Ver- und Entschlüsselung von Datenströmen
  (Schutz von Konfigurationsdateien)

Die hier bereitgestellte Funktionalität wird noch nicht benutzt und wird daher (noch) nicht weiter beschrieben[2].

4 Exceptions

Dieses Paket stellt Klassen zur Ausnahme- (Exceptions) und Fehlerbehandlung (Runtime-Exceptions) bereit. Sie sind projektweit als Basisklassen für eigene Exceptions zu verwenden um ein einheitliches Verhalten zu gewährleisten. So implementieren die bereitgestellten Exceptions gewisse Automatismen, wie beispielsweise eine automatische Fehler-Protokollierung. Ferner werden aber auch einige Komfortfunktionen, wie das Kapseln von inneren Exceptions bereitgestellt. Es gibt zwei zu verwendende Basisklassen:

- SiDiffException
- SiDiffRuntimeException

5 IO

Die zu verarbeitenden Daten, Konfigurationen und Rückgabe-Kanäle werden häufig [3] in Form von Datenströmen übergeben. Um die Handhabung von Datenstömen zu vereinfachen und zu vereinheitlichen[4] bietet dieses Paket unterschiedliche Funktionen an.

[2] **Überarbeitung notwendig!**
[3] Auch zwischen den Modulen von SiDiff
[4] Zugriff erfolgt einheitlich, unabhängig von der Quellen bzw. Senke (Dateien, Resource, Jar oder Bundle, Strings für Zwischenergebnisse)

- IOUtil
  Allgemeine IO Funktionen um Datenstöme anzufordern.
- MapUtil
  Zugriff auf persistente, getypte Maps.
- ResourceUtil
  Zugriff auf Ressourcen und Classloaderverwaltung.
- Serializer
  Allgemeiner Serialisierer (Schnittstelle)

5.1 Map

An vielen Stellen werden Maps benötigt um Komponenten zu konfigurieren oder zu steuern. Die IO-Utils stellen daher einen Mechanismus für typsichere, persistente Maps bereit. Der Inhalt der Maps ist dabei mittels einer XML-Datei folgenden Aufbaus zu spezifizieren[5]:

```
<!ELEMENT Map (MapEntry*)>
<!ATTLIST Map type CDATA #IMPLIED >

<!ELEMENT MapEntry EMPTY>
<!ATTLIST MapEntry
  key CDATA #REQUIRED
  value CDATA #REQUIRED >
```

6 Logging

Das Package „logging“ stellt einen Mechanismus bereit, mit dessen Hilfe unterschiedliche SiDiff-Benutzergruppen über aktuelle Ereignisse innerhalb der Software informiert werden sollen. Die Logausgaben werden bei Bedarf mit Kontextinformationen, wie Datum, Uhrzeit, Aufrufer etc. angereichert und „schön“ formatiert bzw. eingerückt. Die Ausgabe und Formatierung erfolgt dabei entsprechend eines kontextabhängigen Ausgabekanals. Je nach Ausführungskontext können unterschiedliche Kanäle wie Konsole, Datei, OSGI-Logger etc. verwendet werden. Der zu verwendende Ausgabekanal wird entsprechend des Kontextes automatisch bestimmt - kann aber auch manuell gesetzt werden.

6.1 Klassifikation der Logausgabe

Werden Logausgaben übergeben, so müssen diese entsprechend einer passenden Ereignisklasse klassifiziert werden. Folgende Zuordnung ist zu beachten.

[5] Vgl. org.sidiff.common.io.map.dtd

Event	Zielgruppe	Bedeutung
MESSAGE	Endanwender	Allgemeine Mitteilungen über das was gerade passiert. z.B. 'Lade model x'
NOTICE	Endanwender	Hinweis für fortgeschrittenen User. z.B. Abstieg in Subsequenzen etc.
ERROR	Endanwender	Fehlermeldung
WARNING	Entwickler	(Potentielle) Fehler wie 'summe gewichte !=1 oder Zugriff auf Attribut fehlgeschlagen
EVENT	Entwickler	SiDiff interne Ereignisse. z.B. „Mach Created A-B“
SIGNAL	Entwickler	Native Calls etc.
DEBUG	Entwickler	Informationen zum internen Zustand von Komponenten zur Fehlersuche

6.2 Aufruf und Benutzung

An Entwickler gerichtete Nachrichten sind durch ein **assert** zu kapseln. Dies ermöglicht die bedingte Ausführung, was einen Laufzeitvorteil für den Endanwender darstellt.

```
assert(LogUtil.print(LogEvent.WARNING,"Message",exception));
```

Sollen Entwicklernachrichten generiert werden, so muss die Auswertung von Assertions aktiviert werden. Zu diesem Zweck muss der Java-VM **-enableassertions** oder kurz **-ea** übergeben werden.

6.3 Konfiguration

Das Logging kann zum Start einer Applikation über Java-Properties konfiguriert werden. Diese werden der Java-VM wie folgt übergeben:

java -D*PropertyName*=“*PropertyValue*“

Die folgenden Properties können gesetzt werden:

Property	Argument	Bedeutung
LOGCHANNEL	Channelname	Name des zu verwendenden Ausgabekanals
LOGMODULES	Modulnamen	durch „,“ getrennt - Module die geloggt werden sollen. defaultmäßig alle.
LOGEVENTS	Eventnamen	durch „,“ getrennt - Ereignisse die geloggt werden sollen. defaultmäßig „MESSAGE,ERROR“

Als Name ist jeweils auch der Wildcard „*“ erlaubt.

6.4 Ausgabekanäle

Zur Ausgabe von Meldungen sind folgende Kanäle möglich:

Channelname	Bedeutung
ConsoleLogChannel	Meldungen werden auf der Konsole ausgegeben
EclipseLogChannel	Meldungen werden an die Eclipse-Fehlerkonsole geleitet
FileLogChannel	Meldungen werden in eine Datei geschrieben
OSGILogChannel	Meldungen werden an den OSGI-Logservice übergeben
TeePipeChannel	Meldungen an mehrere Channels übergeben (schachtelbar)

6.4.1 FileLogChannel

Der „FileLogChannel“ benötigt den Pfad zur Protokolldatei. Dieser muss durch ein weiteres Property spezifiziert werden.

Property	Argument	Bedeutung
FILENAME	Protokolldatei	Absoluter Pfad im Dateisystem

6.4.2 TeePipeChannel

TODO: Wird z.Z. implementiert.

6.4.3 EclipseLogChannel

TODO: Wird z.Z. implementiert.

7 Collection Utilities

Die Java-Klassenbibliothek stellt vielfältige Containerklassen für unterschiedliche Einsatzzwecke, und mit unterschiedlichen Eigenschaften bereit. Instanzen dieser Klassen werden häufig für den Datenaustausch genutzt. Dies hat oft den Nachteil, dass bei einer schrittweisen Verarbeitung (Filterung, Vereinigung etc.) eine Kopie der Kollektion angelegt werden muss. Erschwerend kommt hinzu, dass häufig nur die ersten Elemente einer solchen n-fachen Kopie letztendlich verarbeitet werden. Daher kann es sinnvoll sein die „Verarbeitungslogik“ in Form von Selektoren zu implementieren und Sichten auf Kollektionen zu definieren.
Ferner tritt immer wieder das Problem auf, dass Objekte einer *Collection* zur Verarbeitung klassifiziert werden müssen. Diese Problemdomäne wird ebenfalls adressiert.
Zusätzlich bietet dieses Paket aber auch spezielle Collection-Klassen mit speziellen Eingenschaften.

7.1 Die Utilities

7.1.1 Selektoren

Beispiel:

```
public static Selector<EObject> byInstance(final EClass eClass) {
  return new Selector<EObject>(){
    public boolean select(EObject item) {
```

```
        return eClass.isInstance(item);
      }
    };
  }
```

DefaultSelectors stellt dabei allgemeine[6] Selektoren bereit (Die an den entsprechenden Stellen zu verwenden sind!).

7.1.2 Klassifikatoren

Beispiel:

```
public final static Classifier<EdgeType, Edge> EDGE_BY_TYPE =
 new CollectionUtil.Classifier<EdgeType, Edge>(){
   public EdgeType classify(Edge item) {
     return item.getEdgeType();
   }
 };
```

7.1.3 ClassificationUtil

Klassifiziert übergebene Kollektionen anhand eines ebenfalls zu übergebenden Klassifikators. Das Ergebnis der Berechnung ist eine (u.U. sortierte) Map, die als Werte Listen oder Mengen enthält.

7.1.4 FilterUtil

Filtern Kollektionen gemäß übergebener Selektoren. Dabei wird eine **modifizierbare Kopie** der übergebenen Kollektion erzeugt.

7.1.5 ViewUtil

Erzeugen Sichten auf die übergebenen Kollektionen. Z.B. Vereinigung, Filterung etc. Es liegt in der Natur der Sache, dass die erzeugte Sicht **„Read-Only“** ist.

7.1.6 SetUtil

Typische Mengenoperationen (Vereinigung, Durchschnitt, Differenz) auf Sets. Dabei wird eine **modifizierbare Kopie** der übergebenen Mengen berechnet.

7.1.7 CollectionUtil

Test und Konvertierung von Kollektionen. Z.B. ob zwei Kollektionen die gleichen Elemente enthalten...

[6] Alles, Nichts, Teilmenge einer Aufzählung

7.2 Spezielle Collections

7.2.1 MapWithDefault

Diese Map erweitert (Decorator Pattern) eine gegebene Map um die Möglichkeit einen Defaultwert für nicht vorhandene Schlüssel zu setzen.

7.2.2 CrossMap

Bei dieser Klasse handelt es sich um eine spezielle Implementierung einer Map, die zu jedem Wert auch den entsprechenden Schlüssel effizient bestimmen kann.

7.2.3 ValueMap

Speichert zu einem Schlüssel eine Menge von Werten. Für jeden in der Map enthaltenen Wert besteht eine effiziente Möglichkeit den entsprechenden Schlüssel abzufragen.

8 lcs

Generische Implementiertung einer LCS auf beliebigen Sequenzen

9 xml

Dieses Paket bietet Funtionen zur Verarbeitung und zur Manipulation von XML-Dokumenten, insbesondere zum

- Laden und Speichern von XML-Dokumenten (mit DOM/SAX+eigenem XML-Writer)
- Auswerten von XPath
- Validieren von XML-Dokumenten (gegenüber lokal vorliegenden DTDs durch Mapping mittels Resolver)

Die Funktionen sind wie folgt zugeordnet:

Klasse	Technologieeinordnung	Funktion/Einsatzkontext
XMLParser	SAX/DOM	Scanner zur Verwendung mit einem ContentHandler Parser zur Erzeugung eines DOM Baumes.
XMLDOMCreator	DOM	Erzeugen und serialisieren eines XML-DOM
XMLWriter	Eigenentwicklung	Programmatische Erzeugung von XML Dokumenten
XMLTransformer	XSLT	XSLT Transformation von XML Dokumenten
XMLResolver	Global	Verwaltet Mapping zwischen DTD-Deklarationen und internen Ressourcen fest. (zur Validierung)

10 Utilities

In diesem Paket sind Hilfsklassen und Funktionen zusammengefasst, die keine größere Gruppe von Funktionen bilden. Funktionen werden dabei i.d.R. durch einzelne Klassen angeboten.

10.1 StringUtil

Wandelt übergebene Objekte in eine menschenlesbare Stringrepräsentation um. Dies ist insbesondere für die Synthese von Debug-Ausgaben[7] nützlich, die anschließend protokolliert werden können. Die Art und Weise dieser Serialisierung wird durch „StringResolver“ definiert, die kontextabhängig konfiguriert werden können!

10.2 ReflectionUtil

Kapselt die Java-Reflection API und bietet komfortable (korrekt getypte) Zugriffsmethoden. Da intern das ResourceUtil verwendet wird, ist die Quelle der Klasse transparent[8].

10.3 StatisticUtil

Ermöglicht es Statistiken anzulegen[9]. Speziell

- Laufzeitmessungen zwischen Aufrufen (Start/Stop Timer)
- Zählen von Aufrufen

10.4 SiDiffUIUtil

Adhoc GUI für schnelle Prototypen. Öffnet Fenster mit entsprechendem Text oder auch interpretiertem HTML.

[7] Der innere Zustand von Objekten kann formalisiert werden
[8] Ermöglicht das reflektive Instanziieren von Klassen in fremden Bundles oder Fragmenten!
[9] Müsste mal überarbeitet werden