10. OLAPLINE-Anwendertreffen

26. und 27. April 2017
Schloss Garath | Düsseldorf

Zwei Tage Weiterbildung und Networking integriert:
Vorträge, Workshops und Erfahrungsaustausch rund um TM1

OLAPLINE GmbH
+49 211 9708 140
www.olapline.de

Hauptsitz Düsseldorf
Garather Schlossallee 19
40595 Düsseldorf

Geschäftsstelle Berlin
Albert-Einstein-Str. 14
12489 Berlin

Niederlassung Spremberg
Dresdener Str. 30
03130 Spremberg

JAVA EXTENSIONS und TI

Ein Vortrag von Christoph Pingel

OLAPLINE GmbH
+49 211 9708 140
www.olapline.de

Hauptsitz Düsseldorf
Garather Schlossallee 19
40595 Düsseldorf

Geschäftsstelle Berlin
Albert-Einstein-Str. 14
12489 Berlin

Niederlassung Spremberg
Dresdener Str. 30
03130 Spremberg

TI und JAVA

Datentypen

TI: Zeichenkette (String), Numerisch (Zahl)

JAVA: Objekte und atomare Typen

Vergleich

TI		JAVA	
numerisch	⇔	double/long	(atomar)
Zeichenkette	⇔	String	(Objekt)

Weshalb JAVA ?

→ Nutzen von vorhandenen Bibliotheken, Frameworks, Klassen
Netzwerkkommunikation und andere Schnittstellen vorhanden

→ einfachere Realisierung von komplexen Algorithmen/Hilfsstrukturen

→ Möglichkeit der dynamischen Parameterverwendung,
Überladen von Methoden

→ Abgeschlossene Module (Pakete) für spezielle Aufgaben/Funktionen
TI Code übersichtlicher/nachvollziehbar
Wiederverwendbarkeit
Reduktion von Fehlern bei Implementierung

Weshalb JAVA ?

→ Platformunabhängigkeit

→ Vorhandenes Wissen nutzen

→ Sicherheit / Datenschutz für Prozess-Code

→ bessere Dokumentation, Community

Gibt es Nachteile ?

→ längere Verarbeitungszeit als bei TI-Funktionsaufrufen

→ zusätzliche Konfiguration

→ Sicherheitsaspekt / Angriffsvektor

→ weiteres Expertenwissen für Erweiterungen / Fehlersuche / Wartung nötig

Konfiguration TM1

→ TM1 Installation: allgemeine JAVA-policy für TM1

[path to TM1]\tm1_64\configuration\javaextensions.policy

```
grant {
        permission java.security.AllPermission;
};
```

→ TM1 Datenbank-Konfiguration: separate Freischaltung der integrierten JAVAVM

tm1s.cfg

JavaJVMPath=[path to TM1]\tm1_64\bin64\jre\7.0\bin\classic\jvm.dll

→ TM1 Datenbank-Konfiguration: Speicherort für JAVA-Klassen Paket (jar-Datei)

[database data-path]/}javaextensions/user/

TM1 Start der Datenbank

→ Log-Dateiauszug tm1server.log

```
Using javaextensions.policy from [path to
TM1]\tm1_64\bin64\..\configuration\javaextensions.policy
 Creating Java Virtual Machine with arguments:
 -Djava.security.policy=[path to
TM1]\tm1_64\bin64\..\configuration\javaextensions.policy
 -Djava.class.path=java_classes;[path to
TM1]\tm1_64\bin64\javatiapi.jar;[path to
TM1]\tm1_64\bin64\tm1javaticore.jar
 end JVM args
 --------------------Session Start--------------------
 TM1 Build Number:  10.2.20500.75
 [com.ibm.cognos.tm1.jos.Main] - JavaTI started
 [com.ibm.cognos.tm1.jos.Main] - JavaTI loading extensions from:
[database data-path]\}javaextensions\user
 [com.ibm.cognos.tm1.jos.extensions.ExtensionManager] - Insert new
extension: [database data-path]\}javaextensions\user\myextension.jar
 [com.ibm.cognos.tm1.jos.extensions.ExtensionManager] - Starting
extension: [database data-path]\}javaextensions\user\myextension.jar
```

JAVA Entwicklungsumgebung

→ javatiapi.jar Bibliothek in Projekt einbinden

[Installationspfad]\tm1_64\bin64\ **javatiapi.jar**

→ jar-Datei erzeugen, zB mit Apache Ant

JAVA Implementierung

→ Entwicklung statischer Klassen bzw. Prozeduren
Gültigkeitsbereich: Prozess und alle Unterprozesse
→ Zugriff auf TM1 mittels der Klasse TIFunctions

```
import com.ibm.cognos.tm1.javati.TM1UserThread;
import com.ibm.cognos.tm1.javati.ti.TIFunctions;

public static double berechneWert(double m) {
      static TIFunctions ti = TM1UserThread.getTIFunctions();
      double value = ti.CellGetN( "meinCube", "2017",
                                  "563528785", "Ist" );

      return value * Math.special(m);
}
```

TI Implementierung

→ Die zwei Möglichkeiten des Aufrufs von JAVA-Prozeduren in TI-Prozessen

(numerisch) n_value = ExecuteJavaN(...) ;
(Zeichenkette) s_value = ExecuteJavaS(...) ;

→ Beispiel für einen numerischen Rückgabewert und einem Parameter

```
n_result = ExecuteJavaN( 'mein_package.Klasse.berechneWert' , 42 ) ;
```

TI Implementierung

→ Gültigkeitsbereich der verwendeten JAVA Variablen

- aufrufender Prozess
- Unterprozesse

→ TM1UserThread.getTIFunctions() muss für jede aufgerufene Prozedur initialisiert sein: Für größte Unabhängigkeit der Prozeduren voneinander sollte daher die statische Instanz von TIFunctions abgerufen werden

Verwendung von Threads

→ Aufgaben / Berechnungen können gezielt auf freie CPU Kerne ausgelagert werden

→ Möglichst kurzer LOCK der Datenbank

→ parallele Berechnungen möglich

Achtung: Hintergrundprozesse können nicht einfach abgebrochen werden

Fazit

→ Objektorientierte Vorteile nutzen

→ komplexe Berechnungen mit Hilfsstrukturen können ausgelagert werden

→ Schnittstellen für jeden Zweck, Wrapper

→ Parallelisierung möglich

Fazit

TM1 ⟺ Welt

OLAPLINE GmbH
+49 211 9708 140
www.olapline.de

Hauptsitz Düsseldorf
Garather Schlossallee 19
40595 Düsseldorf

Geschäftsstelle Berlin
Albert-Einstein-Str. 14
12489 Berlin

Niederlassung Spremberg
Dresdener Str. 30
03130 Spremberg

www.olapline.de

OLAPLINE GmbH
+49 211 9708 140
www.olapline.de

Hauptsitz Düsseldorf
Garather Schlossallee 19
40595 Düsseldorf

Geschäftsstelle Berlin
Albert-Einstein-Str. 14
12489 Berlin

Niederlassung Spremberg
Dresdener Str. 30
03130 Spremberg