JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Protokoll

DEZSYS-04 "VERTEILTE OBJEKTE MIT RMI"

INSY
4CHIT 2015/16

Otahal Maro

Version 1

Note:

Betreuer: Borko

Begonnen am 01. April 2016

Beendet am 07. April 2016

Inhaltsverzeichnis

1 Einführung

Verteilte Objekte haben bestimmte Grunderfordernisse, die mittels implementierten Middlewares leicht verwendet werden können. Das Verständnis hinter diesen Mechanismen ist aber notwendig, um funktionale Anforderungen entsprechend sicher und stabil implementieren zu können.

Ziele

Diese Übung gibt eine einfache Einführung in die Verwendung von verteilten Objekten mittels Java RMI. Es wird speziell Augenmerk auf die Referenzverwaltung sowie Serialisierung von Objekten gelegt. Es soll dabei eine einfache verteilte Applikation in Java implementiert werden.

1.1 Voraussetzungen

- Grundlagen Java und Software-Tests
- Grundlagen zu verteilten Systemen und Netzwerkverbindungen
- Grundlegendes Verständnis von nebenläufigen Prozessen

1.2 Aufgabenstellung

Folgen Sie dem offiziellen Java-RMI Tutorial, um eine einfache Implementierung des PI-Calculators zu realisieren. Beachten Sie dabei die notwendigen Schritte der Sicherheitseinstellungen (SecurityManager) sowie die Verwendung des RemoteInterfaces und der RemoteException.
Implementieren Sie ein Command-Pattern [2] mittels RMI und übertragen Sie die Aufgaben/Berechnungen an den Server. Sie können am Client entscheiden, welche Aufgaben der Server übernehmen soll. Die Erweiterung dieser Aufgabe wäre ein Callback-Interface auf der Client-Seite, die nach Beendigung der Aufgabe eine entsprechende Rückmeldung an den Client zurück senden soll. Somit hat der Client auch ein RemoteObject, welches aber nicht in der Registry eingetragen wird sondern beim Aufruf mittels Referenz an den Server übergeben wird.

1.3 Quellen

- "The Java Tutorials - Trail RMI"; online: http://docs.oracle.com/javase/tutorial/rmi/
- "Command Pattern"; Vince Huston; online: http://vincehuston.org/dp/command.html
- "Beispiel Konstrukt für Command Pattern mit Java RMI";
- Michael Borko; online: https://github.com/mborko/code-examples/tree/master/java/rmiCommandPattern

2 RMI Allgemein

Remote Method Invocation (RMI) ist der Aufruf einer Methode eines entfernten Java-Objekts und realisiert daraus Remote Procedure Call (RPC). Dabei befindet sich das Objekt in verschiedenen JVMs

RPC: Ist die Realisierung von Interprozesskommunikation. Es ermöglicht den Aufruf von Funktionen aus anderen Adressräumen.

2.1 Was wird benötigt

Wenn ein Objekt über RMI verschickt werden soll oder über RMI Methodenaufrufe erhält, so muss dessen Klasse bestimmte Interfaces implementieren und teilweise von speziellen Klassen erben.

Jedem JDK liegt der RMI compiler rmic bei. Dieser generiert aus Klassen, deren Objekte über RMI Methodenaufrufe erhalten sollen, die entsprechenden Stellvertreterklassen (den Stub und das Skeleton). Diese Proxies sorgen für das Verpacken der Nachrichten

Damit eine Java-Anwendung von einer anderen VM angesprochen werden kann, muß sie sich bei der lokalen Registry registrieren. Dies kann wahlweise manuell über die Anwendung rmiregistry oder mit ein paar Zeilen java-sourcecode geschehen. [4]

2.2 Vor- und Nachteile RMI

Vorteile:

- hohes Maß an Ortstransparenz
- Einsatz eines Namensdienstes (RMI- Registry)
- dynamisches Laden von Code möglich

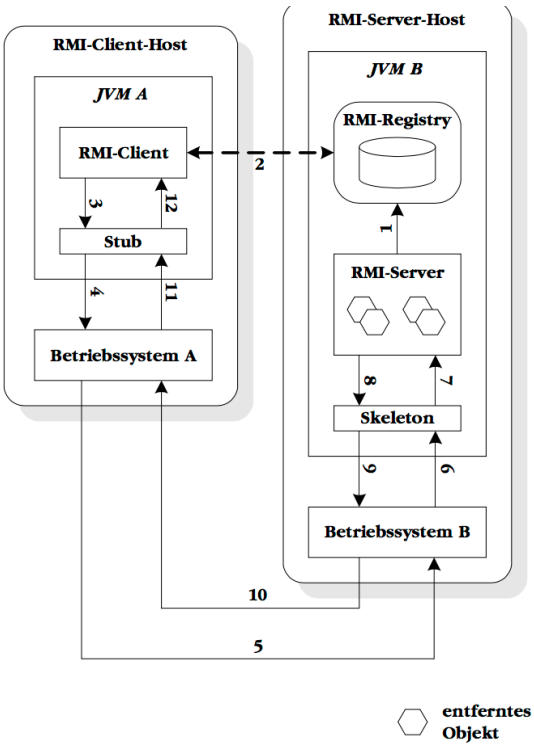
Nachteile:

- Integration mit anderen Verteilungstechniken schwierig
- Synchronisation der entfernten Aufrufe durch Programmierer notwendig
- Aufruf entfernter Methoden kann fehlschlagen

  [6]

2.3 Struktur einer RMI-Anwendung

- RMI-Client
- Stub (-Objekte)
- Skeleton (-Objekte)
- RMI-Registry
- RMI-Server und entfernte Objekte



Stub:

- Ein Stub-Objekt ist ein (client seitiger) Stellvertreter für das entfernte Objekt beim RMI-Server
-

Skeleton:

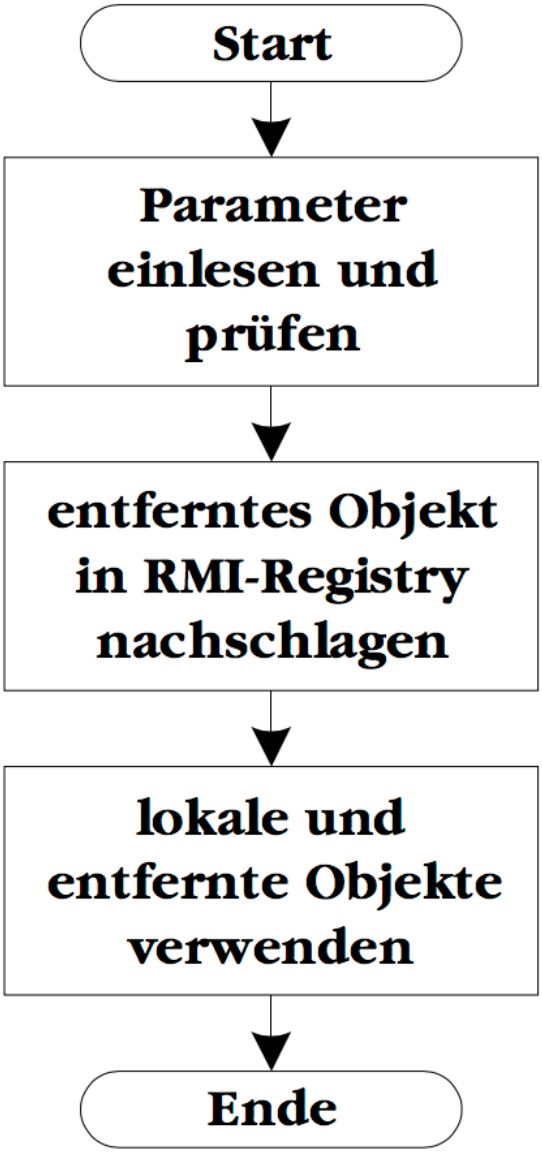
- Ein Skeleton-Objekt ist ein (server-seitiger) Stellvertreter für das aufrufende Objekt.

RMI-Registry:

- Die RMI-Registry implementiert einen Namensdienst, der eine Abbildung von Dienstnamen auf entfernte Objekte realisiert.

[6]

2.4 Arbeitsweise eines RMI-Clients



[6]

3 Ergebnisse

3.1 Aufgabenstellung 1: RMI Tutorial Implementieren

Folgen Sie dem offiziellen Java-RMI Tutorial, um eine einfache Implementierung des PI-Calculators zu realisieren. Beachten Sie dabei die notwendigen Schritte der Sicherheitseinstellungen (SecurityManager) sowie die Verwendung des RemoteInterfaces und der RemoteException.

3.2 GIT Repository Clonen

https://github.com/mborko/code-examples/tree/master/java/rmiTutorial

3.3 Erklärung der einzelnen Klassen/Interfaces

Client -> ComputePi.java:

`ComputePi.java` dient als Client der dem Server einen Task (`Pi`) schicken soll und das Ergebnis ausgeben soll.

Registry wird lokalisiert:

```
Registry registry = LocateRegistry.getRegistry(args[0]);
```

comp verwendet registry um den gewünschten Eintrag zu finden:

```
Compute comp = (Compute) registry.lookup(name);
```

Hier wird der Task gestartet

```
Pi task = new Pi(Integer.parseInt(args[1]));
BigDecimal pi = comp.executeTask(task);
```

Client -> Pi.java:

Klasse um die Zahl Pi zu berechnen. Der Task wird als remote Object an den Server gesendet. Wobei es call by value sein muss.

Compute -> Compute.java

Muss Remote extenden und gibt die execute() Methode vor. Und wirft eine RemoteException.

Compute -> Task.java

Generischer execute Aufruf.

Engine -> ComputeEngine.java

Compute wird implementiert und der Task wird gestartet.

Der Server stub wird erstellt:

```
Compute stub =(Compute) UnicastRemoteObject.exportObject(engine, 0);
```

Erstellt und binded die Registry

```
Registry registry = LocateRegistry.createRegistry(1099);
registry.rebind(name, stub);
```

4.2 Starten des Servers bzw. Clients

Zuerst wird der Server gestartet:

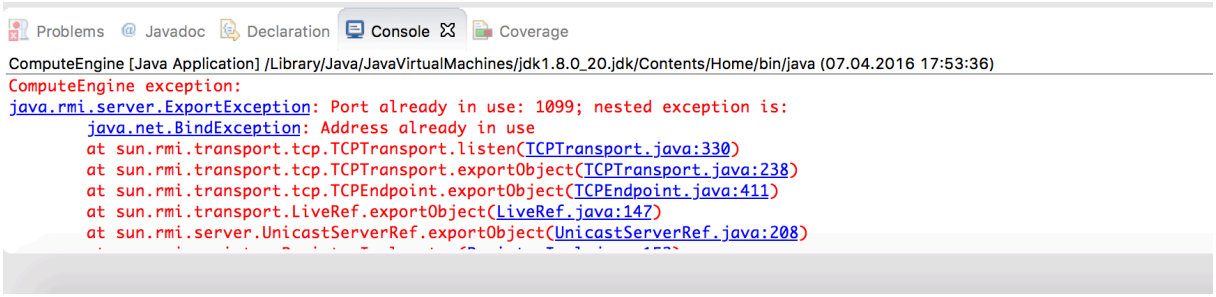
```
ComputeEngine [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_20.jdk/Contents/Home/bin/java (07.04.2016 14:32:14)
ComputeEngine bound
```

Anschließend wird der Client gestartet:

→ Anzahl der Pi stellen

```
ComputePi [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_20.jdk/Contents/Home/bin/java (07.04.2016 18:40:16)
3.14159265358979323846
```

Es kommt folgende Fehlermeldung, wenn schon eine Registry läuft



4.3 Probleme

Bei mir war oft der Fehler unter Mac OSX, dass der Port besetzt war.

Daher hab ich mir nmap runtergeladen um die besetzten Ports anzuschauen.

Folgender Befehl zeigt die Ports deren Status und deren Service an:

```
nmap -Pn 10.0.106.1
```

➔ `1099/tcp open rmiregistry`

Unter Recherche im Internet habe ich folgenden Befehl gefunden, welcher bei mir immer funktioniert hat:

```
lsof -ti TCP:1099 | xargs kill
```

4.3 Ant Mac OSX

1. Neueste Mac-Version von ant unter diesem Link runterladen: http://mirror.klaus-uwe.me/apache//ant/binaries/apache-ant-1.9.6-bin.zip
2. Das Zip-File entpacken
3. Dann muss in den Ordner wo das build.xml gespeichert ist gegangen werden: `cd /Users/marootahal/Downloads/code-examples-master/java/rmiTutorial`
4. Dann wird der Server ausgeführt: `/Users/marootahal/Downloads/apache-ant-1.9.6/bin/ant engine`
5. Und anschließend der Client: `/Users/marootahal/Downloads/apache-ant-1.9.6/bin/ant compute`

4.4 Command Pattern

Übung:

Im Package `commandpattern` ist eine Erstellung eines Interfaces notwendig.
In diesem Fall `Command.java`. Dieses Interface hat nur eine Methode `public void execute()`. Das Interface muss Serilizable implementieren.

`CommandPi.java` implementiert das Interface `Command.java`. Über die `execute()` Methode wird Pi berechnet und anschließend an den stub des Clients gesendet.

Verzeichnis

[1] "The Java Tutorials - Trail RMI"; online: http://docs.oracle.com/javase/tutorial/rmi/

[2] "Command Pattern"; Vince Huston; online: http://vincehuston.org/dp/command.html

[3] "Beispiel Konstrukt für Command Pattern mit Java RMI"; Michael Borko; online: https://github.com/mborko/code-examples/tree/master/java/rmiCommandPattern

[4] Eigenes Repository: https://github.com/motahal/DEZSYS-04-VERTEILTE-OBJEKTE-MIT-RMI-

[5] https://de.wikipedia.org/wiki/Remote_Method_Invocation

[6] http://www.informatik.uni-marburg.de/~mathes/download/k6.pdf

Zeitaufzeichnung

Aufgabe	**Dauer**
Java RMI Tutorial lauffähig	3 h
Command-Pattern mittels RMI	2 h
Protokoll	3 h
Gesamt:	**8 h**