PI-Calculator

DESYS-4AHIT

Patrick Malik & Thomas Taschner

08.01.2015

Contents

**No table of contents entries found.**

# Aufgabenstellung

## Distributed PI Calculator

Als Dienst soll hier die beliebig genaue Bestimmung von pi betrachtet werden. Der Dienst stellt folgendes Interface bereit:

|  |  |
| --- | --- |
| 1  2  3  4 | *// Calculator.java*  **public** **interface** Calculator {  **public** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Abigdecimal+java.sun.com&btnI=I%27m%20Feeling%20Lucky) pi (**int** anzahl\_nachkommastellen);  } |

Ihre Aufgabe ist es nun, zunächst mittels Java-RMI die direkte Kommunikation zwischen Klient und Dienst zu ermöglichen und in einem zweiten Schritt den Balancierer zu implementieren und zwischen Klient(en) und Dienst(e) zu schalten. Gehen Sie dazu folgendermassen vor:

1. Entwicklen Sie ein Serverprogramm, das eine CalculatorImpl-Instanz erzeugt und beim RMI-Namensdienst registriert. Entwicklen Sie ein Klientenprogramm, das eine Referenz auf das Calculator-Objekt beim Namensdienst erfragt und damit pi bestimmt. Testen Sie die neu entwickelten Komponenten.
2. Implementieren Sie nun den Balancierer, indem Sie eine Klasse CalculatorBalancer von Calculator ableiten und die Methode pi() entsprechend implementieren. Dadurch verhält sich der Balancierer aus Sicht der Klienten genauso wie der Server, d.h. das Klientenprogramm muss nicht verändert werden. Entwickeln Sie ein Balanciererprogramm, das eine CalculatorBalancer-Instanz erzeugt und unter dem vom Klienten erwarteten Namen beim Namensdienst registriert. Hier ein paar Details und Hinweise:
   * Da mehrere Serverprogramme gleichzeitig gestartet werden, sollten Sie das Serverprogramm so erweitern, dass man beim Start auf der Kommandozeile den Namen angeben kann, unter dem das CalculatorImpl-Objekt beim Load-Balancer gespeichert wird. Damit soll der Server nun seine exportierte Instanz an den Balancer übergeben, ohne es in die Registry zu schreiben. Verwenden Sie dabei ein eigenes Interface des Balancers, welches in die Registry gebinded wird, um den Servern das Anmelden zu ermöglichen.
   * Das Balancierer-Programm sollte nun den Namensdienst in festgelegten Abständen abfragen um herauszufinden, ob neue Server Implementierungen zur Verfügung stehen.
   * Java-RMI verwendet intern mehrere Threads, um gleichzeitig eintreffende Methodenaufrufe parallel abarbeiten zu können. Das ist einerseits von Vorteil, da der Balancierer dadurch mehrere eintreffende Aufrufe parallel bearbeiten kann, andererseits müssen dadurch im Balancierer änderbare Objekte durch Verwendung von synchronized vor dem gleichzeitigen Zugriff in mehreren Threads geschützt werden.
   * Beachten Sie, dass nach dem Starten eines Servers eine gewisse Zeit vergeht, bis der Server das CalculatorImpl-Objekt erzeugt und beim Namensdienst registriert hat sich beim Balancer meldet. D.h. Sie müssen im Balancierer zwischen Start eines Servers und Abfragen des Namensdienstes einige Sekunden warten.

Testen Sie das entwickelte System, indem Sie den Balancierer mit verschiedenen Serverpoolgrössen starten und mehrere Klienten gleichzeitig Anfragen stellen lassen. Wählen Sie die Anzahl der Iterationen bei der Berechung von pi entsprechend gross, sodass eine Anfrage lang genug dauert um feststellen zu können, dass der Balancierer tatsächlich mehrere Anfragen parallel bearbeitet.

## Gruppenarbeit

Die Arbeit ist als 2er-Gruppe zu lösen und über das Netzwerk zu testen! Nur localhost bzw. lokale Testzyklen sind unzulässig und werden mit 6 Minuspunkten benotet!

## Benotungskriterien

o 12 Punkte: Java RMI Implementierung (siehe Punkt 1)  
o 12 Punkte: Implementierung des Balancers (siehe Punkt 2)

o davon 6 Punkte: Balancer

o davon 2 Punkte: Parameter - Name des Objekts  
o davon 2 Punkte: Listing der Server (dyn. Hinzufügen und Entfernen)  
o davon 2 Punkte: Testprotokoll mit sinnvollen Werten für Serverpoolgröße und Iterationen

## Quellen

An Overview of RMI Applications, Oracle Online Resource, <http://docs.oracle.com/javase/tutorial/rmi/overview.html> (last viewed 28.11.2014)

# Designüberlegung

# Requirementsanalyse

* Aufgabe
  + Interpretation
* Aufgabe 1: Serverprogramm & CalculatorImpl
  + Implementieren des Calculator Interfaces, erzeugen eines CalculatorImpl-Objekts und an die registry des Servers zu binden
* Aufgabe 1: Client
  + Verbinden zur registry des Servers, Aufrufen der Methode pi() via RMI auf dem Calculator Objekt.
  + So gestalten, dass für Aufgabe 2 nichts mehr geändert werden muss.

**Aufgabe 2**

* Implementierung des Balancers ohne den Client ändern zu müssen
  + Alles was in der registry des Balancers liegt, sollte ein Objekt mit Namen „Calculator“ sein, wie vom Client erwartet, dieses sollte auch Calculator implementieren.
* Name des Servers sollte anzugeben sein
  + Wird mit den anderen Argumenten geparsed und in den Server gespeichert.
* Abfragen welche Server derzeit aktuell sind
  + Wir verwenden eine concurrentCollection um alle Server einzutragen, die Server tragen sich ein, wenn sie connecten, tragen sich bei disconnect aus und bei abstürzen eines Servers wird die Anfrage weitergeleitet und der Server aus der Collection gelöscht
* Threadsichere Anwendung
  + Der einzige Punkt in unserem Code der gesichert werden muss, ist die Collection auf die die Server beim Registrieren zugreifen und diese ist concurrent.
* Warten um sicherzugehen, dass Server gestartet sind
  + Unsere Server fahren bei der Initialisierung alles hoch, erzeugen ihre registry und erledigen alles was nötig ist, erst danach connecten sie sich mit dem Balancer und sind somit bei Registrierung arbeitsfähig.
* Listing der Server mit dynamischen hinzufügen und löschen
  + Mithilfe einer Collection wird das hinzufügen und Löschen der Server vom Balancer gehandhabt.

# Arbeitsaufteilung

|  |  |
| --- | --- |
| **Aktion** | **Person** |
| Designen der Klassen | Taschner,Malik |
| Schreiben der Dokumentation | Taschner, Malik |
| Schreiben der Kommentare | Taschner |
| Testen (JUnit) | Taschner, Malik |
| Testen (im Netzwerk, nicht automatisiert) | Taschner, Malik |
| Implementierung Logger | Malik |
| Implementierung CalculatorImpl | Taschner |
| Implementierung Client | Taschner, Malik |
| Implementierung Run | Taschner, Malik |
| Implementierung BalancerInterface | Taschner, Malik |
| Implementierung Server | Taschner, Malik |
| Implementierung ServerInterface | Taschner, Malik |
| Implementierung ArgumentParser | Malik |
| Implementierung MySecurityManager | Malik |
| Implementierung PIArgs | Malik |
| ExceptionHandling | Taschner |

# Zeiteinteilung

# Arbeitsdurchführung/Lessons Learned

# Testbericht