

A thick dark grey vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date 15.1.2015.

15.1.2015

LoadBalancer

AUFGABENDURCHFÜHRUNG

<https://github.com/aayvazyan-tgm/>

Several thin, curved, wavy lines in black and grey originate from the bottom left and extend upwards and to the right.

Ari Ayvazyan, Helmuth Brunner

Inhaltsverzeichnis

Aufgabenstellung	2
Design	3
Durchführung	5
Integrierte Patterns	6
Technologie Beschreibung	8
Testdurchführung	9
Starten der Webservlets.....	9
Starten des LoadBalancers	10
Custom Load Balacing Methode.....	12
Round Robin Methode	12
Weighted Round-Round Methode	12
Quellen	14

Aufgabenstellung

Es soll ein Load Balancer mit mindestens 2 unterschiedlichen Load-Balancing Methoden (jeweils 7 Punkte) implementiert werden (ähnlich dem PI Beispiel [1]; Lösung zum Teil veraltet [2]). Eine Kombination von mehreren Methoden ist möglich. Die Berechnung bzw. das Service ist frei wählbar!

Folgende Load Balancing Methoden stehen zur Auswahl:

- Weighted Round-Round
- Least Connection
- Least Connected Slow- Start Time
- Weighted Least Connection
- Agent Based Adaptive Balancing / Server Probes

Um die Komplexität zu steigern, soll zusätzlich eine "Session Persistence" (2 Punkte) implementiert werden.

Auslastung

Es sollen die einzelnen Server-Instanzen in folgenden Punkten belastet werden können:

- Memory (RAM)
- CPU Cycles
- I/O Zugriff (Harddisk)

Bedenken Sie dabei, dass die einzelnen Load Balancing Methoden unterschiedlich auf diese Auslastung reagieren werden. Dokumentieren Sie dabei aufkommenden Probleme ausführlich.

Tests

Die Tests sollen so aufgebaut sein, dass in der Gruppe jedes Mitglied mehrere Server fahren und ein Gruppenmitglied mehrere Anfragen an den Load Balancer stellen. Für die Abnahme wird empfohlen, dass jeder Server eine Ausgabe mit entsprechenden Informationen ausgibt, damit die Verteilung der Anfragen demonstriert werden kann.

Modalitäten

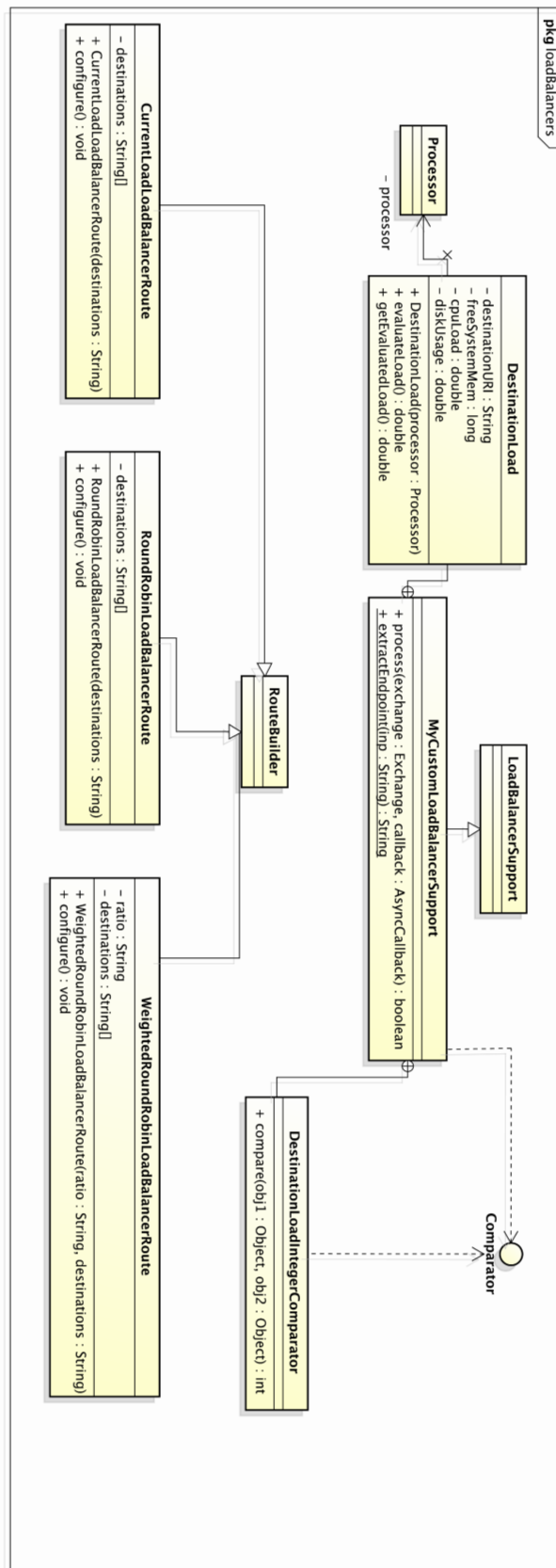
Gruppenarbeit: 2 Personen

Abgabe: Protokoll mit Designüberlegungen / Umsetzung / Testszenarien, Sourcecode (mit allen notwendigen Bibliotheken), Java-Doc, Jar

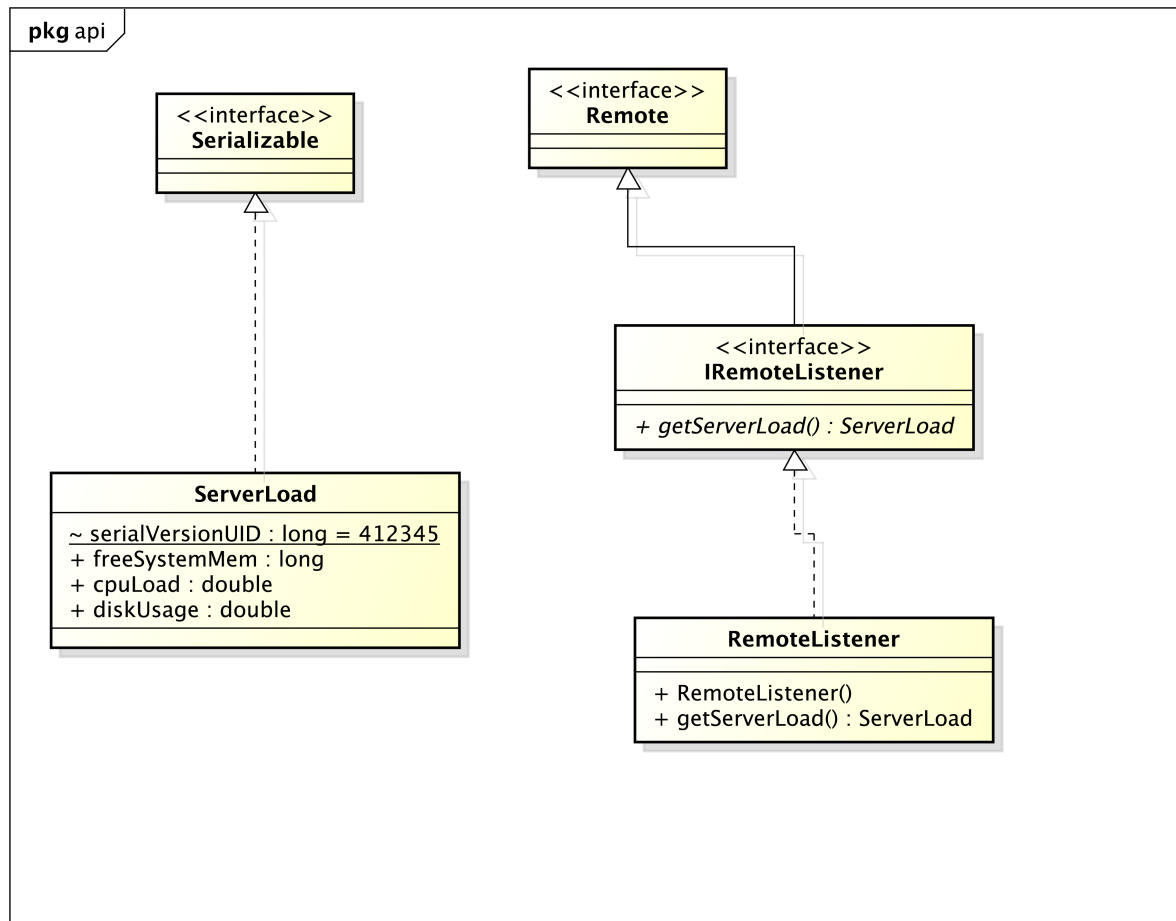
Viel Erfolg!

Design

Load Blancer:



RMI Server:



Durchführung

Im ersten Schritt haben wir ein bisschen recherchiert und uns verschiedene Framework, vorzüglich für Java angeschaut. Dabei ist uns nur ein Tool ins Auge gesprungen und dieses war Apache Camel und somit haben wir uns für dieses Framework entschieden.

Wir verwenden Apache Camel um zwischen den einzelnen Servern zu routen, der Server ist direkt in die Applikation Eingebung und mit kann ganz einfach mit dem Build-Tool Gradle erstellt werden. Infolge wird auch das Servlet am Server deployed.

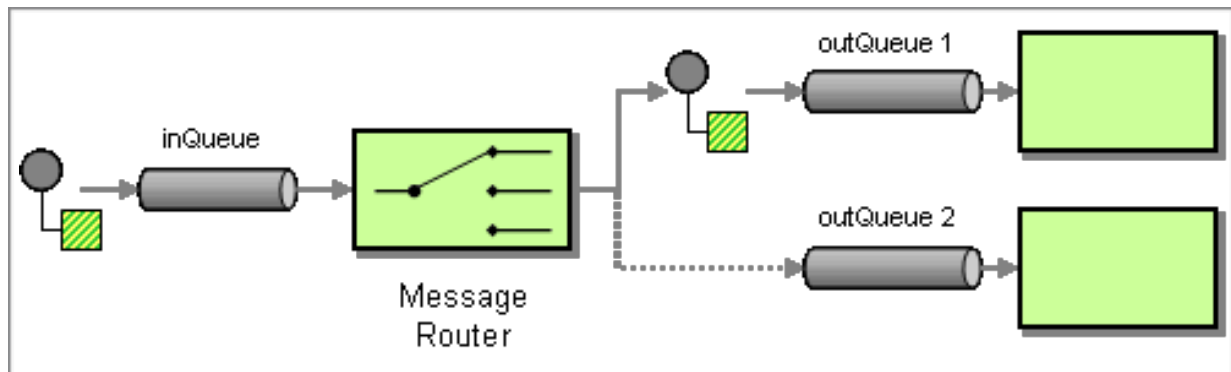
Als erstes haben wir die Balancing Methode Round Robing implementiert. In weiterer Folge sind dann CustomLoadBalancing und Weight Round Robing hinzugeführt worden.

Für den CustomLoadBalancer haben wir mittels RMI ein Client-Server Architektur implementiert, mittels dieser Implementierung ist es uns möglich Auslastungen sowie andere Systemwerte vom Server auszulesen und an den Load Balancer weiterleiten.

Integrierte Patterns

Folgende Patterns wurden integriert:

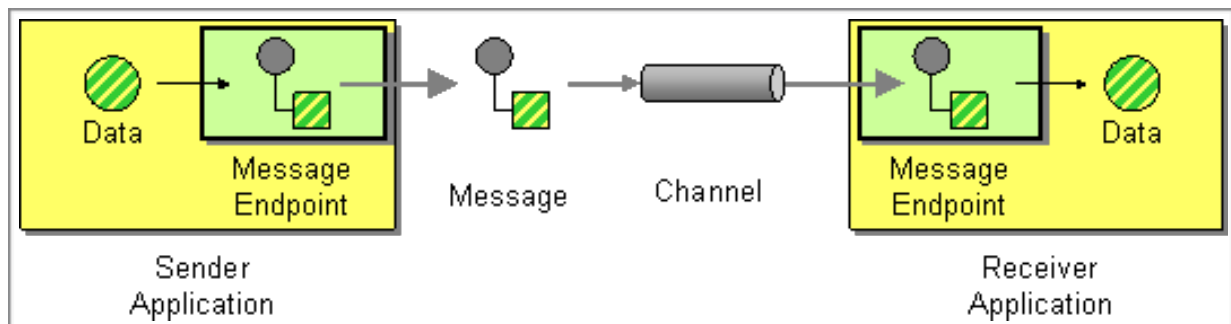
- Message Router



[3]

Beim Message Router Pattern wird von einer Input-Destination die richtige Output-Destination gefunden.

- Message Endpoint



[4]

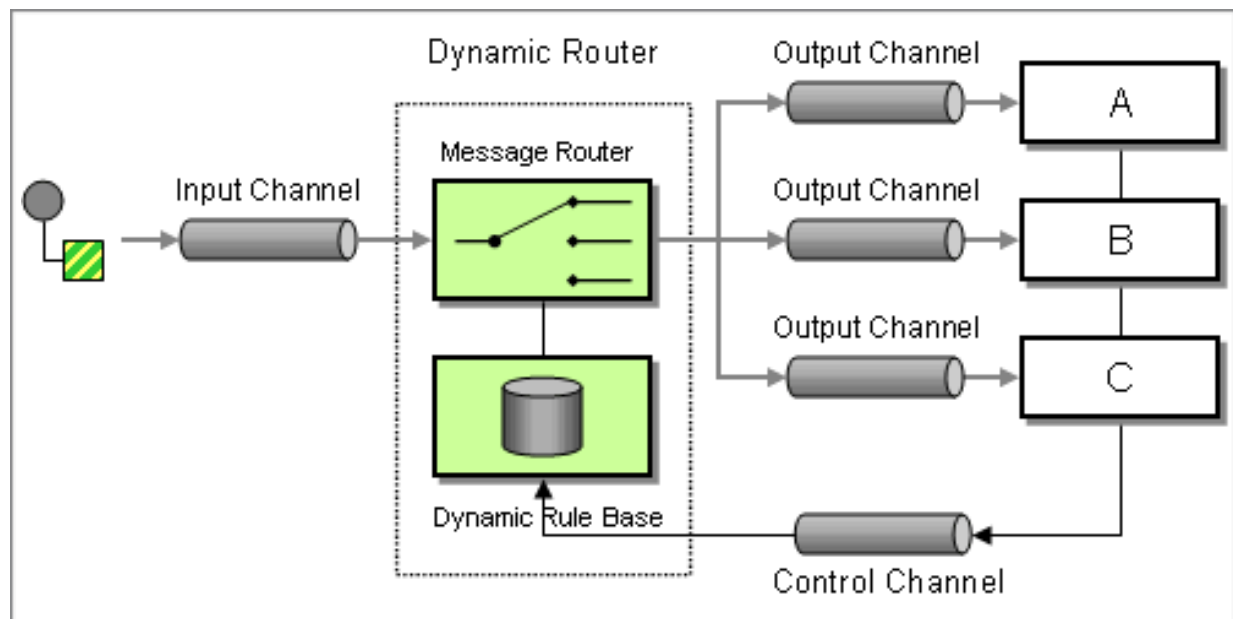
Bei diesem Pattern werden Nachrichten von einem Sender an eine Empfänger-Application weitergeleitet.

Code Snippet:

```

@Override
public void configure() throws Exception {
    from("jetty:http://0.0.0.0:8081/rr?matchOnUriPrefix=true")
        .routeId("LOADBALANCER")
        .loadBalance()
        .roundRobin()
        .to(destinations)
    ;
}
  
```

- Dynamic Routing



[5]

Das Dynamic Routing Pattern ist das Herz eines LoadBalancers. Die Nachrichten werden unter festgelegten Regeln an die einzelnen Destination verteilt.

Technologie Beschreibung

Apache Camel ist eine regelbasierte Routingengine, mit der Regeln definiert werden können.

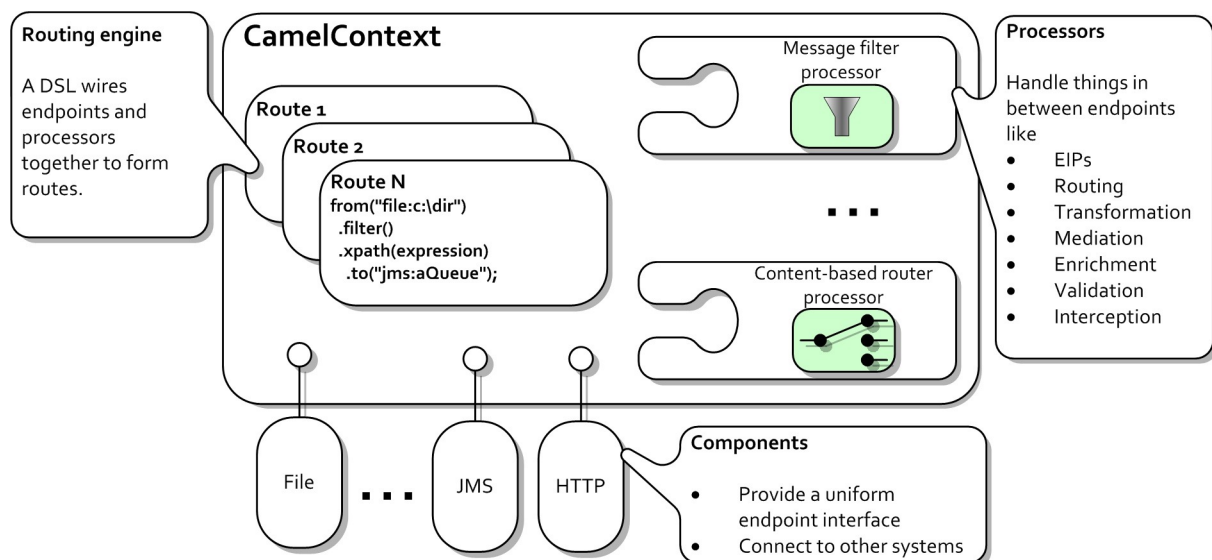
Weil Apache Camel mit URI (Uniform Resource Identifiers) arbeitet kann es ein viel Zahl von Protokollen ansprechen, wie zum Beispiel HTTP, JMS oder AMQP, weiters kann Apache Camel mit JBI, SCA, Apache ActiveMQ, RabbitMQ, Apache MINA oder Apache CXF zusammenarbeiten.

Weiters unterstütz Apache Camel Bean Binding und bietet die Möglichkeit populäre Frameworks wie Spring, Blueprint oder Guice einzubinden.

Folgende Grafik veranschaulicht das Prinzip von Apache Camel.

Es gibt mehrere Routes an die die Messages weiter geleitet werden, sowie die Routing Methode kann auch definiert werden.

[7]



[6]

Testdurchführung

Wir verwenden Gradle als Build-Tool mit diesem kann die Applikation gestartet werden.

Es wird einmal der Loadbalancer gestartet der zwischen den Servern wechselt und die Server mit den Webapplikationen.

Zu erst den :installApp Befehl ausführen.

```
./gradlew :installApp
```

Nun gibt es im build - Folder ein Verzeichnis install/Camel\ Load\ Balancer/bin. In diesem Verzeichnis befinden sich zwei Dateien einmal ein mit einer .bat Endung, diese ist für Windows Anwender und das File ohne Endung ist für alle Unix Systeme.

Starten der Webservlets

Wenn kein Parameter angegeben ist werden 2 Servlets erstellt mit Folgender Adressen:

localhost:8083 und localhost:8084

```
> ./LoadBalancerClientServlet
```

Oder mit Parameter, hier können die Adressen gesetzt werden

```
> ./LoadBalancerClientServlet localhost:8084 localhost:8085
```

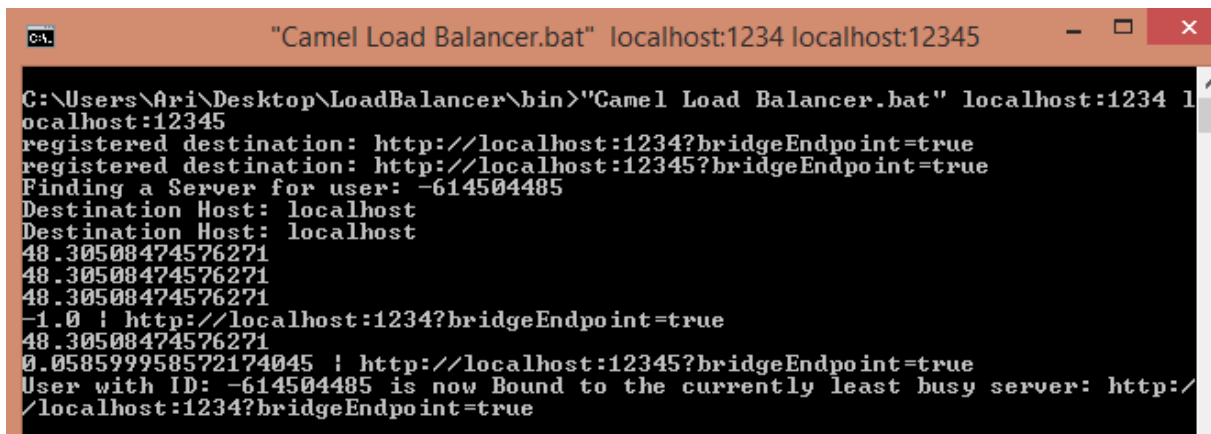
Starten des LoadBalancers

Wiederum im build/install/Camel\ Load\ Balancer/bin die Datei des jeweiligen OS ausführen.

Unter Linux:

> ./Camel\ Load\ Balancer

Unter Windows:



```

C:\Users\Ari\Desktop\LoadBalancer\bin>"Camel Load Balancer.bat" localhost:1234 localhost:12345
localhost:12345
registered destination: http://localhost:1234?bridgeEndpoint=true
registered destination: http://localhost:12345?bridgeEndpoint=true
Finding a Server for user: -614504485
Destination Host: localhost
Destination Host: localhost
48.30508474576271
48.30508474576271
48.30508474576271
-1.0 ! http://localhost:1234?bridgeEndpoint=true
48.30508474576271
0.058599958572174045 ! http://localhost:12345?bridgeEndpoint=true
User with ID: -614504485 is now Bound to the currently least busy server: http://localhost:1234?bridgeEndpoint=true
  
```

Eingabe alle Comando Befehle:

The image shows three overlapping command prompt windows. The top window, titled "Camel Load Balancer.bat" localhost:1234 localhost:12345, shows the execution of a script that registers destinations. The middle window, titled "LoadBalancerClientServlet 12345", shows the execution of a script that starts an RMI server and Jetty. The bottom window, titled "LoadBalancerClientServlet.bat 1234", shows the execution of a script that starts an RMI server and Jetty, with detailed logging output.

```

C:\Users\Ari\Desktop\LoadBalancer\bin>"Camel Load Balancer.bat" localhost:1234 localhost:12345
registered destination: http://localhost:1234?bridgeEndpoint=true
registered destination: http://localhost:12345?bridgeEndpoint=true

C:\Users\Ari\Desktop\Servlets\bin>LoadBalancerClientServlet 12345
RMI Server started
Starting Jetty ..
2015-01-16 02:11:48.468:INFO::main: Logging initialized @665ms
2015-01-16 02:11:48.515:INFO:oejs.Server:main: jetty-9.2.0.RC0
2015-01-16 02:11:48.531:INFO:oejsh.ContextHandler:main: Started o.e.j.s.ServletContextHandler@38cccef</,null,AVAILABLE>
2015-01-16 02:11:48.566:INFO:oejs.ServerConnector:main: Started ServerConnector@22a71081<HTTP/1.1><localhost:1234>
2015-01-16 02:11:48.567:INFO:oejs.Server:main: Started @780ms
Started Servlet/s

C:\Users\Ari\Desktop\Servlets\bin>LoadBalancerClientServlet.bat 1234
RMI Server started
Starting Jetty ..
2015-01-16 02:12:23.673:INFO::main: Logging initialized @195ms
2015-01-16 02:12:23.720:INFO:oejs.Server:main: jetty-9.2.0.RC0
2015-01-16 02:12:23.737:INFO:oejsh.ContextHandler:main: Started o.e.j.s.ServletContextHandler@2f333739</,null,AVAILABLE>
2015-01-16 02:12:23.774:INFO:oejs.ServerConnector:main: Started ServerConnector@449b2d27<HTTP/1.1><localhost:12345>
2015-01-16 02:12:23.775:INFO:oejs.Server:main: Started @314ms
Started Servlet/s
  
```

Custom Load Balancing Methode

Bei dieser Methode werden die Auslastungen von den Server herangezogen und so die Anfragen der Benutzer verteilt.

Aufruf durch folgende URL:

```
localhost:8081/c
```

Round Robin Methode

Die Anfragen der Clients werden immer zwischen den einzelnen Servern verteilt.

Aufruf durch folgende URL:

```
localhost:8081/rr
```

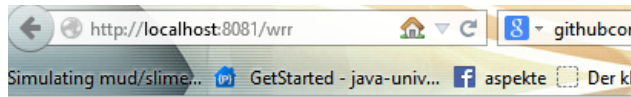
Weighted Round-Round Methode

Hier bei wird der Load Balancer so einstellt das zum Beispiel Server 1, 3 Anfragen verarbeitet und der Server 2 nur 1 Anfrage verarbeitet. Weil der Server 1 mehr Last aufnehmen kann wir diesem auf auch mehrere Anfragen zugeteilt.

Aufruf durch folgende URL:

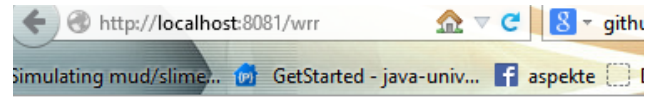
```
localhost:8081/wrr
```

Die Server die ausgeführt werden, einmal der Server 4 und der Server 1.



Prime Searcher: 4

Started at Fri Jan 16 02:16:33 CET 2015
Last prime discovered was 163627 at Fri Jan 16 02:19:08 CET 2015



Prime Searcher: 1

Started at Fri Jan 16 02:18:26 CET 2015
Last prime discovered was 5519 at Fri Jan 16 02:18:34 CET 2015

Quellen

- [1] "Praktische Arbeit 2 zur Vorlesung 'Verteilte Systeme' ETH Zürich, SS 2002", Prof.Dr.B.Plattner, übernommen von Prof.Dr.F.Mattern (<http://www.tik.ee.ethz.ch/tik/education/lectures/VS/SS02/Praktikum/aufgabe2.pdf>)
- [2] <http://www.tik.ee.ethz.ch/education/lectures/VS/SS02/Praktikum/loesung2.zip>
- [3] <http://www.eaipatterns.com/MessageRouter.html>
- [4] <http://www.eaipatterns.com/MessageEndpoint.html>
- [5] <http://www.eaipatterns.com/DynamicRouter.html>
- [6] http://java.dzone.com/sites/all/files/figure1_1.jpg
- [7] <http://camel.apache.org>