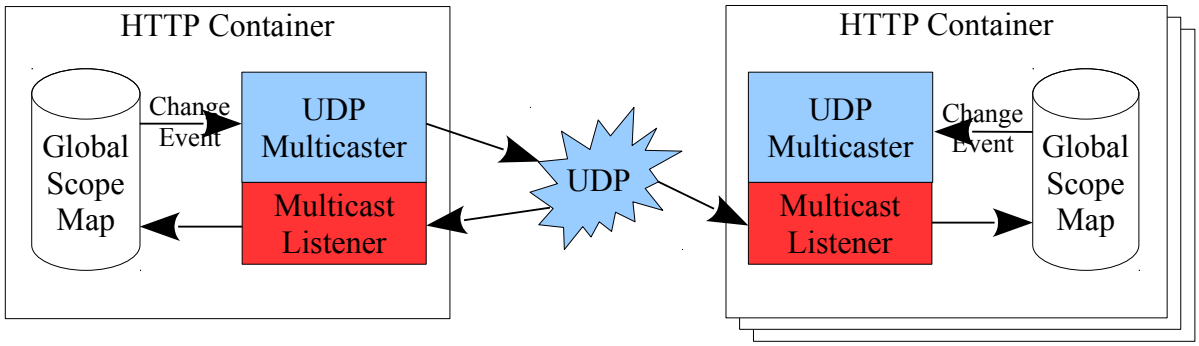
Verteilte Systeme

Übung D - Einzelarbeit

Bearbeitungszeit 2 Wochen

Diese Übung adressiert den Umgang mit UDP-Multicast. Die Klasse *HttpContainer1* verfügt nur über einen eingeschränkten Global-Scope Mechanismus dessen Wirkungsbereich an den Grenzen des Container-Prozesses endet. Dieser soll per UDP-Multicast so erweitert werden dass der Inhalt des Global-Scope innerhalb eines Clusters von Http Containern synchronisiert wird:



Zu beachten ist dabei dass der *Multicast-Listener* Thread des Senders beim Multicast ebenfalls benachrichtigt wird. Daher muss die UDP-Pakete Ursprungsinformation (die Prozess-ID) beinhalten um solche Feedback-Nachrichten aussortieren zu können. Des Weiteren ist unbedingt zu beachten dass Änderungen an der Map, die durch den *Multicast-Listener* ausgelöst werden, keine Change-Events auslösen dürfen. Andernfalls droht das Fluten des Netzwerks mit einer exponentiell ansteigenden Zahl von endlos-rekursiven Multicasts!

Aufgabe 1: Integration in den HTTP-Container

Kopiert die Klasse *HttpContainer1* nach *HttpContainer2*. Übernehmt in der Main-Methode des Containers die zu verwendende virtuelle UDP Multicast-Adresse sowie den dazugehörigen Port als Socket-Adress-Parameter (z.B. Parameter-Index 2). Erlaubte Multicast-Adressen befinden sich dabei im Bereich:

- Ipv4: `224.0.0.0 bis 239.255.255.255`
- Ipv6: `ff00:0000:0000:0000:0000:0000:0000:0000` bis `ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff`

Benennt die Klasse *UdpMulticaster1Skeleton nach UdpMulticaster1* um. Erweitert die Main-Methode des Containers so dass dort eine Instanz von *UdpMulticaster1* erzeugt, und diese anschließend als Listener der Global-Scope Map registriert wird. Übergebt bei der Konstruktion des UDP-Multicasters unbedingt den Delegierten dieser Map, denn falls ihr hier die Global-Scope Map selbst übergebt droht die oben beschriebene Endlos-Rekursion!

Code-Beispiel:

```
final CallbackMap<Serializable> globalMap = HttpDynamicResourceFlavor.globalMap();

final VetoableChangeListener listener = new UdpMulticaster1(multicastSocketAddress, globalMap.getDelegateMap());

globalMap.getListeners().add(listener);

...

globalMap.getListeners().remove(listener);
```

Aufgabe 2: UdpMulticaster1

Implementiert einen Cluster-Scope im Push-Design, folgt dazu den Kommentaren im Code von *UdpMulticaster1*. Zur Serialisierung/Deserialisierung von Objekten in Byte-Arrays und zurück verwendet Object-Streams auf Basis von ByteArray-Streams – die Klasse `de.htw.ds.util.Serializables` enthält Methoden die diese Vorgänge bereits geeignet abstrahieren. Verwendet anonyme Datagram-Sockets zum Versenden der UDP Multicast-Nachrichten, so wie im Chat-Beispiel *de.htw.ds.chat.UdpChatPeer* gezeigt. Beachtet beim Nachrichten-Empfang dass Pakete, die vom eigenen Prozess aus gesendet werden (erkennbar an der beinhalteten Prozess-ID), ignoriert werden müssen.

Startet zwei *HttpContainer2*-Instanzen, zum Testen bevorzugt auf derselben Maschine. Kopiert dazu die Java-Quelldatei `de.htw.ds.http.handler.MemoChat.java` aus `distributed-systems-plugins.jar` in den Kontext-Pfad des Http-Containers – das Package dieser Klasse muss dabei als Verzeichnis-Baum abgebildet werden. Chat-Einträge werden von dieser Klasse im globalen Scope verwaltet, Memo-Einträge dagegen im Session-Scope. Prüft nun im Web-Browser ob die Chat-Einträge auf der MemoChat-Seite (`http://adresse:port/de.htw.ds.http.handler.MemoChat.java`) korrekt auf beide Container-Instanzen verteilt werden.

Aufgabe 3: UdpMulticaster2

Das obige **Push-Design** hat den Nachteil dass bei Ausfall einer HttpContainer2-Instanz der ganze Cluster neu gestartet werden muss, um wieder synchrone Global-Scope Einträge zu erreichen. Dies soll nun in einem **Pull-Design** durch Verwendung einer Datenbank-Tabelle für den globalen Scope vermieden werden.

Installiert dazu eine MySQL-Instanz, oder verwendet alternativ den von der HTW zur Verfügung gestellten MySql-Server. Spielt in Eure MySQL-Instanz das Script `http-mysql.ddl` ein um die Datenbank „http“ und deren Tabelle „GlobalScope“ zu erzeugen. Passt die Datei `http-mysql.properties` gegebenenfalls in situ an Eure Datenbank-Installation an. Ein MySQL-Driver für JDBC (`mysql-connector-java-5.1.22.jar`) befindet sich im lib-Verzeichnis des Share-Laufwerks, dieser muss jedoch noch für Eclipse konfiguriert werden:

- Runtime only: Runtime-Konfiguration von HttpContainer2 → ClassPath → User Entries → Add External JARs
- Runtime & Compiler: Projekt → Properties → Java Build Path → Libraries → Add External JARs

Die zweite Option erlaubt es gegen die Klassen in der JAR-Datei zu codieren, während die Erste dies nur per Java Reflection API, oder die Verwendung dieser Klassen mittels eines Factory-Mechanismus erlaubt.

Benennt sodann die Klasse *UdpMulticaster2Skeleton* in *UdpMulticaster2* um, und integriert diese wie in Aufgabe 1 beschrieben in Euren *HttpContainer2*. Die Idee des Pull-Designs ist, dass die Datenbank-Tabelle „GlobalScope“ pro Schlüssel im globalen Scope eine Zeile beinhaltet, welche den assoziierten Wert einer Variable serialisiert als Bytefolge speichern kann. Damit müssen in den Multicast-Nachrichten nur noch die Schlüssel, nicht aber die Werte der Scope-Variablen transferiert werden, da die Nachrichten-Empfänger diese Werte aus der Datenbank abfragen können. Die Klasse *GlobalScopeConnector* kapselt alle dazu notwendigen Datenbank-Operationen, und eine Instanz davon ist im UdpMulticaster2 bereits als Instanzvariable verfügbar.

Dies hat zum einen den Vorteil dass die Nachrichten nicht mehr die für UDP zulässige Paketlänge überschreiten werden, außer ein Schlüssel ist exzessiv lang. Zudem kann im Konstruktor von *MapMulticaster2* der Global-Scope mit dem Inhalt dieser Tabelle initialisiert werden, so dass nach einem Neustart des Containers der Global-Scope auf demselben Stand wie die anderen Server im Cluster ist. Beachtet dabei dass während dieser Initialisierungs-Prozedur keine parallel eintreffenden Änderungs-Nachrichten verloren gehen können, da Java bei Multicast-Sockets im Gegensatz zu Server-Sockets eingehende Pakete puffert!

Startet zwei Container-Instanzen, und prüft ob die Chat-Einträge im MemoChat immer noch korrekt verteilt werden. Startet einen der Container danach neu, und prüft ob dieser korrekt mit den bereits existierenden Chat-Einträgen initialisiert wird.