Verteilung & Kommunikation

Push Prinzip mit RMI

- Remote-Interface definiert Registrierungsmethode, die als Dienst zur Verfügung stehen
- Der Logger Server implementiert das Interface und erzeugt eine Remote Instanz vom Typ RMIServerImpl.
- Remote Instanz wird bei der RMI Registry registriert.
- Logger Viewer erzeugt Remote Push Receiver und exportiert diesen in RMI Runtime.
- Logger Viewer holt sich Remote-Referenz zur RMIServerImpl und ruft dann Registrierungsmethode auf und übergibt den RMIReceiver für das Remote Push.
- Der Logger Server ruft am Remote Push Receiver bei Bedarf die push-Methode auf.

Komponenten verteilen

RMI Registry:

- Registry-Tool lässt sich mit Klasse `LocateRegistry`
- `createRegistry` erzeugt lokale Registry (nur Registries auf lokaler Plattform verschiedenen Ports und eigener `RMIClientSocketFactory erzeugbar)
- Nach dem Erzeugen der Registry darf das Programm nicht terminieren, da sonst die Registry beendet wird
- Das Registry-Tool sollte in einen inaktiven Wartezustand versetzt werden

Verwendung der RMI Registry vom Server:

- RMI Server holt Hilfe bei `LocateRegistry` (`getRegistry`)
- `getRegistry` ermöglicht Referenz zur lokalen Registry

Start der verteilten RMI Applikation:

- Beim Start des RMI Servers:

```
java.rmi.ServerException: RemoteException occurred
in server thread; nested exception is: -java.rmi.UnmarshalException: error unmarsh
alling
arguments; nested exception is: -java.lang.ClassNotFoundException:
ch.hslu.vsk.vs08.fibo.dist.RemoteFibonacci...
```

- Bei Ausführung von RMI Applikationen wird immer Byte-Code geladen. Der Byte-Code wird nach folgendem Muster gesucht:
  - In lokalen Verzeichnissen
  - In entfernten Verzeichnissen

Anwendung mit verteilten Objekten:

- RMI Anwendungen können entweder:
  - Remote-Objekte (RO) mit RMI Naming Methoden bei RMI-Registry registrieren
  - Entfernte Objektreferenzen als Teil ihrer normalen Operation übergeben/zurückgeben
- Kommunikation mit RO durch RMI-Middelware (transparent)
- Wird Klassen-Bytecode für Objekte, die als Parameter oder Rückgabewerte übergeben werden, benötigt, stellt RMI die erforderlichen Mechanismen zum Laden des Codes eines Objekts sowie zum Übertragen seiner Daten bereit

Codebase erstellen

Codebase einrichten:

- Property `java.rmi.server.codebase` gibt die Codebase an, für Klassen welche die JMV benötigt
- Codebase wird gesetzt mit `System.setProperty(String key, String value)`
  - key: Name des System Property ("java.rmi.server.codebase").
  - value: System Property Argument ("http://localhost:8080/")
- Sobald entfernter Code geladen wird, braucht es einen Security Manager

Codebase zur Verfügung stellen:

- Am einfachsten ist es die Klassen-Bytecodes für die benötigten Objekte per Http-Server zur Verfügung zu stellen
- Http-Server muss Zugriff auf die `.class` Dateien haben, keine Java Archiv (.jar) Dateien
- JDK stellt mit ClassServer (tool.jar) einen einfachen Http-Server zur Verfügung. Argumente:
  - **port** <port> Server Port
  - **dir** <dir> Root-Verzeichnis

- **verbose** protokolliert den Zugriff auf der Konsole
- **stop** beendet den Server

Security Manager aktivieren

Sicherheitsmodell:

- lokaler Code: Zugriff auf alle Ressourcen
- Code aus Netz: Zugriff stark eingeschränkt

Security Manager (SM) aktivieren:

- `java.lang.SecurityManager`
- Aktivieren des SM über Kommandozeile (`Djava.security.manager`)
- Instanziieren im Code mit `System.setSecurityManager(new SecurityManager())`
- Prüfen ob aktiviert: `System.getSecurityManager()`
- benutzerdefinierte Sicherheitslinie in ASCII-File (policy file)

Rechte durch Policy vergeben:

- Rechte in policy file
- policy file besteht aus `grant` -Anweisungen
- Schlüsselwort `permission`

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

- Bei Vergabe von Rechten können zusätzlich angegeben werden:
  - Codebase: Rechte für Klassen von bestimmten Ort
  - Signierung: Nur Recht wenn Code signiert
  - Principal: Gewährt bestimmte Rechte für authentifizierte Benutzer

Policy in RMI Applikation definieren:

- vor Aktivierung des SM angeben
  - Policy file per Kommandozeile definieren: `Djava.security.policy=rules.policy`
  - Policy file im Code definieren:
    `System.setProperty("java.security.policy", "rules.policy");`
- Mit der Angabe der Policy-Datei, kann auch ein Verzeichnis definiert werden. Wird kein Pfad angegeben sucht der Security Managers die Policy im root-Verzeichnis

Registry Setup:

```
grant {
permission java.net.SocketPermission "localhost:1099", "listen";
permission java.net.SocketPermission "localhost:8080", "connect,resolve";
permission java.net.SocketPermission "localhost:1024-", "accept,resolve";
permission java.net.SocketPermission "*:1024-", "connect,resolve";
permission java.net.SocketPermission "*:1024-", "accept,resolve";
};
```