Verteilte Systeme (I6, Bc)

Blatt 04: Java-RMI-Programmierung

Sommersemester 2017

Bearbeitung im Praktikum ab 01.06.2017

***Hinweise*:**

- Die RMI-Registry muss vor Start des RMI-Servers mit `rmiregistry` gestartet werden. Dazu sollten keine Root-Rechte benötigt werden.
- Für den Start Ihres RMI-Servers sollten Sie keine Root-Rechte brauchen.
- Arbeiten Sie bei diesen Aufgaben innerhalb eines Rechners und nicht rechnerübergreifend. Bei rechnerübergreifender Arbeitsweise müssen jetzt Java-Berechtigungen richtig konfiguriert werden, damit der RMI-Client auf den RMI-Server zugreifen kann.
- Unter Windows brauchen Sie ggf. Admin-Rechte, um die Registry zu starten, wenn eine Firewall läuft.
- Verwenden Sie das Paket `rmilernen` für Ihre eigenen Klassen.
- Verwenden Sie für jede Aufgabe ein eigenes Verzeichnis zur Ablage aller zur Aufgabe gehörigen Dateien.

Aufgabe 1: RMI-Server und RMI-Client - Basic

1.1 RMI-Server

Schreiben Sie einen RMI-Server, der die folgende Funktionalität zur Verfügung stellt:

1. Methode `rmiSquare()`,
   die einen long-Wert als Eingabe erhält, den Wert mit sich selbst multipliziert und das Ergebnis als Returnwert zurückgibt.
2. Methode `rmiAdd()`,
   die zwei long-Werte addiert und die Summe als Returnwert zurückgibt.
3. Methode `rmiConcat()`,
   die zwei Zeichenketten als Eingabeparameter nimmt und die Konkatenation der beiden Zeichenketten als Returnwert zurückliefert.
4. Methode `rmiSplit()`,
   die eine Zeichenkette als Eingabeparameter nimmt, die Zeichenkette beim ersten White-Space-Zeichen (Leerzeichen, Tabulator, Carriage Return, Line Feed) aufspaltet und die beiden Teile als Ergebnis liefert.
5. Methode `rmiIncrement()`,
   die einen long-Wert incrementiert, aber als Returntyp void hat.
6. Außerdem soll der RMI-Server eine Methode `rmiShutdown()` bieten, über die der RMI-Server beendet werden kann. Beim Beenden soll der Server auch seine Registrierung beim Binder (RMI-Registry) löschen.

7. Methode `rmiServernameAtWithException()`,
   die einen int-Wert `i` als Index entgegen nimmt und den i-ten Buchstaben des Servernamens als Returnwert zurückgibt. Ist der Index kleiner als `0` oder größer gleich der Länge des Servernamens, so wird eine normale Java-Ausnahme geworfen.

Machen Sie durch geeignete Ausgaben auf der serverseitigen Konsole kenntlich, wenn eine dieser Interface-Methode beim RMI-Server durchlaufen wurde.

1.2 RMI-Client

Schreiben Sie einen RMI-Client, um Ihren RMI-Server zu testen.

- Jede der oben genannten Interface-Methoden (außer `rmiShutdown()`) soll dabei mindestens zweimal, und zwar mit verschiedenen Eingabewerten aufgerufen werden.
- Statten Sie Ihren Client mit einer Aufrufoption aus, mit der es möglich ist, den Server zu beenden.
- Das Ziel der RMI-Aufrufe (Registrierter Name des Interfaces des RMI-Servers) soll bei Ihrem Client über Kommandozeilenargumente konfigurierbar sein.

***Hinweise*:**

1. Realisieren Sie zuerst die Schnittstellenbeschreibungsklasse mit den Interface-Methoden. Überlegen Sie sich dazu die Signaturen der zu realisierenden Methoden. Ansatzweise sind diese ja schon in der Aufgabenstellung beschrieben.
2. Implementieren Sie nun das Interface zur Schnittstellenbeschreibungsklasse in einer eigenen Klasse und übersetzen Sie diese Klasse.
3. Implementieren Sie jetzt Ihren RMI-Client.
4. Nutzen Sie die Beispiele aus dem Skript für Ihre Implementierung.
5. Mit der Methode
   ```
   String[] split(String regex)
   ```
   der Klasse `String` lässt sich eine Zeichenkette leicht bezüglich Trennzeichen zerlegen. Der reguläre Ausdruck für White-Space-Zeichen lautet `\s`.
6. Der Standardport der RMI-Registry ist 1099. Mit `netstat -nl|grep 1099` oder `ps -a` können Sie prüfen, ob die RMI-Registry läuft.
7. Starten Sie die RMI-Registry im Verzeichnis zu dieser Aufgabe.

Aufgabe 2: RMI-Server und RMI-Client - BasicPlus

Schreiben Sie einen RMI-Server, der folgende Funktionalität bietet:

1. Methode `rmiQuad()`,
   die einen int-Wert als Eingabe hat und als Ergebnis ein Feld mit den Quadratzahlen von 1 bis zum Wert der Eingabe zurück gibt. Ist z.B. der Eingabewert gleich `3`, dann ist das Ergebnis das Feld `{1,4,9}`.
2. Methode `rmiTwice()`,
   die eine Zeichenkette `name` als Eingabe hat und als Ergebnis den Wert von `name` als ein Element und `name` mit sich selbst konkatentiert als ein anderes Element zurück liefert. Ist z.B. der Eingabewert gleich `"ab12"`, dann besteht das Ergebnis aus den beiden Elementen `"ab12"` und `"ab12ab12"`.

3. Methode `rmiReaddir()`,
   die eine Zeichenkette `dirname` als Eingabe hat und als Ergebnis den Inhalt des Verzeichnisses `dirname` zurückgibt. D.h. das Ergebnis besteht aus einer verketteten Liste (kein String-Array) von Datei- und Verzeichnisnamen bzw. aus einer Fehlerkennung, wenn das Verzeichnis `dirname` nicht existiert oder nicht gelesen werden kann.
4. Außerdem soll der RMI-Server eine Methode `rmiShutdown()` bieten, über die der RMI-Server beendet werden kann. Beim Beenden soll der Server auch seine Registrierung beim Binder (RMI-Registry) löschen.

Schreiben Sie zum Testen Ihres neuen RMI-Servers auch einen RMI-Client. Dieser RMI-Client soll jede Methode des Interfaces mindestens einmal aufrufen.

***Hinweise*:**

1. Mit der Klasse `java.io.File` lässt sich ein File-Objekt durch `new File(dirname)` erstellen. Die Klasse `File` kennt die Methoden
   ```
   boolean isDirectory()
         // prüft, ob das Objekt ein Verzeichnis repräsentiert
   String[] list()
         // listet Verzeichnisinhalt auf, falls Objekt ein Verzeichnis darstellt
   ```
2. Übergeben Sie dem Client die Aufrufparameter für `rmiQuad()` und `rmiReaddir()` als Kommandozeilenparameter.
3. Starten Sie die RMI-Registry im Verzeichnis zu dieser Aufgabe.

Aufgabe 3: Mehrere Interfaces in einem RMI-Server - BasicAndPlus

Schreiben Sie einen RMI-Server BasicAndPlus, der drei Remote-Objekte exportiert, d.h. drei Remote-Objekte bei der RMI-Registry registriert.

- Das erste Remote-Objekt soll nur das Interface des Remote-Objekts von Aufgabe "Basic" haben.
- Das zweite Remote-Objekt soll nur das Interface des Remote-Objekts von Aufgabe "BasicPlus" haben.
- Das dritte Remote-Objekt soll sowohl das Interface des Remote-Objekts von Aufgabe "Basic2 als auch das Interfache des Remote-Objekts von Aufgabe "BasicPlus" haben.

Schreiben Sie zum Testen Ihres neuen RMI-Servers auch einen RMI-Client. Dieser RMI-Client soll jede Methode der Interfaces mindestens einmal aufrufen. Außerdem soll dieser Client ausgeben, welches Objekt er jeweils für einen Remote-Methodenaufruf nutzt, d.h. er soll die systemweit eindeutige Id des entsprechenden Remote-Objekts ausgeben und den Namen des Remote-Objekts.

Aufgabe 4: Mehrere RMI-Server parallel starten

Erstellen Sie auch für diese Aufgabe ein neues Verzeichnis und starten Sie die RMI-Registry in diesem neuen Verzeichnis.

Starten Sie die RMI-Server der Aufgaben "Basic", "BasicPlus" und "BasicAndPlus" parallel in ihren jeweiligen Verzeichnissen. Achten Sie darauf, dass die Namen der Remote-Objekte, die von den verschiedenen RMI-Servern registriert werden, nicht miteinander in Konflikt stehen.

Führen Sie dann die zugehörigen Clientprogramme nacheinander aus. Beenden Sie durch einen zweiten Aufruf der jeweiligen Clientprogramme dann die RMI-Server durch Remote-Aufrufe von den Clients aus. Schauen Sie abschließend nach, ob die RMI-Registry noch läuft.

Als Lösung der Aufgabe wird eine Datei `a04rmi-lsg.txt` mit folgendem Inhalt erwartet:

- Beschreibung des Vorgehens, der genauen Aufrufe der RMI-Registry, der RMI-Server und RMI-Clients. Verweise auf Shell-Skripte als Beschreibung sind erlaubt.
- Beschreibung des Inhalts der RMI-Registry, nachdem sich die Remote-Objekte zu Aufgabe "Basic", "BasicPlus" und "BasicAndPlus" registriert haben, und zwar als Tabelle mit den Spalten:
    - Remote-Objektname, Implementierungsklasse des Remote-Objekts, Liste der Interfaces des Remote-Objekts

***Hinweis*:** Es gibt zwei Lösungsansätze.

Ansatz 1:

a. Stellen Sie als Codebase alle Interfaces und Klassen der Remote-Interfaces der Remote-Objekte über einen HTTP-URL zur Verfügung.

b. Nutzen Sie dazu beim Start des Servers die Option
`-Djava.rmi.server.codebase=<HTTP-URL>`

Ansatz 2:

a. Stellen Sie beim Start der RMI-Registry alle Verzeichnisse als Codebase ein, in der sich die Interfaces und Klassen der Remote-Interfaces der Remote-Objekte befinden.

b. Nutzen Sie dazu beim Start der RMI-Registry die Option
`-Djava.rmi.server.codebase=<FILE-URL>`

c. Die Option für die Codebase kann beim Start der RMI-Registry auch mehrfach angegeben werden.

Optionale Aufgabe 5: RMI-Server und -Client auf verschiedenen Maschinen starten

Verwenden Sie für diese Aufgabe zwei verschiedene Maschinen M1 und M2. Starten Sie auf M1 die RMI-Registry geeignet. Starten Sie auf der gleichen Maschine M1 Ihren RMI-Server Basic (ggf. auch weitere Server). Starten Sie auf Maschine M2, die mit Maschine M1 über Netzwerk verbunden ist, Ihren RMI-Client Basic.

Als Lösung der Aufgabe wird eine Datei `a05rmi-lsg.txt` mit folgendem Inhalt erwartet:

- Beschreibung der Aufrufe der RMI-Registry, des RMI-Servers Basic und RMI-Clients Basic. Verweise auf Shell-Skripte als Beschreibung in `a05rmi-lsg.txt` und Verweise auf andere Aufgaben ist zulässig. Geben Sie auch die Betriebssysteme und deren Version sowie die jeweiligen Java-Versionen an. Definieren Sie ggf. Testszenarien.

***Hinweis*:**

1. Falls der Lookup für das Remote-Objekt funktioniert, aber die Verbindung beim Aufruf einer Methode des Remote-Objekts fehlschlägt, müssen Sie ggf. beim Registrieren den Host-Anteil Ihres Registrierungsnamens des Remote-Objekts auf die IP der Servermaschine ändern.

