Remote Method Invocation - RMI

Jan-Ole Giebel

Hochschule für angewandte Wissenschaften Hamburg
Fortgeschrittenes Programmieren mit Java

8. Juni 2012

Inhalt

Begriffsklärung

Konzepte im Softwareengineering

Worauf man bei der Entwicklung achten sollte

Bei Methoden:

- KISS (Keep it stupid simple)
- Do only one thing

Bei Klassen:

- Single Responsibility Principle
- Dependency Inversion Principle

Diese Prinzipien erhöhen und vereinfachen die Testbarkeit der Software und dadurch die Zufriedenheit beim Kunden.

Problemstellung

Java Runtime Environments

Klassische Javaprogramme

- alle Objekte in einer JRE
- alle Objekte auf einem Host

Weitere Anwendungsfälle

- Anwendungen verteilt auf verschiedene JREs
- Server-Client-Systeme
- Peer-To-Peer

Problemstellung

Java Runtime Environments

Klassische Javaprogramme

- alle Objekte in einer JRE
- alle Objekte auf einem Host

Weitere Anwendungsfälle

- Anwendungen verteilt auf verschiedene JREs
- Server-Client-Systeme
- Peer-To-Peer

Referenzen auf entfernte Host und verschiedene JREs?

Netzwerkkommunikation

Protokolle OSI-Layer-3

Wichtige Layer 3 Protokolle

- IPv4
- IPv6
- IPX

Adressierung bei IPv4

- 32bit Adresse als Identifizierung des Host
- Port zur Verbindung mit Anwendung
- Beispiel 192.168.0.2:8080

Netzwerkkommunikation

Protokolle OSI-Layer-4

Wichtige Layer 4 Protokolle

- Transmission Control Protocol (TCP) → verbindungsorientiert
- User Datagram Protokol (UDP) → verbindungslos

RMI

Hintergrund

Historische Wurzeln

- Remote Procedure Call (RPC) → RFC[1] 1057 und 5531
- Grundidee von RPC ist der Zugriff auf entfernte Funktionalitäten
- Grundidee von James E. White entstand 1976 → RFC 707

Eigenschaften von RMI

- Ermöglich Zugriff auf Objekte in fremden JREs und auf entfernten Hosts
- Netzwerkverbindung via TCP/IP
- Entwicklung ab J2SE 5.0 nicht abwärtskompatibel

[1] Request or Comments
http://www.ietf.org/rfc.html

Grundlegende Technik

Komponenten

Registry

- Verwaltungsinstanz für Objekte
- Port 1099 (Default)
- Zuordnung (Bind) von Objekt auf eindeutigen Namen

Server

- Gibt Objekte via Registry frei
- Zugriff auf Objekte über Randomport (Default)

Client

- Fragt Objektreferenzen auf Registry an
- Kenntnis der Objekte über Interfaces

Grundlegende Technik

Interface java.rmi.Remote

Funktionalität

- Muss mit dem funktionalen Interface des Objektes erweitert werden
- Markiert das implementierende Interface als Remote Interface
- Registry gibt nur Objekte vom Typ Remote zurück

Grundlegende Technik

Interface java.rmi.Remote

Funktionalität

- Muss mit dem funktionalen Interface des Objektes erweitert werden
- Markiert das implementierende Interface als Remote Interface
- Registry gibt nur Objekte vom Typ Remote zurück

Remote macht ein Objekt im Kontext RMI deklarativ nutzbar.

Grundlegende Technik

Klasse UnicastRemoteObject

Funktionalität

- Jedes konkrete RMI-Objekt muss UnicastRemoteObject erweitern (extends)
- Ermöglicht den Export eines Objektes an einen speziellen Port
- Jede Methode kann eine RemoteException werfen (throws RemoteException)

Grundlegende Technik

Klasse UnicastRemoteObject

Funktionalität

- Jedes konkrete RMI-Objekt muss UnicastRemoteObject erweitern (extends)
- Ermöglicht den Export eines Objektes an einen speziellen Port
- Jede Methode kann eine RemoteException werfen (throws RemoteException)

UnicastRemoteObject macht eine Objekt im Kontext RMI funktional nutzbar.

Serialisierung

Technik

Grundlegendes

- Standardtechnologie zum Übertragen von kompletten Objekten
- Objekte werden ein- und ausgepackt
- Anwendung bei Deep Copy, FileIO und Netzwerkübertragung

Serialisierung

Interface Serializable

Funktionalität

- Markerinterface
- Keine Methoden
- serialVersionUID zum Versionsabgleich

Serialisiert wird nicht

- Alle klassengebundenen Methoden und Attribute
- Inhalt von als transient markierten Attributen (sinnvoll bei gespeicherten Passwort)

Serialisierung

Interface Serializable

Funktionalität

- Markerinterface
- Keine Methoden
- serialVersionUID zum Versionsabgleich

Serialisiert wird nicht

- Alle klassengebundenen Methoden und Attribute
- Inhalt von als transient markierten Attributen (sinnvoll bei gespeicherten Passwort)

Serialisierung und Deserialisierung anhand von Klassen.

Sicherheit

Policy-File

Motivation im Kontext RMI

- Beschränkung von Zugriffen auf die JRE
- Beschränkung von Verbindungen zur JRE

Prinzip der Policy-Files

- Policy-Files . . . /jre/lib/security/java.policy und . . . /<USER_HOME>/.java.policy
- Zusätzliche Policy-Files beim ausführen von „java“als Commandline Parameter möglich
- Policytool von Java ermöglicht einfaches erstellen und bearbeiten der Files

Sicherheit

Policy-File

Motivation im Kontext RMI

- Beschränkung von Zugriffen auf die JRE
- Beschränkung von Verbindungen zur JRE

Prinzip der Policy-Files

- Policy-Files . . . /jre/lib/security/java.policy und . . . /<USER_HOME>/.java.policy
- Zusätzliche Policy-Files beim ausführen von „java“als Commandline Parameter möglich
- Policytool von Java ermöglicht einfaches erstellen und bearbeiten der Files

Policies addieren sich zu übergeordneten Policy-Files

Sicherheit

Beispiel Policy-File

```
1 grant codeBase "file:/home/teadmin/rmi_observer_test/
      RMIObserverTest_fat_server.jar"{
2   permission java.security.AllPermission;
3 };
```

Funktionalität

- Auf jar-Archiv angewendete Policy
- Komplette Rechtefreigabe

Sicherheit

Beispiel Policy-File

```
1 grant codeBase "file:/home/teadmin/rmi_observer_test/
      RMIObserverTest_fat_server.jar"{
2   permission java.security.AllPermission;
3 };
```

Funktionalität

- Auf jar-Archiv angewendete Policy
- Komplette Rechtefreigabe

Komplette Rechtefreigabe nur im Kontext des jar-Archivs sicherheitskritisch.

Sicherheit

Bind auf entfernten Hosts

Gefahrenquellen

- Liste der Registrierten Objekte einsehbar→Registrieren von schädlichen Objekten unter unschädlichem Namen
- Angriffe auf Registry-Host durch bind von vielen Objekten (Heap Size Exception)

Sicherheit

Bind auf entfernten Hosts

Gefahrenquellen

- Liste der Registrierten Objekte einsehbar→Registrieren von schädlichen Objekten unter unschädlichem Namen
- Angriffe auf Registry-Host durch bind von vielen Objekten (Heap Size Exception)

Binden von Objekten ausgehend von entfernten Hosts ist verboten.

Sicherheit

Bind auf entfernten Hosts

Gefahrenquellen

- Liste der Registrierten Objekte einsehbar→Registrieren von schädlichen Objekten unter unschädlichem Namen
- Angriffe auf Registry-Host durch bind von vielen Objekten (Heap Size Exception)

Binden von Objekten ausgehend von entfernten Hosts ist verboten.
Einfachste Lösung: Server und Registry auf gleichem Host realisieren

Anforderungen

Mindestanforderung an RMI-Objekte

Remoteinterface

```
1 public interface RemoteObjectInterface extends java.rmi.Remote{
2         public void RemoteMethod throws RemoteException();
3 }
```

Konkretes Remoteobjekt

```
1 public class ConcreteRemoteObject extends UnicastRemoteObject implements
      RemoteObjectInterface{
2         public ConcreteRemoteObject() throws RemoteException{
3                 super();
4         }
5
6         public void RemoteMethod throws RemoteException(){
7                 System.out.println("do something");
8         }
9 }
```

Anforderungen

Mindestanforderung an RMI-Server

RMI-Server mit Registry

```
1 public class RMIServer{
2         public RMIServer() throws RemoteException{
3                 Registry reg = LocateRegistry.createRegistry();
4                 reg.bind("RemoteObjectName",new ConcreteRemoteObject());
5         }
6 }
```

Anforderungen

Mindestanforderung an RMI-Ansprache

RMI-Client

```
1  public class RMIClient{
2          private String hostname = "localhost";
3          private int port = 8888;
4          public void fetchFromRegistry(){
5                  Registry reg = LocateRegistry.retrieveRegistry(hostname,port
                         );
6                  RemoteObjektInterface obj = (RemoteObjektInterface) reg.
                         lookup("RemoteObjektName");
7          }
8
9          public void fetchFromRegistryAlternative(){
10                 RemoteObjektInterface obj = (RemoteObjektInterface)Naming.
                         lookup("rmi://"+hostname+":"+port+"/RemoteObjectName")
                         ;
11         }
12 }
```

Anforderungen

Mindestanforderung an RMI-Daten

RMI-Daten

Unter RMI-Daten fallen:

- Parameter in Methoden
- Rückgabewerte von Methoden

```
1 public class RMIData implements Serializable{
2         private static final long serialVersionUID = 42L;
3         ...
4 }
```

Interfaces

Clean Code

Entwickle gegen Interfaces, nicht gegen Klassen

- Interfaces repräsentieren Konzepte, Klassen die konkrete Implementierung
- Ändern sich Details der Implementierung, bleibt das Konzept gleich
- Abgrenzung der Fehlerauswirkung im Fehlerfall

In der Implementierung weisst man dem Interface-Typ den Klassen-Typ zu.

```
1 InterfaceTyp obj = new KlassenTyp();
```

Interfaces

Clean Code - Beispiel

Beispiel Katze

- Besitzer und Tierarzt haben eine Katze vor sich sitzen
- Besitzer wird von seinem Haustier Katze gebissen, aber die Katze zeigt Reue
- Tierarzt wird von dem Patienten Katze gebissen, aber die Katze zeigt keine Reue

Auswirkungen

- Änderung an der Implementierung der Methode Patient.beisst() hat keine Auswirkung auf Katze.beisst()
- Klasse gehorcht dem SRP[2] und dem DIP[3]

[2] Single Responsibility Principle
[3] Dependency Inversion Principle

Interfaces

Clean Code - RMI

Einsatz bei RMI

- Konkrete Implementierungen stehen auf dem Server zur Verfügung
- Verschiedene Clients nutzen die Implementierungen nach gleichem Konzept

RMI erlaubt je nach Policy sehr vielen Usern auf Objekte zuzugreifen. Durch SRP und DIP wird die Änderungsauswirkung stark begrenzt.

Beispiel

Observer-Pattern

Resource Management im Testprozess

- Mehrere unterschiedliche Testumgebungen
- Tests sollen auf speziellen Umgebungen ausgeführt werden
- Zugriff auf die Umgebung soll zentral und automatisiert ablaufen
- Testausführung vom Rechner des Testers mittels Framework wie JUnit

Beispiel

Observer-Pattern

Resource Management im Testprozess

- Mehrere unterschiedliche Testumgebungen
- Tests sollen auf speziellen Umgebungen ausgeführt werden
- Zugriff auf die Umgebung soll zentral und automatisiert ablaufen
- Testausführung vom Rechner des Testers mittels Framework wie JUnit

Idee: Resourcemanager ist bei einer RMI-Registry registriert und informiert mittels Observer-Pattern die Tests über freie Ressource.

Implementierung in Java

Interface Observer

Interface Observer als Klassendiagramm

- Besitzt nur eine Methode für das Update
- Unterstützt Push und Pull

Abbildung: UML-Darstellung des Interfaces Oberver

Implementierung in Java

Klasse Observable

Klasse Observable als Klassendiagramm

- Basisklasse für alle Subjects
- Unterstützt Push und Pull
- Vor jedem Notify muss das Objekt als geändert markiert werden

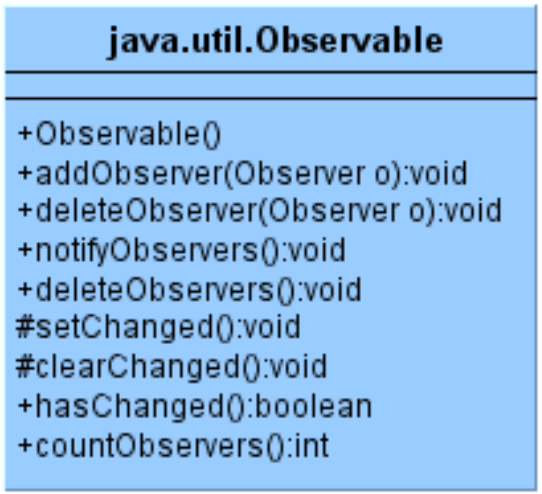

Abbildung: UML-Darstellung der Klasse Observable

Implementierung in Java

Probleme der Javaimplementierung

Observer

- Kein Remote
- Observer ist nur auf einem Host in einer JRE nutzbar

Observable

- Konkrete Implementierung
- Verstoß gegen SRP und DIP
- Alle Observables und Observer sind von Änderungen betroffen
- Kein Remote
- Observable ist nur auf einem Host in einer JRE nutzbar

Implementierung in Java

Probleme der Javaimplementierung

Observer

- Kein Remote
- Observer ist nur auf einem Host in einer JRE nutzbar

Observable

- Konkrete Implementierung
- Verstoß gegen SRP und DIP
- Alle Observables und Observer sind von Änderungen betroffen
- Kein Remote
- Observable ist nur auf einem Host in einer JRE nutzbar

Eigenimplementierung nötig.

Erweiterung für RMI

Definition der Remoteobjekte

Ein Remote stellt entfernte Funktionalität bereit

- Subject
- Observer

Implementierung

- Interfaces für Konzepte definieren
- Interfaces erweitern Remote
- Konkrete Klasse anhand der Interfaces implementieren
- Klasse muss UnicastRemoteObject erweitern und dessen Konstruktor benutzen

Erweiterung für RMI

Definition der Datenobjekte

Was sind alles Datenobjekte?

- Alle Rückgabewerte, die kein Remote sind
- Alle Typen der Übergabeparameter, die kein Remote sind

Implementierung

- Interfaces für Konzepte definieren
- Klassen anhand der Interfaces implementieren
- Datenobjekte müssen Serializable implementieren
- Interfaces der Konzepte sollten Serializable erweitern

Erweiterung für RMI

Klassendiagramm

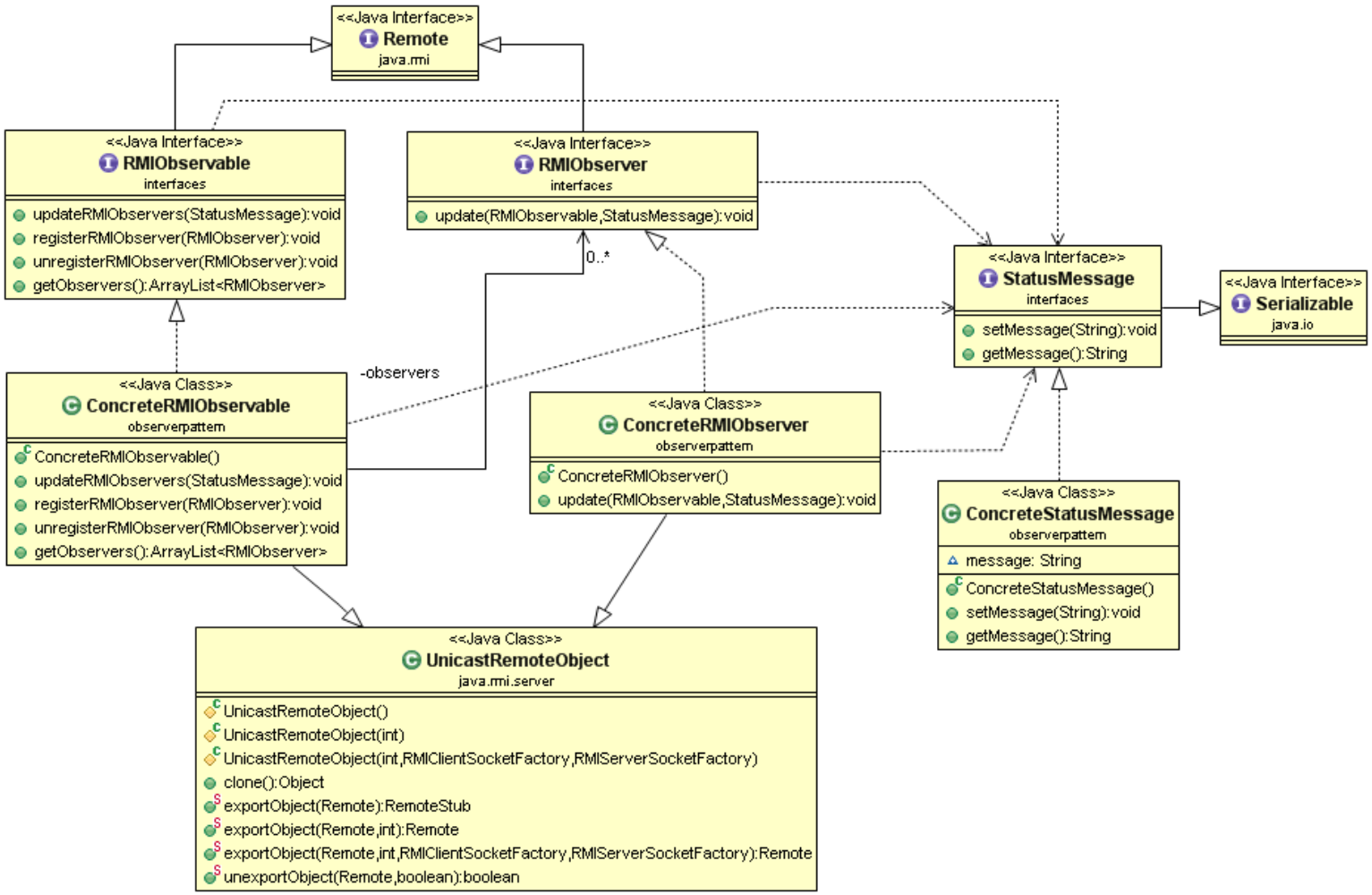

Abbildung: UML-Darstellung der Klasse Observable

Fehlerquellen

Häufigste Fehlerquellen

Interfaces und Klassen

- Remote wird nicht erweitert
- Serializable wird nicht erweitert
- Der Konstruktor von UnicastRemoteObject wird von erweiternden Klassen nicht benutzt

Netzwerk

- Netzwerk nicht verfügbar, langsam, instabil,...
- Sicherheitsrichtlinien im Routing
- Portfreigabe an Firewalls

Fehlerquellen

Häufigste Fehlerquellen

Interfaces und Klassen

- Remote wird nicht erweitert
- Serializable wird nicht erweitert
- Der Konstruktor von UnicastRemoteObject wird von erweiternden Klassen nicht benutzt

Netzwerk

- Netzwerk nicht verfügbar, langsam, instabil,. . .
- Sicherheitsrichtlinien im Routing
- Portfreigabe an Firewalls

Fehlererkennung und Vermeidung durch „gutes“Exception handling

Quellen

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: „Design Patterns. Elements of Reusable Object-Oriented Software.“, Addison-Wesley Longman, Amsterdam, 1st ed., Reprint. (31. Oktober 1994)
- Robert C. Martin, „Clean Code. A Handbook of Agile Software Craftmanship.“, Prentice Hal, 1st ed., (1.August 2008)
- Guido Krüger, Heiko Hansen: „Handbuch der Java-Programmierung“, Addison-Wesley, 5.Auflage, 2007
- Oracle, Java-API