

## J Überblick über die 9. Übung

J Überblick über die 9. Übung

- Remote Events
- ServiceEvents
- Security-Policies
- RMI Versionen

MW - Übung

Übungen zu Middleware  
© Universität Erlangen-Nürnberg • Informatik 4, 2006

J-Uebung9.fm 2006-12-18 13.46

J.1

## J.1 Distributed Events

J.1 Distributed Events

- Probleme
  - ◆ Netzwerkausfall
  - ◆ Reihenfolge
  - ◆ Event-Senke kann verschwinden
- Lösungen:
  - ◆ Exceptions
  - ◆ Sequenznummer
  - ◆ Lease
- zusätzlich: ein Identifikations-Objekt (Handback) vom Listener, welches bei jedem Event wieder mitgegeben wird.

MW - Übung

Übungen zu Middleware  
© Universität Erlangen-Nürnberg • Informatik 4, 2006

J-JINI++.fm 2006-11-27 12.00

J.2

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 1 RemoteEvent

J.1 Distributed Events

- Basisklasse aller Jini Events

```
package net.jini.core.event;
import java.rmi.MarshalledObject;

public class RemoteEvent implements Serializable {
    public RemoteEvent( Object source,
                       long eventID,
                       long seqNum,
                       MarshalledObject handback)

    public long getID();
    public long getSequenceNumber();
    public MarshalledObject getRegistrationObject();
}
```

- Enthält: Proxy des Senders, EventID, Sequenznummer, Identifikations-Objekt, welches bei der Registrierung angegeben wurde.

MW - Übung

Übungen zu Middleware  
© Universität Erlangen-Nürnberg • Informatik 4, 2006

J-JINI++.fm 2006-11-27 12.00

J.3

## 2 RemoteEventListener

J.1 Distributed Events

- Nur ein Interface, das von einer Event-Senke implementiert werden muß

```
public interface RemoteEventListener extends
    java.rmi.Remote, java.util.EventListener {

    public void notify(RemoteEvent theEvent)
        throws UnknownEventException,
        java.rmi.RemoteException;
}
```

- **UnknownEventException** bei unbekannten Events (z.B. Quelle und Typ passt nicht zusammen), der Sender sollte daraufhin den Versand dieses Eventtyps einstellen.

MW - Übung

Übungen zu Middleware  
© Universität Erlangen-Nürnberg • Informatik 4, 2006

J-JINI++.fm 2006-11-27 12.00

J.4

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

### 3 EventRegistration

J.1 Distributed Events

- Wird von der Event-Quelle bei der Registrierung zurückgegeben

```
package net.jini.core.event;
import net.jini.core.lease.Lease;
public class EventRegistration
    implements java.io.Serializable {
    public EventRegistration(    long eventID,
                                Object source,
                                Lease lease,
                                long seqNum);

    public long getID()
    public Object getSource();
    public Lease getLease();
    public long getSequenceNumber();
}
```

- Enthält: EventID, Quelle, Lease, aktuelle Sequenznummer
- Kein Standard-Interface für Registrierung!

Übungen zu Middleware

© Universität Erlangen-Nürnberg • Informatik 4, 2006

J-JINI++.fm 2006-11-27 12.00

J.5

### 4 Beispiel

J.1 Distributed Events

- sehr einfacher Beispiel-Code um ein Event anzubieten

```
protected RemoteEventListener listener = null;

public EventRegistration
    addRemoteListener(RemoteEventListener listener)
        throws java.util.TooManyListenersException {
    if (this.listener == null)
        this.listener = listener;
    else
        throw new java.util.TooManyListenersException();
    return new EventRegistration(myEvID, proxy, null, curSN);
}

protected void fireNotify(long eventID, long seqNum) {
    if (listener == null) return;
    RemoteEvent remoteEvent = new RemoteEvent(proxy, eventID,
                                                seqNum, null);
    listener.notify(remoteEvent);
}
```

Übungen zu Middleware

© Universität Erlangen-Nürnberg • Informatik 4, 2006

J-JINI++.fm 2006-11-27 12.00

J.6

### 5 ServiceEvent

J.1 Distributed Events

- Problem: Wie erfahre ich, wenn sich ein passender Service nach meiner Suche anmeldet?
- Lookup Service verschickt **ServiceEvents**
- Anmelden mittels notify

```
public interface ServiceRegistrar {
    public static final int TRANSITION_MATCH_NOMATCH = ...
    public static final int TRANSITION_NOMATCH_MATCH = ...
    public static final int TRANSITION_MATCH_MATCH = ...

    public EventRegistration
        notify(ServiceTemplate tmpl,
              int transitions,
              RemoteEventListener listener,
              java.rmi.MarshalledObject handback,
              long leaseDuration)
            throws java.rmi.RemoteException;

    ...
}
```

Übungen zu Middleware

© Universität Erlangen-Nürnberg • Informatik 4, 2006

J-JINI++.fm 2006-11-27 12.00

J.7

### 5 ServiceEvent (2)

J.1 Distributed Events

- Die Events werden über das Standard-Interface **net.jini.core.event.RemoteEventListener** entgegengenommen
- Im Event sind Informationen über die Änderung enthalten
- **net.jini.core.lookup.ServiceEvent**

```
public abstract class ServiceEvent
    extends net.jini.core.event.RemoteEvent
{
    public ServiceID getServiceID();
    public int getTransition();
    public abstract ServiceItem getServiceItem();
}
```

Übungen zu Middleware

© Universität Erlangen-Nürnberg • Informatik 4, 2006

J-JINI++.fm 2006-11-27 12.00

J.8

## 6 Beispiel

- Bei jeder Änderung benachrichtigen lassen:

```
public class RegistrarObserver implements RemoteEventListener {
    ...
    ServiceRegistrar registrar = ....;
    int transitions = ServiceRegistrar.TRANSITION_MATCH_NOMATCH |
        ServiceRegistrar.TRANSITION_NOMATCH_MATCH |
        ServiceRegistrar.TRANSITION_MATCH_MATCH;

    RemoteEventListener proxy =
        (RemoteEventListener) exporter.export(this);

    ServiceTemplate template = new ServiceTemplate(null,null,null);

    try {
        registrar.notify(template, transitions,
            proxy, null, Lease.ANY);
    } catch (RemoteException e) { e.printStackTrace(); }
    ...
}
```

## 7 Jini Browser

- Programm, um in Lookup Services zu suchen und aufzulisten

- Browser starten

```
java -cp ${JINI_HOME}/lib/browser.jar
-Djava.security.policy=my.policy
-Djava.rmi.server.codebase=
    http://webserver:4711/browser-dl.jar
    com.sun.jini.example.browser.Browser
```

- ◆ Browser registriert sich für ServiceEvents
- ◆ Lookup Service braucht Proxy, um Browser Events zuzustellen
- ◆ Proxy-Code muss über WWW-Server geladen werden

- Web-Server starten

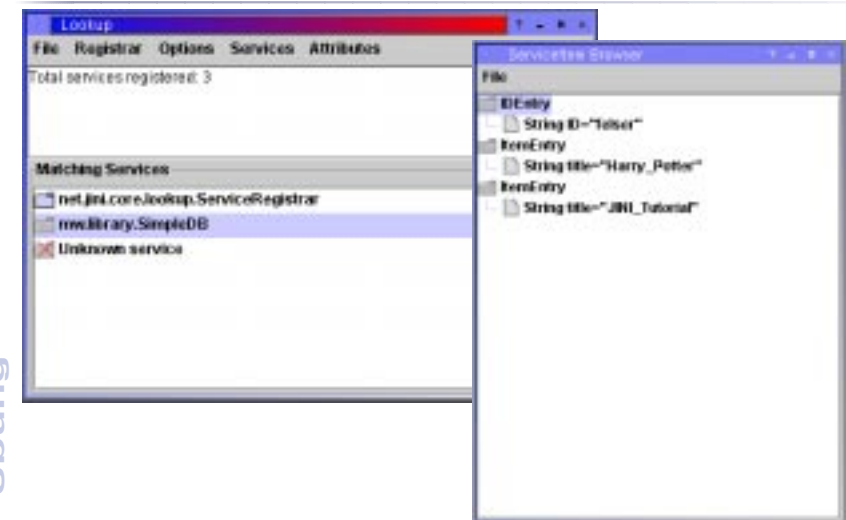
```
java -jar ${JINI_HOME}/lib/tools.jar
    -port 4711 -dir ${JINI_HOME}/lib/ -verbose
```

## 6 Beispiel (2)

- ServiceEvent auswerten:

```
public void notify(RemoteEvent evt)
    throws RemoteException, UnknownEventException {
    try {
        ServiceEvent sevt = (ServiceEvent) evt;
        switch (sevt.getTransition()) {
            case ServiceRegistrar.TRANSITION_NOMATCH_MATCH:
                System.out.println("nomatch -> match"); break;
            case ServiceRegistrar.TRANSITION_MATCH_MATCH:
                System.out.println("match -> match"); break;
            case ServiceRegistrar.TRANSITION_MATCH_NOMATCH:
                System.out.println("match -> nomatch"); break;
        }
        if (sevt.getServiceItem() == null) {
            System.out.println("now null");
        } else {
            Object service = sevt.getServiceItem().service;
            System.out.println("Service is " + service.toString());
        }
    } catch (Exception e) { e.printStackTrace(); }
}
```

## 7 Jini Browser (2)



## J.2 Security-Policies

- Legen das Verhalten des SecurityManagers fest
- Standard Policy in: `$JAVA_HOME/jre/lib/security/java.policy`
- Benutzer Policy in: `$HOME/.java.policy`
- Zusätzliche Policy wird mit der Eigenschaft `java.security.policy` angegeben

```
java -Djava.security.policy=URL Klassenname
```

## 1 Policy-File Syntax

- Zugriff erlauben

```
grant signedBy "signer_name1,s_n2,..", codeBase "URL" {
    permission permission_class_name "target_name", "action",
    signedBy "signer_names";
    ....
};
```

- Schlüsselquelle angeben (erzeugen mit Hilfe von `keytool`)

```
keystore "keystore_url", "keystore_type";
```

- Wem wird ein Recht gegeben
  - ◆ `signedBy`: das Recht nur signiertem Code (aus signiertem JAR-File) geben
  - ◆ `codeBase`: nur Code von einer bestimmten Quelle bekommt das Recht
- Welches Recht wird vergeben
  - ◆ spezifiziert durch Permission-Klasse und Argumente
  - ◆ `signedBy`: die Permission Klasse selbst muss signiert sein

## 2 Beispiele

- Jedem alles erlauben:

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

- Erlaubnis abhängig von der Herkunft des Codes

```
grant codebase "file:/local/java-lib/jini-2.1/lib/*" {
    permission java.security.AllPermission;
};
```

- ◆ Endung bestimmt, welchen Klassen die Erlaubnis gewährt wird:
  - .../lib/= alle Klassen, keine JAR-Archive
  - .../lib/\*= alle Klassen und JAR-Archive
  - .../lib/\*-= alle Klassen und JAR-Archive, auch in Unterverzeichnissen

## 2 Beispiele (2)

- Nur bestimmte Rechte geben

```
grant {
    permission java.net.SocketPermission
        "129.132.200.35", "connect,accept";
};
```

- genau spezifizierte Rechte

```
grant signedBy "sysadmin", codeBase "file:/home/sysadmin/*" {
    permission java.security.SecurityPermission
        "Security.insertProvider.*";
    permission java.security.SecurityPermission
        "Security.removeProvider.*";
    permission java.security.SecurityPermission
        "Security.setProperty.*";
};
```

## 1 Traditionelles RMI

### ■ implizit:

```
public class x extends UnicastRemoteObject implements Remote
{ ... }
```

### ■ explizit:

```
public class x implements Remote {
    public static void main(String[] args) throws Exception {
        Remote demo = new x();
        // this exports the RMI stub to the Java runtime
        RemoteStub stub =
            UnicastRemoteObject.exportObject(demo);
    }
}
```

## 2 Exporter

### ■ Nachteil des traditionellen RMI:

- ◆ kein explizites "unexport"
- ◆ keine Auswahl des zu verwendenden Protokolls: RMI over HTTP, IIOP, SSL
- ◆ explizites Erzeugen des Stubs und Skeletons

### ■ Exporter

- ◆ explizites Exportieren und "Unexportieren"
- ◆ verschiedene Aufrufprotokolle
- ◆ Beispiel:

```
Exporter exporter1 = new JrmpExporter();
Exporter exporter2 = new IiopExporter();

Remote proxy = exporter1.export(demo);

exporter1.unexport(true);
```

- ◆ Ein Exporter ist immer nur für ein Objekt verantwortlich

## 3 Jeri Exporter

### ■ Jini Extensible Remote Invocation

- ◆ Jini Trust-Modell
- ◆ Stubs müssen nicht mehr explizit erzeugt werden
- ◆ verschiedene Transportprotokolle (HTTP, HTTPS, SSL, TCP)
- ◆ flexibler GC
- ◆ bessere Konfigurierbarkeit

### ■ Konfiguration mittels zweier Parameter

- ◆ ServerEndpoint: Kommunikationsendpunkt des gewünschten Protokolls
- ◆ InvocationLayerFactory: Stub-, Skeleton Generator

### ■ Beispiel:

```
Exporter exporter = new BasicJeriExporter(
    TcpServerEndpoint.getInstance(0), new BasicILFactory());
```

## 3 Jeri: Exportierte Interfaces

### ■ Der Proxy implementiert alle Remote-Interfaces des Originalobjekts

### ■ Beispiel:



- ◆ IfaceImplProxy implementiert keine InnerInterfaces!

```
Iface iface = (Iface) ifaceImplProxy // class cast error
Iface iface = (Iface) ifaceRemoteImplProxy // okay
```

## 4 Konfiguration

J.3 RMI Proxies

### ■ Konfiguration

- ◆ des Transport-Protokolls (z.B. TCP, Firewire)
- ◆ des Aufrufprotokolls (z.B. IIOP, JRMP, Jeri)

zur Laufzeit möglich

### ■ Schnittstelle zu Konfigurationsdaten:

```
public interface Configuration {
    Object getEntry( String component,
                    String name, Class type)
    ...
}
```

- ◆ Ein Konfigurationseintrag besteht aus: Komponentename, Name und Klasse des zu konfigurierenden Objekts
- ◆ Implementiert z.B. in `net.jini.config.ConfigurationFile`

## 4 Konfiguration (2)

J.3 RMI Proxies

### ■ Configuration Objekt von einem ConfigurationProvider erzeugt

- ◆ ohne weitere Angaben wird ein ConfigurationFile Objekt erzeugt
- ◆ Beispiel:

```
Configuration config =
    ConfigurationProvider.getInstance(
        new String[]{"myconfig.file"});
```

### ■ In einer Konfigurationsdatei kann man dann die gewünschte Klasse angeben

### ■ Format einer Konfigurationsdatei:

```
import benötigte Packages
KomponentenName {
    Objektname = new Klassenname();
}
```

## 4 Konfiguration (3)

J.3 RMI Proxies

### ■ Beispiel einer Konfigurationsdatei:

```
import net.jini.jrmp.*;
import net.jini.iiop.*;
import net.jini.jeri.BasicILFactory;
import net.jini.jeri.BasicJeriExporter;
import net.jini.jeri.tcp.TcpServerEndpoint;

JrmpExportDemo {
    exporter = new JrmpExporter();
}
IiopExportDemo {
    exporter = new IiopExporter();
}
JeriExportDemo {
    exporter = new BasicJeriExporter(
        TcpServerEndpoint.getInstance(0),
        new BasicILFactory());
}
```

## 4 Konfiguration - Beispiel

J.3 RMI Proxies

### ■ Konfiguration aus Datei laden

```
import java.rmi.*;
import net.jini.config.*;
import net.jini.export.*;

String CONFIG_FILE = "jeri/jeri.config";
String[] configArgs = new String[] {CONFIG_FILE};

// get the configuration
Configuration config =
    ConfigurationProvider.getInstance(configArgs);

// use this to construct an exporter
Exporter exporter = (Exporter) config.getEntry(
    "JeriExportDemo",
    "exporter",
    Exporter.class);

// export an object of this class
Remote proxy = exporter.export(new ConfigExportDemo());
```