JINI

RIVER NETWORK ARCHITECTURE

Panoramica

JINI è un middleware (Java)

JINI è un'architettura di rete

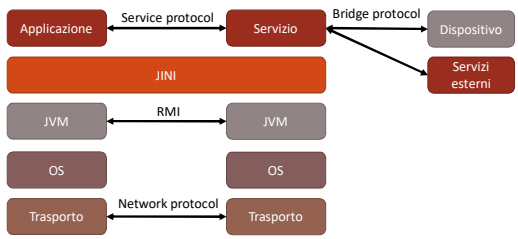
JINI è un modello di programmazione/progettazione

JINI **non** è un acronimo!

JINI offre supporto per la realizzazione di

- **Reti federate**:
  - Aggregazione (dinamica) di dispositivi e (sotto)reti
  - Dispositivi eterogenei (a patto che abbiano una JVM)
  - Gestione decentralizzata
- **Orientate ai servizi**:
  - Pubblicazione/ricerca/uso/composizione di servizi
  - Plug & Play

Panoramica



Servizi

I servizi sono il componente base di un sistema JINI
- I membri del sistema offrono servizi
- Cooperano per condividere l'accesso a tali servizi
  - Aggregatori
  - Proxy

I servizi sono combinati per svolgere compiti più complessi

Composizione dinamica, basata su specifiche

Popolazione variabile

Comunicazione specifica per ogni servizio

Infrastruttura

Modello di comunicazione basato su (java) RMI

Sistema di sicurezza distribuito
- Estende il modello di sicurezza della JVM
- Integrato con SM di RMI
- Autenticazione distribuita (e.g. Kerberos)

Protocolli di discovery/join
- Registrazione e scoperta di servizi

Servizio di lookup
- Ricerca di oggetti che implementano un dato servizio
- (proxy locali per interagire con il servizio)

Modello di programmazione

Discovery

Lookup

Leasing
- Le referenze hanno una componente temporale (expire)
- Rinnovo

Eventi distribuiti
- Estende il modello a eventi di JavaBeans
- Garanzie su delivery e timeliness

Transazioni distribuite
- 2PC

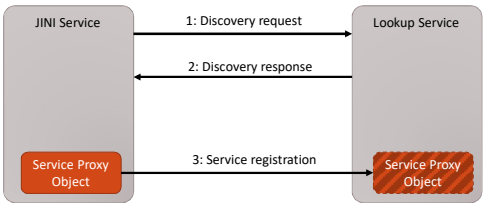
Servizio di Lookup

Descrive:
- Funzionalità
- Interfaccia
- (attribute-based search)

Il lookup stesso è un servizio:
- Può essere referenziato/aggregato da altri servizi
- Può interfacciarsi con directory non-JINI (cf. CORBA)

Registrazione
- Discovery: fase di ricerca di un servizio di lookup
- Join: registrazione del nuovo servizio presso tale servizio di lookup

Registrazione di un servizio



DiscoveryListener

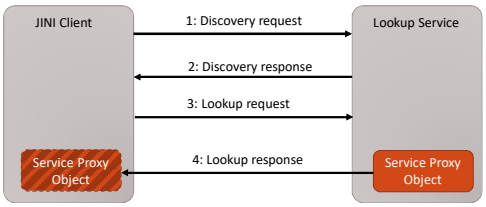
Discovery e Join sono gestiti internamente (mediante la calsse LookupDiscovery)

Per offrire un servizio basta implementare:
- `net.jini.discovery.DiscoveryListener`

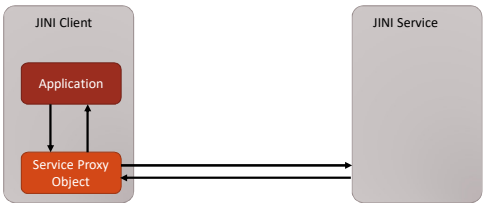
Le callbacks:
- `discovered(DiscoveryEvent dev)`
  - Invocato da JINI al rilevamento di nuovi servizi di lookup
- `discarded(DiscoveryEvent dev)`
  - Invocato da JINI quando un servizio di lookup presso cui il servizio è registrato non è più raggiungibile

DiscoveryEvent.getRegistar() fornisce l'elenco dei servizi presso cui potersi registrare.

Ricerca di un servizio



Invocazione di un servizio



Leasing

Lease:
- Parte delle (JINI) reference
- garantisce il diritto di accesso ad un servizio per un dato periodo di tempo

Negoziato

Parte del protocollo di servizio

Consente la de-allocazione unilaterale delle risorse
- (resistenza ai fallimenti?)

Può essere
- Esclusivo o meno
- Delegabile

Utile per servizi aggiornabili

Transazioni

JINI supporta le transazioni distribuite (2PC)

Le operazioni offerte da un servizio possono essere incapsulate in transazioni

Tuttavia:
- La semantica è delegata all'implementazione delle operazioni
- JINI non specifica il comportamento di Commit e Abort
- Nulla vita che Abort modifichi lo stato del sistema

Eventi

L'infrastruttura JINI supporta gli eventi

JINI gestisce la notifica/propagazione di questi

Tuttavia, gli eventi possono giungere
- Più volte
- Fuori ordine
- Non arrivare proprio

Semantica demandata al programmatore
- (qualche helper/util)

Controllo soft-realtime mediante leasing

Eventi

Un evento è un'istanza di **RemoteEvent**
- Si tratta di un oggetto remoto
- Lo stato dell'evento è centralizzato
- Riduce l'impatto di eventi «voluminosi»

Come per eventi «locali» si utilizzano ascoltatori i.e. istanze di **RemoteEventListener**

Un oggetto espone eventi e.g. mediante metodi
- addEventListenerXXXX(RemoteEventListener)
- removeEventListenerXXXX(RemoteEventListener)

Implementazioni

Implementazioni:
- Apache River
- GigaSpaces
- Blitz

Non solo RMI
- SOAP
- TCP
- UDP
- CORBA

JavaSpaces

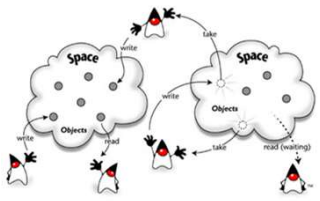
JAVA TUPLE SPACES

Panoramica

Spazi di tuple (entry)

Operazioni:
- Write
- Read
- Take

Varianti
- _ifExists
- Timeout
- Async
- Transactions



Entry

Le tuple sono istanze di `net.jini.core.entry.Entry`

Devono avere un costruttore pubblico privo di argomenti

I campi pubblici sono gli elementi della «tupla»

```
public class PersonEntry implements Entry{
        private int version  = 123;
        public String name;
        public int age;

        public PersonEntry(){}
        public String toString(){…}
}
```


Vs Linda

Le tuple (o entries) sono tipate
- Sottotipi
- Polimorfismo
- Name-based (vs position-based)
- Contengono oggetti
- Espongono metodi

Spazi multipli

Transazioni su più spazi

Leasing

GC

Non esiste un equivalente di *eval*

Matching

Read e take accettano templates:
- Sono entry
- Campi null o vettori vuoti sono wildcard

Vengono considerati solo i campi pubblici (no side-effect)

In presenza di più match il risultato è non-deterministico.

(esempio…)

L'ordine di valutazione dei campi deve essere ininfluente

Peculiarità

Spazi multipli

È possibile replicare interi spazi

Gli spazi possono essere persistenti (e.g. DB)

Le operazioni (anche tra più spazi) possono essere raggruppate in transazioni

Mobilità del «codice»
- Entry espongono metodi
- Campi contengono oggetti che espongono metodi
- Oggetti remoti
- Immutabili all'interno dello spazio (invocazione in locale)

Eventi remoti (JINI)

Eventi

JavaSpaces supporta eventi distribuiti

Sfrutta l'infrastruttura JINI

I listener sono associati a template
- Ricevono solo eventi (write, …) relativi a entry che «matchano»

Supporto a leasing e transazioni
- Listener limitati al contesto di una transazione
- Notify non tiene traccia delle transazioni (lasciato alle specifiche impl.)

API dello spazio

```
package net.jini.space;
…
public interface JavaSpace {
  public final long NO_WAIT = 0;

  Lease write( Entry e, Transaction txn, long lease )
  Entry read( Entry tmpl, Transaction txn, long timeout )
  Entry take( Entry tmpl, Transaction txn, long timeout )
  EventRegistration notify( Entry tmpl, Transaction txn,
    RemoteEventListener listener, long lease,
    MarshalledObject handback )
}
```

Esercizi

1. Implementare JavaSpace
2. Chat federata e distribuita
   1. Le singole stanze possono essere centralizzate (riutilizzate la chat RMI)
   2. Servizi di directory
   3. Ricerca stanze per argomento (sfruttate javaspaces)
3. Composizione dinamica dei servizi
   1. Autenticazione
   2. Replicazione
   3. Storico