Servers Activatable

- Nelle lezioni precedenti abbiamo detto che una referenza remota ad un server di tipo `UnicastRemoteObject` rimane valida finchè il server è in esecuzione.
- Quando il server termina, o viene de-esportato, la referenza diventa inutilizzabile. Ovvero, un ulteriore utilizzo da parte del client produce una `ConnectException`, a segnalare che la connessione è rifiutata.
- Tale inconveniente viene superato lavorando con server `Activatable`. Una referenza remota ad un server Activatable è valida anche quando il server non è in esecuzione. Ovvero può essere riutilizzata nel momento in cui il server viene rilanciato.
- Ogni volta che si usa uno stub di un server attivabile per invocare un metodo del server associato, il *sistema di attivazione* si prende cura di rilanciare il server nel caso in cui questi non sia attivo.

Demone di attivazione RMID

- Questo per lo meno finchè il server attivabile rimane registrato col *sistema di attivazione*, ovvero il demone di attivazione che si occupa di gestire i server attivabili e che viene lanciato col comando `rmid`.
- Il demone di attivazione è un server RMI che implementa l'interfaccia remota `ActivationSystem`.
- Lo scopo di RMID è quello di lanciare (una o più) JVMs in cui verranno lanciati "gruppi" di server Activatable.
- Poichè il demone di attivazione è esso stesso un server remoto, esiste un meccanismo di bootstrap per ottenere una sua referenza invocando il metodo statico `ActivationGroup.getSystem`.

Stub di un server Activatable

- Stub a server unicast contengono tutta l'informazione necessaria per invocare i metodi di un particolare server.

- Stub a server attivabili oltre a contenere tale informazione, mantengono anche una referenza al demone d'attivazione. Per essere precisi uno stub di un server attivabile contiene:
  - un (oggetto di tipo) `RemoteRef`
  - un (oggetto di tipo) `ActivationID`.
- L'istanza di `ActivationID` è un identificatore globale unico che contiene:
  - una referenza (ovviamente remota) al demone d'attivazione
  - un'istanza di `ObjectID` per identificare il server attivabile in questione.
- Tali stub possono essere salvati in un file ed essere riutilizzati anche dopo uno shut-down.
- La comunicazione client-server funziona nel seguente modo:
  1. Quando viene creato uno stub di un server attivabile, esso contiene informazioni sul demone di attivazione (all'interno

del campo `ActivationID`), ma non sul server di cui è una referenza (il campo `RemoteRef` è a null).

2. Quando un client ottiene per la prima volta lo stub ad un server attivabile (ad esempio facendo una lookup sul registro RMI), non sa che lo stub ottenuto in realtà non è ancora una referenza al server attivabile.

3. La prima volta che il client invoca un metodo remoto attraverso lo stub, esso in realtà non invoca il server (che potrebbe non essere stato ancora lanciato), bensì il demone di attivazione (la cui referenza si trova nel campo `ActivationID`) chiedendogli informazioni sul server desiderato.

4. Quando il demone di attivazione viene interrogato a proposito di un server, prima controlla se il server è stato già lanciato. Se non è stato lanciato allora si prende cura che venga lanciato.

5. Più precisamente, in fase di set-up il sistema di attivazione avrà lanciato una o più JVM, ed all'interno di ciascuna JVM sarà stato lanciato un *gruppo di attivazione*. Un gruppo di attivazione è anch'esso un server RMI, della classe

`ActivationGroup`, ed è proprio il gruppo di attivazione che ha il compito di attivare i server attivabili che gli competono.

6. Una volta lanciato il server attivabile, il demone ritorna allo stub (ovvero al client) una `RemoteRef` per contattare il server direttamente.

7. In tal modo, dopo la prima invocazione, lo stub non comunicherà più col demone d'attivazione, e tutte le invocazioni remote saranno inoltrate direttamente al server.

Modifiche per l'attivazione

Vediamo adesso quali sono i cambiamenti da apportare nel codice per realizzare un server attivabile. Essenzialmente è necessario fare due cose:

- modificare il server in modo da renderlo `Activatable`;

- modificare il codice di lancio (ovvero la procedura main).

Rendere un server `Activatable`

- Fino adesso abbiamo usato la classe `UnicastRemoteObject` per collegare un server col sistema RMI.

- Questo può avvenire o estendendo `UnicastRemoteObject` oppure utilizzando uno dei metodi `UnicastRemoteObject.exportObject`.

- Per collegare un server con il *sistema di attivazione* RMI, possiamo fare in maniera simile:

  - o estendiamo la classe `Activatable`, e nel costruttore del server invochiamo il costruttore della superclasse. Si noti che il costruttore non può essere omesso e come prima istruzione deve avere un'invocazione al costruttore della superclasse.

  - oppure usiamo uno dei metodi `Activatable.exportObject`.

- I costruttori della classe `Activatable` hanno la seguente segnatura:

```
protected Activatable(ActivationID id, int port)
                      throws RemoteException;
protected Activatable(ActivationID id, int port,
                      RMIClientSocketFactory csf,
                      RMIServerSocketFactory ssf)
                      throws RemoteException;
protected Activatable(String location,
                      MarshalledObject data,
                      boolean restart,
                      int port,
                      RMIClientSocketFactory csf,
                      RMIServerSocketFactory ssf)
                      throws RemoteException;
```

- Col primo costruttore si attiva ed esporta un server attivabile su una data porta. Mettendo la porta a 0 si lascia al sistema di attivazione la scelta della porta su cui esportare il server. Il campo di tipo `ActivationID` è l'identificativo di attivazione dell'oggetto.
- Il secondo costruttore fornisce i socket per client e server.

- Il terzo costruttore oltre ai socket e la porta contiene un parametro `location` per definire la locazione da cui vanno caricate le classi del server; `data` fornisce il valore iniziale del server; infine, se `restart` è posto a 'true' il server è riattivato ogni qual volta viene riattivato il sistema di attivazione (magari a seguito di un crash); se è posto a 'false' la riattivazione avviene solo a seguito di un'invocazione remota da parte del client.

- I metodi per esportare server Activatable ritornano uno stub al server ed hanno la seguente segnatura:

```
public static Remote exportObject(Remote obj,
                                  ActivationID id,
                                  int port)
                                  throws RemoteException;
public static Remote exportObject(Remote obj,
                                  ActivationID id,
                                  int port,
                                  RMIClientSocketFactory csf,
                                  RMIServerSocketFactory ssf)
```

```
                                      throws RemoteException;
public static Remote exportObject(Remote obj,
                                  String location,
                                  MarshalledObject data,
                                  boolean restart,
                                  int port)
                                  throws RemoteException;
```

- Nota che un `MarshalledObject` è una wrapper class; un'istanza di `MarshalledObject` ha solo uno scopo: incapsulare un'istanza di un'altra classe serializzabile.

- Istanze di MarshalledObject sono create passando appunto un'istanza di una classe serializzabile.

- Una volta creata, un'istanza di MarshalledObject ha un solo metodo `get()` per recuperare l'istanza incapsulata.

- MarshalledObject sono espressamente usati per passare dati a server che verranno lanciati in un secondo tempo.

Come modificare il codice di lancio

Il codice di lancio (Setup) di un server `Activatable` è in generale più complesso di quello di un server `UnicastRemoteObject`. L'operazione contiene 5 differenti fasi:

1. Vengono creati uno o più *gruppi di attivazione* (activation group). Ciascun gruppo di attivazione andrà in esecuzione su una JVM differente; sarà cura del gruppo d'attivazione di lanciare (all'interno di quella JVM) uno o più server attivabili. Un gruppo di attivazione è descritto da un oggetto di tipo `ActivationGroupDesc` (activation group descriptor) che contiene le informazioni necessarie al sistema di attivazione per lanciare la JVM del gruppo.

2. Ciascun gruppo di attivazione deve essere registrato col sistema di attivazione. L'operazione di registrazione ritorna un oggetto `groupID`, di tipo `ActivationGroupID`, che identifica il gruppo di attivazione.

3. Dopodichè, per ogni server attivabile, si crea un oggetto di tipo `ActivationDesc` (activation descriptor). Un activation descriptor

contiene 4 informazioni fondamentali che caratterizzano un server attivabile:

(a) Il `groupID` del gruppo a cui appartiene il server attivabile;

(b) Il nome della classe dell'implementazione del server attivabile;

(c) Il codebase da cui il gruppo d'attivazione dovrà caricare il codice del server ogni volta che dovrà lanciarlo;

(d) Un marshalled object che può contenere parametri di inizializzazione del server.

4. Ciascun activation descriptor dovrà essere registrato col sistema di attivazione.

5. Infine, se necessario, si possono registrare gli stub dei server attivabili su un registro RMI.

Vediamo allora quale potrebbe essere un esempio di codice di lancio (Setup) in cui viene creato un gruppo di attivazione, e un server attivabile all'interno del gruppo. Ovviamente, potrebbero esservi più gruppi di attivazione con più server attivabili.

Il programma di Setup potrebbe essere lanciato nel seguente modo:

```
java -classpath .....
     -Djava.security.policy=setup.policy
     -Djava.rmi.server.codebase= ...dove stanno le interfaccie remote..
     -Dactivation.impl.codebase=...codebase del codice del server att...
     -Dactivation.classeserver= ‘‘activation.ServerSuperPippo’’
     -Dactivation.policy=group.policy
     activation.Setup
```

Dove:

- setup.policy è il file di policy del codice di Setup;
- activation.ServerSuperPippo è il nome completo della classe del server attivabile;
- group.policy è il file di policy del gruppo d’attivazione.

```
public class Setup{.......................
 public static void main(String[] args) throws Exception {
   MarshalledObject data =...;  // Valore iniziale del server
   String policyGroup  = System.getProperty(``activation.policy'');
   String implCodebase = System.getproperty(``activation.impl.codebase'');
   String class = System.getproperty(``activation.classeserver'');
   try {Properties pr = new Properties();
     pr.put(``java.security.policy'', policyGroup);
     pr.put(``activation.impl.codebase'', implCodebase);
     pr.put(``java.class.path'', ``no_classpath'');
     ActivationGroupDesc groupD = new ActivationGroupDesc(pr,null);
     ActivationGroupID groupID =
                 ActivationGroup.getSystem().registerGroup(groupD);
     ActivationDesc actD =
                 new ActivationDesc(groupID,class,implCodebase,data);
     IntRemota stub = (IntRemota)Activatable.register(actD);
     Naming.bind(``SuperPippo'',stub);}
   catch (Exception e) {........}
 }}
```

Esaminiamo il codice dentro il comando “try”.

1. **Fase 1** Le prime cinque linee di codice servono a creare un oggetto di tipo `ActivationGroupDesc` (activation group descriptor) che conterrà l’informazione necessaria per creare/ricreare un gruppo di attivazione. Utilizzeremo un’istanza della classe `Properties` per definire le proprietà di startup della VM in cui verrà lanciato il gruppo di attivazione. Nel nostro caso tale istanza conterrà

    - informazioni sul file di policy con cui dovrà essere lanciato il gruppo d’attivazione,
    - il codebase in cui è contenuta l’implementazione del server, e che verrà menzionato nel file di policy *group.policy* per dare i permessi al server di attivazione.
    - “azzeramento” del classpath per evitare che il gruppo di attivazione tenti di caricare il codice dell’implementazioni dei server dal classpath locale.

2. **Fase 2**: Si registra il gruppo di attivazione col sistema di attivazione (rmid) passando l’istanza di `ActivationGroupDesc` appena

creata, ed ottenendo come risultato un'istanza di `ActivationGroupID` che identifica il gruppo d'attivazione creato. Tale referenza al gruppo d'attivazione può essere usata per registrare oggetti all'interno del gruppo, oppure per de-registrare il gruppo dal sistema d'attivazione. La referenza al gruppo d'attivazione va salvata, tipicamente in un file, se si vuole evitare di creare gruppi all'infinito.

3. **Fase 3** Viene creata un'istanza di `ActivationDesc` che contiene la descrizione del server attivabile da lanciare, caratterizzato da:

    - L'identificatore del gruppo.
    - Il nome assoluto della classe (cioè comprensiva di eventuali packages).
    - Il codebase da dove caricare la classe del server al momento dell'attivazione.
    - Un `MarshalledObject`, eventualmente a null, che incapsula un'istanza con cui inizializzare il server.

    Si noti che un server specificato da un'istanza di `ActivationDesc` sarà attivato solo previa invocazione di un client e non a seguito di una riattivazione del sistema di attivazione (di default il parametro `restart` è messo a 'false').

4. **Fase 4** Viene registrato il server attivabile col sistema di attivazione passando l'activation descriptor e ottenendo come risultato uno stub al server remoto.
   - Ciò corrisponde ad un'invocazione remota al demone di attivazione, al quale viene passato una copia dell'ActivationDesc del server attivabile contenente l'informazione necessaria per lanciare il server attivabile.
   - Si noti che lo stub che viene tornato tornato non è altro che un'istanza della classe stub del server attivabile. Tale stub, con versioni java precedenti alla 5.0 devono essere state preventivamente create attraverso la compilazione `rmic`.
   - Lo stub rappresenta una referenza remota persistente. Comunque, per renderla veramente tale è opportuno che il client la memorizzi in un file.

- Il risultato di una registrazione è persistente ad invocazioni di `rmid`. Le registrazioni sono tenute nel database di `rmid` e vi rimarranno finchè il database esiste o il server viene de-registrato.
- A differenza dei server unicast, la de-registrazione di un server dal sistema di attivazione può essere fatta solo dal server stesso.

5. **Fase 5** Eventuale registrazione del server attivabile su un registro RMI.

Il comando `rmid`

Il demone di attivazione `rmid` può essere lanciato settando differenti parameri: `-port`, `-log`, `-stop` `-J`, `-C`. I primi quattro influenzano il demone di attivazione, mentre il quinto influenza le JVMs lanciate dal demone di attivazione.

- **-port**
  - Abbiamo detto che in ogni JVM lanciata dal demone di attivazione viene lanciato un gruppo d'attivazione (esso stesso un server RMI) che si occupa di gestire i dettagli relativi alla creazione di nuovi server attivabili all'interno della JVM.
  - Il demone di attivazione deve comunicare con i vari gruppi di attivazione i quali hanno bisogno della referenza remota del demone di attivazione.
  - Per tale ragione, col comando `rmid`, insieme al demone di attivazione viene lanciato un registro RMI in cui il demone di attivazione si autoregistra col nome

`java.rmi.activation.ActivationSystem`.

  - La proprietà `-port` specifica appunto su quale porta viene messo in ascolto il registro RMI lanciato col demone di attivazione (ad esempio `rmid -port 2067`).
  - Se la porta non viene specificata il valore di default è 1098.
  - Perciò il metodo `ActivationGroup.getSystem`, visto in precedenza, che fornisce la referenza remota del demone di attivazione non fa altro che una `lookup` su tale registro.

- **-log**

  - Specifica il nome della directory su disco che `rmid` usa per scriverci sopra il suo database con le informazioni sulle attivazioni. Ciò non può essere evitato.
  - Se non viene usato il parametro -log i file di log verranno piazzati nella directory corrente dentro una directory con nome "log".
  - Utilizzando il parametro -log è possibile specificare il nome di questa directory.

- **-stop** Tale parametro arresta il demone di attivazione.
- **-J** Specifica un'opzione passata all'interprete java che lancia rmid. Particolarmente importante è il passaggio del file di policy con cui si lancia RMID.
  - A partire da JDK 1.3, per sopperire ad una falla nel sistema di sicurezza di RMI, rmid deve essere lanciato specificando un file di policy:

    `rmid -J-Djava.security.policy=`rmid.policy

    Dove rmid.policy è il file di policy per il demone rmid.
  - Un alternativa all'opzione sopra indicata, da utilizzare solo in fase di sviluppo del software, è:

    `rmid -J-Dsun.rmi.activation.execPolicy=none`

    In cui non viene garantita la sicurezza ma consente di lanciare le applicazione senza eccezioni.
  - Se non si adotta una delle due soluzioni mostrate sopra, l'applicazione del client andrà incontro ad una

`java.security.AccessControlException`

per mancanza di un `ExecPermission` oppure di un `ExecOptionPermission`.

– Se invece si costruisce un file di policy per rmid avremo bisogno di fornire `ExecPermission` a tutti gli eseguibili usati dal demone di attivazione; normalmente solo `java` e `java_g` (durante lo sviluppo).

– Inoltre bisognerà fornire `ExecOptionPermission` per tutte le proprietà di sistema specificate quando si definiscono i gruppi di attivazione, incluso il `java.security.policy` file per il gruppo.

Un esempio di security policy file per rmid potrebbe essere il seguente:

```
grant {
     permission com.sun.rmi.rmid.ExecPermission
                    ``/usr/java/j2sdk1.4/bin/java'';

     permission com.sun.rmi.rmid.ExecPermission
                    ``/usr/java/j2sdk1.4/bin/java_g'';

     permission com.sun.rmi.rmid.ExecOptionPermission
            ``-Djava.security.policy=/home/...../group.policy'';

     permission com.sun.rmi.rmid.ExecOptionPermission
            ``-Djava.security.debug=*'';

     permission com.sun.rmi.rmid.ExecOptionPermission
            ``-Dsun.rmi.*'';
}
```

  - I primi due permessi consentono a `rmid` di eseguire i comandi `java` e `java_g` di java 1.4.
  - Il terzo permesso consente a `rmid` di lanciare i gruppi di attivazione con policy file uguale a group.policy.
  - Il quarto permesso consente ai gruppi di attivazione di usare la proprietà `java.security.debug`.
  - Similmente per il quinto permesso.

- **-C** Questo è un meta-argomento che non influenza il demone di attivazione. In pratica, il suo valore è passato a tutti i processi figli di `rmid`, ovvero i gruppi di attivazione. Nota che non vi è spazio tra -C ed il comando passato. Ad esempio per abilitare i logs in ciascuna JVM si potrebbe scrivere:

```
rmid -J-Djava.security.policy=rmid.policy
     -C-Djava.rmi.server.logCalls=true
```

Infine può essere utile lanciare il demone `rmid` con la proprietà di sistema `sun.rmi.activation.debugExec` messa a `true`. In questo modo il sistema di attivazione visualizza tutti i tentativi di attivare gruppi e server e tutte le eccezioni che ne possono derivare.

Lo shutdown di un server attivabile

Per fare lo shut-down di un server attivabile è necessario fare due cose:

- De-esportare il server. Tale operazione è simile a quella vista per server unicast. Così facendo si comunica al sistema RMI che il server non è più in ascolto su una porta per soddisfare invocazioni remote.
- Informare il demone di attivazione che il server è inattivo e quindi successive invocazioni al server possono essere soddisfatte solo dopo che il server sia stato riattivato dal demone di attivazione.

Entrambe queste operazioni vengono gestite usando metodi statici di `Activatable`. In particolare:

```
public static booelan unexportObject(Remote obj, boolean force)
public static boolean inactive(ActivationID id)
```

- Il primo metodo de-esporta il server. L'argomento force consente di avere un controllo sulla de-esportazione. Se tale parametro è a 'false' e vi sono invocazioni remote pendenti, la de-esportazione fallisce. Se il parametro force è messo a 'true', la de-esportazione ha successo pur essendovi un'invocazione remota pendente alla quale viene ritornata una `RemoteException`.

- Il metodo `inactive()` comunica al demone di attivazione che il server non è più attivo. Ciò consente al demone di attivazione di resettare opportunamente i suoi registri. In tal modo, se un client invoca un metodo del server che è stato appena de-esportato non si avrà un'eccezione poichè il server verrà riattivato dal sistema di attivazione.

L'interfaccia Unreferenced

- Anche un server attivabile può implementare l'interfac. `Unreferenced` e scrivere un metodo `unreferenced`. Non solo, è anche raccomandabile farlo.
- Infatti in caso di assenza di client il metodo unreferenced verrà in-

vocato dal sistema RMI consentendo di implementare una stategia "exit when idle", ovvero:

1. Il server attivabile viene attivato quando riceve la prima invocazione remota.
2. Rimane attivo finchè vi sono remote clients.
3. Uscirà quando il numero dei clienti va a zero.
4. Verrà attivato nuovamente a seguito di una nuova invocazione remota.

La struttura del metodo unreferenced può essere la seguente:

```
public void unreferenced () {
  try {
      //prima rimuovi le referenze locali
      if (Activatable.unexport(server, true)){
          Activatable.inactive(server.getID());
      }
      // Se poi si vuole che il server non venga piu' riattivato
```

```
    // dal demone di attivazione a seguito di un'invocazione
    // da parte di un client, allora bisogner\`a de-registrarlo
    if (Activatable.unexport(server, true){
        Activatable.unregister(server.getID());
    }
    // a questo punto si puo' lanciare il garbage collector locale
}
catch (RemoteException e) {
.......
}
```

Quali server dovrebbero essere Activatable?

Quelli che un client incontra all'inizio della sua interazione con un servizio e che non vengono frequentemetne invocati dai clients.

Clients di server attivabili

I clients di server attivabili sono sostanzialmente indistinguibili dai clients di server unicast. L'unica differenza è che possono incontrare:

1. **Ritardi nell'attivazione.** Tali server devono tener conto di interagire con un server attivabile nel caso in cui intendano settare dei timeout. Quando un client invoca un server che non è attivo il sistema di attivazione deve avere il tempo di attivarlo. In generale tale operazione è sufficientemente veloce. Comunque, se anche il gruppo di attivazione relativo al server non è in esecuzione allora dovrà prima essere lanciato il gruppo, la qualcosa è ben più pesante e richiede un po' più di tempo.

2. **Eccezioni d'attivazione.** Le `ActivateException` estendono `RemoteExeception`.

    - `UnknownGroupException` e `UnknownObjectException` indicano che il client sta usando delle referenze attivabili non valide, ovvero che non si riferiscono ad alcun gruppo di attivazione o server attivabile conosciuto dal sistema di attivazione.

- `ActivateFailedException` indica che il server non può essere attivato.