Java による分散プログラミング　～Jini,.～

Jini はこの分散オブジェクトプログラミングをベースに、いろいろなコンピュータ、家電に入っているプロセッサからスーパーコンピュータまで、ネットワーク上のあらゆる機器（コンピュータ）を「連合(federation)」させるための仕組みを提唱したものである。たとえば、いろいろな家電製品にはいまやプロセッサが入っているが、これをネットワークにつなぎ、RMI（というか、RMI で提供されている標準のプロトコルと Jini によって提供されるサービスの検索機能）でつなぐことによって、いろいろな家電を統一的に制御したり、利用したりできるようになる。Jini のもっとも重要な概念として「サービス」がある。ネットワーク上に接続されているコンピュータを単なるデータを交換する対象と考えるのではなく、なんらかのサービスを提供する対象と考える。そのサービスをお互いに交換することによって、分散システムはなんらかの仕事をする。これまで、いわゆるサーバはサービスを提供する担い手であり、クライアントはそのサーバからサービスを受ける形態が一般的であったが、Jini が想定しているのはネットワーク上の分散システムを構成するコンピュータがお互いにサービスを提供することによって協調作業をするシステムを想定している。

Jini でサービスをネットワーク上のどこからでも利用できる。サービスはネットワーク上を移動するオブジェクトによって提供される。いろいろなサービスがあるとする Jini では、そのサービスを見つけるための機構「Lookup サービス」が提供されている。これによって、ネットワーク上に提供されているサービスを検索し、そのサービスを利用できる。これについては、たとえば DHPC を考えるとわかりやすい。いまでは、ノート PC を単にケーブルを接続するだけで、ネットワークに参加できるが、ケーブルを接続したときにまず、ネットワークのアドレスを管理している DHPC サーバを検索し（これが Lookup、つまり DHCP のサービスの検索）、標準のプロトコルでアドレスやネットマスク、DNS などのアドレスを取得する。また、サービスを提供する側は、Lookup サービスに登録することを Join と呼んでいる。

RMI は個々のコンピュータで提供するオブジェクトを管理する（registry）機能を提供しているが、Jini はこれをネットワーク全体に拡張し、すべてのコンピュータで提供されている機能を検索する機能を提供するものということもできる。

Jini

Jini は以下の３つの部分からなっている。

1. JCP(The Jini Technology Core Platform)
2. JXP (The Jini Technology Extended Platform)
3. JSK(The Jini Technology Software Kit)
4. JSTK(The JavaSpace Technology Kit)

ここでは、上の３つを用いる。Jini を使う前に、以下の手順で Lookup サービスを立ち上げる。

- HTTP サーバの立ち上げ：Jini 自体のプログラムもいろいろなところで動かすことができるようにするために、httpd からロードできるように http サーバを立ち上げておく。
- RMI Activation Demon：　Lookup サービスは activation をつかっているので、rmid を立ち上げておく。
- Jini Lookup サービスの立ち上げ：Jini ではいくつかの lookup サービスのサーバがあるが、その一つである reggie を以下のようにして立ち上げる。
  `Java –jar reggie.jar http://hostname/reggie-dl.jar policy log_file myName`
  ここで、`http:`は上の `httpd` が立ち上がっているホストを指定する。

このほかに、`Lookup` サービスを立ち上げるための `GUI` や、状態を確認する `Browser` がある。

`Jini` の中核になるアイデアはいろいろなサービスを検索して、必要なサービスを利用できる環境を提供することである。さて、実際のプログラムをみてみることにしよう。最初の例は、`TimeService` オブジェクト（以前、`ShowDate` と読んでいたものと同じ）を登録し、それを利用する例である。まず、サーバ側では、

- 登録する `Lookup` サーバに対して、`LookupLocator` を作成し、ここから、`Registrar` を取得する。
- `TimeService` とプロパティから `ServiceItem` を作成し、この `registrar` に対し、登録する。

このサーバーを動かすためには、このプログラムをコンパイルし、起動する際に、このプログラムのクラスを供給する http サーバを用意しておかなくてはならない。このプログラムを Lookup サービスがもちいている http サーバと共用してもいいが、別でもよい。例えば、別にして 8081 にこのクラスのための http サーバーを立ち上げているとすると以下のようにして起動する。

```
Java –Djava.security.policy=policy.txt
–Djava.rmi.server.codebase=http:/HOSTNAME:8081/ setup
```

クライアント側は、

- まず、指定された URL から、LookupLocator をつくり、ここから、Registrar を取得する。
- 検索するクラス（インタフェース）を指定して、テンプレートを作成する。
- このテンプレートより、検索し、オブジェクトを取得する。

LookupLocator は、Lookup サービスが立ち上がっている場所を指定するオブジェクトである。しかし、Lookup サービスをそのものではない。Lookup サービスの本体は、ServiceRegistrar であり、これに対し、以下のようにして取得する。

```
LookupLocator locator = new LookupLocator(“jini://myhost”);
ServiceRegistrar registrar = locator.getRegistrar();
```

オブジェクトを登録するには、サービスのデータを作成して、登録する。

```
ServiceItem sit = new ServiceItem(…);
ServiceRegisteration sre = registrar.regsiter(sit,Lease.FOREVER);
```

ServiceItem は、サービス名、バージョン番号などを含んだ属性とオブジェクトを元に作る。
クライアント側では、検索するテンプレートを作成し、これを元に検索する。

```
ServiceTemplate tmpl = new ServiceTemplate(…);
Object service = registrar.lookup(tmpl);
```

テンプレートには、検索するクラスを指定する他、属性からサービスを探すこともできる。詳細は省略する。

また、これまでのプログラムに policy ファイルを指定して実行するが、これはどのような人がどのような権限で実行できるかを細かくしているものであるが、詳細は省略する。

Lookup サーバの検索

前述の LookupLocator の例は、Lookup サーバがわかっている場合を想定している。Lookup サーバから取得した ServiceRegistrar 内で、いろいろなタイプのサービスを検索できるようになっている点は、RMI と違うものの、RMI と大差がないとも言える。

Jini の想定する環境では、クライアントは Lookup サーバがどこにあるのかを情報を持っていないことがありうる。このためのクラスが、LookupDiscovery である。次の例はこのクラスを使う例である。どこにあるかわからないサーバーを探すために、Multicast のプロトコルを使っている。

Jini の Lookup サービスには「グループ」という概念がある。一つの Lookup サービスは複数のグループに属することができるし、複数の Lookup サービスが同一のグループに属することができる。グループについては、文字列の配列で表現している。但し、空の文字列の場合は特定のグループには属さないと解釈される。

LookupDiscovery が Mulitcast を使って discovery を行うドメインを Jini では、djinn(ジン)と呼んでいる。
discovery には、３つのプロトコルが使われている。
multicast request プロトコル：自分が属する djinn に対して、マルチキャスト Lookup がないかを問い合わせる。
multicast announce プロトコル：Lookup サービスが、自分の存在をアナウンスするためのプロトコル。新しいサービスが加わったり、ネットワーク障害等で切断したり、復旧したりするときに用いる。
unicast discovery プロトコル：特定の Lookup サービスと通信するプロトコル。上の例。
LookupDiscovery オブジェクトは、グループを指定して作成される。

```
lookupDiscovery = new LookupDiscovery(groups);
```

Lookup サーバが見つかったときのインタフェースとして、イベントリスナーモデルを使っている。そのため、リスナーとして登録するオブジェクトは、DiscoveryListerner のインタフェースを定義していなくてはならない。

```
public class TimeServiceImpl implements DiscoveryListener …. {
```

```
  …..
  lookupDiscovery.addDiscoveryListerner(thisObj);
  ….
  public void discovered(DiscoveryEvent ev){ ….. /* 見つかった時の action */ }
  public void discarded(DiscoveryEvent ev) { … /* 切れた時の action */ }
…. }
```

discovered は Lookup サービスが見つかったときに呼び出されるメソッドである。ここで、引数になっているイベントから、ServiceRegistrar を取り出す。これについては、複数の Lookup サービスがありうるので、配列になっていることに注意。

```
   ServiceRegistrar[] regs = ev.getRegistrars();
```

サーバ側では、これを用いて、TimeServiceImpl を TimeService として登録している。登録の手順は前の Unicast のものと同等である。逆に、クライアント側ではどれか一つの ServiceRegistrar から、オブジェクトを取得している。

ちなみに、このプログラムでは Listener に設定してから、検索が終了し、リスナーが呼び出されるまでに時間がかかるため、設定後、sleep して待っている。

Jini: RMI の場合

次の例は、RMI をサービスとして登録するものである。違いは、サービスが Remote を extend して定義し、RMI できるようにしてある。この場合、Remote インタフェースを持つオブジェクトは、クライアント側に移動するのではなく、サーバ側にとどまって、サーバ側で実行される。そのため、このプログラムを実行する時には、rmic によって、スタブを生成しておくことが必要である。また、このプログラムではリスナーが見つかり登録するための部分を別のスレッドにして実行している。

リースの概念、その他

Jini のプログラミングの大きな特徴の一つに「リース」という考え方がある。つまり、あるオブジェクトが他のオブジェクトに貸し出す期間を設定し、その期限が過ぎると使えなくなるというものである。

RMI や RPC はリモートの呼び出しもあたかもローカルなもののように見せることによって、プログラミングを容易にするものである。しかし、Jini ではこの点をあえてユーザに見せるようにしている。例えば、分散環境ではリモートのコンピュータが壊れたり、ネットワークに障害が起こるかもしれない。このような環境ではあらかじめこのようなことを起こることを想定するプログラミングが必要となる。Jini のリースでは、あらかじめ決められた時間が来るとサービスは終了し、リソースは自動的に開放される。リースの期限が期限になる前に、更新するかどうかの対処を行う。リソースの管理を一定時間ごとにできることだけでなく、これにより障害等に強いソフトウエアを作成することができる。

このほかにも、Jini には Java でのイベントリスナーモデルを分散環境に拡張した分散イベント(distributed event)の仕組みがある。イベントも分散オブジェクトとして登録され、lookup サービスを通じてやり取りが行われる。また、データベースの更新などの同期を取るために、トランザクションをサポートする機構などもサポートされている。