第一部分

JavaSpaces 基础知识

现在，关于程序设计人员是否需要了解分布式程序设计已经不是一个问题了。新的问题是，编写一个分布式应用程序需要付出多大的努力。JavaSpaces技术降低了编写分布式应用程序的复杂性，减轻了编程人员的负担。它使程序设计人员能够将较多的时间花在详细的应用程序设计之上，而不是花在分布式环境上。尽管这样，分布式环境中的程序设计仍然与传统单机程序设计环境不同。

在本书的这个部分中，介绍 JavaSpaces。JavaSpaces技术提供了一个在分布式环境中进行程序设计的简单机制。本书前四章逐步介绍 JavaSpaces技术及其怎样使用。这样做有助于跳过不必要的难点。由于环境和设置问题导致失败的地方有 很 多 ，在第一次遇到时要判断出这些地方是相当令人头痛的。

阅读完前四章后，就具备了使用 JavaSpaces技术和更进一步理解本书其余内容的基本思想和概念了。

第1章

关于JavaSpaces

- 介绍JavaSpaces
- 使用JavaSpaces的理由

本章概要地介绍两个内容。首先介绍什么是 JavaSpaces。从非常具体的（且简单的）接口定义到JavaSpaces究竟处于什么位置这种更为抽像的问题进行介绍。

在具有这种总体性的了解之后，我们将对 JavaSpaces应用做某种高层次的描述，这些描述在后面的章节中还要进一步扩充。本章举例说明了怎样将 JavaSpaces用作一种分离通信、应用程序构造和并行计算的机制。

1.1 什么是JavaSpace

“什么是JavaSpace”这个问题的答案有多种。根据提问题的角度不同，实际上对其有不同的理解。

可以从以下几个方面来了解JavaSpace：

- 纯对象风格。
- 作为Jini服务。
- 具有共享分布式通信的机制。
- 对象存储机制。

从纯对象的观点，所有 JavaSpace都是稍后要考察的重要的 JavaSpace接口的实现。它确实是得出那些有趣结果的一个非常小的接口。

这里是插入JavaSpaces相关术语的一个好地方。在本书中谈到“JavaSpaces技术”或JavaSpaces时，一般指的是JavaSpace的实现。在仅看到JavaSpace或space（空间）时，应该理解为JavaSpaces的一个具体运行实例。

从Jini的观点来看，JavaSpace是一个利用Jini基础结构并向其他Jini客户机和服务提供其功能的一个Jini服务。关于JavaSpaces怎样适应Jini世界还要碰做更多的介绍。

JavaSpaces提供了一种完成共享分布式计算的机制。这可能是vaSpace所提供的一种最重要的功能了。

JavaSpaces还提供了一个非常有趣且简单的对象存储机制。这并不是说它们是一种对象数据库（它们的确不是），但这确实是一个非常有用的功能。

下面的段落将从这些方面来介绍avaSpace。这些段落能在概念上为理解avaSpace提供一个很好的开端。

1.1.1 JavaSpace接口

JavaSpace实际的接口定义很简短紧凑，请参阅程序清单1-1。

程序清单1-1 JavaSpacejava

```
package net.jini.space;

import net.jini.core.entry.*;
import net.jini.entry.*;
import net.jini.core.transaction.*;
import net.jini.core.event.*;
import net.jini.core.lease.*;
import java.rmi.*;

 public interface JavaSpace {
    Lease write(Entry entry, Transaction txn, long lease)
        throws TransactionException, RemoteException;

    long NO_WAIT = 0;
Entry read(Entry tmpl, Transaction txn, long timeout)
   throws UnusableEntryException, TransactionException,
          InterruptedException, RemoteException;

Entry readIfExists(Entry tmpl, Transaction txn,
                   long timeout)
   throws UnusableEntryException, TransactionException,
          InterruptedException, RemoteException;

Entry take(Entry tmpl, Transaction txn, long timeout)
   throws UnusableEntryException, TransactionException,
          InterruptedException, RemoteException;

Entry takeIfExists(Entry tmpl, Transaction txn,
```

```
                    long timeout)
        throws UnusableEntryException, TransactionException,
               InterruptedException, RemoteException;

    EventRegistration notify(Entry tmpl, Transaction txn,
                             RemoteEventListener listener,
                             long lease,
                             MarshalledObject handback)
        throws TransactionException, RemoteException;

    Entry snapshot(Entry e) throws RemoteException;
}
```

正如所见，程序清单1-1中列出的七个方法可用来提供某些非常复杂的行为的机制。不过，在遇到复杂问题时，最好从简单的地方入手。幸运的是，这些接口都很简单。

1. Entry

在学习实际的方法前，应该对Entry（项）类给予某种特殊的关注。请注意，每个方法都以一个项为参数，七个方法中有五个返回一个项。显然，在项和JavaSpaces之间具有相当重要的联系。这种联系就是，项是你放入一个JavaSpace或从一个JavaSpace取出的东西。

下面是net.jini.core.entry.Entry的接口定义：

```
package net.jini.core.entry;
public interface Entry extends java.io.Serializable {
}
```

这甚至比JavaSpace接口更简单，其中根本就没有方法。Entry接口是marker接口的一个例子。它本身不增加任何特定的指示功能。它所提供的是一个特定的类可放入一个空间的指示。

请注意，Entry接口位于程序包net.jini.core.entry之中，但JavaSpace接口在net.jini.space中。Entry接口并不仅仅用作JavaSpace用法的一个标记。Entry实际上提供了可供任何Jini服务（在1.1.2节“Jini和JavaSpaces”中作为一个可搜索的程序块学习）使用的公共接口。

除了提供关于哪些类可以放入空间的指示外，项还定义了一个JavaSpace实现怎样使用它的项实例的语义。

在建造一个实现项的类时，需要遵循以下几条规则：

- Entry子类的每个字段必须是公共的。（字段可以是非公共的，但不能把它们保存在空间中。）
- 字段不能为简单字段。它们必须是对象。
- 字段必须是可串行化的。
- 必须提供一个公共的无参数的构造函数。

第3章中介绍了许多Entry的内容，但简要地说，在那里这些规则是为了提供搜索较大项组的简单有效的机制。这实际是说，JavaSpace是什么？是一个实现Entry接口的类实例的集合。

现在回到JavaSpace接口本身的方法上。

2. Read

read方法用于在JavaSpace中查找项。从本质上来说，它提供了一种搜索JavaSpace的方法。

```
Entry read(Entry tmpl, Transaction txn, long timeout)
   throws UnusableEntryException, TransactionException,
          InterruptedException, RemoteException;
```

第一个参数是一个项，用作执行搜索的模板。如果项的某个字段为空，则空间中相同类型的任意项内的同一个字段都将匹配。这里，“类型”一词用来表示匹配项可以是与模板相同的类或者是模板的子类。

如果模板中的一个字段不为空，则相同类中其他项内的字段必须精确匹配。第3章将详细介绍匹配的确切含义。如果找到一个匹配，就会返回匹配项。如果空间中有不止一个匹配项，则空间可以返回任意匹配项。至于返回哪个项不能保证。这表示不应该指望读项的顺序会有什么对应关系（如到达的顺序等）。

第二个参数提供了一个Transaction实例，应该在其下执行读操作。第4章将介绍怎样对JavaSpaces使用事务处理。

最后一个参数为长整型，以毫秒为单位，此值说明在read方法中对于匹配项的出现在空间中要等待多久。这表示，如果一个匹配项在第一次调用read方法时不在空间中，则read方法将等待此项被添加到空间中，等待时间为此参数给出的超时值。如果等待时间超过超时值后还没有项匹配，则返回空。

3. readIfExists

readIfExists方法是一个非常类似于read的方法。它具有与read完全相同的参数和返回值。不过，readIfExists在这些参数的使用上稍有不同。它也是用来搜索空间的，而且使用相同的匹配模板实例的规则。read和readIfExists的不同之处在于超时值的处理。

readIfExists方法试图匹配作为第一个参数传入的模板项。如果未匹配，它立即返回，而不是像read方法那样等待匹配项。既然它立即返回，那么还要超时参数干什么呢？

这个问题涉及JavaSpaces怎样处理事务。一个匹配项可能位于空间中，但可能在至今尚未完成的事务处理下写过。这表示此匹配项并不真正对readIfExists方法可见。超时参数指出readIfExists等待完成此未结束的事务处理要等多久。

因此，read方法要等待直至找到匹配项或超时期满。而readIfExists方法仅在恰好存在与未结束事务处理之下的匹配项相同的匹配项时等待。

第4章将详细介绍JavaSpaces和事务处理的相互作用。

4. take

take方法也具有与read方法相同的参数和返回值。它使用项模板的相同匹配规则，其超时值与read的超时值类似，即等待直到匹配项出现为止。但其重要的差别在于，如果找到一个匹配项，不仅返回给方法的调用者，而且还从空间中删除它。

另外，如果多个客户机调用take方法且它们匹配空间中相同的项，则只有一个客户机取得该项，而其他客户机得到空返回值。

5. takeIfExists

就像readIfExists方法对应于read方法一样，takeIfExists方法对应于take方法。即，它的超时参数指出等待具有一个匹配项的未结束的事务处理完成要等待多长时间。

6. write

write方法用于把项放入空间。

```
Lease write(Entry entry, Transaction txn, long lease)
        throws TransactionException, RemoteException;
```

write以希望放入空间的项作为第一个参数。请注意，任意种类的项都可以写入一个空间。write方法也使用write所属的Transaction实例以及一个租用参数。

第4章将深入介绍租用参数，简单地说，租用表示write的调用者希望项在空间内呆多久，以毫秒为单位。

write方法的返回值为一个Lease实例。这允许调用者对将项保留在空间中有某些控制权。

7. notify

notfiy方法提供感兴趣的项被写入某个空间时到得通知的异步机制。

```
EventRegistration notify(Entry tmpl, Transaction txn,
                         RemoteEventListener listener,
                         long lease,
                         MarshalledObject handback)
        throws TransactionException, RemoteException;
```

第一个Entry参数指出匹配空间中的项时使用的模板。匹配规则与read方法的相同。与read方法不同的是notify指出调用者对只要写匹配项就得到通知感兴趣，而不是对调用时在那里的项感兴趣。

RemoteEventListener参数告诉空间送回事件给谁。在写一个新项到匹配模板的空间时，此空间发送一个事件到RemoteEventListener，以便它可以处理此事件。

Handback参数作为事件数据的组成部分被发送到监听程序。这给notify方法的请求者提供了

与Listener实例通信的一种途径。

Lease参数指出调用者在接收通知时等待多长时间，时间以毫秒表示。

EventRegistration返回的值为 notify方法的调用者（如 Lease实例）管理其注册提供某些信息。

第4章将介绍通知及怎样使用这些通知。

8. snapshot

它是一种优化空间性能的方法。

```
Entry snapshot (Entry e) throws RemoteException;
```

在利用相同的模板项对一个空间重复调用方法时， snapshot方法有助于提高程序的性能。snapshot工作的方式是你对传递希望优化性能的模板的空间调用它。

调用之后返回一个 Entry实例，此 Entry实例代表传入的项。基本上，空间会记住此新项实际上是旧项。在传递的这个新代表项调用空间时，避免了串行处理的大量开销。在模板项很大且串行化代价很高时，这样做可以极大地改善性能。

有一件需要注意的重要事情是，这仅对在其上调用 snapshot方法的空间有效。如果在另一个不同的空间上调用某个方法并传递快照项，则新空间将不承认快照项代表原模板项。

另一个要点是取回的项与传入的项根本不同。不应该将新项与已经拥有的任何项进行比较。

1.1.2 Jini和JavaSpaces

Jini是JavaSpaces建立在其上的一种技术基础。如果对 JavaSpace技术怎样适应Jini没有正确地理解，实际上是不可能在 JavaSpace中进行编程的。

Jini提供了一种可以在其上建立分布式计算系统的基础。这听上去可能会有似曾相识的感觉，因为前面我们曾经说过 JavaSpaces是一种分布式计算的机制。

对Jini进行透彻的讨论已经超出了本书的范围。如果希望对 Jini有更好的理解， W.Keith Edwards撰写的《Core Jini》(Prentice Hall出版社, 2001出 版)一书是一本很好的参考书籍。 Web站点www.jini.org也是了解Jini的更多信息的好所在。

本书的目的是提供足够的信息和背景知识以说明 JavaSpaces适合在Jini内的哪些地方使用，并说明对JavaSpace程序员是必需的或将被证明是极为有用的 Jini机制。

图1-1图示了JavaSpaces与Jini的关系。

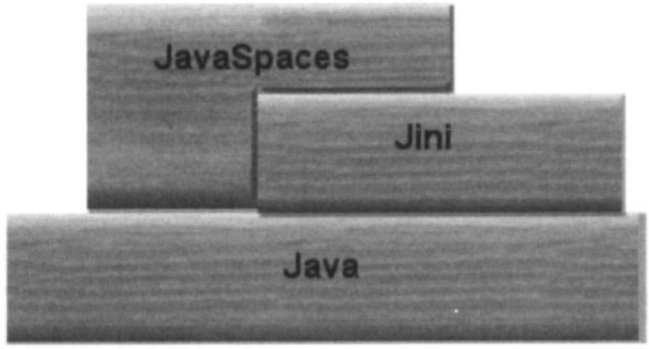

图1-1 JavaSpaces与Jini

JavaSpace是一个Jini服务。Jini服务对其他Jini服务和客户机提供功能。它通过Jini查找机制使自己可为潜在的用户所用。（第3章将详细介绍查找功能。）

因为JavaSpace是一个Jini服务，客户机可以将可从JavaSpace得到的功能与其他Jini服务和库机制相结合。当然，也可以使用所有其他的Java功能。作为Jini组成部分的优点在于Jini自身的基本功能及其他可得到的服务（如JavaSpaces本身）所提供的手段。

Jini的基本功能可分为五个方面：

- Discovery（发现）
- Lookup（查找）
- Leasing（租用）
- Event（事件）
- Transaction（事务处理）

从一个非常高的层次上看，这些功能拥有定义良好的任务。提供Discovery和Lookup作为查找东西的手段。Jini中提供的Discovery协议就是用来找到Lookup服务的东西。Lookup服务是用来找到其他Jini服务的机制。

作为分布式计算的支持机制提供了Leasing、Event和Transaction。在前面关于JavaSpaces的段落中已经简略地提到过Leasing。JavaSpaces的write方法返回一个net.jini.core.lease.Lease实例。Lease接口是作为Jini的组成部分提供的。Leasing表示事物（包括程序服务）都具有生存期这个概念。通过使不活动的事物具有自动消失的能力，Jini能够自动消除可能的废物。

Events（正如前面的“Notify”一节所述）提供一种异步通知有关成分的手段。当发出事件的一个服务中某个感兴趣的事情发生时，客户机寄存器希望接收到相应的事件。这样客户机程

序可以继续完成其他任务而不用等待该事件的发生。

Transaction提供防止部分故障的机制。这个机制决定加入（恰当地加入）一个事务处理的所有操作都成功，或者全都失败。这有助于防止出现不一致情况。

另一个对JavaSpaces应用程序有重要影响的机制是动态代码下载。这使服务和客户机能够利用直到实际运行时才会遇到的类。

1.1.3 共享分布式计算

租用、事件和事务处理的产生是由分布式程序设计的特性所决定 的。相对于一般的本地计算来说，分布式计算环境中的基础环境易出错。

因此，不应该假定服务总是存在，而是应该预见到分布式环境中不可避免地会存在问题。分布式进程通信的“ 标准 ”方法是使它们互相联络，然后直接互相传递消息。这些消息对程序员来说可以作为远程方法调用或数据包出现，但要点是在进程间建立直接连接。图-2示出了两个进程的直接通信。

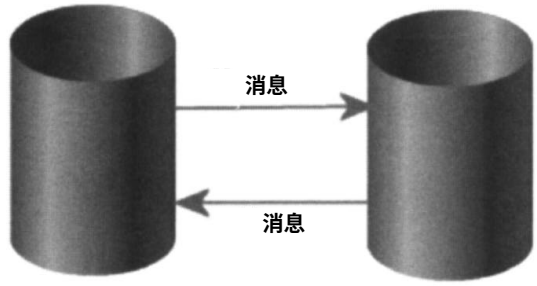

图1-2 进程对进程的通信

JavaSpaces引入了一种不同的模型。JavaSpace提供了进行通信的一种中介模型。图-3示出这种模型。

初看上去，这似乎在分布式系统中又引入了一个可能出问题的环节。但实际功能是分离了进程。不用操心特定进程通信的细节，进程 1（图1-3中）所要操心的是写一个项到 JavaSpace。进程2无需关心项是怎样进入JavaSpace的，它只要取走它们，然后做自己的工作即可。

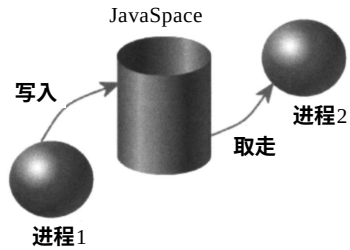

图1-3 利用JavaSpace进行进程通信

对进程进行分离有几个好处。如果进程 2失败，并不会影响进程 1，进程1仍然能够继续完成自己的任务。如果需要添加另一个进程到图中（在图 1-2所示的紧密耦合模型中），必须更改一个或多个程序的代码，或者必须一开始就编写涉及多个进程的复杂代码。而在图 1-4中要添加另一个进程只需在图中简单地加上它即可。

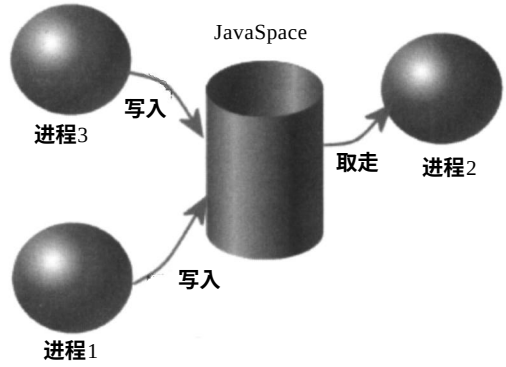

图1-4 添加另一进程

在图1-4中，进程3可以很轻松地向空间写入项了。因为进程 1不需要知道进程2的细节，所以添加进程3也不需要更改它。这里，进程 2并不关心空间中的项来自何处，它只需使用它们即可。

这种松散的耦合模型对于降低分布式程序设计的复杂性特别有用。在第二部分中，我们将进一步介绍这种模型对程序设计的影响，以及能够获得何种好处。

1.1.4 永久对象仓库

JavaSpaces的最后一个特点是对象数据仓库。写入一个空间的项都是正式的 Java对象。

但请注意，JavaSpaces并不是一个对象数据库。 Entry实例在位于JavaSpace中时并不是活动的，只是能够作为拷贝访问。这表示不能直接更改空间中的一个项。例如，如果在一个空间的某行中两次写相同的 Entry实例，则此空间中将会有两个项。因此，空间中不存在维护项的对象标识的概念。

Sun Microsystems提供的JavaSpaces有两种版本。一个是临时版本，另一个是永久版本。两者之间的差别为：在临时版本中，当空间消失时，写入的项也消失。而在永久版本中，一个空间启动和停止多次后，项仍然存在。

永久JavaSpace是可靠的。如果运行空间的机器崩溃，可以重新启动，并且成功写入空间中的项仍然存在。在一个成品环境中，知道数据不会消失是一件非常有用的事情。不过，这并不是说临时版本不能用于成品环境。如果在某个环境中，崩溃之后项的恢复并不那么重要，则使用临时版本可能会非常合适。

1.2 JavaSpaces的用途

可见，JavaSpace是一个Jini服务，它提供一种共享分布式对象仓库。现在的问题是怎样利用这样的一个实体？虽然JavaSpace接口只有七个方法，但可将这些方法作为建立更多复杂分布式系统的基础。

通过怎样使用JavaSpaces可以反映出JavaSpaces是什么。现在我们来简要地介绍一下几个可以使用JavaSpaces的领域。当然，这并不表示JavaSpaces仅在这些领域中使用，这些只是常见的实例。

1.2.1 信息共享

JavaSpaces可以共享对象。因此，许多JavaSpaces的应用程序可以做这个工作是不奇怪的。

可通过空间方便地交换信息。写到一个空间的每个项都可以从该空间的任一个客户机读出。读取者和写入者不需要互相了解。所需知道的只是何种项放入了此空间。

例如，空间可在聊天或通话系统中使用，这种系统已经变得非常流行了。其中，空间被用作共享的留言板，多个客户机可以写入和读取消息。

1.2.2 计算服务

除共享数据以外，JavaSpaces的一个非常有意思的用途是共享分布式系统计算。

一般在这种用法中，计算一个（或多个）问题块的产生器可能很昂贵，但这种问题可以分成并行任务。

产生器把问题块发送到某个JavaSpace。然后，运行在分离的机器上的该空间的客户机使用这些块，进行计算并将完成了的块返回给JavaSpace。然后再把这些完成的块装配起来。

1.2.3 工作流

JavaSpaces也很适合于管理工作流环境。可以把工作流环境与纯计算服务环境区分开来，在工作流环境中所调度的工作可以由人而不是纯计算完成。

1.3 本章小结

本章对JavaSpaces进行了概述。介绍了JavaSpaces本身的实际接口（比较少），并尽量让读者理解JavaSpace是什么以及它可能在什么地方有用。下一章将介绍怎样获得JavaSpaces并让它运转在你的机器上。你也许会遇到许多人第一次用JavaSpaces开发应用程序时所遇到的问题。

第2章

获得和安装JavaSpaces

- Jini安装
- Sun社区资源许可协议
- JavaSpace安装
- 获得一个运行例子
- 代码库说明

在进一步学习使用JavaSpaces之前，需要了解怎样获得JavaSpaces以及怎样安装它。本书将使用下列产品：

- Java 2 Software Development Kit（Java 2软件开发包，SDK），Standard Edition, v 1.3（标准版，1.3版）
- Jini Technology Starter Kit v1.1（Jini技术启动包1.1版）

JavaSpace接口自身作为Jini安装的组成部分包含在Jini Technology Extended Platform（JXP）中。Sun的JavaSpaces（Outrigger）实现也包含作为Jini Software Kit的组成部分，因此这里不单独列出。如果你已经安装了它们，可以直接跳到本章后面的运行JavaSpaces应用程序部分。

在安装了JavaSpaces后，可以运行作为Jini组成部分提供的一个例子。按照本章所讲的方法去做后，你应该得到一个很好的Jini环境，这样可以继续编写自己的程序了。

在钻研程序前花点时间做准备对于长期稳定运行是很有好处的。JavaSpaces（及Jini）的开发人员开始时遇到的大量问题都可以归结为环境设置方面的问题。

2.1 获得Java

如果你至今还未在系统上安装Java 2版本，那么首先需要获得它。可在java.sun.com/j2se/上找到各种Java 2版本。

Jini 1.1（下一步应该取得它）列出它已经针对下面的Java版本进行了测试：

- Sparc上的Solaris 2.6、Solaris 2.7和Solaris 8的最新版本上的Java 2 SDK Standard Edition, v 1.2.2_006 Solaris Reference Implementation
- Sparc上的Solaris 2.6、Solaris 2.7、Solaris 8及X86上的Solaris 8的最新FCS版本上的Java 2 SDK Standard Edition, v 1.2.2_05a Solaris Production版
- Windows 2000(2nd Edition)和Windows NT 4.0 Service Pack 6(SP6)上的Java 2 SDK Standard Edition, v 1.2.2_006 Windows 95/98/2000/NT Production Release
- Windows 2000(2nd Edition)和Windows NT 4.0(SP6)上的Java 2 Standard Edition(J2SE) v 1.3.0 Production Release for Windows 95/98/2000/NT
- Sparc上的Solaris 2.6、Solaris 2.7、Solaris 8以及X86上Solaris 8的最新FCS版本上的Java 2 SDK Standard Edition, v 1.3.0 Solaris Production

如果你没有使用这些产品，建议你升级到1.3版。虽然这不应该成为问题（Sun ’s Jini Starter Kit是纯Java的，应该在任意的Java 2运行时上都可以工作），但基于相同的软件基础可以帮助避免学习本书中的例子时可能产生的误解。

现在访问java.sun.com/j2se/。

在这里将会看到各种可得到的Java版本。选择一个想要的版本。这样做将换到某一页，比方说换到java.sun.com/j2se/1.3/的页面。

在这里，着手选择适合于将要在其上运行Java的操作系统（OS）平台的下载。这时，在上述的Web页上，有Solaris SPARC/x86、Linux x86和Microsoft Windows平台的链接。

在点击适合你的OS的链接后，会来到具有各种链接的一个页面。我们感兴趣的是下载Java SDK的SDK和文档。此文档文件让你将SDK的文档安装到本地机器上。

在每个特定OS的SDK的下载页面上，都有该OS安装说明的链接。它们都是很好很清晰的说明，如果你以前没有安装过SDK，遵照这些说明进行SDK的安装是最容易的。例如，对于Windows平台，可在java.sun.com/j2se/1.3/install-windows.html找到1.3 SDK的安装说明。

这应该会进行得相当顺利。如果不是这样，表示此安装页含有每个OS都会遇到的某些常见问题。

2.2 获得和安装Jini

如果你的Jini版本低于1.1版，则需要获得Jini 1.1，因为它添加了大量的类，这些类在本书的学习过程中要用到。

Sun Microsystems Jini主页总是包含最新的链接，在其上可以找到最新的Jini软件。此页面位于www.sun.com/jini/。

除最新版本的链接外，这个页面还包含用于Jini开发工作的信息和其他非常有用的Web站点的链接。在安装的这一点上，要特别关注Java Developer Connection Web站点的链接。这个页面位于developer.java.sun.com/。

此页面包含各种关于Java的常见有用信息，但特别要注意的是，这里在能够访问Jini的下载页前需要注册。注册过程很简单。只要单击注册链接，并遵循说明步骤进行即可。在完成这个过程后，就能够转到Jini 1.1下载页面了。此页面位于www.sun.com/software/communitysource/jini/download.html。

这里有一个继续下载的链接。第一次要按照这个链接进行实际下载，将会要求签署Sun社区资源许可协议（Sun Community Source License，SCSL）。

2.2.1 Sun社区资源许可协议（SCSL）

因为在继续下载之前需要接受SCSL，这里对它做一个简介。SCSL是一个用来允许开发人员访问特定许可产品（这里是Jini）的许可证。

当要求你签署SCSL时，可以浏览它。SCSL的当前链接可在www.sun.com/jini/licensing/处找到。

除了链接到SCSL外，一般这个页面还含有Sun的关于SCSL和Jini的Frequntly Asked Questions（FAQ，常见问答）。看看这个页面很有好处，因为Sun会定期对它进行修改和更正。本书或者Sun站点上的FAQ都不能取代对应的法律意见。

SCSL本身是一个法律文献，而且像所有法律文献一样，多数人读起来会觉得很沉闷。但SCSL的意图是很令人激动的。这个意图就是向开发人员开放源代码而同时保持兼容性。最重要的是，Jini的SCSL的主要动力是保持Jini核心的完整性。如果你希望研制自己的Jini核心版本或者发行源代码或二进制代码，需要给予更多的关注。如果主要是出于学习的目的阅读Jini核心源代码，则有一个低滚动条可以跳过一些许可条款。

为了达到允许供某些开发人员用于教学，供某些开发人员用于扩展Jini的目的，SCSL定义了两种级别（一般是三个级别Research、Internal和Commercial，但Sun简化了SCSL。）的权利和责任，分别是：Research Use（研究用途）的许可和Commercial Use（商业用途）的许可。

研究用途的许可用来帮助学习Jini技术。它允许出于研究、个人用途以及教学的目的对Jini源代码进行广泛的实验。如果你不打算出卖或分发你的源代码或应用程序，则Research Use就可以满足要求了。

商业用途的许可授予出于商业目的发布修改过的、兼容代码的权力。这句话中关键的词是兼容。为了发布属于商业用途的代码和应用程序，这些代码和应用程序必须通过Jini Technology Core Platform Compatibility Kit的测试。这有助于保证所有Jini版本都能协同工作。在你开发自己的产品时，可能符合研究许可。除非在你打算发行自己的产品，否则不需要变更许可。不管哪种许可版本都不涉及费用。

总之，如果你打算公开使用或销售Jini核心或库服务的修改版本，很可能需要研读商业用途许可。

2.2.2 获得Jini

在认可SCSL之后，将会来到一个选择合适的下载站点的页面，然后到达实际的Jini下载页面。有两个不同的下载文件：

- Jini Starter Kit——此文件具有类似于JINI-1.1-G-CS.zip这样的名称。如果是新版本，名称可能会稍有不同。此文件包括实际的Jini基础结构。
- Jini Technology Compatibility Kit——此文件具有类似于JINITCK-1.1A-G-CS.zip的文件名。如果是新版本，名称可能会稍有不同。此文件包含可用来测试Jini服务、扩展或应用程序的代码。这种测试就是前面关于商业用途许可的讨论中提到的兼容性测试。

Jini Starter Kit是我们目前要下载的文件。

2.2.3 安装Jini

刚才下载的Jini Starter Kit文件是一个ZIP文件，它包括Jini的二进制代码、源代码和支持文档。这些压缩到一起的文件提取到根目录jini1_1下。为完成提取，可选择喜欢的解压缩文件的工具。例如，可用如下的命令使用Java 2 SDK所带的jar实用程序：

```
jar xvf JINI-1.1-G-CS.zip
```

此命令提取所有文件放入当前目录的jini1_1目录下。当然，也可以用别的zip工具完成此项工作。在完成了提取后，应该在jini1_1目录中看到下面的目录：

- Doc——此目录包含本Jini版本的所有文档。这些文档包括Jini的Java文档、Jini例子的文档、一个安装文件、许可证文件和版本说明。
- Example——此目录包含帮助运行所提供的例子的文件。

- Lib —— 此目录包含Jini的JAR文件。可将这些文件想像为组成Jini运行时的实际二进制文件。
- Policy —— 此目录包含作为Jini 1.1分发包组成部分的Jini服务的策略文件模板。关于策略文件在第9章中有详细介绍，它们基本上是定义Java安全的一种手段。
- Source —— 此目录存放Jini 1.1的源代码。

除了这几个目录外，文件index.html提供了与Jini文件的一个链接，此文件位于jini1_1目录中。可用你自己喜欢的浏览器来观看它。如果由于某种原因导致这些目录不存在或不全，则应该查看一下是否确实提取了所有内容。

2.3 运行JavaSpace

现在机器上已经安装好了Jini，安装JavaSpaces就很容易了。自Jini 1.1起，JavaSpaces作为Jini扩展程序包的组成部分发布。这使安装更简单了一些。而且还包含了好几个有用的 JavaSpaces例子程序。这些例子程序如下：

- 射线跟踪（ray-tracing）例子。
- 图书采购员（book-buyer）例子。
- JavaSpaces服务浏览器。

本书并不详细研究这些例子怎样工作，运行它们的目的是进行测试。只要其中某一个例子正常运行，就表示Jini已经正确安装，使JavaSpaces工作所需要的各种成分已经启动。所有这些例子的源代码都包含在Jini安装程序包中。在能够启动任意一个例子（或者你自己的 JavaSpace程序）前，需要启动一些支持服务。

2.4 启动支持服务

Jini和Java提供了许多服务。一般情况下，下面的这组服务对于本书中大多数例子已经足够了：

- Hypertext Transport Protocol（http，超文本传输协议）服务器 —— 这个服务器用来下载类文件到客户机，以便在客户机与服务打交道时使用。
- RMID（Remote Method Invocation Activation Daemon，远程方法调用激活守护进程） —— 这个服务不是Jini的组成部分，而是Java 2 Standard Edition本身的组成部分。它提供一个控制可激活对象（如Transaction Manager）的激活守护进程。这提供了使当前未使用的服务进入睡眠，并在以后的某个时刻唤醒的能力。只要存在一个查找服务，运行就需要RMID守护进程运行。因为Sun的查找实现是一个可激活的服务，所以为了支持它，总是需要运行RMID守护进程。

- Jini查找服务 —— 这是用来找到其他Jini服务的服务。此服务处于Jini的核心位置。
- Transaction Manager（事务处理管理器） —— 这是管理事务处理的Jini服务。
- JavaSpaces服务 —— 这是实际支持来自Sun的JavaSpaces的基础结构。对于这个服务后面还要详细介绍。

看上去似乎需要启动许多东西，但一般来说，在使它们运行起来后，很长时间可以不去理会它们。

有两种启动这些支持服务的方法。Sun Microsystems提供了启动服务的一个样例图形用户接口（Graphical User Interface，GUI）。也可以直接从命令行启动它们。下几节将研究这两种启动服务的方法。我们先来看看利用所提供的GUI启动这些服务。

2.4.1 利用GUI进行启动

作为应用启动程序的例子，与Jini一起提供了类com.sun.jini.example.launcher.StartService。这是一个启动服务的相当简单和快捷的办法。

在Windows平台上，可执行下面的命令启动GUI：

```
java -cp C:\jini1_1\lib\jini-ext.jar;C:\jini1_1\lib\jini-
examples.jar com.sun.jini.example.launcher.StartService
```

在UNIX平台上，可执行下面的命令启动GUI：

```
java -cp /files/jini1_1/lib/jini-ext.jar:/files/jini1_1/lib/jini-
examples.jar com.sun.jini.example.launcher.StartService
```

这上面的两种情况下，都需要放入在机器上实际安装Jini的正确路径。执行了上述命令后，将会看到如图2-1所示的窗口。

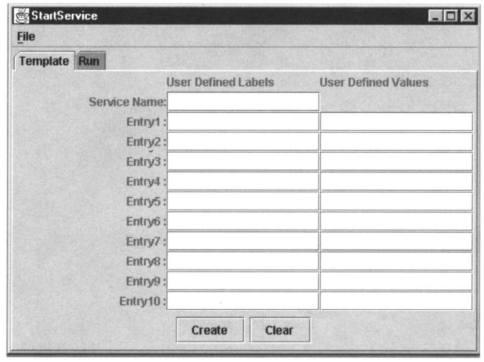

图2-1 初始GUI窗口