IceGrid 应用
配置手册
V 2.1

中诚信资讯科技有限公司

目录

1. 概述

1.1 配置目标

本文档是描述 Ice 中间件中的 IceGrid 服务的应用配置，通过使用 IceGrid 服务来实现：

1. 服务器端服务分布式部署。
2. 服务器端服务按需激活。
3. 服务器端服务多节点负载均衡。
4. 注册服务主/从热备（Master/Slaves）
5. 集成 IceBox 服务

1.2 实验环境

1. 硬件：hp 服务器，3 台
2. 操作环境：Red Hat 5
3. 服务器程序：ServerApp.jar
4. 说明：实际应用中，服务器节点可任意扩充、操作系统可被更换、服务器程序可用实际项目的服务程序替换，本文档所描述的配置方式具有通用性，适用但不局限于当前实验环境。

1.3 局限

本文档不详细描述 IceGrid 服务的运行机制和实现原理，不详细介绍服务器端和客户端程序的实现，主要描述 IceGrid 服务应用的配置步骤、主要配置项及验证配置结果等。

2. 配置过程

2.1 服务器端配置

配置步骤：

1. 创建主注册服务（Master）的配置文件 config_master.grid，文件名称可以任意

2. 创建从注册服务（Slave）的配置文件 config_slave.grid， 文件名称可以任意
3. 创建各节点服务的配置文件 config.node，文件名称可以任意
4. 创建分布式应用配置文件 app.xml，文件名称可以任意，但格式最好定义成 xml
5. 运行 Ice 提供的工具，启动我们的分布式应用，主要有如下两个工具：icegridnode 和 icegridadmin。详细启动过程如下：

   1） icegridnode --Ice.Config=config_master.grid 启动主注册服务

   2） icegridnode --Ice.Config=config_slave.grid 启动从注册服务

   3） icegridadmin --Ice.Config= config_master.grid -e "application add app.xml" 部署分布式服务

   icegridadmin --Ice.Config= config_master.grid -e "application update app.xml" 重新部署分布式服务

   4） icegridnode --Ice.Config=config.node 将各节点注册到注册服务的注册表中

配置文件清单：

假设有 n 个节点（n > 0），其中从注册服务有 x 个，(x > 0)

config_master.grid ---------- 主注册服务配置文件 --------- 1 份

config_slave.grid ---------- 从注册服务配置文件 --------- x 份

config.node ---------- 节点配置文件 ---------- n 份

app.xml ---------- 部署配置文件 ---------- 1 份

通常情况下，由于注册服务占用资源很少，所以一般都会和一个节点集成在一起，并且可以和节点服务在一个进程中运行。因此，如果假设服务部署到 n 个服务器，通常情况下配置文件清单如下：

config_master.grid -- 主注册服务配置文件 -- 1 份 -- 主注册服务信息+节点信息

config_slave.grid --- 从注册服务配置文件 -- x 份 -- 从注册服务信息+节点信息

config.node --------- 节点配置文件 ---- n-1-x 份 -- 节点信息

app.xml ---------- 部署配置文件 -------- 1 份 -- 部署信息

其中 app.xml 要和 config_master.grid 放在一台服务器上，下面的各章节将详细介绍各配置文件。

2.1.1 主注册服务配置

config_master.grid 的内容：

```
#
# The IceGrid Instance Name
#
IceGrid.InstanceName=IceGridRDDataSource        # 1

#
# The IceGrid locator proxy.
#
Ice.Default.Locator=IceGridRDDataSource/Locator:default -h 10.0.5.201 -p 12000:default -h
10.0.5.202 -p 12000      #2

#
# IceGrid registry configuration.
#
IceGrid.Registry.Client.Endpoints=default -p 12000     #3
IceGrid.Registry.Server.Endpoints=default       #4
IceGrid.Registry.Internal.Endpoints=default     #5
IceGrid.Registry.Data=master       #6
IceGrid.Registry.PermissionsVerifier=IceGridRDDataSource/NullPermissionsVerifier        #7
IceGrid.Registry.AdminPermissionsVerifier=IceGridRDDataSource/NullPermissionsVerifier#8
IceGrid.Registry.SSLPermissionsVerifier=IceGridRDDataSource/NullSSLPermissionsVerifier#9
IceGrid.Registry.AdminSSLPermissionsVerifier=IceGridRDDataSource/NullSSLPermissionsVeri
fier      #10

#
# IceGrid SQL configuration if using SQL database.
#
#Ice.Plugin.DB=IceGridSqlDB:createSqlDB      #11
#IceGrid.SQL.DatabaseType=QSQLITE      #12
#IceGrid.SQL.DatabaseName=register/Registry.db      #13
#

#
#Ice Error and Standard output Set
#
#Ice.StdErr=master/stderr.txt       #14
#Ice.StdOut= master /stdout.txt     #15

#
#Trace Registry properties
```

```
#
Ice.ProgramName=Master    #16
IceGrid.Registry.Trace.Node=3      #17
IceGrid.Registry.Trace.Replica=3    #18

#
# IceGrid node configuration.
#
IceGrid.Node.Name=node_1                    #19
IceGrid.Node.Endpoints=default              #20
IceGrid.Node.Data=node_1                    #21
IceGrid.Node.CollocateRegistry=1            #22
#IceGrid.Node.Output=node_1                 #23
#IceGrid.Node.RedirectErrToOut=1            #24

# Trace properties.
#
IceGrid.Node.Trace.Activator=1              #25
#IceGrid.Node.Trace.Adapter=2               #26
#IceGrid.Node.Trace.Server=3                #27

#
# Dummy username and password for icegridadmin.
#
IceGridAdmin.Username=mygrid                #28
IceGridAdmin.Password=mygrid                #29
```

配置项说明：

#1 为这个应用实例指定一个唯一的标识

#2 注册服务的端点信息(主注册服务和所有的从注册服务)，节点注册时要用到

#3 客户端访问注册服务器的端点信息

#4 服务访问注册服务器的端点信息，通常是 default

#5 内部访问端点信息，通常是 default，节点用这个端口和注册服务通信

#6 注册服务的数据目录的路径

#7 设定防火墙安全代理，从而控制客户端访问注册表时可用的权限

#8 设定防火墙安全代理，从而控制注册表管理者可用的权限

#9 设定 SSL 安全代理，从而设定客户端访问注册表时的 SSL 安全访问机制

#10 设定 SSL 安全代理，从而设定注册表管理者的 SSL 安全访问机制

#11 指定 Ice 对象序列化的机制，如果不设置，默认用 Freeze 机制

#12 指定使用数据库的类型

#13 指定使用数据库的名称

#14 指定标准错误输出文件

#15 指定标准输出文件

#16 指定主注册服务的名称

#17 指定主注册服务跟踪节点信息的级别（0~3），默认为 0

#18 指定主/从热备注册服务的跟踪级别（0~3），默认为 0

#19 定义节点的名称，必须唯一

#20 节点被访问的端口信息，注册服务使用这个端点和节点通信，通常设为 default

#21 节点的数据目录的路径

#22 定义节点是否和注册服务并置在一起，设为 1 时并置，设为 0 时不并置

#23 节点标准输出信息重定向蹈的目录路径，会自动生成输出文件

#24 节点上的服务程序的标准错误重定向到标准输出

#25 激活器跟踪级别，通常有 0，1，2，3 级，默认是 0

#26 对象适配器跟踪级别，通常有 0，1，2，3 级，默认是 0

#27 服务跟踪级别，通常有 0，1，2，3 级，默认是 0

#28 IceGrid 管理器登录该应用的用户名

#29 IceGrid 管理器登录该应用的密码

未涉及的属性还有一些，如果需要请参考官方文档。

2.1.2 从注册服务配置

config_slave.grid 的内容：

```
#
# The IceGrid locator proxy.
#
Ice.Default.Locator=IceGridRDDataSource/Locator:default -h 10.0.2.241 -p 12000:default -h
10.0.2.242 -p 12000      #1

#
# IceGrid registry configuration.
#
IceGrid.Registry.Client.Endpoints=default -p 12000     #2
```

```
IceGrid.Registry.Server.Endpoints=default       #3
IceGrid.Registry.Internal.Endpoints=default     #4
IceGrid.Registry.Data=slave_1       #5
IceGrid.Registry.ReplicaName=slave_1     #6
IceGrid.Registry.PermissionsVerifier=IceGridRDDataSource/NullPermissionsVerifier        #7
IceGrid.Registry.AdminPermissionsVerifier=IceGridRDDataSource/NullPermissionsVerifier#8
IceGrid.Registry.SSLPermissionsVerifier=IceGridRDDataSource/NullSSLPermissionsVerifier#9
IceGrid.Registry.AdminSSLPermissionsVerifier=IceGridRDDataSource/NullSSLPermissionsVeri
fier      #10

#
# IceGrid SQL configuration if using SQL database.
#
#Ice.Plugin.DB=IceGridSqlDB:createSqlDB      #11
#IceGrid.SQL.DatabaseType=QSQLITE      #12
#IceGrid.SQL.DatabaseName=register/Registry.db      #13
#

#
#Ice Error and Standard output Set
#
#Ice.StdErr=slave_1/stderr.txt      #14
#Ice.StdOut=slave_1 /stdout.txt     #15

#
#Trace Registry properties
#
Ice.ProgramName=Slave_1    #16
IceGrid.Registry.Trace.Node=3      #17
IceGrid.Registry.Trace.Replica=3    #18

#
# IceGrid node configuration.
#
IceGrid.Node.Name=node_2                    #19
IceGrid.Node.Endpoints=default              #20
IceGrid.Node.Data=node_2                    #21
IceGrid.Node.CollocateRegistry=1            #22
#IceGrid.Node.Output=node_2                 #23
#IceGrid.Node.RedirectErrToOut=1            #24

# Trace properties.
#
IceGrid.Node.Trace.Activator=1              #25
```

```
#IceGrid.Node.Trace.Adapter=2                #26
#IceGrid.Node.Trace.Server=3                 #27

#
# Dummy username and password for icegridadmin.
#
IceGridAdmin.Username=mygrid                 #28
IceGridAdmin.Password=mygrid                 #29
```

配置项说明：

其实这个文件和主注册配置文件基本一样，差别只有一点：

1. *没有指定应用实例名，因为在主注册服务中已经有了定义*
2. *多了第 6 行，IceGrid.Registry.ReplicaName=slave_1，指定从注册服务的名称*

其它的基本就没有差别了，大部分属性项在 config_master.grid 里面都有定义，为了方便阅读，下面也将用到的各项给出说明：

#1 注册服务的端点信息(主注册服务和所有的从注册服务)，节点注册时要用到

#2 客户端访问注册服务器的端点信息

#3 服务访问注册服务器的端点信息，通常是 default

#4 内部访问端点信息，通常是 default，节点用这个端口和注册服务通信

#5 注册服务的数据目录的路径

#6 指定从注册服务的名称

#7 设定防火墙安全代理，从而控制客户端访问注册表时可用的权限

#8 设定防火墙安全代理，从而控制注册表管理者可用的权限

#9 设定 SSL 安全代理，从而设定客户端访问注册表时的 SSL 安全访问机制

#10 设定 SSL 安全代理，从而设定注册表管理者的 SSL 安全访问机制

#11 指定 Ice 对象序列化的机制，如果不设置，默认用 Freeze 机制

#12 指定使用数据库的类型

#13 指定使用数据库的名称

#14 指定标准错误输出文件

#15 指定标准输出文件

#16 指定从注册服务运行时程序名称

#17 指定从注册服务跟踪节点信息的级别（0~3），默认为 0

#18 指定主/从热备注册服务的跟踪级别（0~3），默认为 0

19 定义节点的名称，必须唯一

20 节点被访问的端口信息，注册服务使用这个端点和节点通信，通常设为 default

21 节点的数据目录的路径

22 定义节点是否和注册服务并置在一起，设为 1 时并置，设为 0 时不并置

23 节点标准输出信息重定向蹈的目录路径，会自动生成输出文件

24 节点上的服务程序的标准错误重定向到标准输出

25 激活器跟踪级别，通常有 0，1，2，3 级，默认是 0

26 对象适配器跟踪级别，通常有 0，1，2，3 级，默认是 0

27 服务跟踪级别，通常有 0，1，2，3 级，默认是 0

28 IceGrid 管理器登录该应用的用户名

29 IceGrid 管理器登录该应用的密码

2.1.3 应用部署配置

app.xml 配置文件内容：

```
1<icegrid>
2  <application name="RTDSSystem">
3    <server-template id="RTDSSystemServer">
4      <parameter name="index"/>
5      <server id="RTDSSystemServer-${index}" exe="java"
                              activation="on-demand">
6        <adapter name="RTDataSysytem" endpoints="tcp"
                            replica-group="ReplicatedRTDataSysytemAdp"/>
7        <option>-jar</option>
8        <option>ServerApp.jar</option>
9      </server>
10   </server-template>
11
12   <replica-group id="ReplicatedRTDataSysytemAdp">
13     <load-balancing type="round-robin"/>
14     <object identity="RTDataSource"
                              type="::RTDataSystem::RTDataSource"/>
15   </replica-group>
16
17   <node name="node_1">
18     <server-instance template="RTDSSystemServer" index="1"/>
19     <server-instance template="RTDSSystemServer" index="11"/>
20     <server-instance template="RTDSSystemServer" index="111"/>
```

```
21    </node>
22    <node name="node_2">
23      <server-instance template="RTDSSystemServer" index="2"/>
24      <!--server-instance template="RTDSSystemServer" index="22"/-->
25      <!--server-instance template="RTDSSystemServer" index="222"/-->
26    </node>
27    <node name="node_3">
28      <server-instance template="RTDSSystemServer" index="3"/>
29      <!--server-instance template="RTDSSystemServer" index="33"/-->
30      <!--server-instance template="RTDSSystemServer" index="333"/-->
31    </node>
32  </application>
33</icegrid>
```

配置文件结构分析：

IceGrid里，部署是一个在注册服务中表述一个应用（Application）的过程，而部署配置文件就是来描述这些配置信息的文件，这个配置文件是用xml标记性语言来描述的。通常一个部署应该包含如下信息：

1. 应用标签（application）,name属性定义这个应用的名字
2. 服务（server），一个逻辑上的服务器，能够通过exe命令而启动的一个服务程序。

   activation属性，是设置服务的启动方式，on-demand是最常用的方式，另外还有always等启动方式；option标签是exe执行命令命令行的参数；
3. 适配器（adpter），定义服务器端的适配器。

   name属性唯一标志这个适配器；

   endpoints属性指定端点信息；

   replica-group属性标示该适配器是个可复制组集群，并指定这个可复制组的名称；

   register-process属性定义了是否这个节点是否可以被icegrid关闭；
4. 节点（node），它应该代表了一个物理上的节点。

   name属性指定节点的名字，并且是唯一的。
5. 可复制组（replica-group），一组对象适配器的集合。

   id属性唯一标识一个可复制组；

   load-balancing子项中type属性指定负载均衡策略，icegrid提供了四种负载均衡策略： Random（随机方式）

   Adaptive(适配方式)

   Round Robin(最近最少使用)

Ordered（顺序方式）

object子项定义适配器绑定的服务对象信息。其中identity属性指定对象的标识，type属性指定了对象的层次结构类型。这两个属性都可以唯一的标识一个服务对象。

6. 服务模板（server-temple），服务模板是对服务的一个抽象，避免了重复定义。这样，在节点中描述服务时只需要实例化它的服务模板就可以了。

   id属性唯一标识一个服务模板；

   parameter子项定义服务模板的参数，可包含多个，主要实例化服务时用；

   server子项就是上面2中的服务定义；

   另外还有一些特殊的服务模板，比如：icebox服务模板，它的定义和通用的服务模板的定义不太一样。

解析app.xml文件：

通过对配置文件结构的分析，来解析一下app.xml。

第1行，标识这是一个icegrid的配置文件；

第2行，标识应用的名称为*RTDSSystem*，这个名称是唯一的；

第3~10行，定义了一个服务模板*RTDSSystemServer*，并有一个参数index；

其中5~9定义了这个模板包含的服务定义，第6行是这个服务包含的对象适配器的定义；

第12~15行，是对可复制组的定义，包括服务对象的定义和负载均衡策略；

第17~21行，是对节点node_1的定义，指定了节点的名称，包含的服务（3个服务）；

第22~26行，是对节点node_2的定义

第27~31行，是对节点node_3的定义

最后两行是闭合标签，至此一个icegrid的分布式部署配置文件就完成了。

部署配置文件的扩展：

app.xml中对服务模板、适配器、服务对象等的配置都是一个，事实上这些可以在文件中定义多个，比如可以有多个服务模板，一个服务里可以有多个适配器，可以有多个可复制组，一个节点里可以有多个不同类型的服务等。

另外，app.xml可以包含其它的xml。

2.1.4 节点配置

config.grid 文件的内容：

```
#
# The IceGrid locator proxy.
#
Ice.Default.Locator=IceGridRDDataSource/Locator:default -h 10.0.2.241 -p 12000:default -h
10.0.2.242 -p 12000      #1

#
# IceGrid node configuration.
#
IceGrid.Node.Name=node_2              #2
IceGrid.Node.Endpoints=default        #3
IceGrid.Node.Data=node_2              #4
IceGrid.Node.Output=node_2            #5
IceGrid.Node.RedirectErrToOut=1       #6

# Trace properties.
#
IceGrid.Node.Trace.Activator=1                #7
#IceGrid.Node.Trace.Adapter=2                 #8
#IceGrid.Node.Trace.Server=3                  #9
```

配置项说明：

事实上，这个文件里面的配置项，在 config_slave.grid 中都有描述，但这里也列出来，方便阅读。

#1 注册服务的端点信息(主注册服务和所有的从注册服务)，节点注册时要用到

#2 定义节点的名称，必须唯一

#3 节点被访问的端口信息，注册服务使用这个端点和节点通信，通常设为 default

#4 节点的数据目录的路径

#5 节点标准输出信息重定向的目录路径，会自动生成输出文件

#6 节点上的服务程序的标准错误重定向到标准输出

#7 激活器跟踪级别，通常有 0，1，2，3 级，默认是 0

#8 对象适配器跟踪级别，通常有 0，1，2，3 级，默认是 0

#9 服务跟踪级别，通常有 0，1，2，3 级，默认是 0

2.2 客户端配置

客户端的配置很简单，和分布式相关的配置就一项，添加如下：

```
#
# The IceGrid locator proxy.
#
Ice.Default.Locator=IceGridRDDataSource/Locator:default -h 10.0.2.241 -p 12000:default -h 10.0.2.242 -p 12000 #注册服务的端点信息(主注册服务和所有的从注册服务)，用于定位
```

3.结果验证

3.1 程序方式

1. 启动服务器
   1） icegridnode --Ice.Config=config_master.grid 启动主注册服务和节点 1
   2） icegridnode --Ice.Config=config_slave.grid 启动从注册服务和节点 2
   3） icegridadmin --Ice.Config=config_master.grid -e "application add app.xml" 部署分布式服务
   4） icegridnode --Ice.Config=config.node 启动节点 3
2. 启动客户端，进行多次远程调用，根据执行情况就可以判断服务器端是否配置成功。

3.2 工具方式

用 Ice 官方提供的可视化管理工具 IceGridGUI.jar 来验证和管理 icegrid 的部署。

1. 打开 dos 窗口，在命令行下进入 C:\Program Files\ZeroC\Ice-3.4.1\bin 目录下，然后运行“java –jar IceGridGUI.jar”，弹出 IceGrid Admin 的主界面，如下图所示：

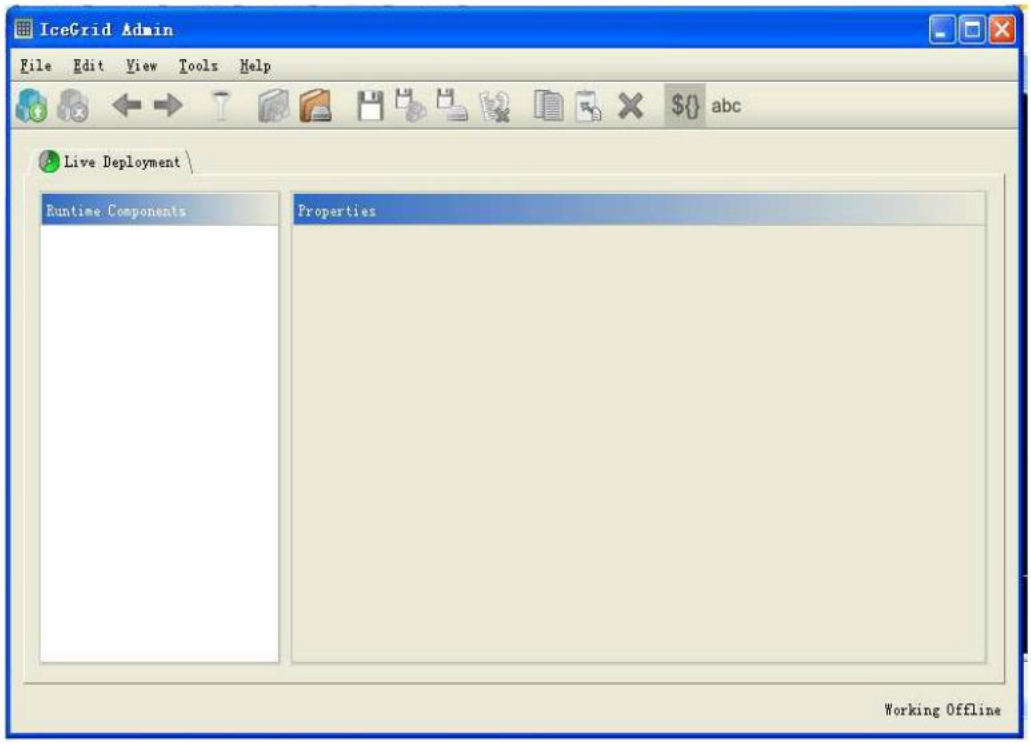

图 3.2.1 主界面

2. 点击按钮 File—>login，弹出登录对话框，输入用户名、密码，Ice Instance Name 和 ice Registry endpoint，点击“OK”按钮，就可以进入该部署的管理界面

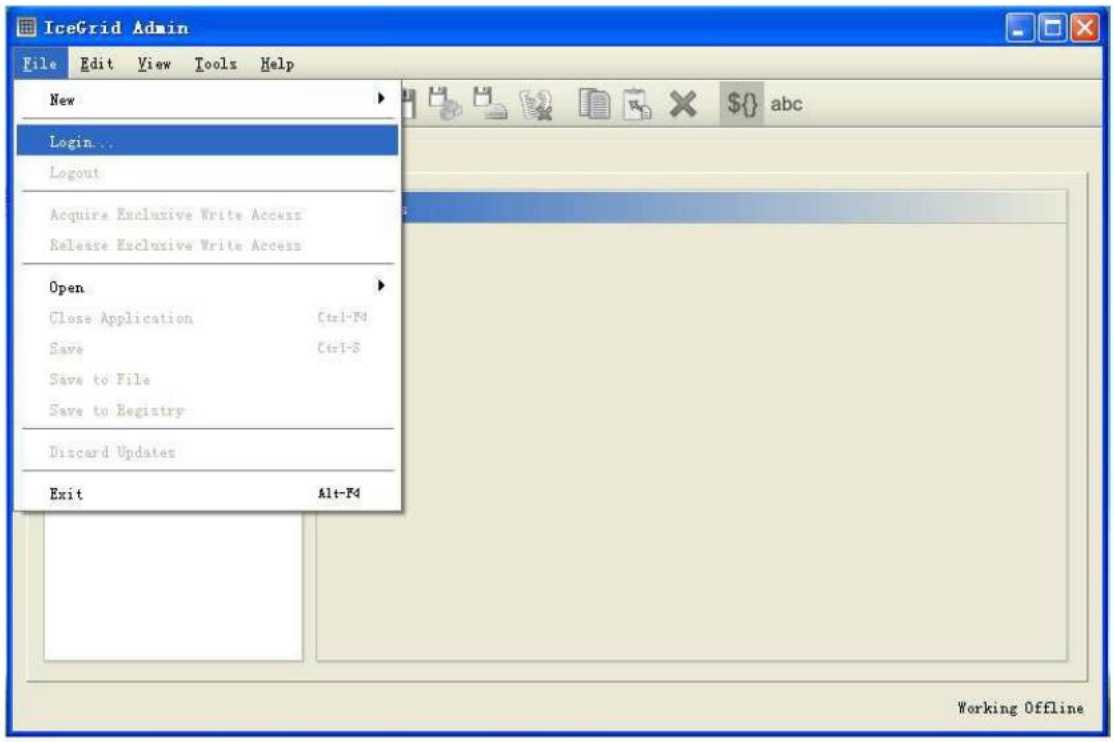

图 3.2.2 File 菜单

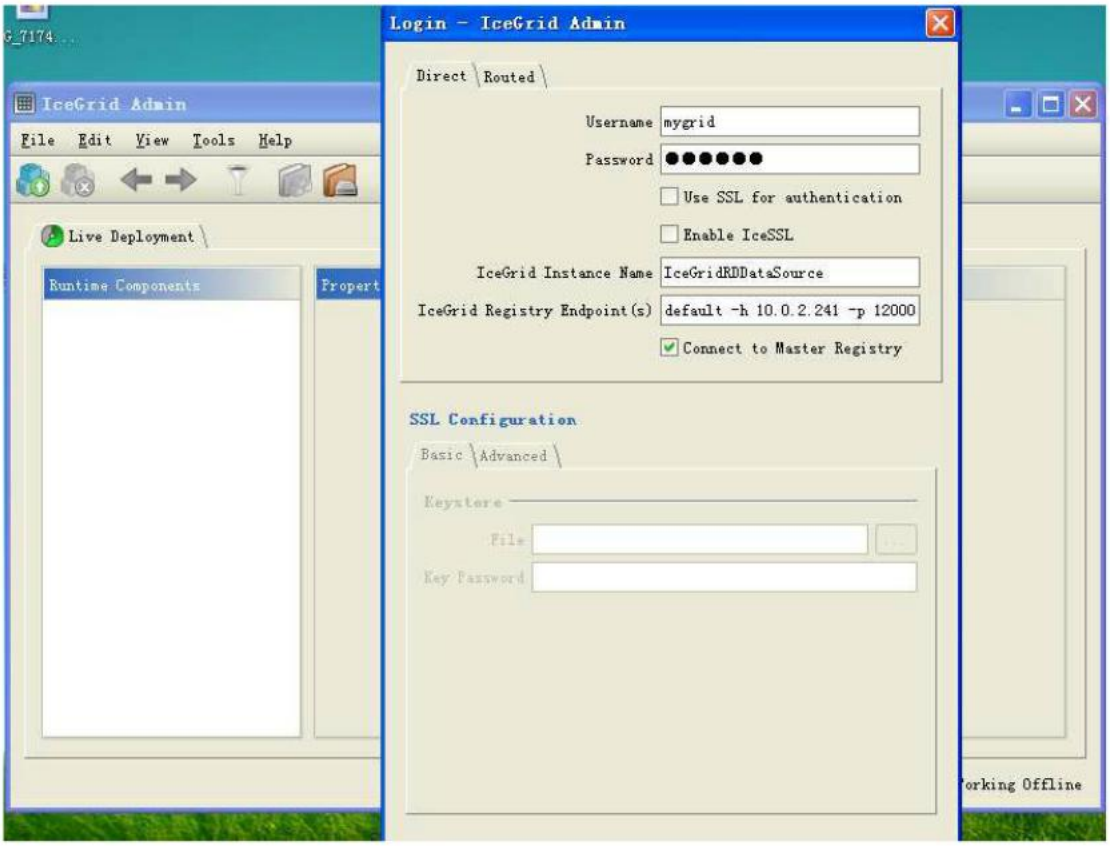

图 3.2.3 登陆窗口

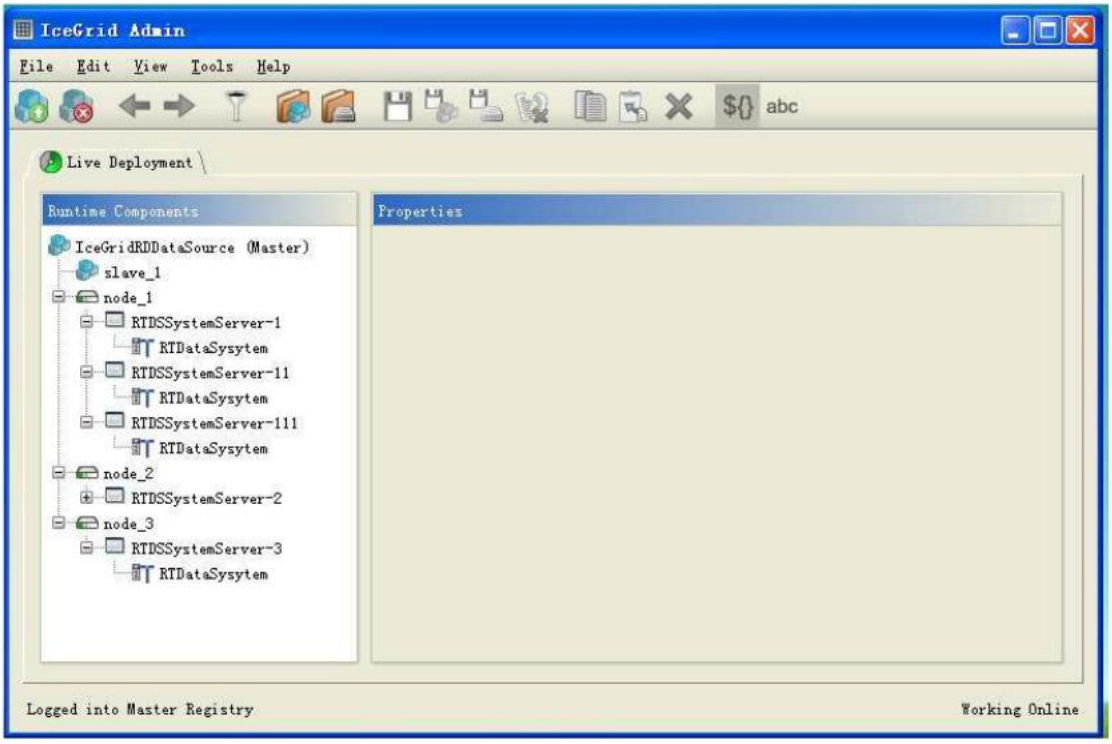

图 3.2.4 实时管理（服务已加载，但未激活）

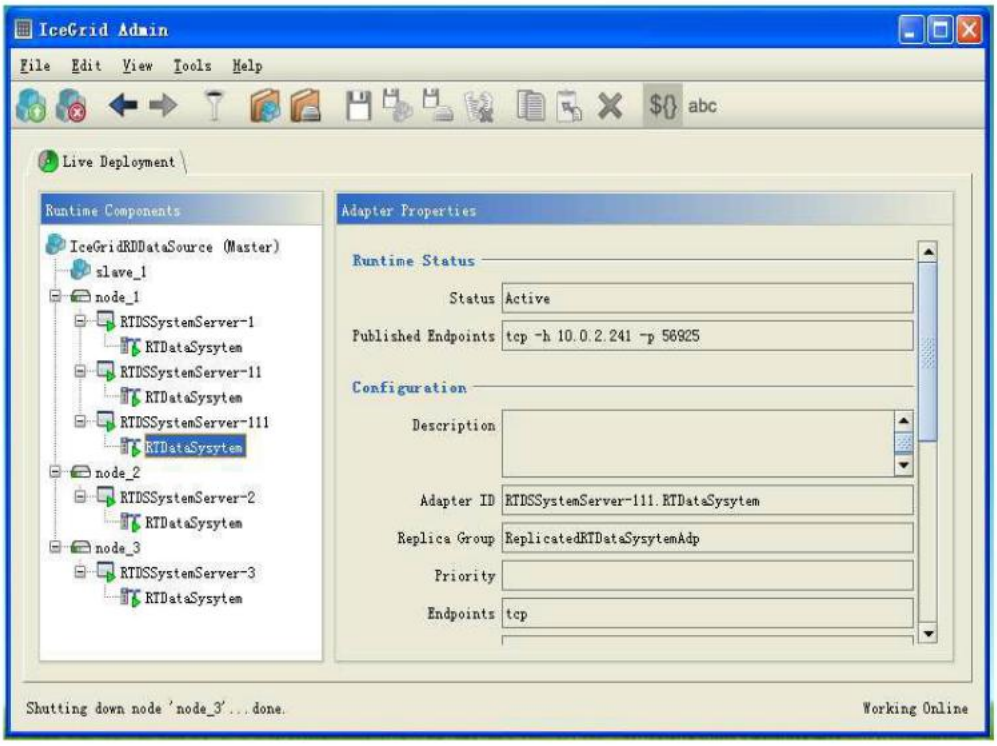

图 3.2.5 实时管理（服务已激活）

3. 由图 3.2.4 和 3.2.5 可以看到，该应用部署的名字是 IceGridRDDataSource，它进行了主/从注册服务的热备（Master/slave_1）,当 Master 崩溃后，slave_1 就会接收客户端的请求；同时该应用有三个节点（node_1,node_2,node_3）,每个节点都分布了同一类服务，通过负载均衡的方式共同完成客户端的请求，其中 node_1 部署了三个，node_2 和 node_3 分别部署了一个。
4. 注册服务（Master/Slave）可以通过该管理工具被关闭，在其上面点击右键，选取弹出的菜单项“shutdown”。只要存在一个活动状态的注册服务，整个应用依然能够提供服务。由于该工具登陆时默认使用的是主注册服务（Master），所以，当关闭 Master 后管理界面将被关闭，如果想再次进入管理界面，需要重新登陆并选择 slave 注册服务器，因为主注册服务已经被关闭。

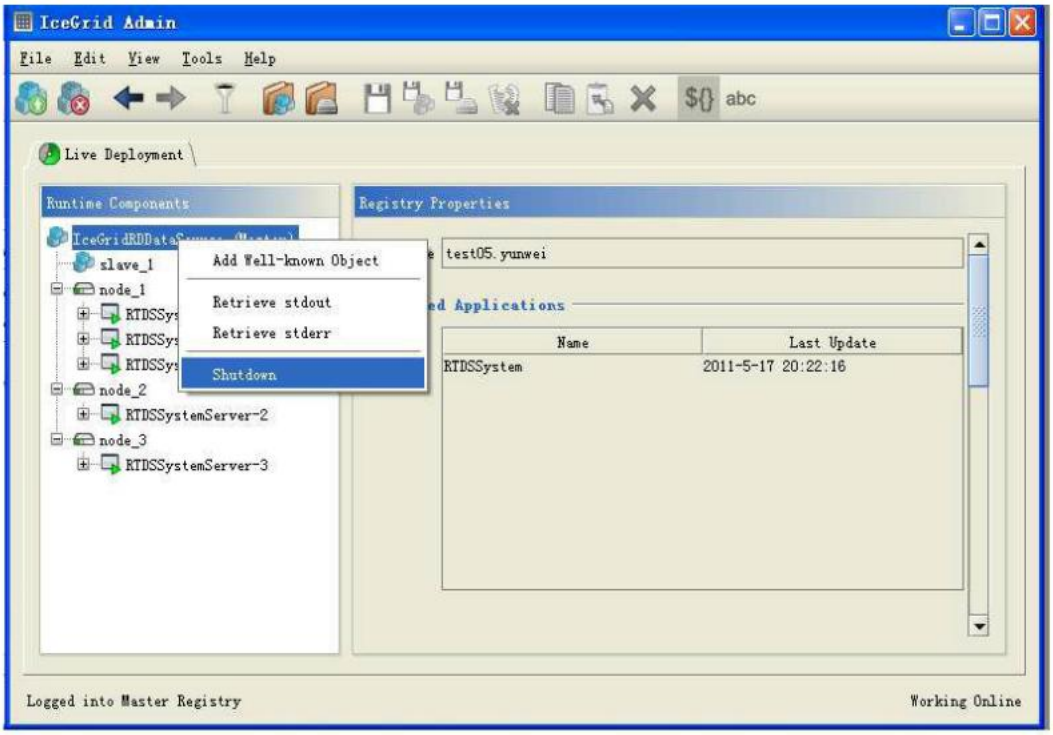

图 3.2.6 关闭 Master

5. 同注册服务器一样，各个节点也是可以用同样的方式被关闭，同样由于主注册服务和节点 1 的应用服务集成在一起，当节点 1 被关闭时，主注册服务也将结束。如果希望再次进入管理界面，需要用从注册服务（slave）重新登陆。

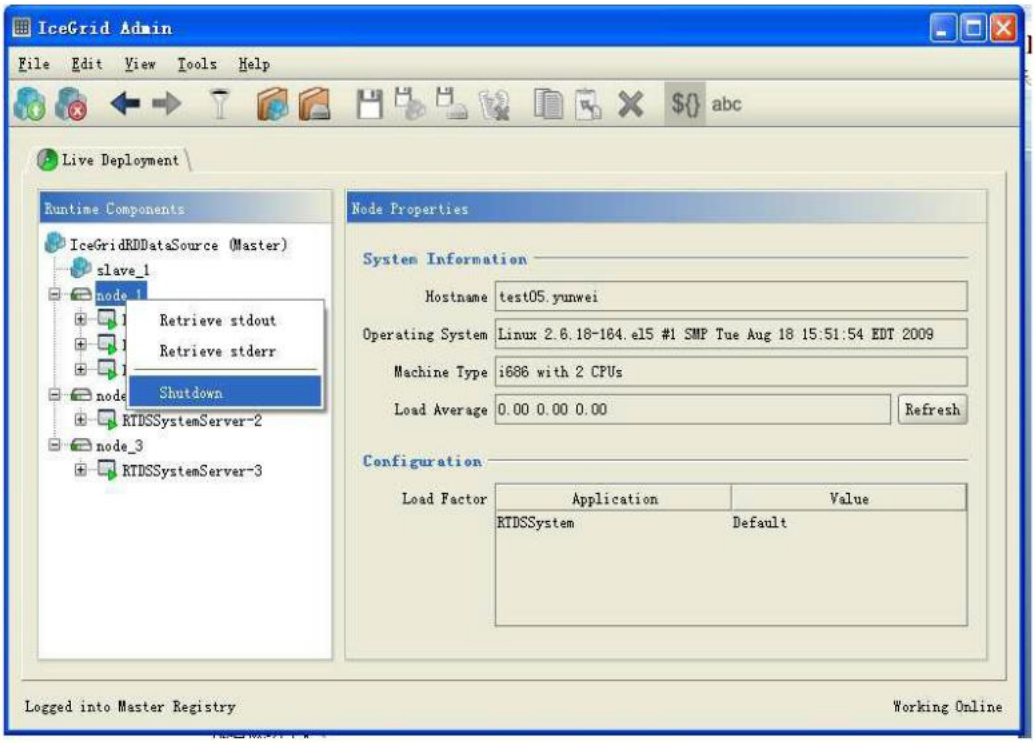

图 3.2.7 关闭节点

6. 每个节点上的服务可以被启动、停止，也可以被设置为有效或失效，还有一些其它的功能菜单，这里不再详述，这些操作通过右键菜单都可以方便的进行。

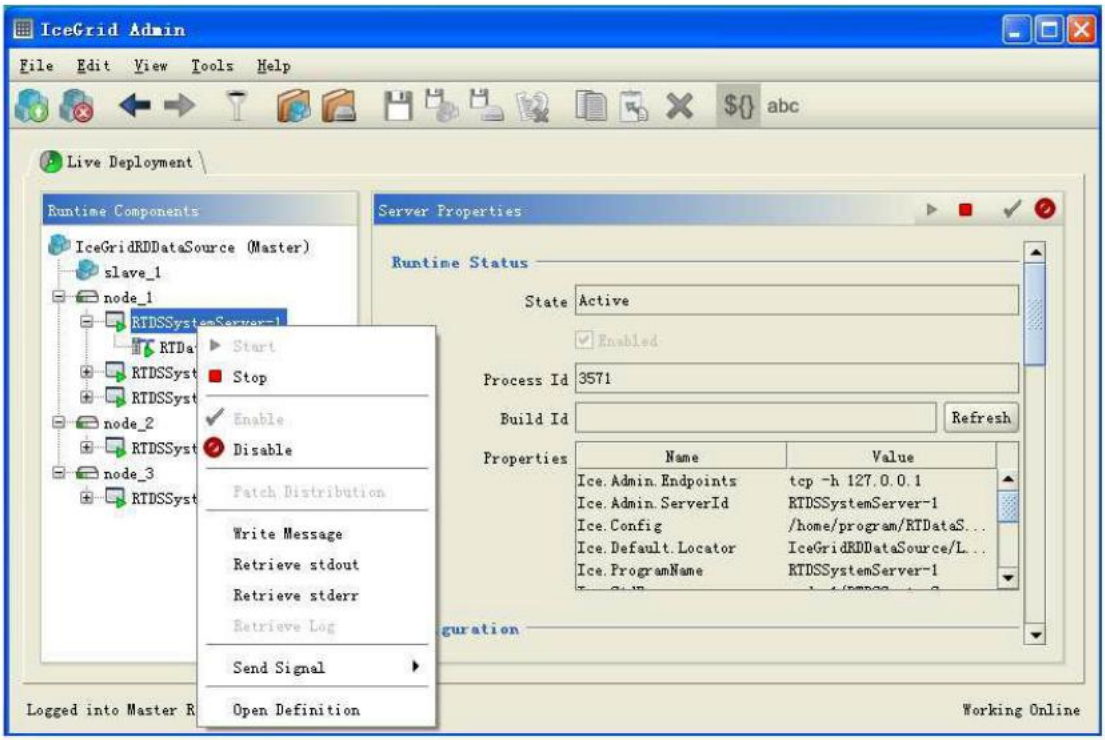

图 3.2.8 关闭节点

7. 如果我们能看到部署的各个注册服务、节点、各个应用服务，并能操纵正常，就说明部署成功了。

4. 高级应用配置

4.1 集成 IceBox

在文档《IceBox 开发和配置》（当前是 1.0 版）中，介绍了一个 IceBox 服务程序的开发方法和单独应用中配置和管理的过程。在实际的应用中，IceBox 服务通常集成到 IceGrid 中，并通过 IceGrid 进行激活和部署。

本章节中 IceBox 服务是集成在 IceGrid 中，并通过 IceGrid 进行部署，所以 IceBox 服务的配置信息不再同《IceBox 开发和配置》中一样在 config.icebox 中描述，而是直接配置在部署文件 app.xml 中。那也就是说，IceGrid 集成 IceBox 服务，只需要在 app.xml 文件中添加 Icebox 服务相关的配置信息就可以了。事实上，有关 Ice 所有的配置信息（除 IceGrid 自身的配置信息），都可以添加到 app.xml 中，并通过 icegrid 部署后生效。

下面各节详细描述 IceBox 服务的集成过程。

4.1.1 IceBox 服务程序编写

请参考文档《IceBox 开发和配置》，这里不再详述。由于 IceBox 服务相关的配置信息都放在了 app.xml 中，并且服务是通过 IceGrid 按需激活的，因此这里程序代码略有调整。下面列出 IceBox 服务的实现代码：

文件名：ServerService.java

```
import main.java.DataSource;
import IceBox.Service;

public class ServerService implements IceBox.Service {
    /**
     * @param name 配置文件中的service名称
     * @param communicator对象,由IceBox.ServiceManager负责创建和销毁。
     *          可能同时被其他服务共享使用（由配置文件决定），object Adapter的名
     *          称必须是唯一的；
     * @param args  配置文件中的参数列表
     * @Override
     **/
    public void start(String name, Ice.Communicator communicator,
                      String[] args){
        //创建objectAdapter，名称有配置文件决定
```

```
        Adapter = communicator.createObjectAdapter(
                              "RTDataSystem-"+name);

        //创建servant
        String RTDataSourceIdentity = communicator.getProperties().
                              getProperty("RTDataSource.Identity");
        DataSource objDataSrc = new DataSource("dataSource");
        Adapter.add(objDataSrc,
            communicator.stringToIdentity(RTDataSourceIdentity));

        Adapter.activate();
    }

    /**
     *
     * @param args
     * @Override
     *
     **/
    public void stop()
    {
        Adapter.destroy();
    }

    private Ice.ObjectAdapter Adapter;
}
```

4.1.2 IceGrid 集成 IceBox 服务

IceGrid 集成 IceBox 只和部署文件（app.xml）有关，IceBox 服务（service）的粒度和普通的 server 是一样的，因此 IceBox service 的部署和普通的 server 非常类似，它同样有模板、服务（service）和实例化的概念，可以将 IceBox service 理解为一个特殊的 server。

为了能更清楚的描述这个集成配置的过程，在 IceGrid 配置的基础上，添加 IceBox 服务。具体目标如下：

1. 集成 ServerService 服务（service），并且 ServerService 服务（service）使用的服务对象和之前 server 的服务对象使用同一个（type--::RTDataSystem::RTDataSource）
2. 在节点 1（node_1）上添加 IceBox 服务功能（IceBox-Node1），这个 IceBox 服务包含了 5 个 ServerService 服务；同样的在节点 2（node_2）上也添加一个 IceBox 服务功

能（IceBox-Node2），也包含了 5 个 ServerService 服务

3. 这些 IceBox 服务中分布的多个服务（service）和之前已经存在的服务（server）一起通过 IceGrid 实现负载均衡

为了实现上述的功能，需要添加 IceBox 服务的相关配置，首先看一下此时 app.xml 的变化，变化和添加部分用浅灰阴影标出。

app.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<icegrid>
  <application name="RTDSSystem">
    <server-template id="RTDSSystemServer">
      <parameter name="index"/>
      <server id="RTDSSystemServer-${index}" exe="java" activation="on-demand">
        <adapter name="RTDataSysytem" endpoints="tcp"
                 replica-group="RTDataSystemGroup"/>
        <option>-jar</option>
        <option>ServerApp.jar</option>
    </server>
    </server-template>

<!— begin 服务模板定义-->
1   <service-template id="RTDSystemService">
2       <parameter name="name"/>
3       <service name="${name}" entry="ServerService">
4           <description>A simple service named after ${name}</description>
5           <properties>
6             <property name="RTDataSource.Identity" value="RTDataSource"/>
7           </properties>
8           <adapter name="RTDataSystem-${name}" endpoints="tcp"
                      id="RTDataSystem-${name}" replica-group="RTDataSystemGroup"
                      server-lifetime="false"/>
11      </service>
12  </service-template>
<!-- end 服务模板定义-->

    <replica-group id="RTDataSystemGroup">
      <load-balancing type="round-robin"/>
      <!--load-balancing type="ordered" /-->
      <!--load-balancing type="adaptive" /-->
      <!--load-balancing type="random" n-replicas="0"/-->
      <object identity="RTDataSource" type="::RTDataSystem::RTDataSource"/>
    </replica-group>
```

```
        <node name="node_1">
            <server-instance template="RTDSSystemServer" index="1"/>
            <server-instance template="RTDSSystemServer" index="11"/>
            <server-instance template="RTDSSystemServer" index="111"/>
    <!— begin IceBox 服务配置  IceBox-Node1-->
1       <icebox id="IceBox-Node1" activation="on-demand" exe="java">
2           <description>A sample IceBox server IceBox-Node1</description>
3           <option>IceBox.Server</option>
4           <properties>
5               <property name="IceBox.InstanceName" value="${server}"/>
6               <property name="Ice.Admin.Endpoints" value="tcp -h 10.0.2.241"/>
7               <property name="IceBox.Trace.ServiceObserver" value="1"/>
8           </properties>
9           <service-instance template="RTDSystemService" name="one"/>
10          <service-instance template="RTDSystemService" name="two"/>
11          <service-instance template="RTDSystemService" name="three"/>
12          <service-instance template="RTDSystemService" name="four"/>
13          <service-instance template="RTDSystemService" name="five"/>
14      </icebox>
    <!— end IceBox 服务配置  IceBox-Node1-->
        </node>
        <node name="node_2">
            <server-instance template="RTDSSystemServer" index="2"/>
            <server-instance template="RTDSSystemServer" index="22"/>
            <server-instance template="RTDSSystemServer" index="222"/>
    <!— begin IceBox 服务配置  IceBox-Node2-->
1       <icebox id="IceBox-Node2" activation="on-demand" exe="java">
2           <description>A sample IceBox server IceBox-Node2</description>
3           <option>IceBox.Server</option>
4           <properties>
5               <property name="IceBox.InstanceName" value="${server}"/>
6               <property name="Ice.Admin.Endpoints" value="tcp -h 10.0.2.242"/>
7               <property name="IceBox.Trace.ServiceObserver" value="1"/>
8           </properties>
9           <service-instance template="RTDSystemService" name="2-one"/>
10          <service-instance template="RTDSystemService" name="2-two"/>
11          <service-instance template="RTDSystemService" name="2-three"/>
12          <service-instance template="RTDSystemService" name="2-four"/>
13          <service-instance template="RTDSystemService" name="2-five"/>
14      </icebox>
    <!— begin IceBox 服务配置  IceBox-Node2-->
        </node>
```

```
    <node name="node_3">
      <server-instance template="RTDSSystemServer" index="3"/>
    </node>
  </application>
</icegrid>
```

app.xml 中增加的 IceBox 服务相关的配置部分如下：

- **服务摸板(Service Template)：**

  可以对比一下 server template 的定义，两者基本上没有什么区别，最大的不同是 Server template 中 server 是指定一个可执行的程序，而 service 中指定的是动态加载的组件入口。以下解释上述配置中的服务模板的定义：

  第 1 行指定定义模板的 id，唯一标志一个服务模板，第 12 是闭合标签；

  第 2 行定义了一个参数 name，默认值是“name”；

  第 3~11 行定义了模板中使用的服务（service），并在该 service 中指定了名称、入口、描述信息、配置属性，定义了一个对象适配器；

  第 4 行，是该服务的描述信息；

  第 5~7 行，是属性定义列表，这里定义了一个属性 RTDataSource.Identity，并指定其值为 RTDataSource；

  第 8 行，定义了一个对象适配器，指定了其 name、endpoints、id、replica-group 等属性信息，这个基本上和 server 中 adapter 的定义没有什么区别

  以上内容就是 service 模板的定义。

- **IceBox 服务(IceBox-Node1)：**

  icebox 服务的定义被包含在分布的服务器节点中，主要包括三部分的信息：

  1. IceBox 服务的启动配置信息
  2. IceBox 的属性配置信息
  3. Service 服务实例化列表

  下面解释这块内容：

  第 1~3 行，指定了 IceBox 服务的名称，启动方式，启动执行程序等

  第 4~8 行，指定了 IceBox 服务的属性配置列表，这里定义了 IceBox 服务的实例名称、管理器访问端点以及 service 被跟踪的级别

  第 9~13 行，实例化了 5 个 service 服务

至此，一个包含了 5 个 serverservice 服务的 IceBox 服务被集成在 node1 中。

- **IceBox 服务(IceBox-Node2)**

同 IceBox 服务(IceBox-Node1)中描述，只是具体 value 有所不同，这里不再解释。

4.1.3 测试验证

验证方式同第 3 章，这里不再赘述。部署完成后，就可以通过 IceGridGUI.jar 程序来进行管理，基本情况如下图所示：

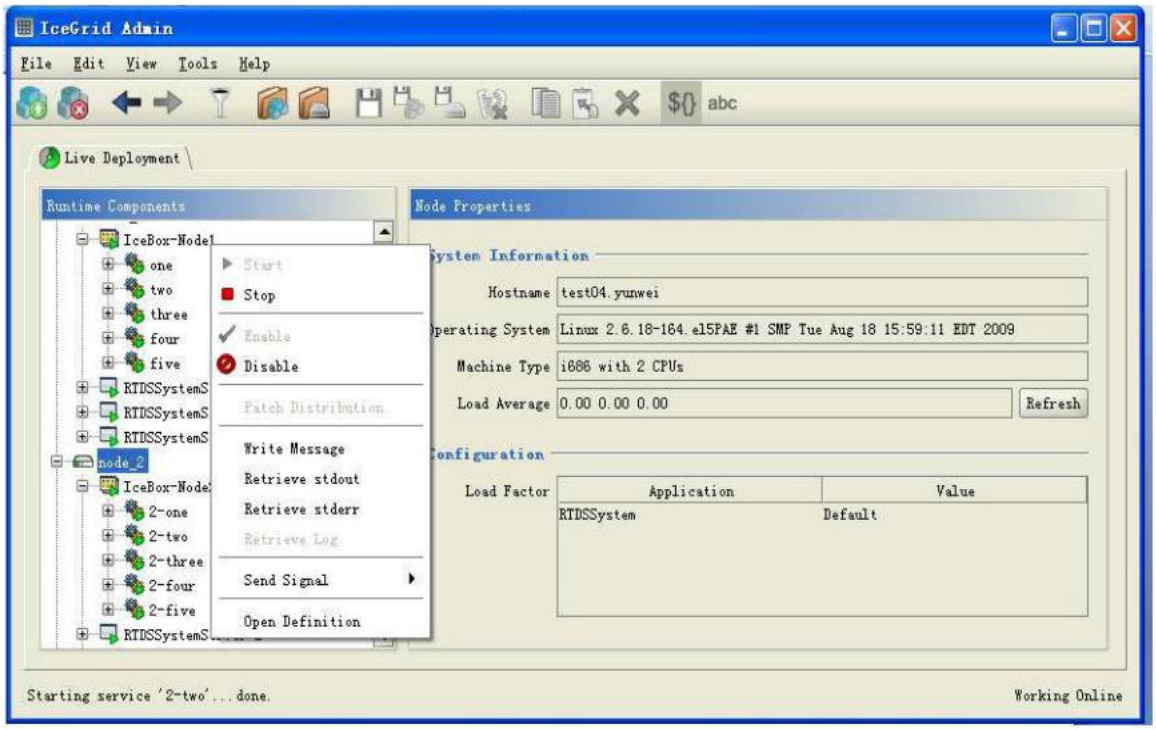

图 4.1.3.1 IceGridGUI 工具管理 IceBox 服务

运行测试程序（jsp）进行远程调用，打开浏览器，在地址栏中输入：http://10.0.2.243:8080/testApp/，按回车键。如果能得到结果，就说明服务器端已经正常工作了，如下图所示：

图 4.1.3.2 jsp 客户端远程调用

查看服务器端 IceBox 服务中 service 执行的情况，以节点 1 为例（对应的服务器 ip：10.0.2.241），监测 service 运行的情况：

tail –f /home/program/RTDataSystem/node_1/IceBox-Node1.out

多刷新几次浏览器，此时 IceGrid 就会通过负载均衡，将客户端的一些请求转移过来，如下图所示：

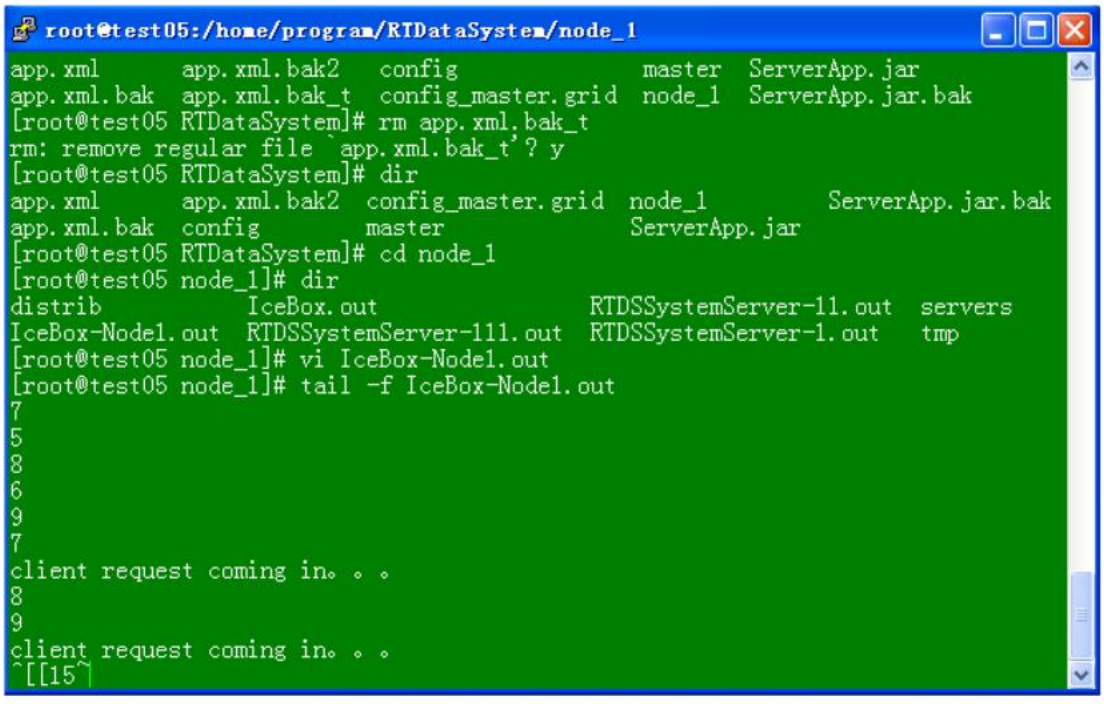

图 4.1.3.3 服务器端 service 响应客户端请求

4.2 集成 IcePatch2

下个版本补充