**Terracotta和SmartFoxServer 2X集群的配置**

**--**用terracotta和SmartFoxServer 2X 构建高可用性和高扩展性的多用户服务器端集群

编写人：陈杰

**一、集群介绍**

在本文中，我们会深入研究如何构建高可用性、无单点故障（no single point of failure）、高横向扩展性和快速故障恢复的SmartFoxServer集群。

**1、什么是集群**

概括的讲，服务器集群是为了提高一个或多个服务的可用性和性能，而协同工作的一个计算机组（a group of computers）。集群中的每个计算机称为一个“节点”（node），根据集群的架构不同，每个节点在集群中可以有不同的角色，起不同的作用。比如可以有通用服务器节点、网关节点、缓存节点、备份节点等。

从客户端的角度看，集群常常被视为一个单独的计算机/实体（computer / entity）。

**2、研究目标**

研究的目标是为了构建基于SmartFoxServer的在线多用户游戏服务的一个原型，使其随着应用的流量和知名度的提升仍能提供高可用性、快速故障恢复和高横向扩展性。

在下面的几章，我们将检验这个服务的架构，使用的技术并分析在本研究中服务器端使用的代码。

**二、Terracotta配置**

1. **Terracotta服务器组的配置**

Terracotta服务器组，主要是为了协调每一个Terracotta客户端(这里指SFS2X)之间的数据通信和同步。为了避免Terracotta服务器成为整个集群中的脆弱和单点故障的点。我们使用Terracotta服务器组的形势。就是使用两台Terracotta服务器，运行在主从模式下。这样当主Terracotta服务器出现故障的时候，从Terracotta服务器可以接着主Terracotta服务器而运行。

整个配置如下tc-config.xml:

<?xml version="1.0" encoding="UTF-8"?>

<!--

All content copyright Terracotta, Inc., unless otherwise indicated. All rights reserved.

-->

<tc:tc-config xmlns:tc="http://www.terracotta.org/config"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.terracotta.org/schema/terracotta-6.xsd">

<!-- Tell DSO where the Terracotta server can be found;

See

- Terracotta Configuration Guide and Reference

- About Terracotta Configuration Files

for additional information. -->

<tc-properties>

<property name="12.l1reconnect.enabled" value="true" />

<property name="12.l1reconnect.timeout.millis" value="5000" />

</tc-properties>

<servers>

<server host="192.168.168.88" name="server1">

<dso-port>9510</dso-port>

<jmx-port>9520</jmx-port>

<data>%(user.home)/terracotta/server1/server-data</data>

<logs>%(user.home)/terracotta/server1/server-logs</logs>

<statistics>%(user.home)/terracotta/server1/server-statistics</statistics>

</server>

<server host="192.168.168.88" name="server2">

<dso-port>9511</dso-port>

<jmx-port>9522</jmx-port>

<data>%(user.home)/terracotta/server2/server-data</data>

<logs>%(user.home)/terracotta/server2/server-logs</logs>

<statistics>%(user.home)/terracotta/server2/server-statistics</statistics>

</server>

</servers>

<clients>

<logs>%(user.home)/terracotta/client-logs</logs>

</clients>

</tc:tc-config>

将该配置文件放在启动start-tc-server.bat或者start-tc-server.sh的目录下，由于使用了多台Server，启动的时候，需要加上Server的名称

上例中启动server1的时候，使用：

cmd> C:\Program Files\Terracotta\terracotta-ee-3.6.1\bin>start-tc-server.bat -n serve

r1

启动server2的时候，将server1改成server2即可

1. **Terracotta客户端的配置—即Terracotta和SmartForServer 2X的整合**

为了让Terracotta可以识别SmartFoxServer 2X,或者说，让SFS2X加入到Terracotta集群中去，我们需要修改SFS2X的启动脚本。

SFS2X原来的启动脚本如下：

@java -cp "./;lib/\*;lib/Jetty/\*;extensions/\_\_lib\_\_/\*" -Dfile.encoding=UTF-8 com.smartfoxserver.v2.Main $1 $2 $3

修改后的启动脚本如下：

@java -Xbootclasspath/p:"D:/Terracotta/terracotta-ee-3.6.1/lib/dso-boot/dso-boot-hotspot\_win32\_160\_22.jar" -Dtc.install-root=D:/Terracotta/terracotta-ee-3.6.1 -Dtc.config=tc-cluster.xml -cp "./;lib/\*;lib/Jetty/\*;extensions/\_\_lib\_\_/\*" -Dfile.encoding=GBK com.smartfoxserver.v2.Main 1% %2 %3

加入了红色的部分让Terracotta识别SFS2X实例以及修改了-Dfile.encoding参数，这样控制台可以显示中文。

注意：上面使用了tc-cluster.xml作为Terracotta客户端的配置文件。

1. **Terracotta客户端的配置文件tc-cluster.xml**

<?xml version="1.0" encoding="UTF-8"?>

<!--

All content copyright (c) 2003-2007 Terracotta, Inc.,

except as may otherwise be noted in a separate copyright notice.

All rights reserved.

-->

<tc:tc-config xmlns:tc="http://www.terracotta.org/config"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.terracotta.org/schema/terracotta-6.xsd">

<!-- Tell DSO where the Terracotta server can be found;

See

- Terracotta Configuration Guide and Reference

- About Terracotta Configuration Files

for additional information. -->

<tc-properties>

<property name="12.l1reconnect.enabled" value="true" />

<property name="12.l1reconnect.timeout.millis" value="5000" />

</tc-properties>

<servers>

<server host="192.168.168.88" name="server1">

<dso-port>9510</dso-port>

<jmx-port>9520</jmx-port>

<data>%(user.home)/terracotta/server1/server-data</data>

<logs>%(user.home)/terracotta/server1/server-logs</logs>

<statistics>%(user.home)/terracotta/server1/server-statistics</statistics>

</server>

<server host="192.168.168.88" name="server2">

<dso-port>9511</dso-port>

<jmx-port>9522</jmx-port>

<data>%(user.home)/terracotta/server2/server-data</data>

<logs>%(user.home)/terracotta/server2/server-logs</logs>

<statistics>%(user.home)/terracotta/server2/server-statistics</statistics>

</server>

</servers>

<clients>

<logs>%(user.home)/terracotta/client-logs</logs>

</clients>

<application>

<dso>

<!-- Our app requires these custom objects/classes to be shared - the following declarations

tells DSO which ones they are. When the app runs under DSO, instances of these classes

will broadcast changes in their state.

A good idiom when writing an app that you intend to cluster via TC DSO, is to group the

classes you wish to share under a single package (although if you follow the MVC pattern

this tends to happen naturally) - this way the list of classes you wish to instrument

can be concise -->

<instrumented-classes>

<include>

<class-expression>demo.cluster.data.\*</class-expression>

</include>

</instrumented-classes>

<!-- These methods (originating from local objects) operates on objects declared as shared. This

section tells DSO to assume a lock on those objects for the duration of the call; essentially this

section declares that all methods found for all classes found for all packages should assume the

behavior described -->

<locks>

<autolock>

<method-expression>\* \*..\*.\*(..)</method-expression>

</autolock>

</locks>

<!-- We declare the following fields a root, making it

available for all instances of our app that runs via DSO -->

<roots>

<root>

<field-name>demo.cluster.data.DataStore.usersInCluster</field-name>

</root>

<root>

<field-name>demo.cluster.data.DataStore.postOffice</field-name>

</root>

</roots>

</dso>

</application>

</tc:tc-config>

application标签上面的部分一定要和Terracotta服务器组的tc-cluster.xml配置一致。

application标签内配置需要共享的数据以及使用的锁机制等信息，具体的可以参考Terracotta官方文档。

注意：扩展中需要共享的数据就是在这里配置。

1. **修改SFS2X的启动代码。**

上面的配置虽然可以将SFS2X成功加入Terracotta集群。但是，由于Terracotta要求所有加载共享数据的ClassLoader，必须命名。

而SFS2X加载每一个扩展之前都会创建一个URLClassLoader实例。使用该实例来加载扩展。这样的目的是为了使得SFS2X支持热部署。

如果不重新定义一个命名的ClassLoader，当在扩展中需要操作共享数据的时候，你会得到一个ClassLoader没有命名的异常。为了解决这个问题，我们需要重新实现SFS2X的加载扩展的机制。

我们使用一个自定义的类EkarmaClassLoader来加载我们需要集群的扩展。

**public** **class** EkarmaClassLoader **extends** URLClassLoader **implements** NamedClassLoader{

**private** **static** **final** String *NAME* = "EkarmaClassLoader";

**private** String loaderName;

**public** EkarmaClassLoader(URL[] classpath, ClassLoader parent) {

**super**(classpath, parent);

**try** {

ClassLoader parentLoader = EkarmaClassLoader.**class**.getClassLoader();

Class<?> namedClassLoader = parentLoader.loadClass("com.tc.object.loaders.NamedClassLoader");

Class<?> helper = parentLoader.loadClass("com.tc.object.bytecode.hook.impl.ClassProcessorHelper");

Method m = helper.getMethod("registerGlobalLoader", **new** Class[] { namedClassLoader });

m.invoke(**null**, **new** Object[] { **this** });

} **catch** (Exception e) {

e.printStackTrace();

}

System.*out*.println("实例化了一个EkarmaClassLoader");

}

**public** String \_\_tc\_getClassLoaderName() {

**return** *NAME*;

}

@Override

**public** **void** \_\_tc\_setClassLoaderName(String arg0) {

**this**.loaderName = arg0;

}

}

注意，这个类实现了NamedClassLoader接口，这个接口是Terracotta提供的。然后我们在实例化EkarmaClassLoader的时候，使用ClassProcessorHelper的静态方法registerGlobalLoader来注册。也许这里，应该放在一个静态的区域来完成，否则每次实例化的时候，都调用了一次registerGlobalLoader.而这里需要的名称是EkarmaClassLoader这个类的名称，而不是每个实例的名称。开始的时候，将所有扩展都使用这个类，结果找不到共享数据引用到的数据。然后，限制只有名称为cluster的扩展才使用这个类来加载。这样变相地只注册了一次该命名的ClassLoader类。后面还待测试。

具体的细节可以参看修改后的启动源码。

完成修改后，将SFS2X\lib目录下的boot.jar替换为我们修改后的boot.jar，同时不要忘记将tim-api-1.3.0.jar也一并放入lib文件夹下。

**三、Terracotta和SFS2X集群的例子**

关于sfs如何实现扩展的一些细节，可以参看SFS集群\目录下SmartFoxServer集群.doc。这个是sfs1.6和terracotta集群的实现。这个例子也同样采用类似的机制，完成部分功能。

源码和配置注意事项，在SFS2X集群扩展实例服务器端和客户端\目录下。

这里略去。

同时，需要注意一点：

在SFS2X\目录下需要有一个cluster-server.properties.这个文件是扩展的例子中需要使用的。这个文件主要配置了本节点的IP和SFS2X运行的端口号。

如果没有配置的话，端口号使用默认的9933.

该配置文件中配置的信息主要是用于唯一标示每一个节点。使用IP:Port来唯一标识每一个节点。