关键点：

（1）如果想提高并发的话，还是在一个node上对于同一个application开多个server instance.

1.名词解释

。Replica groups

A replica group is the term for a collection of replicated object adapters. An

application can create any number of replica groups. Each group requires a

unique identifier.

• Nodes

An application must assign its servers to one or more nodes.

• Servers

A server’s description includes a unique name and the path to its executable. It

also lists the object adapters it creates.

• Object adapters

Information about an object adapter includes its endpoints and any wellknown

objects it advertises. If the object adapter is a member of a replica

group, it must also supply that group’s identifier.

• Objects

A well-known object is one that is known solely by its identity. The registry

maintains a global list of such objects for use during locate requests.

IceGrid uses the term descriptor to refer to the description of an application

代理

直接代理;：对象 和 地址 。例如，代理：  SimplePrinter:default -p 10000

间接代理：两种。第一种仅仅包含well-known objects(众所周知对象)，例如代理  SimplePrinter 就是对象SimplePrinter的代理。第二种包含对象和对象适配器标识，例如，SimplePrinter@PrinterAdapter

2.一个简单的ice架构（ice registry只提供定位服务）

(1)客户端代码：

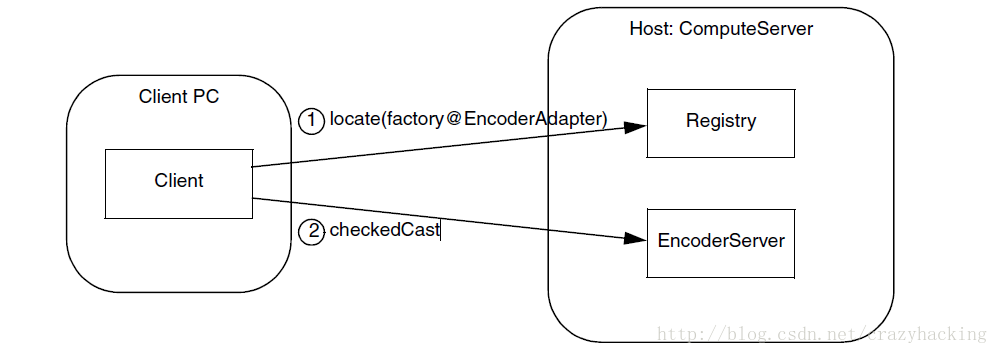
Ice::ObjectPrx proxy =

communicator->stringToProxy("factory@EncoderAdapter");//第一步是定位服务，通过括号内的"对象@对象适配器id"找到了代理

Ripper::MP3EncoderFactoryPrx factory =

Ripper::MP3EncoderFactoryPrx::checkedCast(proxy); //第二步是使用代理得到想要的远端对象。是对远端对象的第一次调用

Ripper::MP3EncoderPrx encoder = factory->createEncoder();



(2)配置文件

registry 配置：

/opt/ripper/registry.cfg contains the following properties:（前五个必选）

IceGrid.InstanceName=inputtest   //如果不写。默认是IceGrid

IceGrid.Registry.Client.Endpoints=tcp -p 4061

IceGrid.Registry.Server.Endpoints=tcp

IceGrid.Registry.Internal.Endpoints=tcp

IceGrid.Registry.Data=/opt/ripper/registry

IceGrid.Registry.DynamicRegistration=1

IceGrid.Registry.AdminPermissionsVerifier=IceGrid/NullPermissionsV

erifier

IceGrid.Registry.AdminCryptPasswords=../etc/passwd //这个选项倒是很有意思，可以设置操作admin的权限

client配置文件：

Ice.Default.Locator=IceGrid/Locator:tcp -h registryhost -p 4061 （在没有写IceGrid的名字的情况下）

server配置，opt/ripper/server.cfg

EncoderAdapter.AdapterId=EncoderAdapter （对象适配器id）

EncoderAdapter.Endpoints=tcp  //不写端口地址，表明系统自动分配端口

Ice.Default.Locator=IceGrid/Locator:tcp -h registryhost -p 4061

（3）启动

启动registry

icegridregistry --Ice.Config=/opt/ripper/registry.cfg

启动server

/opt/ripper/bin/server     --Ice.Config=/opt/ripper/server.cfg

3 使用ice registry来管理server(xml和node)

3.1 使用xml文件描述server,使用node

为了使用icegridadmin管理工具，我们使用xml配置sever文件：

<icegrid>

<application name="Ripper">

<node name="Node1">

<server id="EncoderServer"

exe="/opt/ripper/bin/server"

activation="on-demand">

<adapter name="EncoderAdapter"

id="EncoderAdapter"

endpoints="tcp"/>

</server>

</node>

</application>

</icegrid>

事实上，以上的描述文件会生成一个server的配置文件如下，由node生成（文件位置在类似/opt/ripper/node/servers/EncoderServer/config/config）：

# Server configuration

Ice.Admin.ServerId=EncoderServer

Ice.Admin.Endpoints=tcp -h 127.0.0.1

Ice.ProgramName=EncoderServer

# Object adapter EncoderAdapter

EncoderAdapter.Endpoints=tcp

EncoderAdapter.AdapterId=EncoderAdapter

Ice.Default.Locator=IceGrid/Locator:default -p 4061

# Registry properties

IceGrid.Registry.Client.Endpoints=tcp -p 4061

IceGrid.Registry.Server.Endpoints=tcp

IceGrid.Registry.Internal.Endpoints=tcp

IceGrid.Registry.AdminPermissionsVerifier=IceGrid/NullPermissionsV

erifier

IceGrid.Registry.Data=/opt/ripper/registry

# Node properties

IceGrid.Node.Endpoints=tcp

IceGrid.Node.Name=Node1

IceGrid.Node.Data=/opt/ripper/node

IceGrid.Node.CollocateRegistry=1  //标明registry和node在一个进程中

Ice.Default.Locator=IceGrid/Locator:tcp -p 4061

因为使用协同的registry和node，所以一个命令可以启动registry和node

icegridnode --Ice.Config=/opt/ripper/config

说明：（1）如果没有标明adapter id,那么默认生成id名字为：EncoderServer.EncoderAdapter

3.2 增加节点

（1）server配置

<icegrid>

<application name="Ripper">

<node name="Node1">

<server id="EncoderServer1"

exe="/opt/ripper/bin/server"

activation="on-demand">

<adapter name="EncoderAdapter"

endpoints="tcp"/>

</server>

</node>

<node name="Node2">

<server id="EncoderServer2"

exe="/opt/ripper/bin/server"

activation="on-demand">

<adapter name="EncoderAdapter"

endpoints="tcp"/>

</server>

</node>

</application>

</icegrid>

（2）node配置

IceGrid.Node.Endpoints=tcp

IceGrid.Node.Name=Node2

IceGrid.Node.Data=/opt/ripper/node

Ice.Default.Locator=IceGrid/Locator:tcp -h registryhost -p 4061

（3）客户端

缺陷就是每个server上一个不同的object adapter id，需要进行选择。（这对于增删节点等都很麻烦）

非直接代理：

• factory@EncoderServer1.EncoderAdapter

• factory@EncoderServer2.EncoderAdapter

string adapter;

if ((rand() % 2) == 0)

adapter = "EncoderServer1.EncoderAdapter";

else

adapter = "EncoderServer2.EncoderAdapter";

Ice::ObjectPrx proxy =

communicator->stringToProxy("factory@" + adapter);

Ripper::MP3EncoderFactoryPrx factory =

Ripper::MP3EncoderFactoryPrx::checkedCast(proxy);

Ripper::MP3EncoderPrx encoder = factory->createEncoder();

4 well-known objects(众所周知对象)

4.1  概念

有两种类型的简介代理：一种指定对象标识和对象适配器标志(object@object\_adapter\_id) ；一种仅仅指定对象标志。后者称为众所周知代理（well-known proxy）,它指向well-known object（众所周知对象）。

可以通过以下几个地方声明：

XML文件。          （statically in descriptors,）

icegrid admin的编程接口           • programmatically using IceGrid’s administrative interface,

使用 ice admin工具动态添加         • dynamically using an IceGrid administration tool.

下面是一个例子，well-known对象和它们的代理

Identity Proxy

Object1 Object1:tcp -p 10001

Object2 Object2@TheAdapter

Object3 Object3

4.2 生成well-known对象

（1）xml文件中

<icegrid>

<application name="Ripper">

<node name="Node1">

<server id="EncoderServer"

exe="/opt/ripper/bin/server"

activation="on-demand">

<adapter name="EncoderAdapter"

id="EncoderAdapter"

endpoints="tcp">

<object identity="EncoderFactory"

type="::Ripper::MP3EncoderFactory"/>

</adapter>

</server>

</node>

</application>

</icegrid>

（2）通过编程接口

接口api说明如下，

module IceGrid {

interface Admin {

...

void addObject(Object\* obj)

throws ObjectExistsException,

DeploymentException;

void updateObject(Object\* obj)

throws ObjectNotRegisteredException,

DeploymentException;

void addObjectWithType(Object\* obj, string type)

throws ObjectExistsException,

DeploymentException;

void removeObject(Ice::Identity id)

throws ObjectNotRegisteredException,

DeploymentException;

...

};

};

编程实现，

Ice::ObjectAdapterPtr adapter =

communicator->createObjectAdapter("EncoderAdapter");

Ice::Identity ident =

communicator->stringToIdentity("EncoderFactory");

FactoryPtr f= new FactoryI;

Ice::ObjectPrx factory = adapter->add(f, ident);

IceGrid::AdminPrx admin = // ...

try {

// admin->addObject(factory); // OOPS!，要求首先激活object adapter

admin->addObjectWithType(factory, factory->ice\_id());

} catch (const IceGrid::ObjectExistsException &) {

admin->updateObject(factory);

}

(3)

icegridadmin --Ice.Config=/opt/ripper/config

>>> object add "EncoderFactory@EncoderAdapter"

4.3 使用ice query查找 well-known objects

（1）接口说明：

module IceGrid {

enum LoadSample {

LoadSample1,

LoadSample5,

LoadSample15

};

interface Query {

idempotent Object\* findObjectById(Ice::Identity id);

idempotent Object\* findObjectByType(string type);

idempotent Object\* findObjectByTypeOnLeastLoadedNode(

string type, LoadSample sample);

idempotent Ice::ObjectProxySeq findAllObjectsByType(

string type);

idempotent Ice::ObjectProxySeq findAllReplicas(Object\* proxy);

};

};

(2)示例,

一个含有两个well-known objects的例子

<icegrid>

<application name="Ripper">

<node name="Node1">

<server id="EncoderServer1"

exe="/opt/ripper/bin/server"

activation="on-demand">

<adapter name="EncoderAdapter"

endpoints="tcp">

<object identity="EncoderFactory1"

type="::Ripper::MP3EncoderFactory"/>

</adapter>

</server>

</node>

<node name="Node2">

<server id="EncoderServer2"

exe="/opt/ripper/bin/server"

activation="on-demand">

<adapter name="EncoderAdapter"

endpoints="tcp">

<object identity="EncoderFactory2"

type="::Ripper::MP3EncoderFactory"/>

</adapter>

</server>

</node>

</application>

</icegrid>

客户端查找，

1)使用findAllObjectsByType函数

Ice::ObjectPrx proxy =

communicator->stringToProxy("IceGrid/Query");

IceGrid::QueryPrx query = IceGrid::QueryPrx::checkedCast(proxy);

Ice::ObjectProxySeq seq;

string type = Ripper::MP3EncoderFactory::ice\_staticId();

seq = query->findAllObjectsByType(type);

if (seq.empty()) {

// no match

}

Ice::ObjectProxySeq::size\_type index = ... // random number

Ripper::MP3EncoderFactoryPrx factory =

Ripper::MP3EncoderFactoryPrx::checkedCast(seq[index]);

Ripper::MP3EncoderPrx encoder = factory->createEncoder();

2）使用findObjectByTypeOnLeastLoadedNode函数

Ice::ObjectPrx proxy =

communicator->stringToProxy("IceGrid/Query");

IceGrid::QueryPrx query = IceGrid::QueryPrx::checkedCast(proxy);

Ice::ObjectPrx obj;

string type = Ripper::MP3EncoderFactory::ice\_staticId();

obj = query->findObjectByTypeOnLeastLoadedNode(type,

IceGrid::LoadSample1);

if (!obj) {

// no match

}

Ripper::MP3EncoderFactoryPrx factory =

Ripper::MP3EncoderFactoryPrx::checkedCast(obj);

Ripper::MP3EncoderPrx encoder = factory->createEncoder();

到底怎样才是well-known对象？

1.1 replica group（对象适配器复制集群）

(1)   概念

一个复制集群是一组复制的对象适配器；有一个唯一的标识。

一个复制集群被确定之后，他的标识就可以被用在间接代理中替代适配器标识；比如，一个标识为PrinterAdapters的复制集群可以被用在如下的代理：

SimplePrinter@PrinterAdapters

（2）示例

<replica-group id="ReplicatedAdapter">

<load-balancing type="adaptive" load-sample="5"  n-replicas="2"/>

</replica-group>

load-balancing type 表示负载均衡策略；

load-sample 负载采样时间间隔，以分钟计；

n-replicas  返回两个负载最小的对象适配器的端点地址

负载平衡策略包括四种：Random ，Adaptive，Round Robin，Ordered。

（3）问题

采样的时候，linux系统采样的是load average，这样当机器cpu数目不同时，岂不是无法有效利用cpu了？

问题：

（1）既然registry location服务，那么是不是说一个client连接上之后，请求只会被发送到一个node的server上？没错。

但是通过设置，可以指定server，这样就可以发送到不同的node上的server 了

(2) adapter\_ = communicator->createObjectAdapter(name); //objectadapter名字

  Ice::PropertiesPtr properties = communicator->getProperties();

  const string& locator = properties->getProperty("Ice.Default.Locator");

  LOG(INFO, "locator: " << locator);

  BunnyConf::instance().set\_locator(locator);

  Ice::ObjectPtr object = new HandlerI(); //ice对象

  adapter\_->add(object, communicator->stringToIdentity("M")); // 这个ice对象的标识是“M”

  adapter\_->activate();

（3）当使用registry 时，object id 可以同时使用object:tcp -p 10001 和  object@adapter 两种形式

(4)replica-group意思？

(5)ice box代替了原来的server,包含程序的真正入口。它可以包含多个service,即多个真正的服务。

貌似有了ice box后，server就是ice box ，详细来说，就是 ice box id。

但是server 又可以包含 ice box

(6)对象适配器的意思

顾名思义，还是对象，不过改变了形态，

（7）icebox   service entry 后边的参数的规定是--name=value这种形式的视为communicator的属性，其余才视为start的参数。可是事实上input服务中是可以通过args读取到这种类型的参数的

规定：Any arguments following the entry point specification are examined. If an

argument has the form --name=value, then it is interpreted as a property definition

that appears in the property set of the communicator passed to the service

start operation. These arguments are removed, and any remaining arguments are

passed to the start operation in the args parameter.

start函数：

void start(string name,

Ice::Communicator communicator,

Ice::StringSeq args)

---------------------

作者：crazyhacking

来源：CSDN

原文：https://blog.csdn.net/crazyhacking/article/details/39638245

版权声明：本文为博主原创文章，转载请附上博文链接！