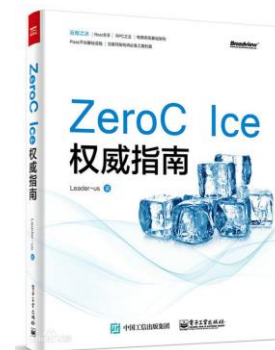




# 分布式高端架构系列

# ZeroC ICE

## ——微服务架构之王

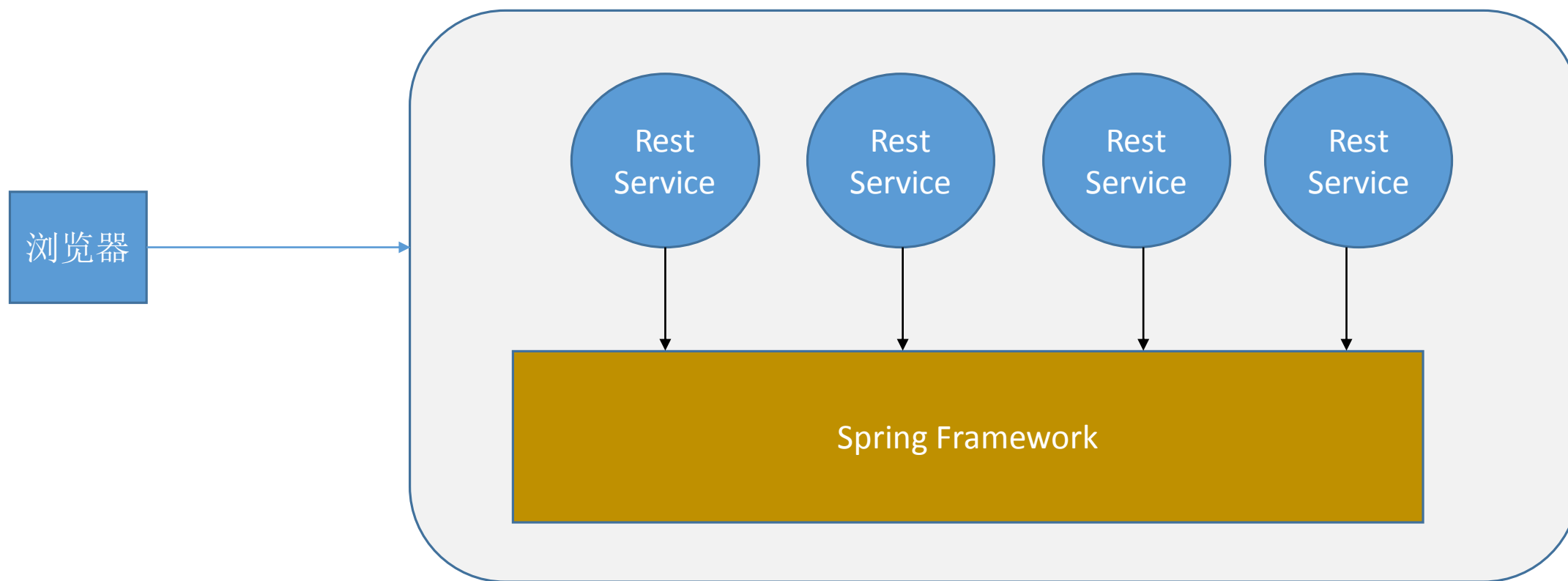


Leader-us@Mycat

# 一：微服务架构概述

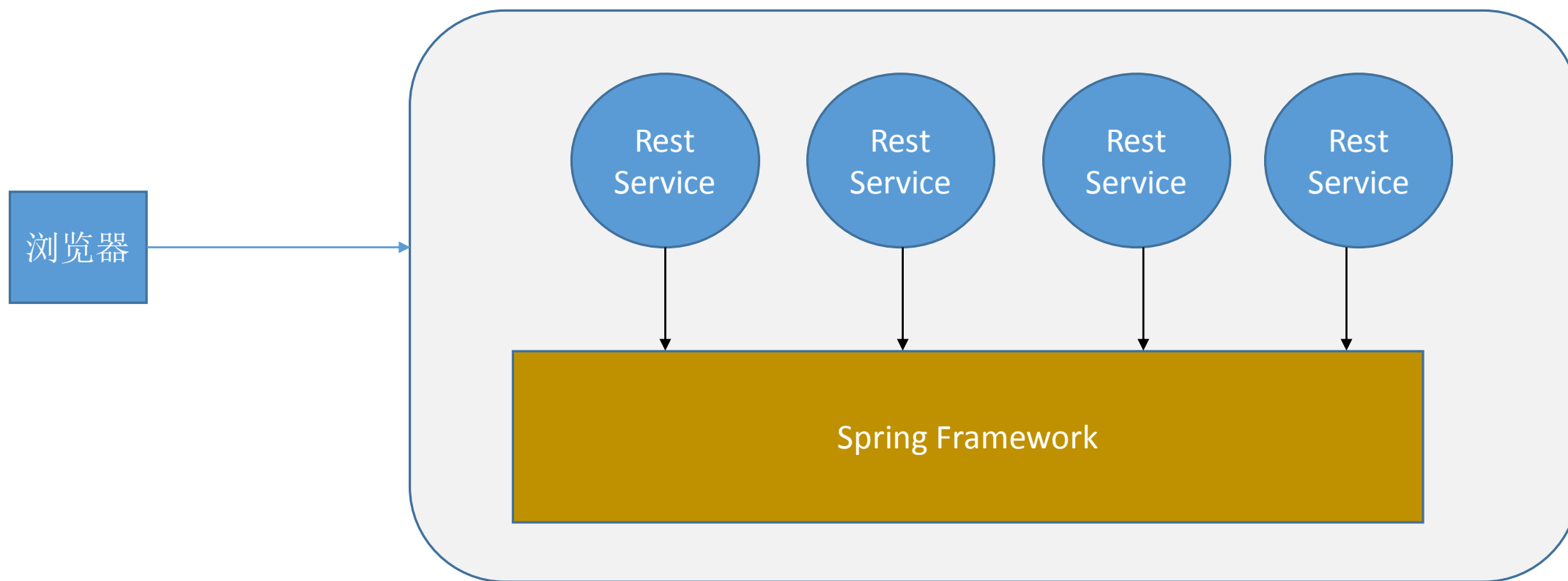
# 1：什么是单体应用

主要业务逻辑都运行在**一个进程**里的程序



# 1：什么是单体应用

主要业务逻辑都运行在**一个进程**里的程序



## 2:单体应用的七宗罪



**No1:先天性缺陷**

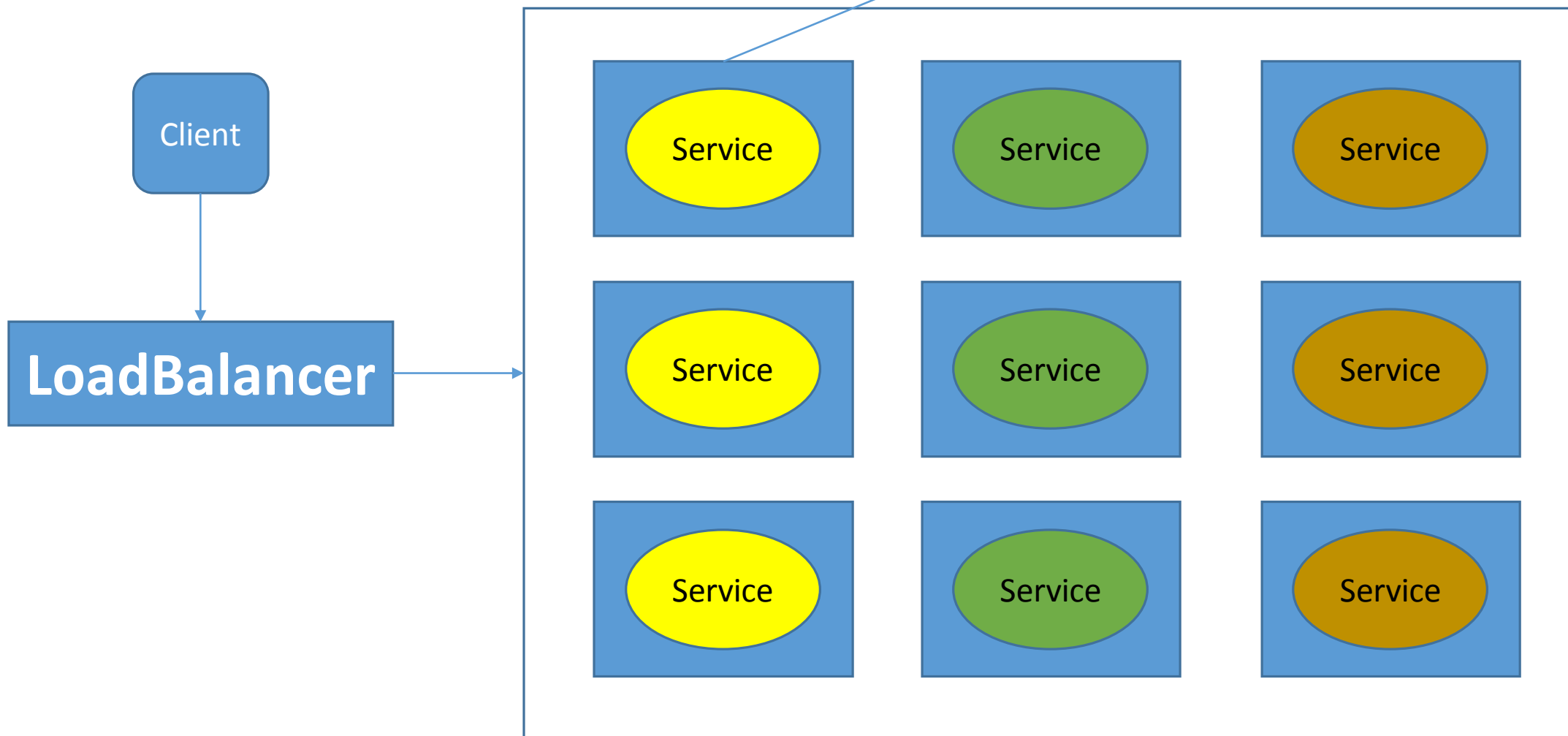
**No2:系统性风险**

**No3:运维风险**

**No4:难以可持续发展**

# 3:微服务架构

每个微服务实例一个进程



# 4 : Ice微服务架构

## grid.xml

```
<icegrid>
  <application name="MyAppGrid">
    <server-template id="xxxServerTemplate">
      <parameter name="id" />
      <icebox id="TicketOrderServer${id}" >
        <service name="xxxService" />
      </icebox>
    </server-template>
    <node name="node1">
      <server-instance template="xxxServerTemplate" id="1" />
      <server-instance template="xxxServerTemplate" id="2" />
    </node>
    <node name="node2">
    </node>
  </application>
</icegrid>
```

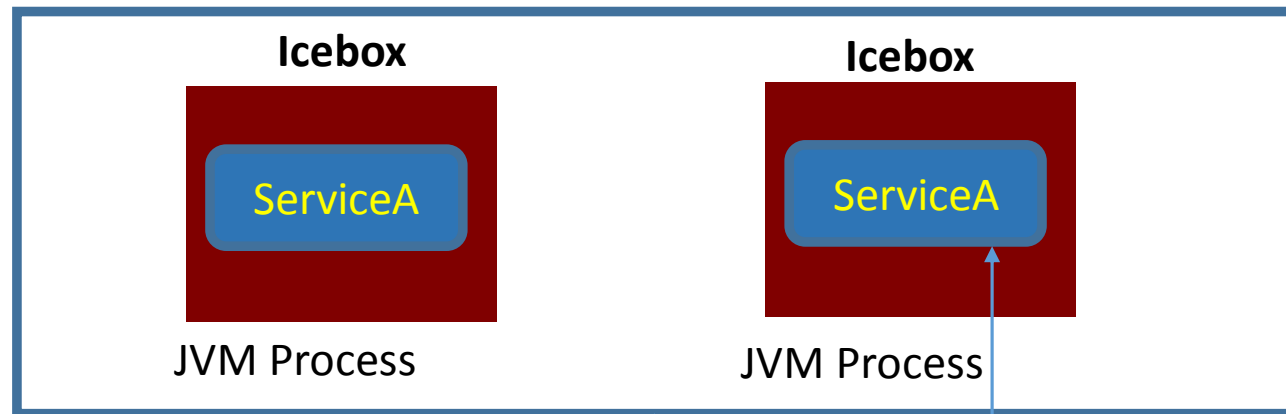
微服务描述及部署文件

## icegridadmin

- >>发布grid
- >>升级grid
- >>停止/重启服务
- >>服务状态查询
- >>.....

运维工具  
命令行&Applet

## Ice Node



JVM Process

JVM Process

服务注册和管理

LocatorService

Master Registry

服务查询

Client

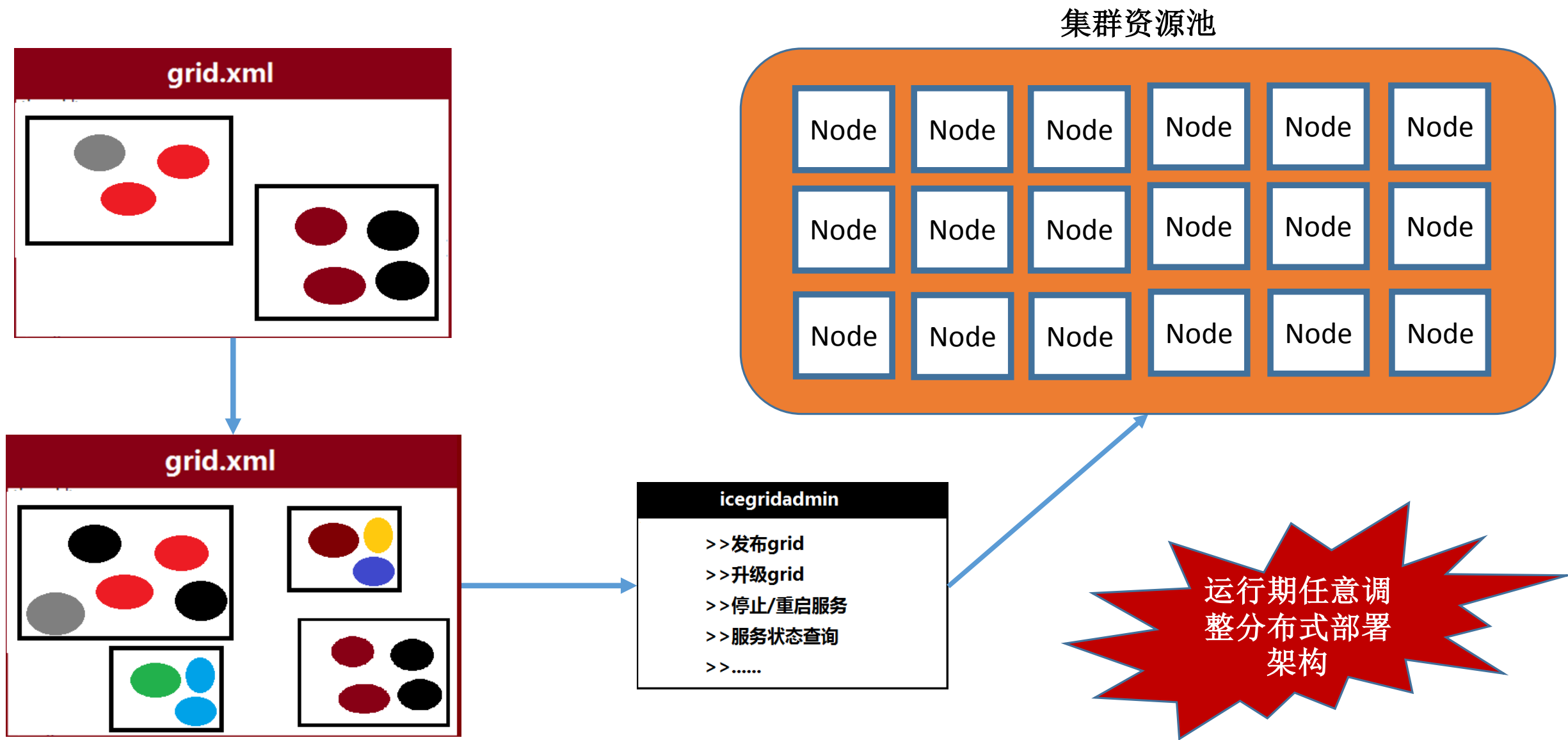
LocatorService

Slave Registry

服务调用

客户端负载均衡机制

# 4 : Ice微服务架构特点





# 5：微服务架构的优点

**No1:先天分布式**

**No2:无状态（尽量）**

**No3:积木式发展**

# 二：Zeroc Ice安装

# 1：安装ICE SDK环境



官方：<https://zeroc.com/>，区分平台，安装包比较大, 是因为各个版本的库文件都包括了


Assemblies	11/26/2015 01:2...	文件夹	
bin	11/26/2015 01:2...	文件夹	
config	11/26/2015 01:2...	文件夹	
include	11/26/2015 01:2...	文件夹	
lib	11/26/2015 01:2...	文件夹	
php	11/26/2015 01:2...	文件夹	
slice	11/26/2015 01:2...	文件夹	
ICE_LICENSE.txt	9/9/2015 07:01 ...	TXT 文件	3 KB
LICENSE.txt	9/9/2015 07:01 ...	TXT 文件	18 KB
THIRD_PARTY_LICENSE.txt	9/9/2015 07:01 ...	TXT 文件	18 KB

命令行工具，如slice2java  
Icegridnode, icegridadmin

Ice.jar,icebox.jar等文件

# 2 : 安装Ice demos

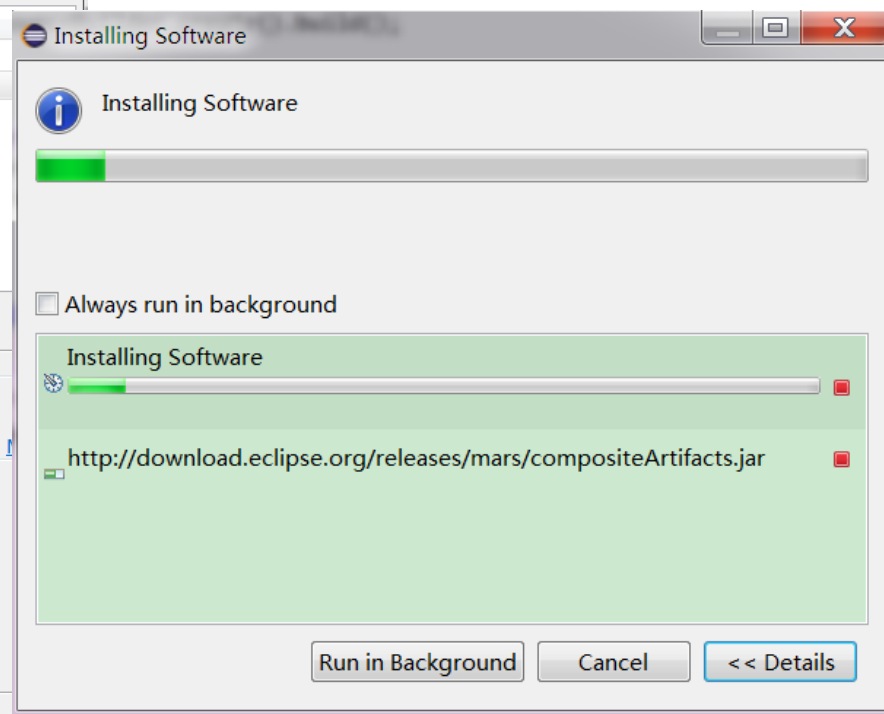
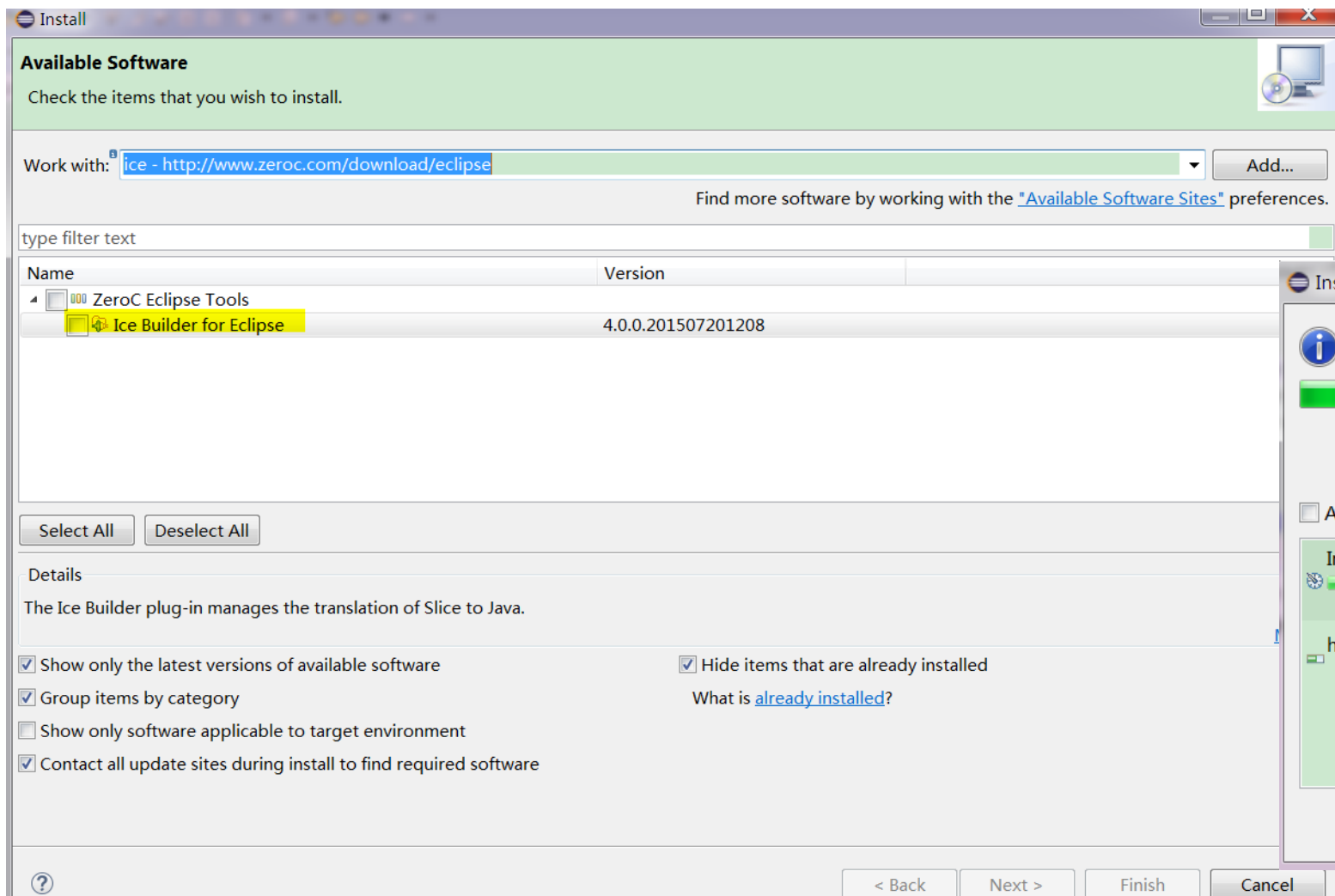
Ice-3.6.1-demos部分包括各个语言的例子，建议安装的时候选择安装

certs	11/26/2015 01:2...	文件夹	
cpp	11/26/2015 01:2...	文件夹	
csharp	11/26/2015 01:2...	文件夹	
java	11/26/2015 01:2...	文件夹	
js	11/26/2015 01:2...	文件夹	
objective-c	11/26/2015 01:2...	文件夹	
php	11/26/2015 01:2...	文件夹	
python	11/26/2015 01:2...	文件夹	
ruby	11/26/2015 01:2...	文件夹	
scripts	11/26/2015 01:2...	文件夹	
visualBasic	11/26/2015 01:2...	文件夹	
CONTRIBUTING.md	9/11/2015 02:43...	MD 文件	
LICENSE	9/11/2015 02:43...	文件	
README.md	9/11/2015 02:43...	MD 文件	
Android	11/26/2015 01:2...	文件夹	
Chat	11/26/2015 01:2...	文件夹	
Database	11/26/2015 01:2...	文件夹	
Freeze	11/26/2015 01:2...	文件夹	
Glacier2	11/26/2015 01:2...	文件夹	
gradle	11/26/2015 01:2...	文件夹	
Ice	11/26/2015 01:2...	文件夹	
IceBox	11/26/2015 01:2...	文件夹	
IceDiscovery	11/26/2015 01:2...	文件夹	
IceGrid	11/26/2015 01:2...	文件夹	
IceStorm	11/26/2015 01:2...	文件夹	
Manual	11/26/2015 01:2...	文件夹	

# 3 : 安装Eclipse Ice插件

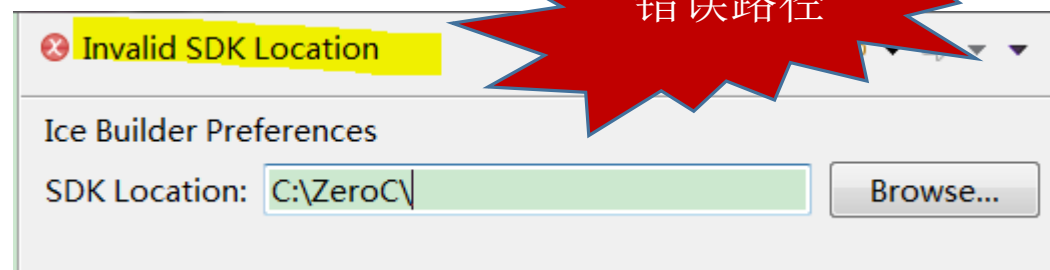
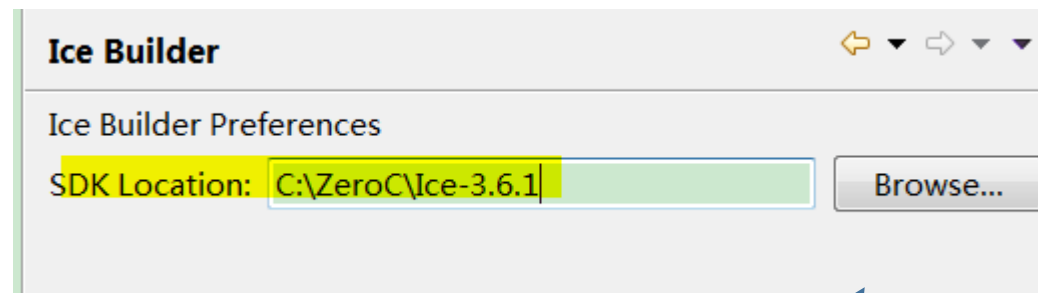
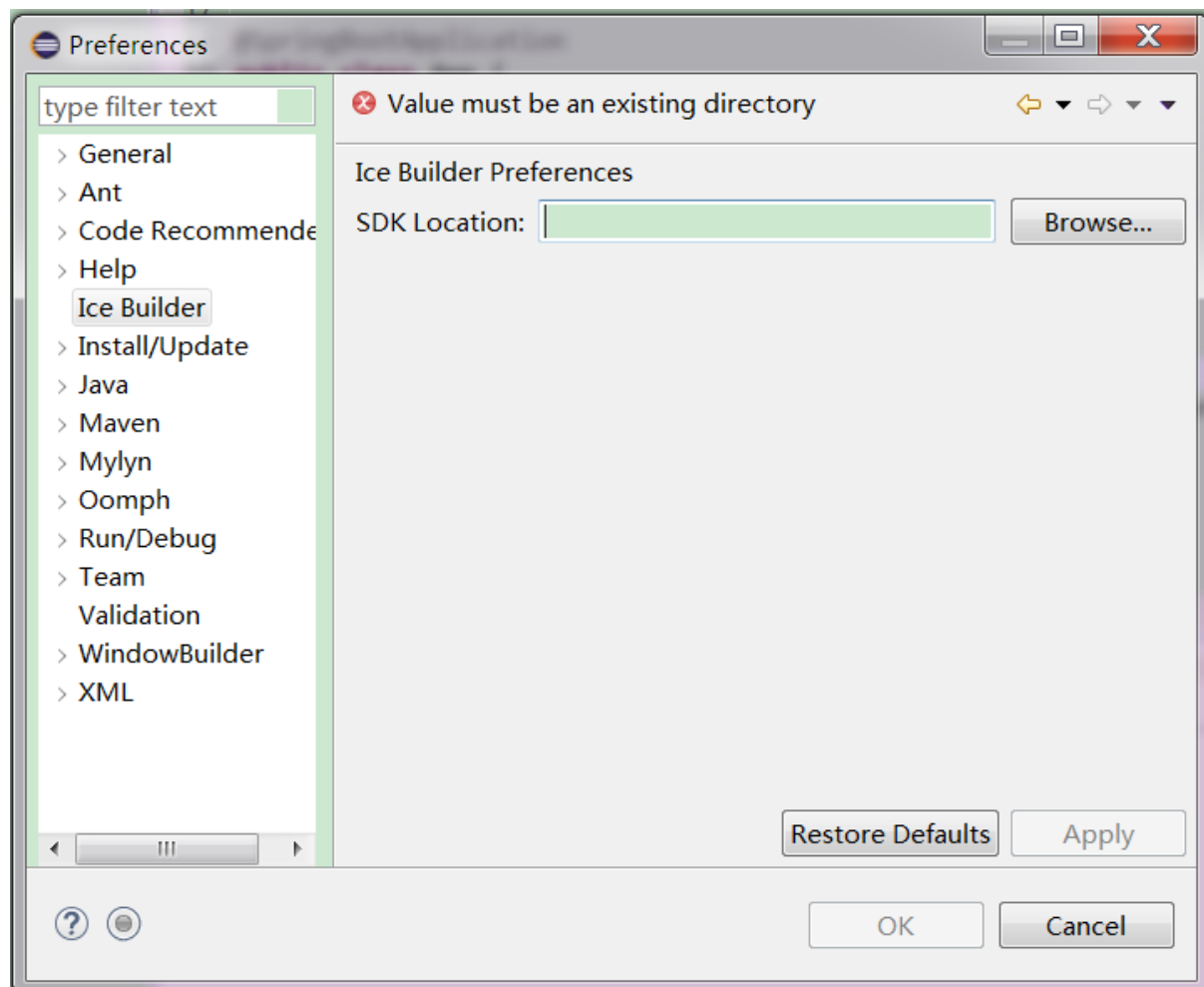
插件用于自动完成slice接口定义到java 接口包的源码生成

插件地址<http://www.zeroc.com/download/eclipse>



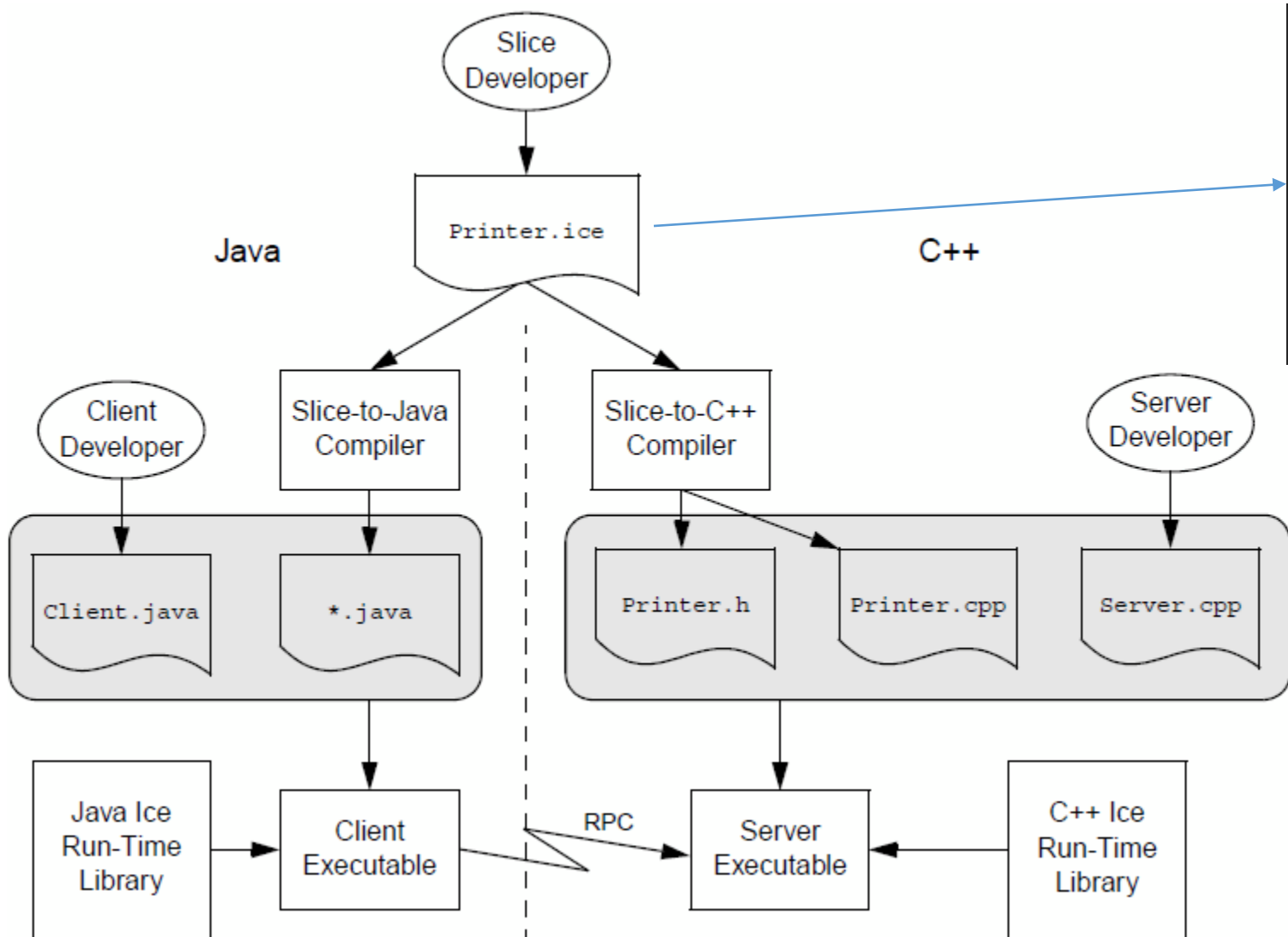
# 4 : Ice插件设置

Ice Builder插件需要设置Ice SDK的路径，eclipse->preferences菜单里进行设置



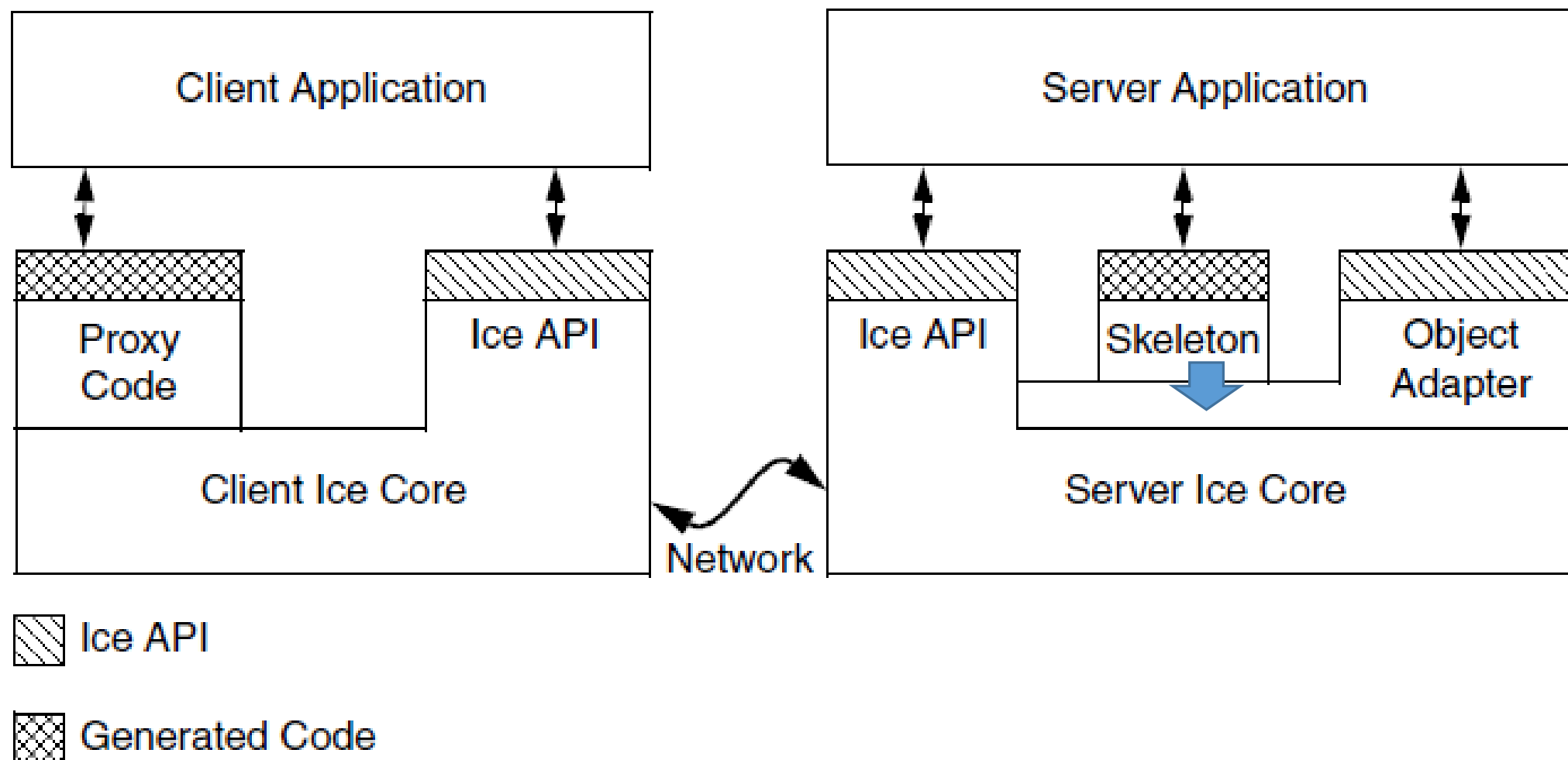
# 三：ZeroC Ice入门

# 1 : 服务定义&Slice





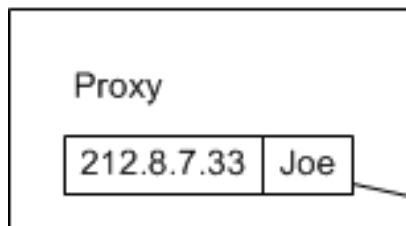
# 2 : I C E 通信原理



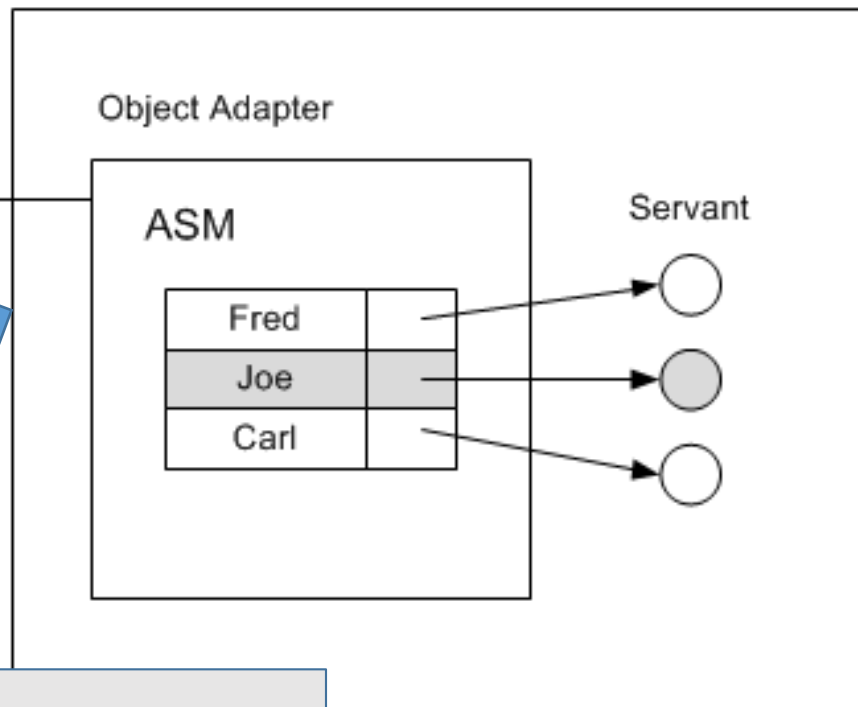
# 3 : 基本概念

An Ice object is an abstraction. Ice objects do not physically exist. The concept of an Ice object is made real by a servant

Client



Server



Endpoint

Joe

212.8.7.33

// 传入远程服务单元的名称、网络协议、IP以及端口，构造一个Proxy对象

```
Ice.ObjectPrx base = ic.stringToProxy("MyService:default -p 20000");
```

// 通过checkedCast向下转型，获取MyService接口的远程

```
MyServicePrx prxy = MyServicePrxHelper.uncheckedCast(base);
```

// 调用服务方法

```
String rt=prxy.hellow();
```

```
Ice.ObjectAdapter adapter = ic.createObjectAdapterWithEndpoints(  
    "MyServiceAdapter", "default -p 20000");
```

// 实例化一个MyService服务对象(Servant)

```
MyServiceImpl servant = new MyServiceImpl();
```

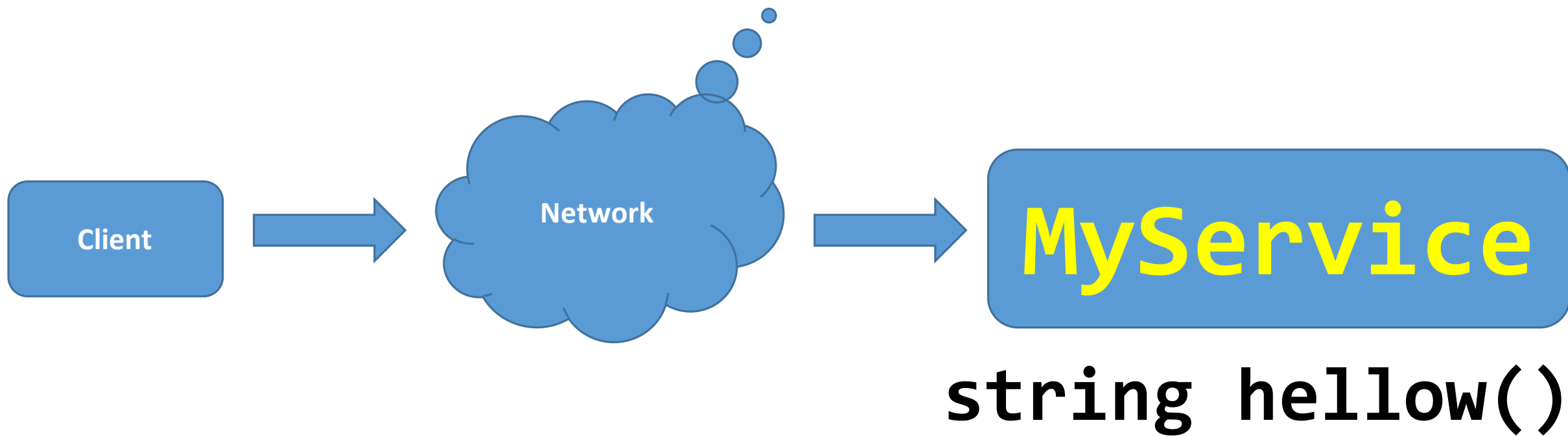
// 将Servant增加到ObjectAdapter中，并将Servant关联到ID为MyService的Ice Object

```
adapter.add(servant, Ice.Util.stringToIdentity("MyService"));
```

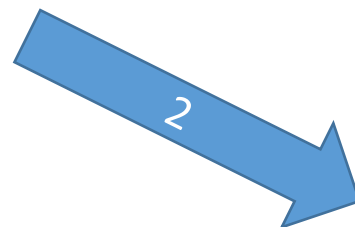
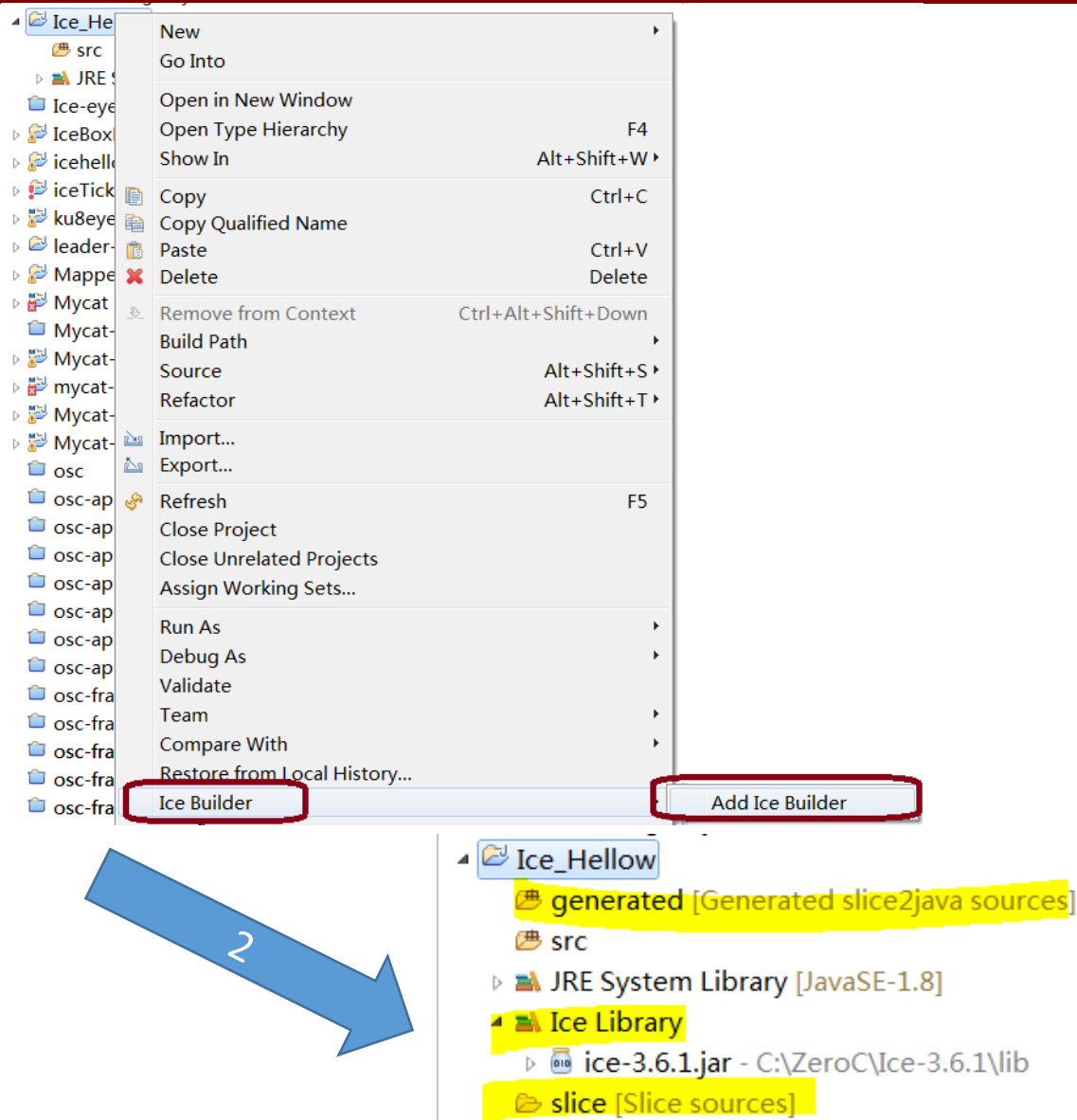
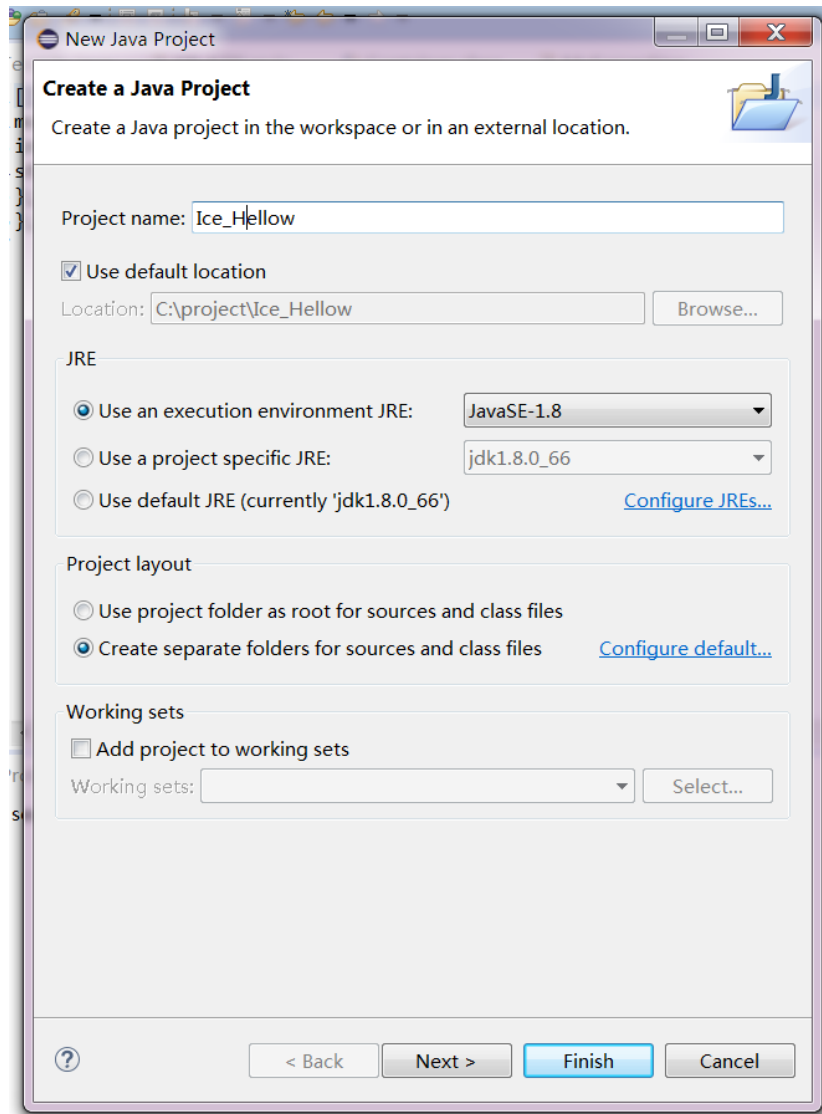
// 激活ObjectAdapter

```
adapter.activate();
```

# 4 : Hello World(1)



# 4 : Hello World(2)



# 4 : Hello World(3)

## MyService服务接口定义

myservice.ice

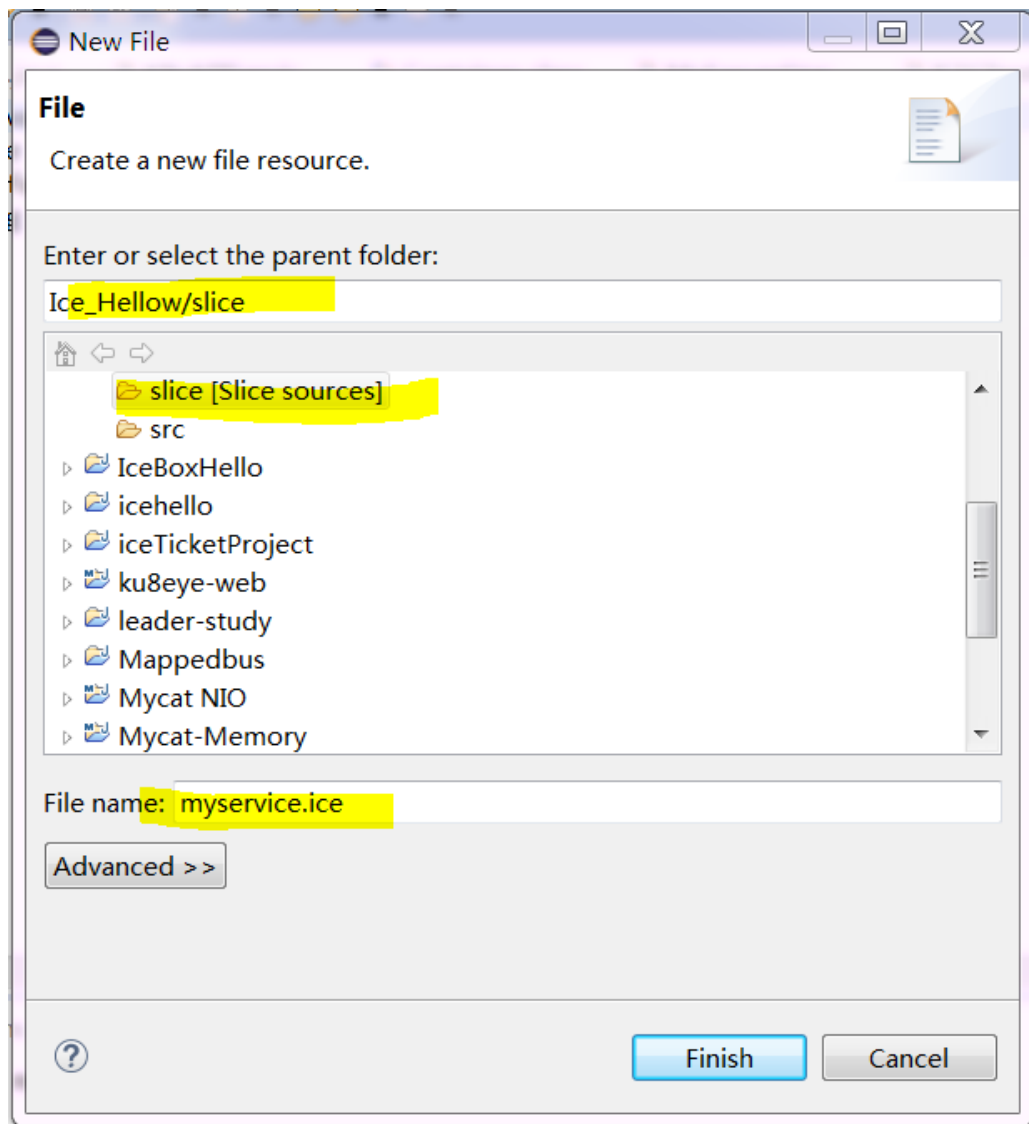
```
[[ "java:package:com.my" ]]  
module demo{  
  interface MyService{  
    string hellow();  
  };  
};
```

映射为Java包名

模块名

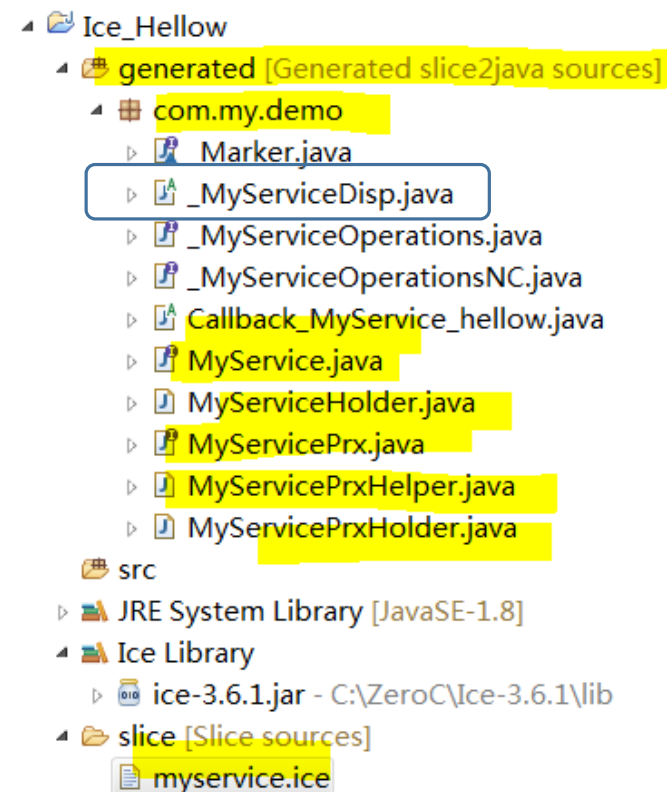
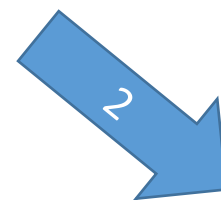
服务接口

# 4 : Hello World(4)



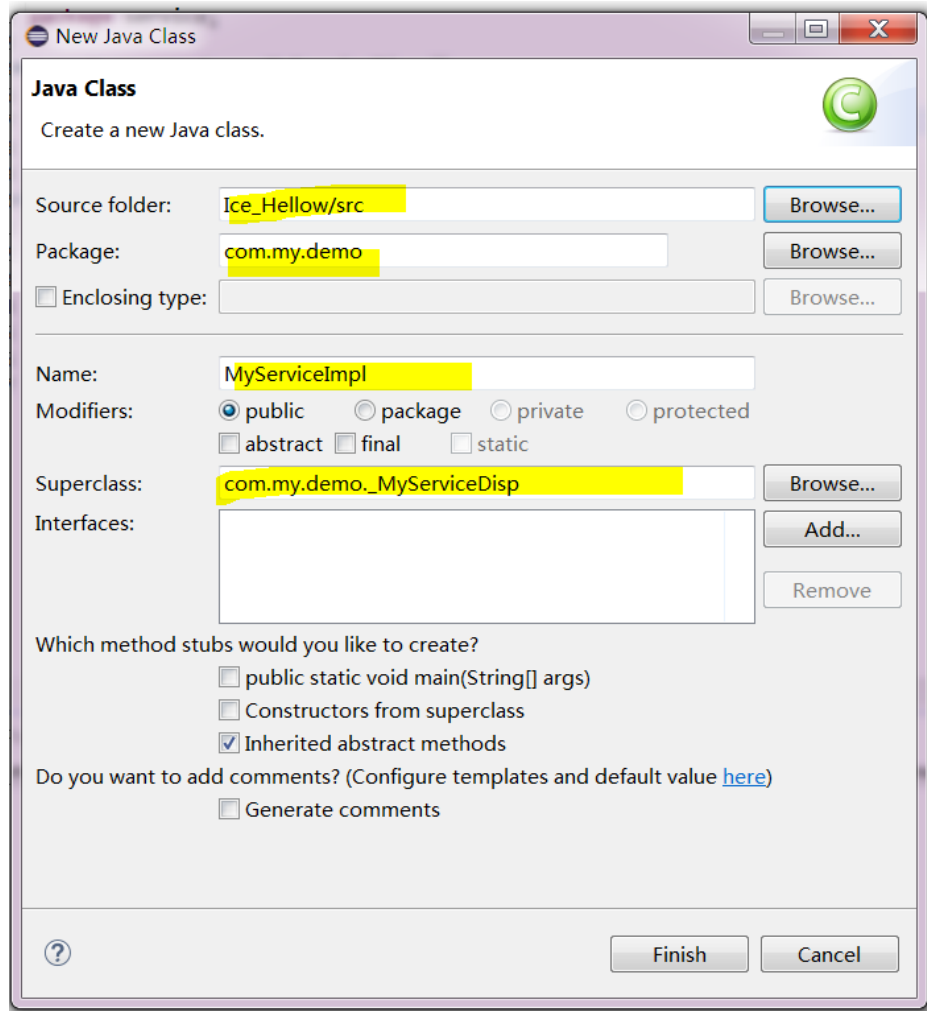
```
*myservice.ice
```

```
1 ["java:package:com.my"]
2 module demo{
3 interface MyService{
4 string hellow();
5 };
6 };
7
```



# 4 : Hello World(5)

## MyService服务端开发



1

```
package com.my.demo;  
import Ice.Current;  
public class MyServiceImpl extends  
MyServiceDisp {
```

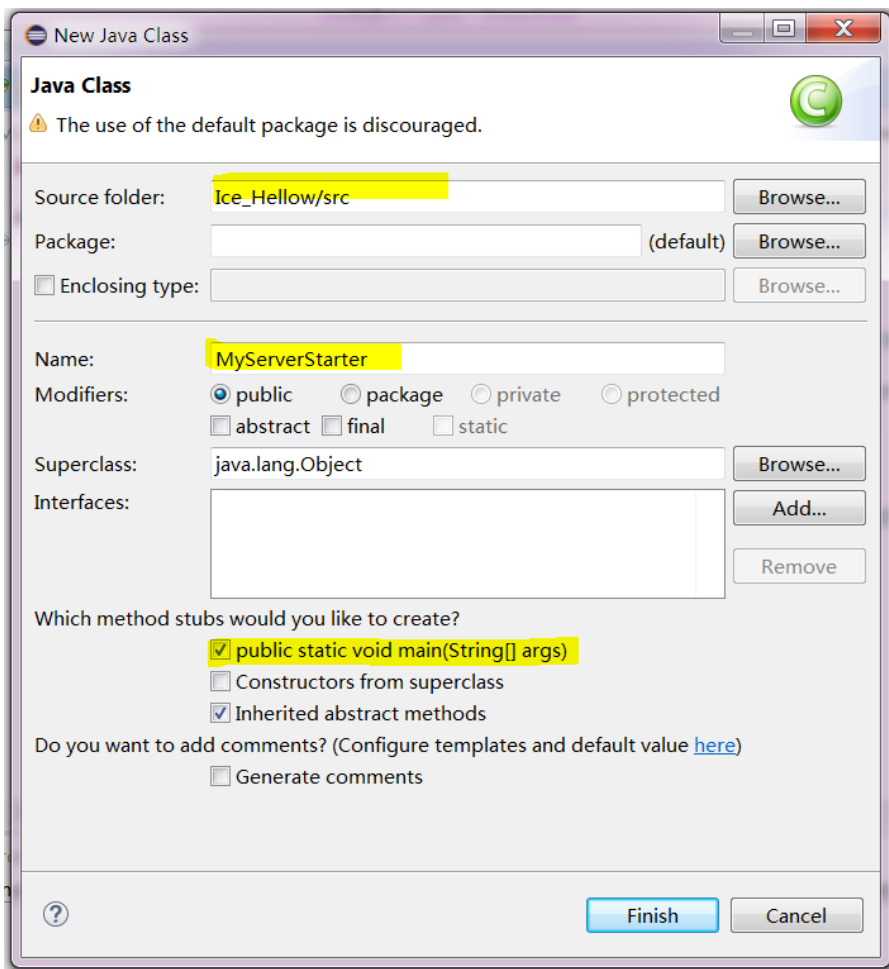
```
@Override  
public String hellow(Current __current) {  
    // TODO Auto-generated method stub  
    return null;  
}  
}
```

2

```
@Override  
public String hellow(Current __current) {  
    return "Hello world";  
}
```

# 4 : Hello World(6)

## Server Starter程序

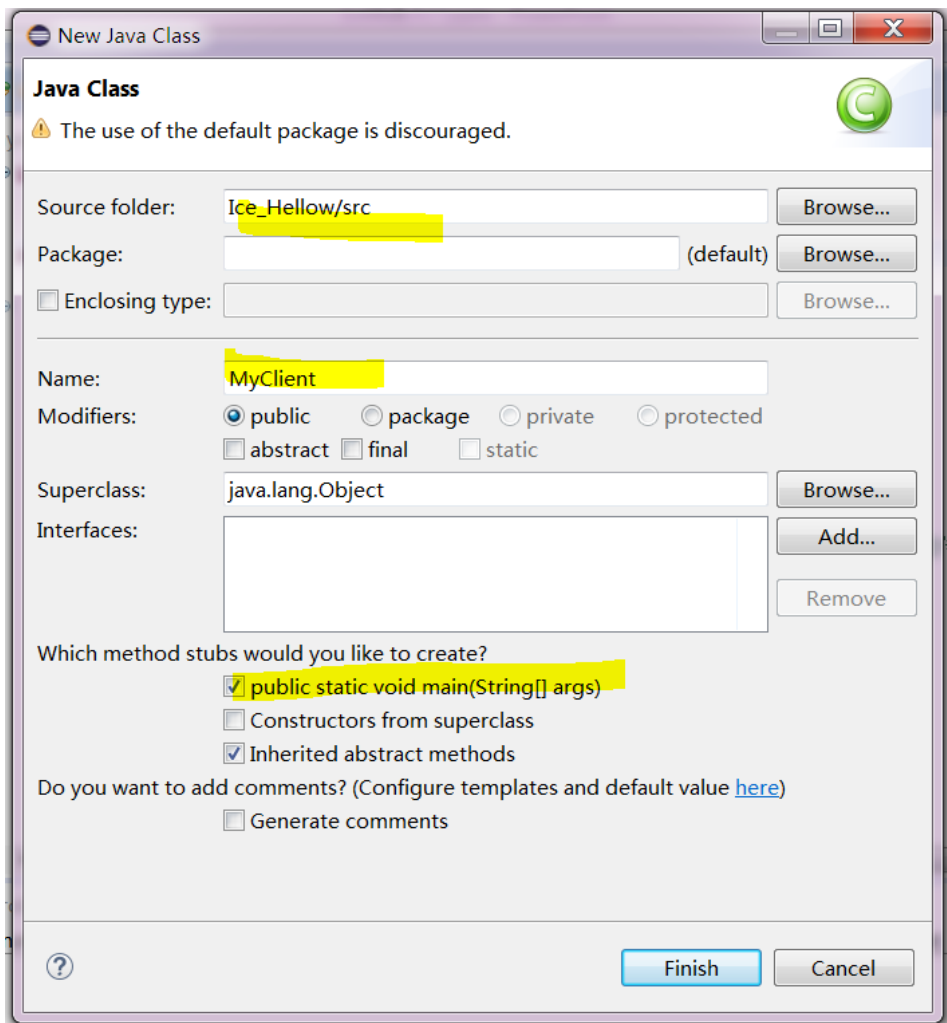


```
public static void main(String[] args) {  
    int status = 0;  
    Ice.Communicator ic = null;  
    try {  
        // 初始化Communicator对象, args可以传一些初始化参数, 如连接超时, 初始化客户端连接池的数量等  
        ic = Ice.Util.initialize(args);  
        // 创建名为MyServiceAdapter的ObjectAdapter, 使用缺省的通信协议 (TCP/IP 端口为20000的请求)  
        Ice.ObjectAdapter adapter = ic.createObjectAdapterWithEndpoints(  
            "MyServiceAdapter", "default -p 20000");  
        // 实例化一个MyService服务对象 (Servant)  
        MyServiceImpl servant = new MyServiceImpl();  
        // 将Servant增加到ObjectAdapter中, 并将Servant关联到ID为MyService的Ice Object  
        adapter.add(servant, Ice.Util.stringToIdentity("MyService"));  
        // 激活ObjectAdapter  
        adapter.activate();  
        // 让服务在退出之前, 一直持续对请求的监听  
        System.out.print("server started ");  
        ic.waitForShutdown();  
    } catch (Exception e) {  
        e.printStackTrace();  
        status = 1;  
    } finally {  
        if (ic != null) {  
            ic.destroy();  
        }  
    }  
    System.exit(status);  
}
```



# 4 : Hello World(7)

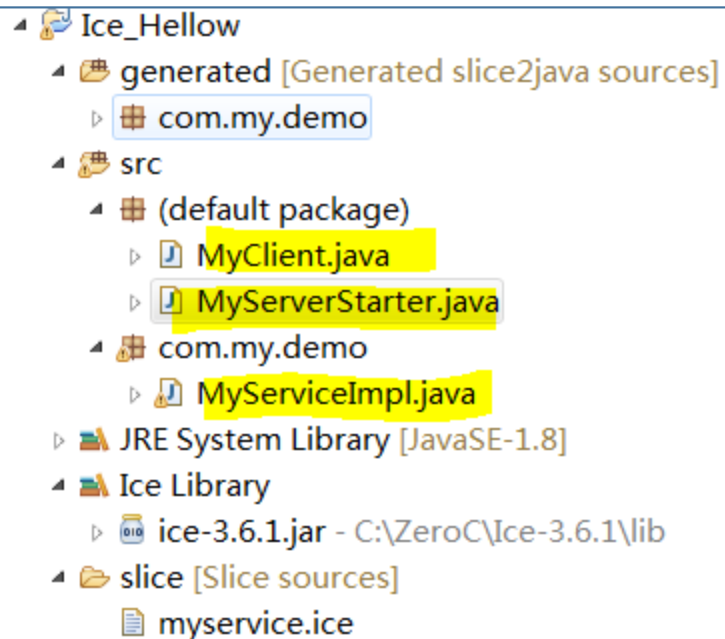
## Client程序



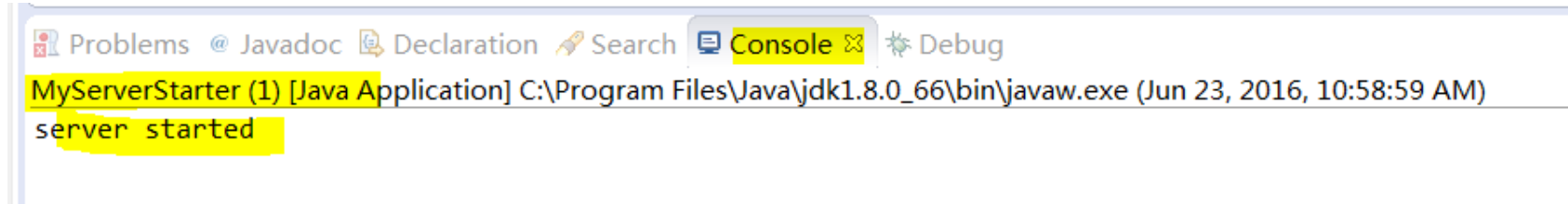
```
public static void main(String[] args) {
    int status = 0;
    Ice.Communicator ic = null;
    try {
        // 初始化通信器
        ic = Ice.Util.initialize(args);
        // 传入远程服务单元的名称、网络协议、IP以及端口，构造一个Proxy对象
        Ice.ObjectPrx base = ic
            .stringToProxy("MyService:default -p 20000");
        // 通过checkedCast向下转型，获取MyService接口的远程，并同时检测根据传入的名称获取服务单元是否OnlineBook的代理接口，如果不是则返回null对象
        MyServicePrx prxy = MyServicePrxHelper.uncheckedCast(base);
        if (prxy == null) {
            throw new Error("Invalid proxy");
        }
        // 调用服务方法
        String rt=prxy.hellow();
        System.out.print(rt);
    } catch (Exception e) {
        e.printStackTrace();
    }
    status = 1;
} finally {
    if (ic != null) {
        ic.destroy();
    }
}
System.exit(status);
}
```

# 4 : Hello World(8)

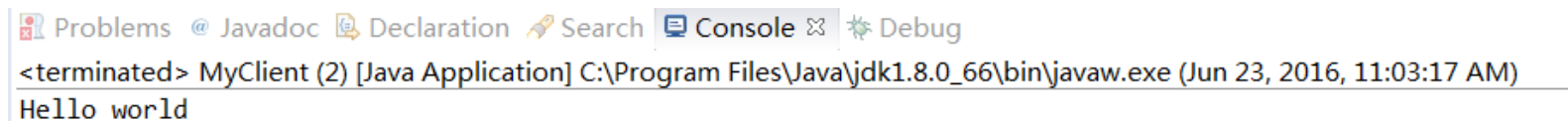
## 完工测试



### 一：启动Server



### 二：启动Client



# 5 : R P C 性能之王



Rpc	并发客户端	每客户端调用次数	总调用次数	执行时间	每秒调用数tps
ice	1	300000	300000	16s	18329
dubbo	1	300000	300000	52s	5675
thrift	1	300000	300000	23s	12832
grpc	1	300000	300000	77s	3896

Rpc	并发客户端	每客户端调用次数	总调用次数	执行时间	每秒调用数tps
ice	100	300000	30000000	361s	83014
dubbo	100	300000	30000000	1599s	18760
thrift	100	300000	30000000	597s	50211
grpc	100	300000	30000000	2186s	13721

从数据可以看出ice, thrift的tps最高, ice是thrift的1.6倍, 是dubbo的4.4倍, 是grpc的6倍, 来自南哥测试报告,<http://i.mycat.io>

# 6 : ICE RPC总结



**很简洁，只依赖一个包**

**高性能，很稳定**

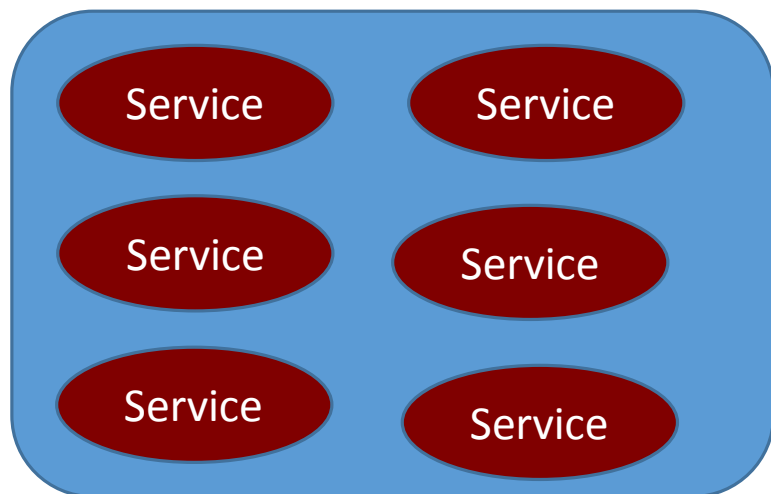
**多语言支持**

# 二：ICE微服务架构实践

# 1 : IceBox(1)

## IceBox

```
java IceBox.Server --Ice.Config=config.icebox
```



Apache Tomcat

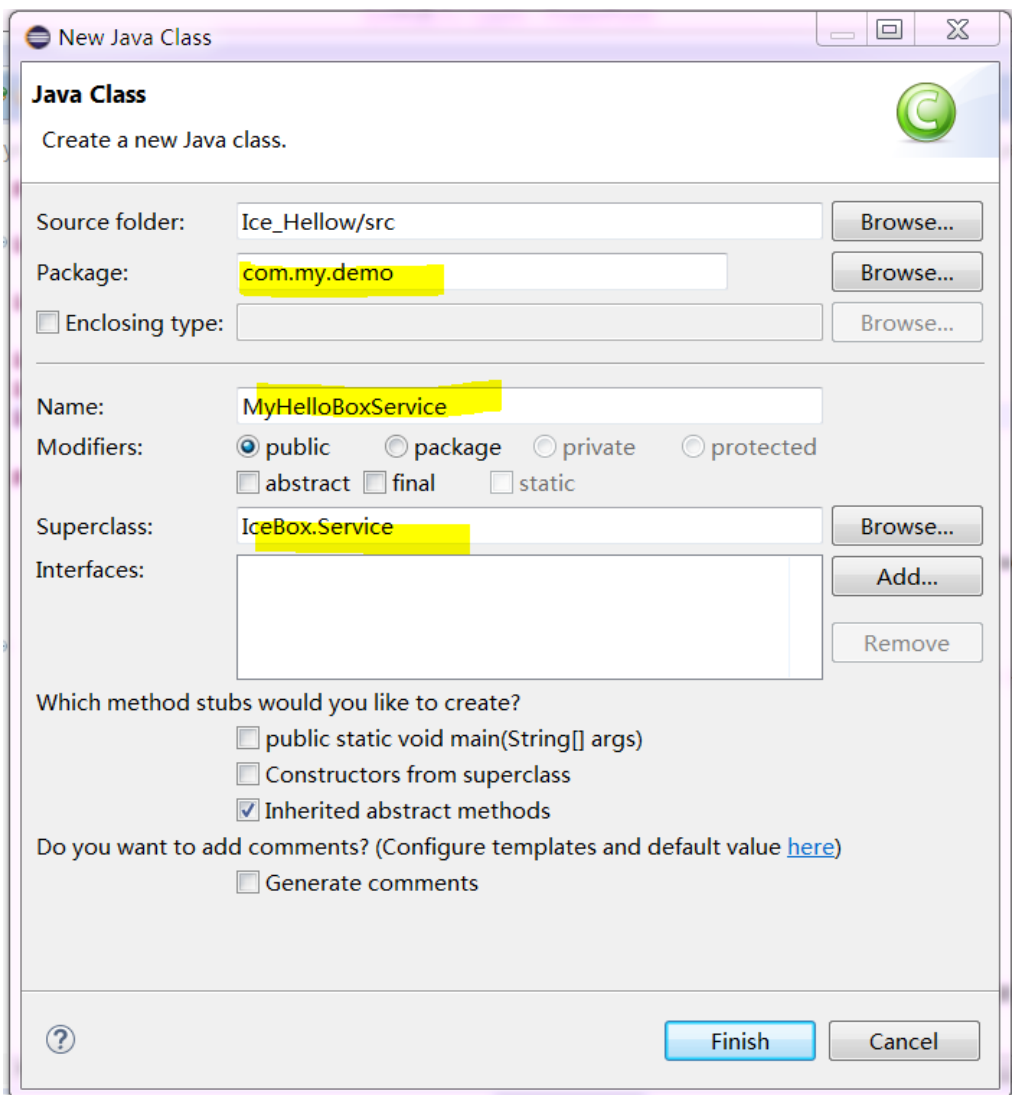
```
public abstract interface IceBox.Service {  
    public abstract void start(java.lang.String arg0, Ice.Communicator arg1, String[] arg2);  
    public abstract void stop();  
}
```



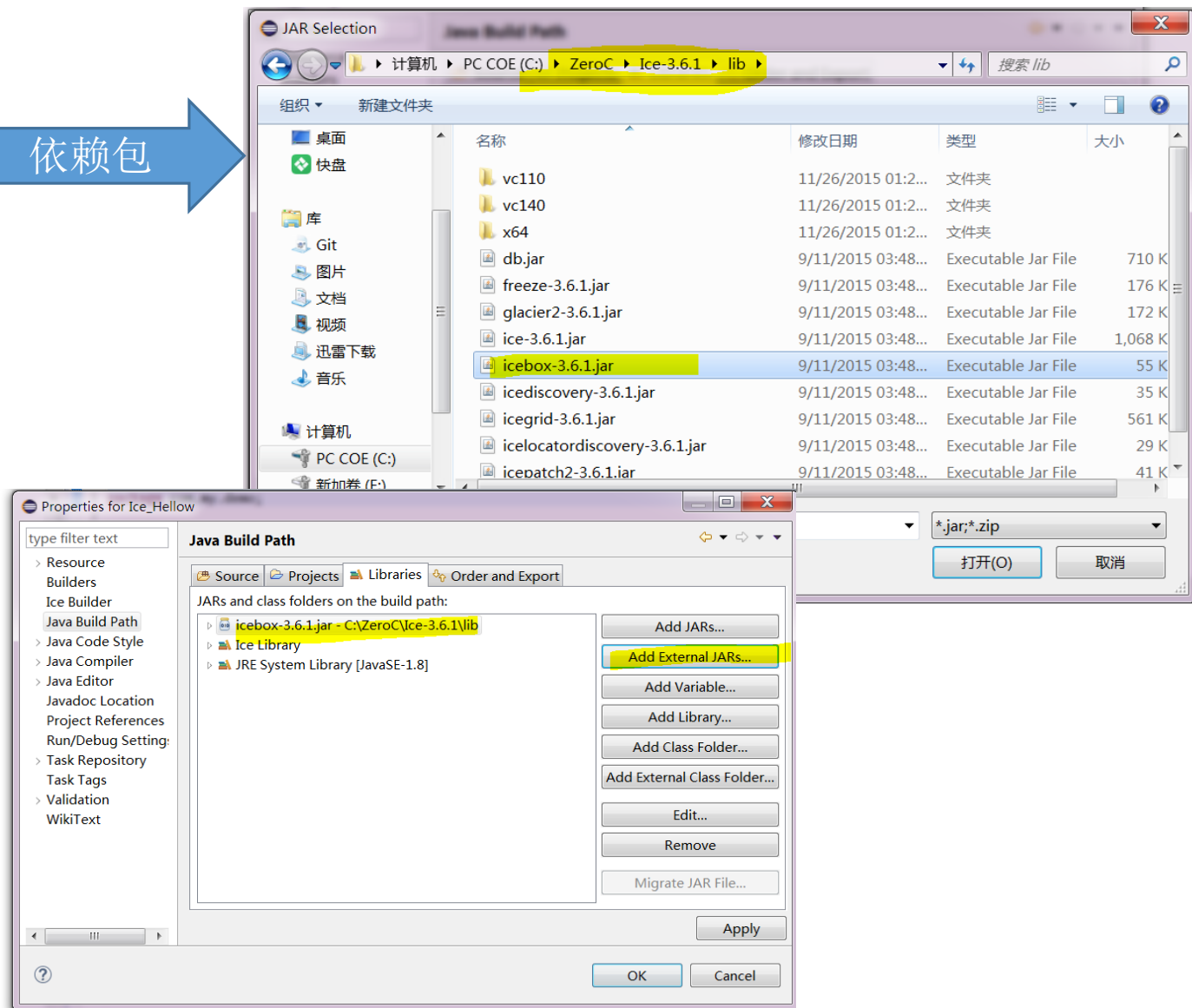
**Servlet**

# 1 : IceBox(2)

## 开发运行在IceBox中的Service



依赖包



# 1 : IceBox(3)



## 开发运行在IceBox中的Service

```
package com.my.demo;
import Ice.Communicator;
import IceBox.Service;
public class MyHelloBoxService implements Service {
    @Override
    public void start(String arg0, Communicator arg1, String[] arg2) {
        // TODO Auto-generated method stub

    }
    @Override
    public void stop() {
        // TODO Auto-generated method stub
    }
}
```

```
@Override
public void start(String name, Communicator communicator, String[] arg2) {
    // IceBox
    // 创建objectAdapter, 这里和service同名
    _adapter = communicator.createObjectAdapter(name);
    // 创建servant
    Ice.Object object = new MyServiceImpl();
    id = communicator.stringToIdentity(name);
    // _adapter.add(object, communicator.stringToIdentity(name));
    _adapter.add(object, id);
    // 激活
    _adapter.activate();
    System.out.println("start service success :" + name);
}

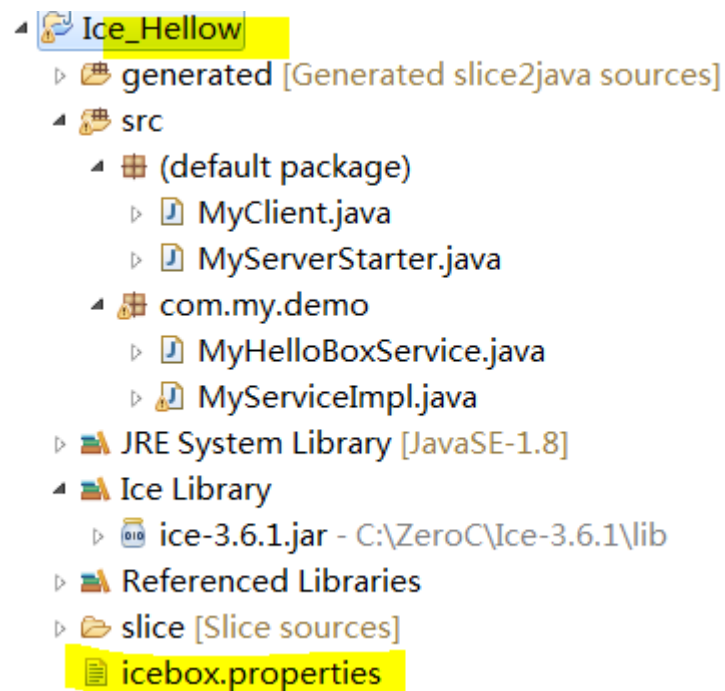
@Override
public void stop() {
    System.out.println("stop service ...");
    _adapter.destroy();
    System.out.println("stop service successs" + id.toString());
}
```



# 1 : IceBox(4)



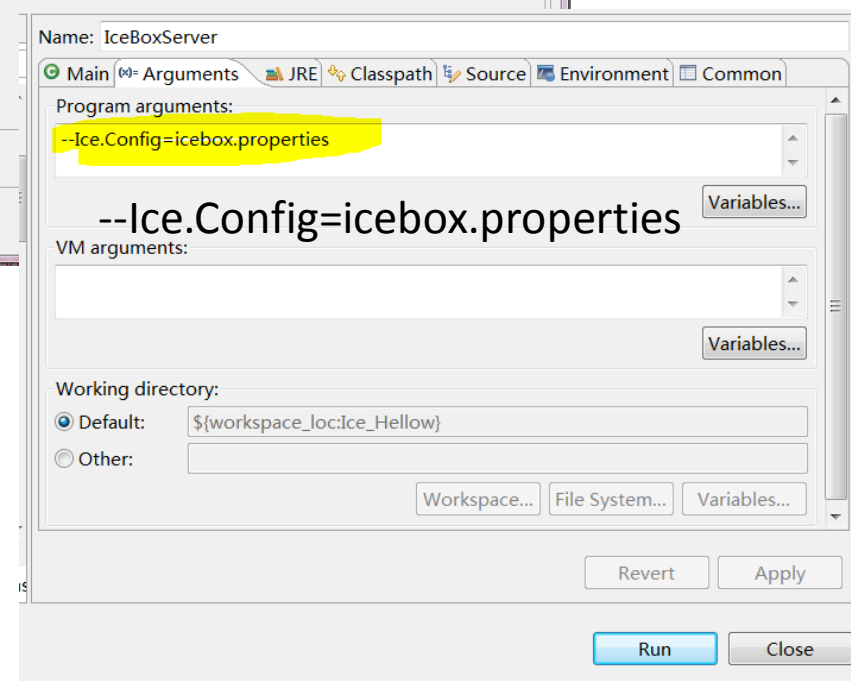
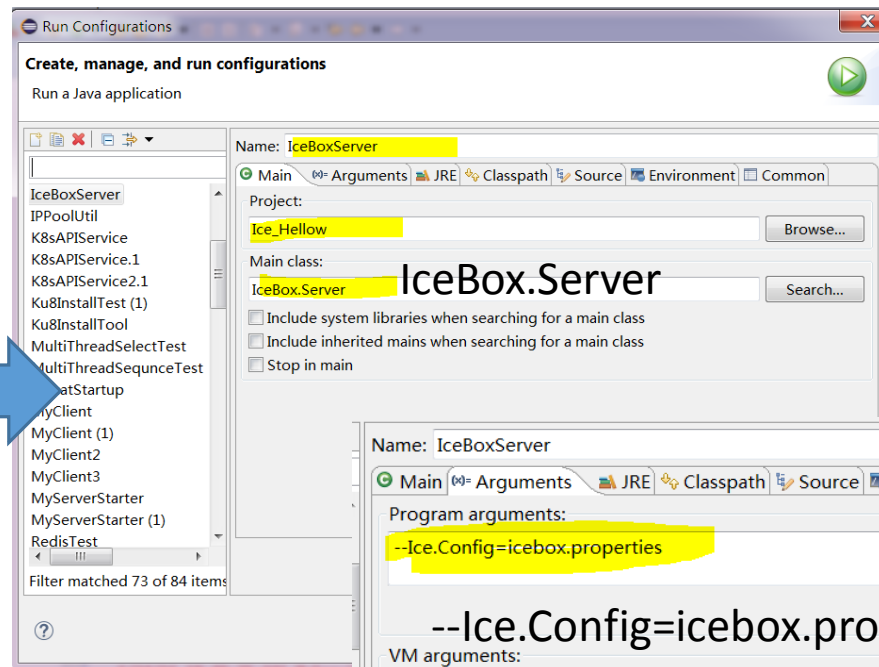
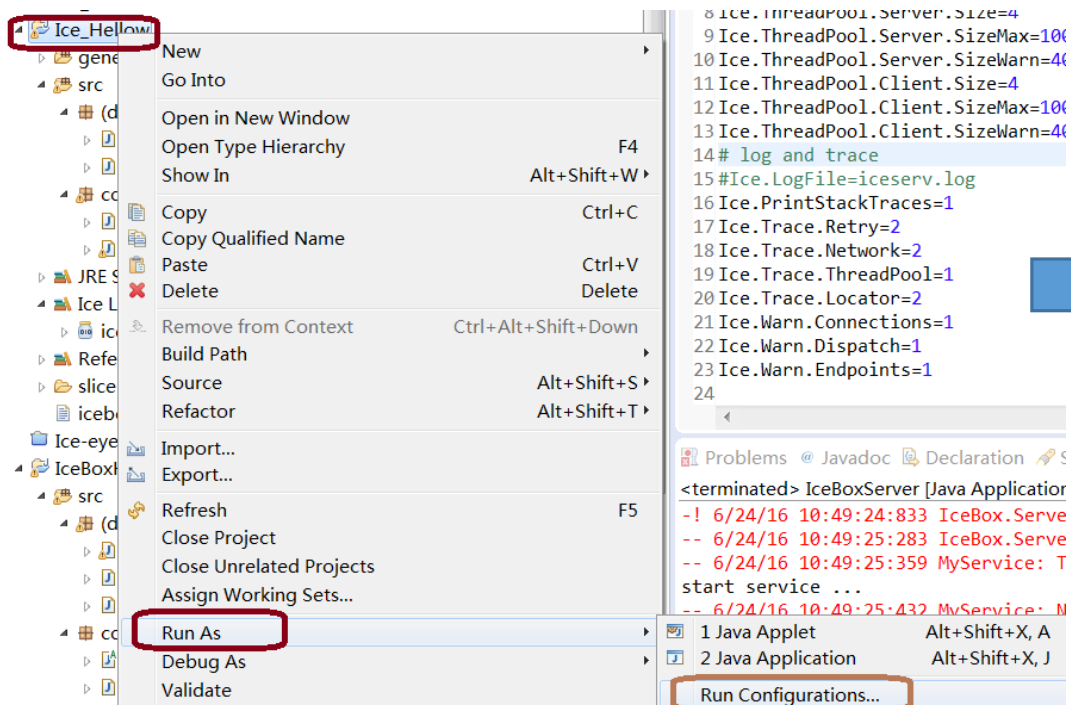
## 编写IceBox的配置文件



```
#server properties
IceBox.InheritProperties=1
IceBox.PrintServicesReady= MyAppIceBox 1
#service define begin
IceBox.Service.MyService=com.my.demo.MyHelloBoxService prop1=1 prop2=2 prop3=3
MyService.Endpoints=tcp -p 20000 -h localhost
#performance properties
Ice.ThreadPool.Server.Size=4
Ice.ThreadPool.Server.SizeMax=100
Ice.ThreadPool.Server.SizeWarn=40
Ice.ThreadPool.Client.Size=4
Ice.ThreadPool.Client.SizeMax=100
Ice.ThreadPool.Client.SizeWarn=40
#for system stronger
Ice.ACM.Client=300
Ice.ACM.Server=300
# log and trace
#Ice.LogFile=iceserv.log
Ice.PrintStackTraces=1
Ice.Trace.Retry=2
Ice.Trace.Network=2
Ice.Trace.ThreadPool=1
Ice.Trace.Locator=2
Ice.Warn.Connections=1
Ice.Warn.Dispatch=1
Ice.Warn.Endpoints=1
```

# 1 : IceBox(5)

## 启动IceBox Server



# 1 : IceBox(6)



## 启动IceBox Server

```
-- 6/24/16 10:57:26:871 IceBox.Server: ThreadPool: creating Ice.ThreadPool.Client: Size = 4, SizeMax = 100, SizeWarn = 40
-- 6/24/16 10:57:26:950 MyService: ThreadPool: creating Ice.ThreadPool.Client: Size = 4, SizeMax = 100, SizeWarn = 40
start service ...
-- 6/24/16 10:57:27:021 MyService: Network: attempting to bind to tcp socket 127.0.0.1:20000
-- 6/24/16 10:57:27:027 MyService: Network: listening for tcp connections
    local address = 127.0.0.1:20000
-- 6/24/16 10:57:27:032 MyService: ThreadPool: creating Ice.ThreadPool.Server: Size = 4, SizeMax = 100, SizeWarn = 40
-- 6/24/16 10:57:27:035 MyService: Network: published endpoints for object adapter `MyService':
    tcp -h localhost -p 20000 -t 60000
-- 6/24/16 10:57:27:044 MyService: Network: accepting tcp connections at 127.0.0.1:20000
start service success
MyAppIceBox 1 ready
```

# 1 : IceBox(7)



## 客户端发起调用

```
-- 6/24/16 10:58:26:910 IceBox.Server: ThreadPool: shrinking Ice.ThreadPool.Client: Size=3
-- 6/24/16 10:58:26:912 IceBox.Server: ThreadPool: shrinking Ice.ThreadPool.Client: Size=2
-- 6/24/16 10:58:26:914 IceBox.Server: ThreadPool: shrinking Ice.ThreadPool.Client: Size=1
-- 6/24/16 10:58:26:956 MyService: ThreadPool: shrinking Ice.ThreadPool.Client: Size=3
-- 6/24/16 10:58:26:957 MyService: ThreadPool: shrinking Ice.ThreadPool.Client: Size=2
-- 6/24/16 10:59:27:045 MyService: ThreadPool: shrinking Ice.ThreadPool.Server: Size=3
-- 6/24/16 10:59:27:048 MyService: ThreadPool: shrinking Ice.ThreadPool.Server: Size=2
-- 6/24/16 10:59:27:051 MyService: ThreadPool: shrinking Ice.ThreadPool.Server: Size=1
-- 6/24/16 11:01:26:964 MyService: ThreadPool: shrinking Ice.ThreadPool.Client: Size=1
-- 6/24/16 11:02:08:179 MyService: Network: trying to accept tcp connection
    Local address = 127.0.0.1:20000
    remote address = 127.0.0.1:53700
-- 6/24/16 11:02:08:208 MyService: Network: accepted tcp connection
    Local address = 127.0.0.1:20000
    remote address = 127.0.0.1:53700
-- 6/24/16 11:02:08:215 MyService: ThreadPool: growing Ice.ThreadPool.Server: Size=2
-- 6/24/16 11:02:08:233 MyService: Network: closed tcp connection
    Local address = 127.0.0.1:20000
    remote address = 127.0.0.1:53700
```

# 1 : IceBox(8)



## 管理IceBox Server

一: **icebox.properties**中增加下面的管理相关的配置

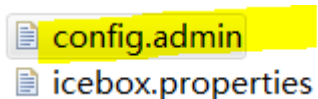
#定义**IceBoxAdmin**名称, 默认是**IceBox**

**Ice.Admin.InstanceName=Box**

#**Ice.Admin**访问**ServiceManager**的端口

**Ice.Admin.Endpoints=tcp -p 9998 -h 127.0.0.1**

二: 新建配置文件: **config.admin**

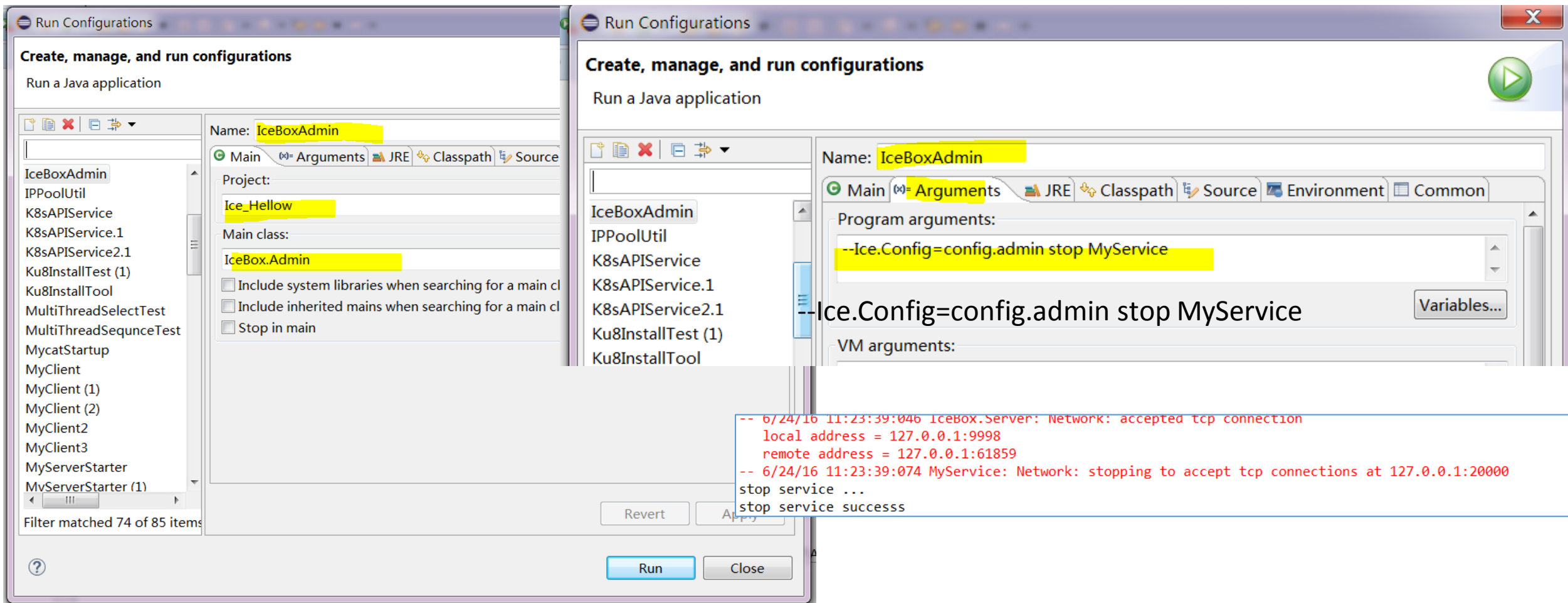


#这里的**Box**要和**Ice.Admin.InstanceName**对应。后面的端口要和**Ice.Admin.Endpoints**的配置相

**IceBoxAdmin.ServiceManager.Proxy=Box/admin -f IceBox.ServiceManager:tcp -p 9998 -h 127.0.0.1**

# 1 : IceBox(9)

## 启动IceBox Admin进程

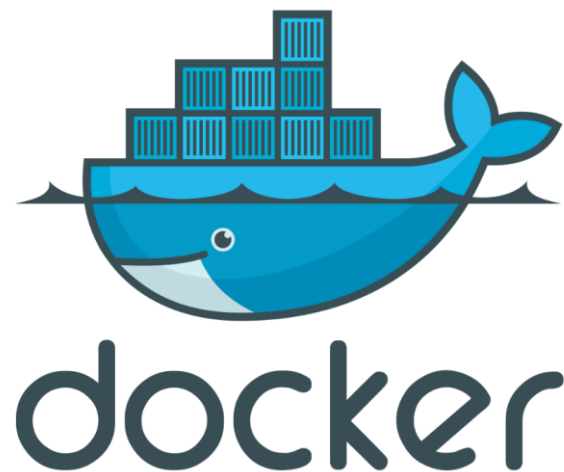


# 1 : IceBox 总结

是一个单独JVM进程

是一个可托管多个“服务”的Server

有API接口可远程管理



# 下一期预告



## 深入IceGrid微服务架构实践



**Leader潜心研究3个月，联合几大高手完善了ice企业框架，并且开源，让ice具有类似docker的特性，远程动态加载微服务的jar包，本视频中首次讲解！**

视频观看地址：<http://www.roncoo.com>

# 谢谢观看

开源项目地址：

<https://github.com/MyCATapache/mycat-ice>

Leader us 高端Java培训报名群： 434568702