# Eclipse下分布式计算环境构建与开发——Eclipse与Slice2Java插件

报告人：尹德春

# 主要内容

1 中间件的概念及作用
2 ICE中间件的介绍
3 ICE 的构成
4 基于ICE中间件的异构环境开发
5 Slice （Specification Language for Ice）
6 实验开发环境
7 Java调用ICE例子程序
8 ICE集群——负载均衡及容错

# 中间件的概念及作用

- **中间件的概念**

中间件是在计算机硬件和操作系统之上,支持应用软件开发和运行的系统软件,它能够使应用软件相对独立于计算机硬件和操作系统平台,为当今的大型分布式应用搭起了一个标准的平台,把大型企业分散的系统和技术组合在一起,实现大型企业应用软件系统的集成。

- **中间件的作用**

中间件具有标准的程序接口和协议,可以实现不同硬件和操作系统平台上的数据共享和应用互操作。在具体实现上,中间件是一个用API 定义的分布式软件管理框架,具有强大的通信能力和良好的可扩展性。中间件在三层结构中起到承上启下的作用,可以大大提高开发效率而缩小应用开发的时间和精力,提高应用开发的成功性。

# ICE中间件的介绍

- **ICE中间件的介绍**

ICE 中间件是由ZeroC 公司推出的新兴的中间件产品。它克服了现存的COBRA 和COM/ DCOM/ COM + 等中间件的缺陷，并具备很更多的优点。

ICE 的主要目标是:

1) 提供适合于异构环境中使用面向对象的中间件平台;

2) 平台为广泛领域中的分布式应用提供一整套强大的特性和功能支持;

3) 平台提供的平台易于学习和使用,避免不必要的复杂性;

4) 平台提供在网络带宽、内存使用、CPU 开销等方面高效的实现;

5) 平台提供内置安全特性的实现,可跨越不安全的网络使用。

# ICE中间件的介绍

- **ICE中间件的介绍**

ICE 中间件构架在应用开发中提供了种种好处:面向对象的语义、消息的同步处理或异步处理支持;多重接口的支持、实现;操作系统和传输协议的无关性、位置透明性、安全性、内建的持久化机制等等,这一切都为使用ICE中间件建构三层或多层应用提供了坚实的基础设施,并提供良好的扩展性和可控的复杂度。

# ICE中间件的介绍

- **ICE 的构成**

ICE 的构成是一系列的软件包,它的主要部分是由 ICEUtil ( ICE 实用工具库) 、Slice ( Specific language for ICE) 、Slice compiler 、ICE core lib( ICE 核心库) 、ICEBox(用来协调一系列应用组件的简易应用服务器) 、ICEPack( ICE 定位服务) 、Freeze ( ICE 内建的持久化机制) 、ICE SSL ( ICE 支持的加密传输协议) 、Glacier 、ICEStorm ( ICE 内建的消息处理机制) 、ICEPatch ( ICE 软件补丁服务) 组成

# 基于ICE中间件的开发

- **基于ICE中间件的异构环境开发**

ICE是一个面向对象的中间件平台。从根本上讲，这意味着ICE提供了一些工具、API和库用以开发面向对象的客户端-服务器的应用程序。

ICE应用程序适合于用在各种异构的环境中：客户端和服务器可以用不同的语言编写，能够运行在不同的操作系统、不同体系结构的机器上，可以用不同的网络技术进行通讯，这些程序的源代码可以在不同的开发环境之间移植。

# Ice对多语言支持的特性

- Slice （Specification Language for Ice）

Ice可以保证多语言之间对象的正确传输，每种语言都有各自的特点、数据类型，Ice是通过Slice （Specification Language for Ice）语言确保各种语言之间正确通信， Slice独立于其它任何语言，Ice可以把Slice代码片断转化为其他语言的描述。

Ice的服务端可以采用C++/Java/Python/C#等任意一种语言实现,客户端也可以采用C++/Java/Python/C#等任意一种语言来实现。多种语言之间采用共同的Slice进行沟通，支持ice到C++,JAVA,C#,Python,Ruby,PHP等多种语言的映射。

# Eclipse下的ICE开发环境

- 实验开发环境：

  **Eclipse （Version: 3.5.2）**

  **Ice-3.4.1.msi**

  **Slice2Java插件**

# Eclipse下的ICE开发环境

- Ice-3.4.1.msi 安装（下载后安装到E:\installed-program\ICE）

## Windows Installers

**Windows Installer for Visual Studio 2008 SP1, Visual Studio 2010, and CodeGear C++Builder 2010**

Ice-3.4.1.msi

Everything needed to develop Ice applications on Windows in all supported programming languages. This installer includes executables, debug and release DLLs, header files, import libraries, PDBs, Java classes, sample programs, third-party dependencies, and the Ice extension for Visual Studio.

*Requirements*

| | |
|---|---|
| C++ | Visual Studio 2008 SP1, Visual C++ 2008 SP1 Express, Visual Studio 2010, Visual C++ 2010 Express, or CodeGear C++Builder 2010 |
| .NET | Visual Studio 2008 SP1 or Visual Studio 2010 |
| Java | Java5 or Java6 |
| Python | Python 2.6.4 |
| Ruby | Ruby 1.8.6 |
| PHP | PHP 5.3.1 |

Note: The Ice extensions for Ruby and PHP included in this installer are compiled with Visual C++ 6.0 (VC6) for compatibility with the the binary distributions available from ruby-lang.org and php.net. Although this installer also includes the VC6 DLLs for Ice and the third-party dependencies required by the Ruby and PHP extensions, general application development with VC6 is not supported.

# Eclipse下的ICE开发环境

- **Slice2Java插件**

## What is the Slice2Java Eclipse Plug-in?

Eclipse is a popular open-source development platform that includes an Integrated Development Environment (IDE) for Java programmers. ZeroC has created a Slice2Java plug-in for Eclipse that automates the translation of Slice files and manages the resulting generated code.

The Slice2Java plug-in provides the following features:

- Handles all aspects of translating your Slice files
- Incrementally recompiles Slice files after modifications
- Maintains dependencies between Slice files
- Highlights compilation errors in your source code
- Manages the generated code to remove obsolete files automatically

The Slice2Java plug-in is only supported with Eclipse Ganymede (3.4).

# Eclipse下的ICE开发环境

- **Slice2Java插件**

## Installation

ZeroC hosts an Eclipse plug-in site that you can add to your Eclipse configuration. Follow these steps to install the Slice2Java plug-in:

1. From the `Help` menu, choose `Software Updates`
2. Select the `Available Software` tab
3. Click `Add Site`
4. In the `Location` field, enter `http://www.zeroc.com/download/eclipse`
5. Select the Slice2Java plug-in and click `Install`

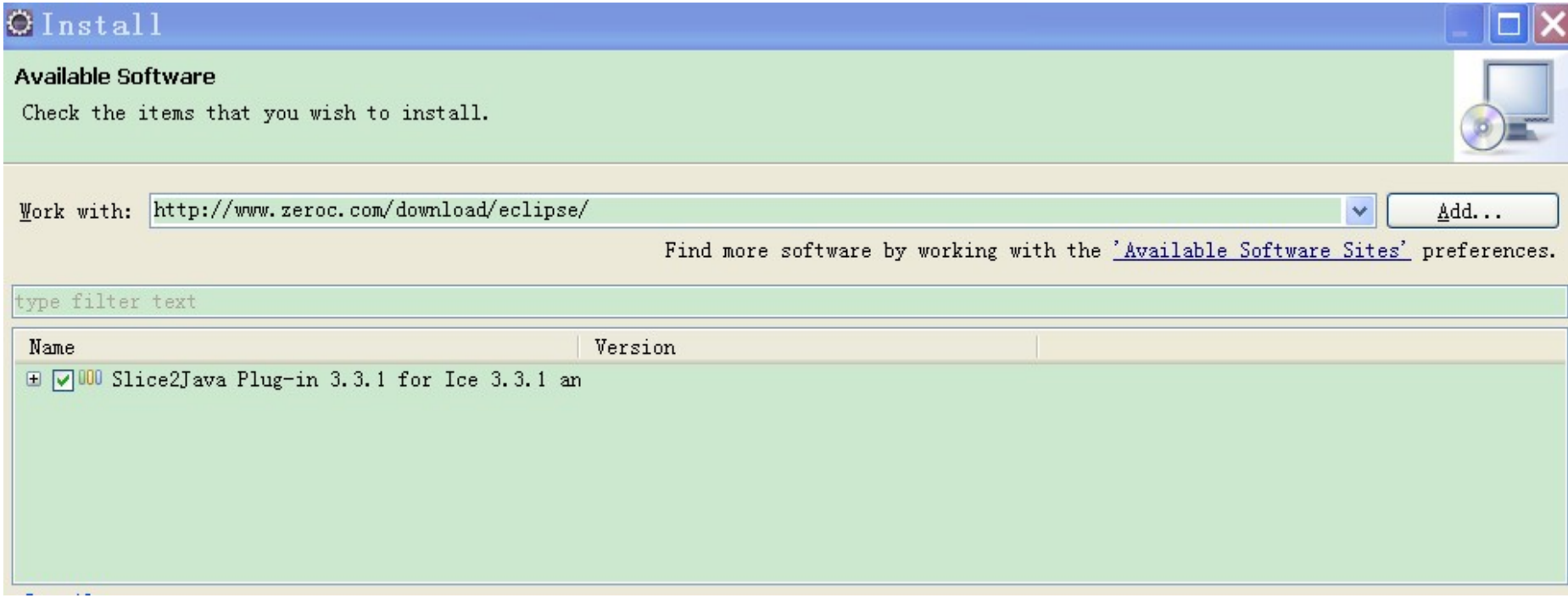
The plug-in requires Ice 3.3.1 or later. Users of Ice for Android 0.1 may also use the plug-in but you must upgrade your Ice installation to version 3.3.1 or later.

Refer to the CHANGES file for a detailed history of the changes in each release.

# Eclipse下的ICE开发环境

- **Slice2Java插件**

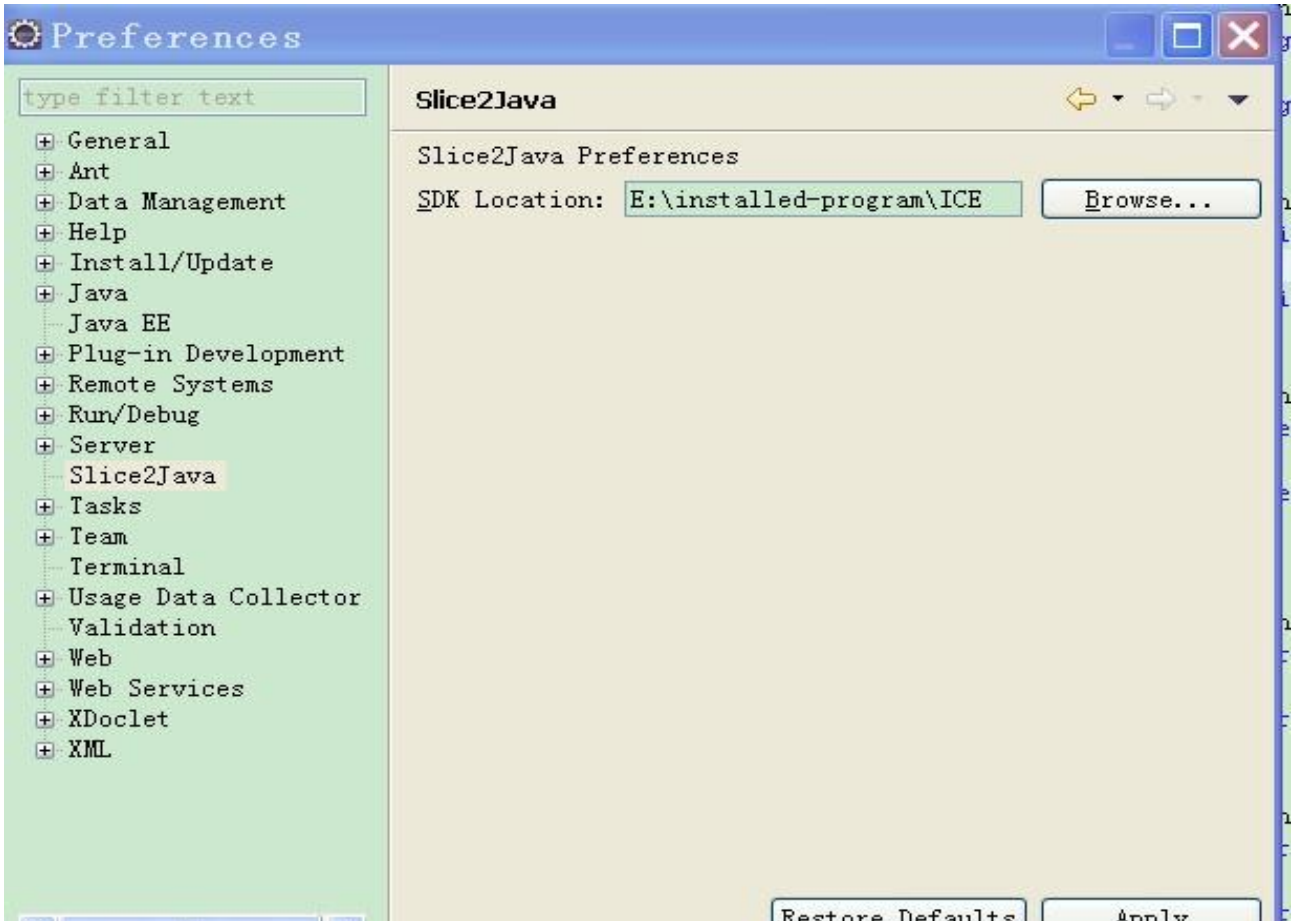
help-install new software - work with...，右侧add按钮，点击add -Add Site - 输入http://www.zeroc.com/download/eclipse/添加好即可。



# Eclipse下的ICE开发环境

- **Slice2Java插件**

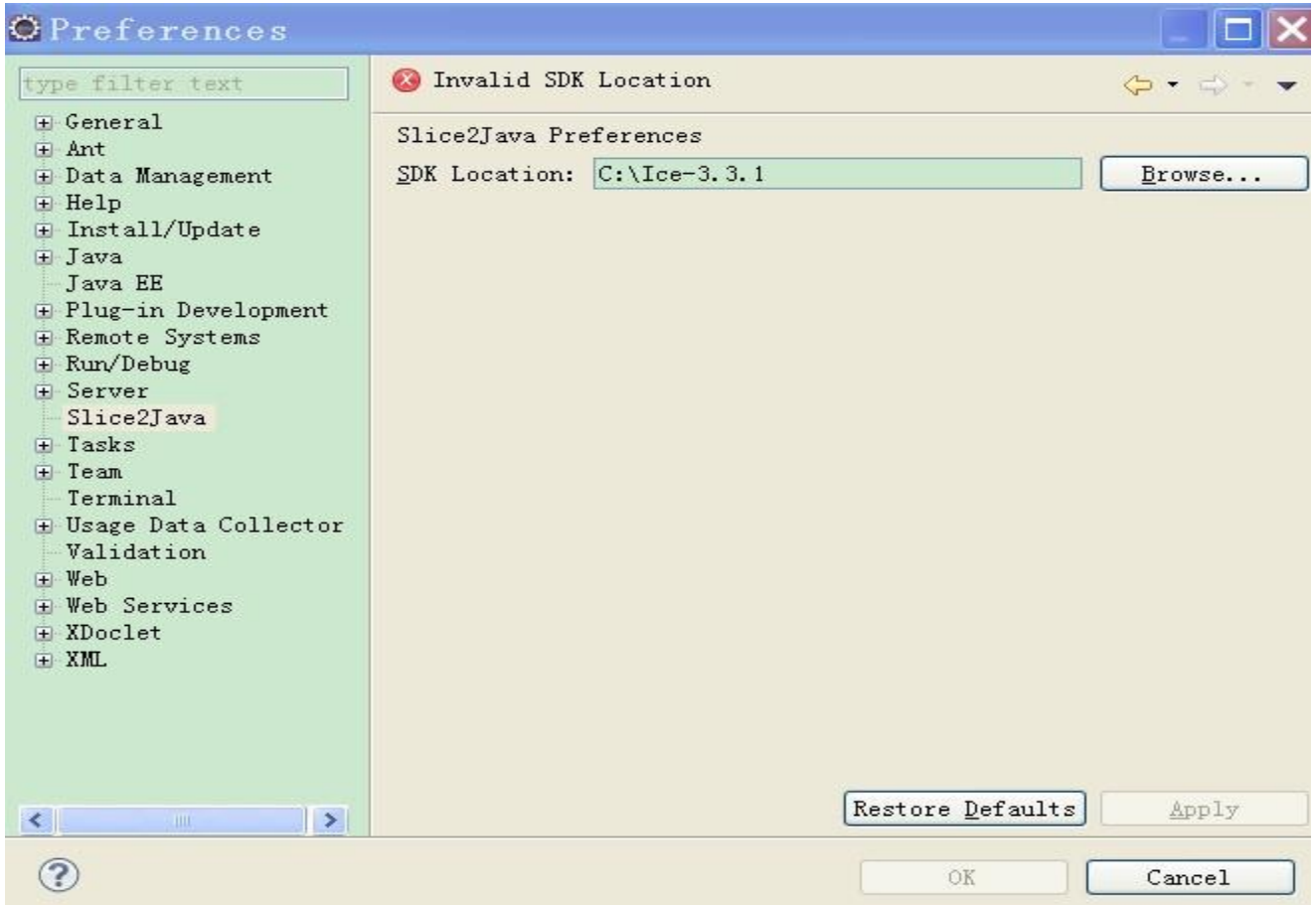
安装成功后：点击Eclipse的Window——preferences,如下图：



# Eclipse下的ICE开发环境

- **Slice2Java插件**

如果指定的路径不对，则显示如下：

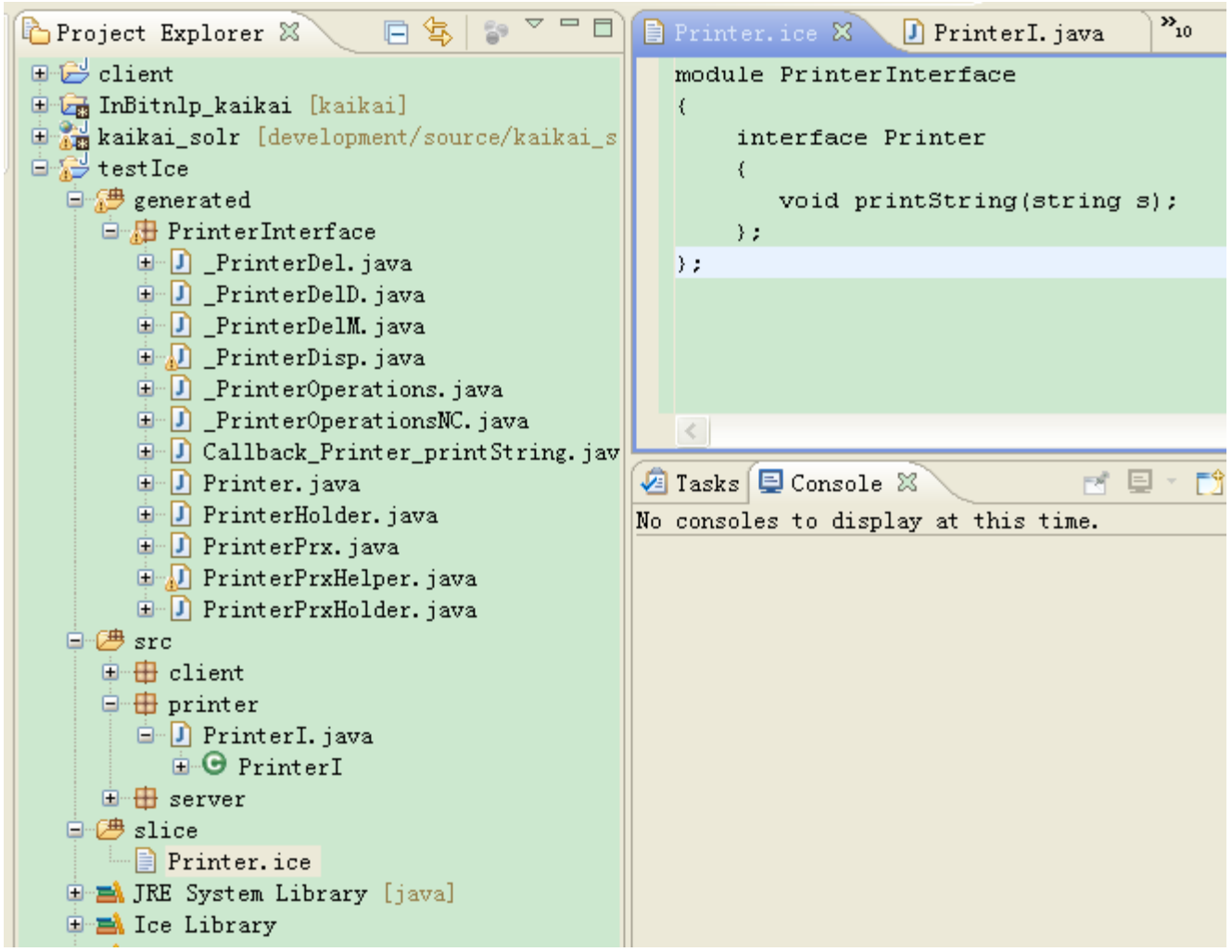


# Java调用ICE例子程序

- 建立Java工程：testIce
- 建立slice文件夹，在其目录下建立：Printer.ice，内容如下：
- ```
  module PrinterInterface
  {
     interface Printer
     {
       void printString(string s);
     };
  };
  ```
  ```

# Java调用ICE例子程序

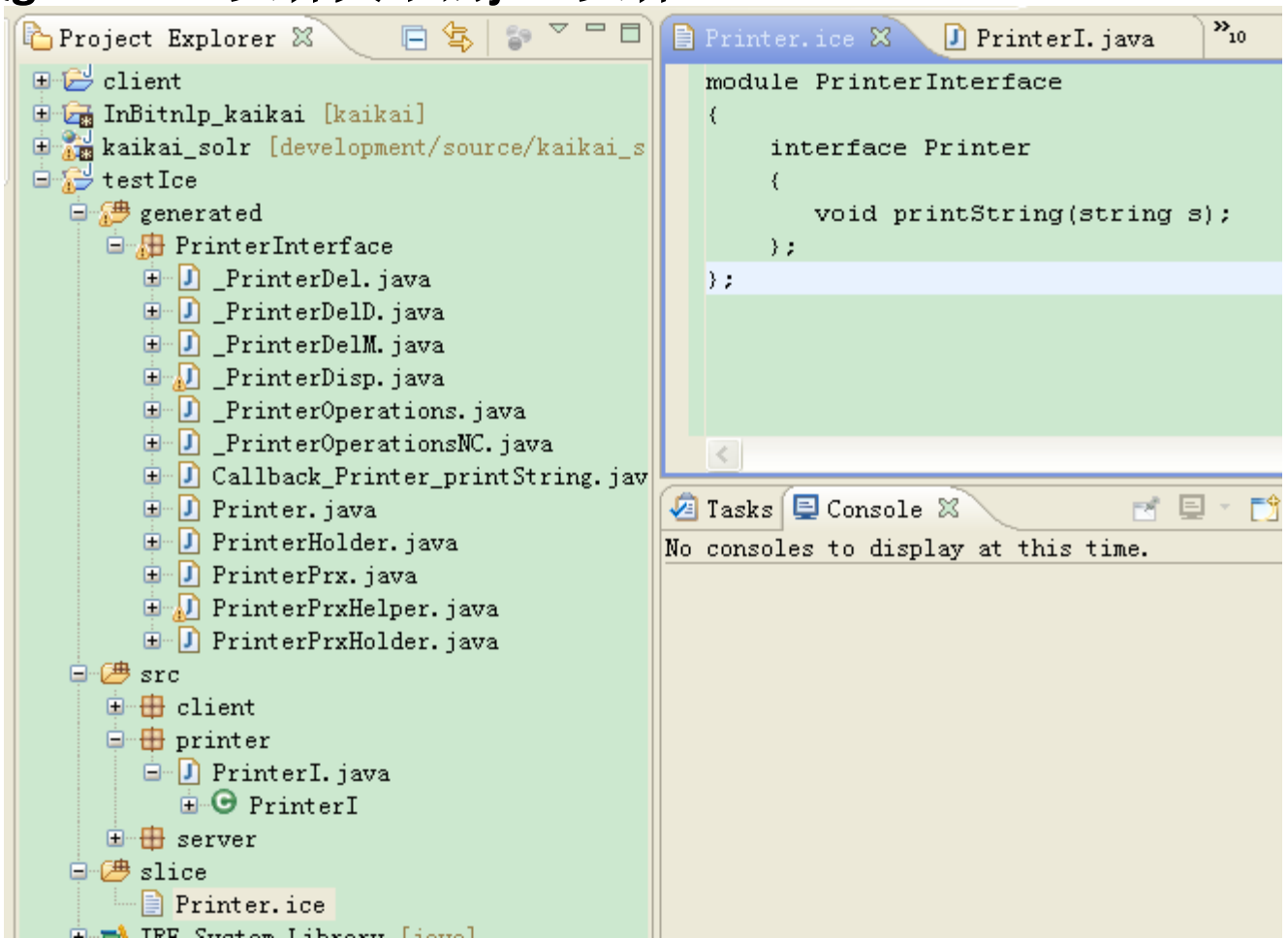
- **Printer.ice**



# Java调用ICE例子程序

- 使用Slice2Java插件

右键点击工程“testIce”——Slice2Java——Add Slice2Java builder，自动生成generated文件夹下的java文件：



# Java调用ICE例子程序

- 编写作为**servant**的**PrinterI**类

```
package printer;
import Ice.Current;
import PrinterInterface._PrinterDisp;
@SuppressWarnings("serial")
public class PrinterI extends _PrinterDisp {
    public void printString(String s, Ice.Current current) {
        System.out.println("server接收到client 传入数据： " + s);
        System.out.println("server返回给client处理结果: " + compute(s));
    }
    public String compute(String s) {
        s = "<server_to_client>" + s + "</server_to_client>";
        return s;
    }
    public void shutdown(Current arg0) {
    }
}
```

# Java调用ICE例子程序

- 编写作为servant的PrinterI类

```
package printer;

import Ice.Current;
import PrinterInterface._PrinterDisp;

@SuppressWarnings("serial")
public class PrinterI extends _PrinterDisp {
    public void shutdown(Current arg0) {
    }

    public void writeMessage(String arg0, int arg1, Current arg2) {
    }

    public void printString(String s, Ice.Current current) {
        System.out.println("server接收到client 传入数据: " + s);
        System.out.println("server返回给client处理结果: " + compute(s));
    }
    public String compute(String s) {
        s = "<server_to_client>" + s + "</server_to_client>";
        return s;
    }
}
```

# Java调用ICE例子程序

- 编写ICE的server

```
package server;
import printer.PrinterI;
public class Server {
 public static void main(String[] a) {
 int status = 0;
 String[] args = { "", "" };
 Ice.Communicator ic = null;
 try {
 ic = Ice.Util.initialize(args);
 Ice.ObjectAdapter adapter = ic.createObjectAdapterWithEndpoints("SimplePrinterAdapter", "default -p 1888");
 Ice.Object object = new PrinterI();
 adapter.add(object, Ice.Util.stringToIdentity("SimplePrinter"));
 adapter.activate();
 ic.waitForShutdown();
 } catch (Ice.LocalException e) {
 e.printStackTrace();
 status = 1;
 } catch (Exception e) {
 System.err.println(e.getMessage());
 status = 1;
 } finally {
 if (ic != null)
 ic.destroy();
 }
 System.exit(status);
 }
```

# Java调用ICE例子程序

- 编写ICE的server

```
package server;

import printer.PrinterI;

public class Server {
    public static void main(String[] a) {
        int status = 0;
        Ice.Communicator ic = null;
        try {
            String[] args = { "", "" };
            ic = Ice.Util.initialize(args);
            // ic = Ice.Util.initialize();
            Ice.ObjectAdapter adapter = ic.createObjectAdapterWithEndpoints(
                    "SimplePrinterAdapter", "default -p 1888");
            Ice.Object object = new PrinterI();
            adapter.add(object, Ice.Util.stringToIdentity("SimplePrinter"));
            adapter.activate();
            ic.waitForShutdown();
        } catch (Ice.LocalException e) {
            e.printStackTrace();
            status = 1;
        } catch (Exception e) {
            System.err.println(e.getMessage());
            status = 1;
        } finally {
            if (ic != null)
                ic.destroy();
        }
        System.exit(status);
    }
}
```

# Java调用ICE例子程序

- 编写client客户端

```
package client;
import PrinterInterface.PrinterPrx;
import PrinterInterface.PrinterPrxHelper;
 public class Client {
    public static void main(String[] args) {
    int status = 0;
    Ice.Communicator ic = null;
    try {
    ic = Ice.Util.initialize(args);
    Ice.ObjectPrx base = ic.stringToProxy("SimplePrinter:default -h 127.0.0.1 -p 1888");
    PrinterPrx printer = PrinterPrxHelper.checkedCast(base);
    if (printer == null)
    throw new Error("Invalid proxy");
    printer.printString("client->server计算请求!");
     } catch (Ice.LocalException e) {
    e.printStackTrace();
    status = 1;
    } catch (Exception e) {
    System.err.println(e.getMessage());
    status = 1;
    } finally {
    if (ic != null)
    ic.destroy();
     }
    System.exit(status);
    }
```

# Java调用ICE例子程序

- 编写client客户端

```
package client;

import PrinterInterface.PrinterPrx;

public class Client {

    public static void main(String[] args) {
        int status = 0;
        Ice.Communicator ic = null;
        try {
            ic = Ice.Util.initialize(args);
            Ice.ObjectPrx base = ic
                    .stringToProxy("SimplePrinter:default -h 127.0.0.1 -p 1888");
            PrinterPrx printer = PrinterPrxHelper.checkedCast(base);
            if (printer == null)
                throw new Error("Invalid proxy");
            printer.printString("client->server计算请求!");
        } catch (Ice.LocalException e) {
            e.printStackTrace();
            status = 1;
        } catch (Exception e) {
            System.err.println(e.getMessage());
            status = 1;
        } finally {
            if (ic != null)
                ic.destroy();
        }
        System.exit(status);
    }

}
```

# Java调用ICE例子程序

- 在一台机器上运行**Server**，另一台运行**Client**，在运行**Server**的机器上输出：

server接收到client 传入数据：client->server计算请求!

server返回给client处理结果: <server_to_client>client->server计算请求!</server_to_client>

如果客户端不断地请求，则服务端不断地处理来自客户端的请求，并把计算结果返回给客户端。客户端只是调用接口函数，并不知道是哪个计算节点提供的计算服务。

说明：工程目录下不能包含中文路径，否则，右键点击工程“MyTestICE”——Slice2Java——Add Slice2Java builder 时候，ice 编译出错：

Errors occurred during the build. Errors running builder 'Slice2Java Builder' on project 'testICE'. internal error reading the generated output list Invalid byte 1 of 1-byte UTF-8 sequence. internal error reading the generated output list Invalid byte 1 of 1-byte UTF-8 sequence.

# ICE集群——负载均衡及容错

- **Single Server**

在同一个主机上，**ICE**服务支持多端口的监听。

**(1)**服务端注册： **tcp -h host -p port1:tcp -h host -p port2**

例如：**IP:172.17.12.101**，需要在**10001**和**10000**同时监听，就可以写成:

**tcp -h 172.17.12.101 -p 10000:tcp -h 172.17.12.101 -p 10001**

运行之后，服务就监听于**10000**和**10001**端口。

**(2)**客户端连接可以采用如下**3**种形式：

**1. tcp -h 172.17.12.101 -p 10000**

**2. tcp -h 172.17.12.101 -p 10001**

**3. tcp -h 172.17.12.101 -p 10000:tcp -h 172.17.12.101 -p 100001**

无论**Server**监听多少个端口，还是只有唯一的一个**Server**在工作。

# ICE集群——负载均衡及容错

- **Multiple Server**

对于**Client**较多的应用或负载要求很高的情况下，我们可以把**Server**程序运行于多台主机之上。通过集群方式合理有效的化解来自**Client**的压力。例如:

**ServerA 172.17.12.101 tcp -h 172.17.12.101 -p 10000**

**ServerB 172.17.12.102 tcp -h 172.17.12.102 -p 10000**

**ServerC 172.17.12.103 tcp -h 172.17.12.103 -p 10000**

**Client**可以如下的连接方式:

**tcp -h 172.17.12.101 -p 10000:tcp -h 172.17.12.102 -p 10000:**

**tcp -h 172.17.12.103 -p 10000**

或是

**tcp -h 172.17.12.101 -p 10000:tcp -h 172.17.12.102 -p 10000**

等多种情况，可以根据应用的具体要求合理有效的构造所需连接主机的字符串。这种连接方式可以有效地利用**ICE**提供的**load balancing**功能,把**Client**的每个请求合理的分配到每个**Server**。从而有效地避免了**Client**大量请求对同一台**Server**的巨大压力。

# ICE集群——负载均衡及容错

- 容错功能

ICE自身提供一种极其强大的集群功能和容错功能。具体体现在如果当某个Server(假设是ServerA)宕机之后,来自Client的请求分配到ServerA服务器上,Client会自动记录ServerA失效状态,会把请求再分配给可正常工作的Server(ServerB,ServerC),对于用户的每次请求都能分配到正常的服务主机上(除非A,B,C都同时宕机).当ServerA回复正常之后,Client会自动感知ServerA工作状态.Client的请求又可以有效地分配到上述A,B,C主机上.这一切对于开发者都是透明的.

# 谢谢