JDK1.5/1.6/1.7 之新特性总结

http://caowei3047.javaeye.com/blog/725079

开发过程中接触到了从 jdk1.5---jdk1.7 的使用，在不同的阶段，都使用过了 jdk 的一些新特性，操作起来更加方面啦！特此总结了下，与喜欢 it 的朋友共勉！呵呵

以下是测试代码：

JDK1.5 新特性：

1.自动装箱与拆箱：

```
Integer iObj = 3;

System.out.println(iObj + 12);
      Integer i1 = 137(-128--127 范围时，为 true);
      Integer i2 = 137(-128--127 范围时，为 true);

      System.out.println(i1 == i2); //false，但是括号中时却返回 ture，原因是 Integer 采用的是享元模式

      Integer i3 = Integer.valueOf(213);
      Integer i4 = Integer.valueOf(213);
      System.out.println(i3==i4);//同上，另一种包装形式
```

2.枚举(常用来设计单例模式)

```
public class EnumTest {

/**
* @param args
*/
public static void main(String[] args) {
      WeekDay1 weekDay = WeekDay1.MON;
      System.out.println(weekDay.nextDay());
      WeekDay weekDay2 = WeekDay.FRI;
      System.out.println(weekDay2);
      System.out.println(weekDay2.name());
      System.out.println(weekDay2.ordinal());
      System.out.println(WeekDay.valueOf("SUN").toString());
```

```
        System.out.println(WeekDay.values().length);
        new Date(300){};
}

public enum WeekDay{

        SUN(1),MON(),TUE,WED,THI,FRI,SAT;
        private WeekDay(){System.out.println("first");}
        private WeekDay(int day){System.out.println("second");}
}

public enum TrafficLamp{
        RED(30){
          public TrafficLamp nextLamp(){
            return GREEN;
          }
        },
        GREEN(45){
          public TrafficLamp nextLamp(){
            return YELLOW;
          }
        },
        YELLOW(5){
          public TrafficLamp nextLamp(){
            return RED;
          }
        };
        public abstract TrafficLamp nextLamp();
        private int time;
        private TrafficLamp(int time){this.time = time;}
}
}
```

3.静态导入

```
import static java.lang.Math.*;

public class StaticImport {
public static void main(String[] args){
        int x = 1;
        try {
          x++;
        } finally {
          System.out.println("template");
        }
```

```
        System.out.println(x);


        System.out.println(max(3, 6));
        System.out.println(abs(3 - 6));

}
}
```

4.可变参数

```
public class VarableParameter {

/**
* @param args
*/
public static void main(String[] args) {

        System.out.println(add(2,3));
        System.out.println(add(2,3,5));
}


public static int add(int x,int... args){
        int sum = x;
/*      for(int i=0;i<args.length;i++){
        sum += args[i];
        }*/

        for(int arg : args){
        sum += arg;
        }
        return sum;
}

}
```

5.内省

```
ReflectPoint pt1 = new ReflectPoint(3,5);

BeanInfo beanInfo = Introspector.getBeanInfo(pt1.getClass());
        PropertyDescriptor[] pds = beanInfo.getPropertyDescriptors();
        Object retVal = null;
        for(PropertyDescriptor pd : pds){
            Method methodGetX = pd.getReadMethod();
            retVal = methodGetX.invoke(pt1);
```

}

jdk1.6 新特性：

1.Web 服务元数据

Java 里的 Web 服务元数据跟微软的方案基本没有语义上的区别,自从 JDK5 添加了元数据功能(Annotation)之后,SUN 几乎重构了整个 J2EE 体 系, 由于变化很大,干脆将名字也重构为 Java EE, Java EE(当前版本为 5.0)将元数据纳入很多规范当中,这其中就包括 Web Services 的相关规范, 加入元数据之后的 Web Services 服务器端编程模型就跟上面看到的 C#片断差不多了, 这显然比以前的 JAX-RPC 编程模型简单(当然, Axis 的编程模型也很简单).这里要谈的 Web 服务元数据(JSR 181)只是 Java Web 服务规范中的一个,它跟 Common Annotations, JAXB2, StAX, SAAJ 和 JAX-WS 等共同构成 Java EE 5 的 Web Services 技术堆栈.

```
package WebServices;

import java.io.File;
import java.io.IOException;
import javax.jws.Oneway;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.xml.ws.Endpoint;

/**
* @author chinajash
*/
@WebService(targetNamespace="http://blog.csdn.net/chinajash",serviceName="HelloService")
public class WSProvider {
        @WebResult(name="Greetings")//自定义该方法返回值在 WSDL 中相关的描述
        @WebMethod
        public String sayHi(@WebParam(name="MyName") String name){
                return "Hi,"+name; //@WebParam 是自定义参数 name 在 WSDL 中相关的描述
        }
        @Oneway //表明该服务方法是单向的,既没有返回值,也不应该声明检查异常
        @WebMethod(action="printSystemTime",operationName="printSystemTime")//自定义该方法在 WSDL 中相关的描述
        public void printTime(){
                System.out.println(System.currentTimeMillis());
```

```
        }
        public static void main(String[] args) {
                Thread wsPublisher = new Thread(new WSPublisher());
                wsPublisher.start();
        }
        private static class WSPublisher implements Runnable{
                public void run() {
                        //发布 WSProvider 到
http://localhost:8888/chinajash/WSProvider 这个地址,之前必须调用 wsgen
命令
                        //生成服务类 WSProvider 的支持类,命令如下:
                        //wsgen -cp . WebServices.WSProvider
                        Endpoint.publish("http://localhost:8888/chin
ajash/WSProvider",new WSProvider());
                }
        }
}
```

如果想看到 Web Services Engine 生成的 WSDL 文件是否遵守上面的元数据, 我们没有必要将上面的 WSProvider 部署到支持 JSR-181 的应用服务器或 Servlet 形式的 Web Services Engine,现在 JDK6 已经提供了一个很简单的机制可以用来测试和发布 Web Services,下面讲讲如何在 JDK6 环境下发布 Web Services 和查看生成的 WSDL

1.将<JDK_HOME>/bin 加入 path 环境变量

2.在命令行下切换当前目录到WSProvider的class文件所在的目录,运行下面命令

```
wsgen -cp . WebServices.WSProvider
```

在这个例子中会生成以下 3 个类的源代码文件及 class 文件

SayHi

SayHiResponse

PrintTime

3.执行如下代码发布 WSProvider 到 http://localhost:8888/chinajash/WSProvider,在这里可以执行 WSProvider 类的 main 方法就可以

```
Endpoint.publish("http://localhost:8888/chinajash/WSProvider",new
WSProvider());
```

4.在浏览器输入 http://localhost:8888/chinajash/WSProvider?wsdl 就可以看到生成的 WSDL 文件，为了节省篇幅,这里就不把生成的 WSDL 文件贴上了，大家可以自己动手试试.

2.脚本语言支持

JDK6 增加了对脚本语言的支持(JSR 223),原理上是将脚本语言编译成 bytecode,这样脚本语言也能享用 Java 平台的诸多优势，包括可移植性，安全等，另外，由于现在是编译成 bytecode 后再执行,所以比原来边解释边执行效率要高很多。

加入对脚本语言的支持后，对 Java 语言也提供了以下好处。
1、许多脚本语言都有动态特性，比如，你不需要用一个变量之前先声明它，你可以用一个变量存放完全不同类型的对象，你不需要做强制类型转换，因为转换都是自动的。现在 Java 语言也可以通过对脚本语言的支持间接获得这种灵活性。
2、 可以用脚本语言快速开发产品原型，因为现在可以 Edit-Run，而无需 Edit-Compile-Run，当然，因为 Java 有非常好的 IDE 支持，我 们完全可以在 IDE 里面编辑源文件，然后点击运行(隐含编译)，以此达到快速开发原型的目的，所以这点好处基本上可以忽略。
3、通过引入脚本语言可以轻松实现 Java 应用程序的扩展和自定义，我们可以把原来分布在在 Java 应用程序中的配置逻辑，数学表达式和业务规则提取出来，转用 JavaScript 来处理。

Sun 的 JDK6 实现包含了一个基于 Mozilla Rhino 的 脚本语言引擎，支持 JavaScript，这并不是说明 JDK6 只支持 JavaScript，任何第三方都可以自己实现一个 JSR-223 兼容的脚本引擎 使得 JDK6 支持别的脚本语言，比如，你想让 JDK6 支持 Ruby，那你可以自己按照 JSR 223 的规范实现一个 Ruby 的脚本引擎类，具体一点，你需要实现 javax.script.ScriptEngine(简单起见，可以继承 javax.script.AbstractScriptEngine)和 javax.script.ScriptEngineFactory 两个接口。当然，在你实现自己的脚本语言引擎之前，先到 scripting.dev.java.net project 这里看看是不是有人已经帮你做了工作，这样你就可以直接拿来用就行。

Scripting API

Scripting API 是用于在 Java 里面编写脚本语言程序的 API， 在 Javax.script 中可以找到 Scripting API，我们就是用这个 API 来编写 JavaScript 程序，这个包里面有一个 ScriptEngineManager 类，它是使用 Scripting API 的入口，ScriptEngineManager 可以通过 jar 服务发现(service discovery)机制寻找合适的脚本引擎类(ScriptEngine)，使用 Scripting API 的最简单方式只需下面三步
1、创建一个 ScriptEngineManager 对象
2、通过 ScriptEngineManager 获得 ScriptEngine 对象
3、用 ScriptEngine 的 eval 方法执行脚本

下面是一个 Hello World 程序

```
/** * @author chinajash */public class HelloScript {public static void main(String[] args) throws Exception
{                 ScriptEngineManager factory = new ScriptEngineManager();//step 1                 ScriptEngine engine = factory.getEngineByName("JavaScript");//Step 2                         engine.eval("print('Hello,
```

Scripting')");//Step 3 } }运行上面程序，控制台会输出 Hello, Scripting 上面这个简单的 Scripting 程序演示了如何在 Java 里面运行脚本语言，除此之外，我们还可以利用 Scripting API 实现以下功能 1、暴露 Java 对象为脚本语言的全局变量 2、在 Java 中调用脚本语言的方法 3、脚本语言可以实现 Java 的接口 4、脚本语言可以像 Java 一样使用 JDK 平台下的类下面的类演示了以上 4 种功能 package Scripting;import java.io.File;import javax.script.Invocable;import javax.script.ScriptEngine;import javax.script.ScriptEngineManager;import javax.script.ScriptException;/** * @author chinajash */public class ScriptingAPITester { public static void main(String[] args) throws Exception { ScriptEngineManager manager = new ScriptEngineManager(); ScriptEngine engine = manager.getEngineByName("JavaScript"); testScriptVariables(engine);//演示如何暴露 Java 对象为脚本语言的全局变量 testInvokeScriptMethod(engine);//演示如何在 Java 中调用脚本语言的方法 testScriptInterface(engine);//演示脚本语言如何实现 Java 的接口 testUsingJDKClasses(engine);//演示脚本语言如何使用 JDK 平台下的类 } public static void testScriptVariables(ScriptEngine engine) throws ScriptException{ File file = new File("test.txt"); engine.put("f", file); engine.eval("println('Total Space:'+f.getTotalSpace())"); } public static void testInvokeScriptMethod(ScriptEngine engine) throws Exception{ String script = "function hello(name) { return 'Hello,' + name;}"; engine.eval(script); Invocable inv = (Invocable) engine; String res = (String)inv.invokeFunction("hello", "Scripting"); System.out.println("res:"+res); } public static void testScriptInterface(ScriptEngine engine) throws ScriptException{ String script = "var obj = new Object(); obj.run = function() { println('run method called'); }"; engine.eval(script); Object obj = engine.get("obj"); Invocable inv = (Invocable) engine; Runnable r = inv.getInterface(obj,Runnable.class); Thread th = new Thread(r); th.start(); } public static void testUsingJDKClasses(ScriptEngine engine) throws Exception{ //Packages 是脚本语言里的一个全局变量,专用于访问 JDK 的 package String js = "function doSwing(t){var f=new

```
Packages.javax.swing.JFrame(t);f.setSize(400,300);f.setVisible(true);}
";                  engine.eval(js);                  Invocable inv =
(Invocable) engine;                  inv.invokeFunction("doSwing",
"Scripting Swing" );         }}Scripting Tool
```

SUN 提供的 JDK6 中有一个命令行工具??jrunscript，你可以在<JDK6_Home>/bin 下面找到这个工具，jrunscript 是一个脚本语言的解释程序，它独立于脚本语言，但默认是用 JavaScript，我们可以用 jrunscript 来测试自己写的脚本语言是否正确，下面是一个在命令行运行 jrunscript 的简单例子

```
jrunscript
js>println("Hello,JrunScript");
Hello,JrunScript
js>9*8
72.0
js>
```

3.JTable 的排序和过滤

原来的 JTable 基本上是只能显示数据，在 JDK6 新增了对 JTable 的排序和过滤功能，下面代码演示了这两个功能

```
/**
* @author chinajash
*/
public class JTableTester {
        static String data[][] = {
                {"China","Beijing","Chinese"},
                {"America","Washington","English"},
                {"Korea","Seoul","Korean"},
                {"Japan","Tokyo","Japanese"},
                {"France","Paris","French"},
                {"England","London","English"},
                {"Germany","Berlin","German"},
        };
        static String titles[] = {"Country","Capital","Language"};
        public static void main(String[] args) {
                DefaultTableModel m = new
DefaultTableModel(data,titles);
                JTable t = new JTable(m);
                final TableRowSorter sorter = new TableRowSorter(m);
                t.setRowSorter(sorter); //为 JTable 设置排序器

                JScrollPane sPane = new JScrollPane();
```

```
                sPane.setViewportView(t);

                JPanel p = new JPanel();
                p.setLayout(new BoxLayout(p,BoxLayout.X_AXIS));
                JLabel l = new JLabel("Criteria:");
                final JTextField tf = new JTextField();
                JButton b = new JButton("Do Filter");
                p.add(l);
                p.add(tf);
                p.add(b);
                b.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                if(tf.getText().length()==0){
                                        sorter.setRowFilter(null);
                                }else{
                                        sorter.setRowFilter(RowFilte
r.regexFilter(tf.getText()));//为 JTable 设置基于正则表达式的过滤条件
                                }
                        }
                });

                JFrame f = new JFrame("JTable Sorting and Filtering");
                f.getContentPane().add(sPane,BorderLayout.CENTER);

                f.getContentPane().add(p,BorderLayout.SOUTH);
                f.setSize(400,300);
                f.setVisible(true);
        }
}
```

运行上面程序，单击 JTable 的某一个 title，这个 title 对应的列就会按照升序/降序重新排列；在下面的 Criteria 文本框中输入"ese"，点击"Do Filter"按钮，JTable 将只显示带有"ese"字符串的行，也就是 China 和 Japan 两行，如果文本框里面什么都没有，点击"Do Filter"按钮，这时 JTable 会显示所有的行。

4.**更简单,更强大的 JAX-WS**

JAX-WS2.0 的来历

JAX-WS(JSR-224) 是 Java Architecture for XML Web Services 的缩写,简单说就是一种用 Java 和 XML 开发 Web Services 应用程序的框架, 目前版本是 2.0,它是 JAX-RPC 1.1 的后续版本, J2EE 1.4 带的就是 JAX-RPC1.1, 而 Java EE 5 里面包括了 JAX-WS 2.0,但为了向后兼容,仍然支持 JAX-RPC. 现在,SUN 又把 JAX-WS 直接放到了 Java SE 6 里面,由于 JAX-WS 会用到 Common Annotation(JSR

250),Java Web Services Metadata(JSR 181), JAXB2(JSR 222), StAX(JSR 173),所以SUN也必须把后几个原属于Java EE范畴的Components下放到Java SE, 现在我们可以清楚地理解了为什么Sun要把这些看似跟Java SE没有关系的Components放进来,终极目的就是要在Java SE里面支持Web Services.

JAX-WS2.0的架构

JAX-WS不是一个孤立的框架,它依赖于众多其他的规范,本质上它由以下几部分组成
1.用来开发Web Services的Java API
2.用来处理Marshal/Unmarshal的XML Binding机制,JAX-WS2.0用JAXB2来处理Java Object与XML之间的映射,Marshalling就是把Java Object映射到XML,Unmarshalling则是把XML映射到Java Object.之所以要做Java Object与XML的映射,是因为最终作为方法参数和返回值的Java Object要通过网络传输协议(一般是SOAP)传送,这就要求必须对Java Object做类似序列化和反序列化的工作,在SOAP中就是要用XML来表示Java object的内部状态
3.众多元数据(Annotations)会被JAX-WS用来描述Web Services的相关类,包括Common Annotations, Web Services Metadata, JAXB2的元数据和JAX-WS2.0规范自己的元数据.
4.Annotation Processing Tool(APT)是JAX-WS重要的组成部分,由于JAX-WS2.0规范用到很多元数据,所以需要APT来处理众多的Annotations.在<JDK_HOME>/bin下有两个命令wsgen和wsimport,就是用到APT和Compiler API来处理碰到的Annotations,wsgen可以为Web Services Provider产生并编译必要的帮助类和相关支持文件,wsimport以WSDL作为输入为Web Service Consumer产生并编译必要的帮助类和相关支持文件.
5.JAX-WS还包括JAX-WS Runtime与应用服务器和工具之间的契约关系

JAX-WS2.0的编程模型

现在用JAX-WS2.0来编写Web Services非常简单,不像JAX-RPC,JAX-WS可以把任意POJO暴露为Web Services,服务类不需要实现接口,服务方法也没有必要抛出RMI异常.下面介绍在JDK6环境下用JAX-WS2.0开发和测试Web Services的步骤
1.编写服务类,并用Web Services Metadata(JSR-181)标注这个服务类,我用我的另一篇BlogJDK6的新特性之十:Web服务元数据中的WSProvider类作为服务类的例子,在此我重复贴一下WSProvider类的源代码:

```
/**
* @author chinajash
*/
@WebService(targetNamespace="http://blog.csdn.net/chinajash",serviceN
```

```
ame="HelloService")
public class WSProvider {
        @WebResult(name="Greetings")//自定义该方法返回值在 WSDL 中相
关的描述
        @WebMethod
        public String sayHi(@WebParam(name="MyName") String name){
                return "Hi,"+name; //@WebParam 是自定义参数 name 在 WSDL
中相关的描述
        }
        @Oneway //表明该服务方法是单向的,既没有返回值,也不应该声明检
查异常
        @WebMethod(action="printSystemTime",operationName="printSyst
emTime")//自定义该方法在 WSDL 中相关的描述
        public void printTime(){
                System.out.println(System.currentTimeMillis());
        }
        public static void main(String[] args) {
                Thread wsPublisher = new Thread(new WSPublisher());
                wsPublisher.start();
        }
        private static class WSPublisher implements Runnable{
                public void run() {
                        //发布 WSProvider 到
http://localhost:8888/chinajash/WSProvider 这个地址,之前必须调用 wsgen
命令
                        //生成服务类 WSProvider 的支持类,命令如下:
                        //wsgen -cp . WebServices.WSProvider
                        Endpoint.publish("http://localhost:8888/chin
ajash/WSProvider",new WSProvider());
                }
        }
}
```

2.用 wsgen 生成上面服务类的必要的帮助类,然后调用用 EndPoint 类的静态方法 publish 发布服务类(步骤请参考我的另一篇 Blog JDK6 的新特性之十:Web 服务元数据),我在这里是将服务类发布到
http://localhost:8888/chinajash/WSProvider

3.用 wsimport 为服务消费者(也就是服务的客户端)生成必要的帮助类,命令如下:

```
wsimport http://localhost:8888/chinajash/WSProvider?wsdl
```

这会在<当前目录>\net\csdn\blog\chinajash 下生成客户端的帮助类,在这个例子中会生成 7 个类

```
HelloService.class
ObjectFactory.class
package-info.class
```

PrintSystemTime.class
SayHi.class
SayHiResponse.class
WSProvider.class
4.在客户端用下面代码即可调用步骤 1 定义的 Web Service
HelloService hs = new HelloService();
WSProvider ws = hs.getWSProviderPort();
System.out.println(ws.sayHi("chinajash"));
ws.printSystemTime();
调用上述代码后客户端控制台输出
hi,chinajash
服务端控制台输出服务器当前系统时间
5.**轻量级 Http Server**

JDK6 的新特性之五:轻量级 Http Server

JDK6 提供了一个简单的 Http Server API,据此我们可以构建自己的嵌入式 Http Server,它支持 Http 和 Https 协议,提供了 HTTP1.1 的部分实现，没有被实现的那部分可以通过扩展已有的 Http Server API 来实现,程序员必须自己实现 HttpHandler 接口,HttpServer 会调用 HttpHandler 实现类的回调方法来处理客户端请求,在这里,我们把一个 Http 请求和它的响应称为一个交换,包装成 HttpExchange 类,HttpServer 负责将 HttpExchange 传给 HttpHandler 实现类的回调方法.下面代码演示了怎样创建自己的 Http Server

```
/**
* Created by IntelliJ IDEA.
* User: Chinajash
* Date: Dec 30, 2006
*/
public class HTTPServerAPITester {
         public static void main(String[] args) {
                  try {
                           HttpServer hs = HttpServer.create(new
InetSocketAddress(8888),0);//设置 HttpServer 的端口为 8888
                           hs.createContext("/chinajash", new
MyHandler());//用 MyHandler 类内处理到/chinajash 的请求
                           hs.setExecutor(null); // creates a default
executor
                           hs.start();
                  } catch (IOException e) {
                           e.printStackTrace();
                  }
         }
}
```

```
class MyHandler implements HttpHandler {
       public void handle(HttpExchange t) throws IOException {
              InputStream is = t.getRequestBody();
              String response = "<h3>Happy New Year
2007!--Chinajash</h3>";
              t.sendResponseHeaders(200, response.length());
              OutputStream os = t.getResponseBody();
              os.write(response.getBytes());
              os.close();
       }
}
```

运行程序后,在浏览器内输入 http://localhost:8888/xx,浏览器输出

6.嵌入式数据库 Derby

Derby 是 IBM 送给开源社区的又一个礼物，是一个 pure java 的数据库，现在已经被列入到 java1.6 中。
不知道对于大数据量的性能如何，但传说中启动 derby 只会给 JVM 添加 2M 的内存，对那些小数据库应用，比如像用 access 那种应该是挺有诱惑力的。
另外，麻雀虽小，五脏俱全，功能要比 access 多得多咯，包括事务处理，并发，触发器都有，管理又简单，因此自己用来做点工具正好合适。
废话少说，介绍一下我折腾了半天的经验吧。
我的 Derby 配置过程：
1，下载 db-derby-10.1.3.1-bin.tar.gz,derby_core_plugin_10.1.3.zip 和 derby_ui_plugin_1.1.0.zip，把两个插件安装到 eclipse 上
2，打开 ecllipse,新建一个 project
3，右键这个 project，选择 Apache Derby，再选择 add apache derby native，发现只是给我的 project 添加了几个 derby 的 jar，还不是在我看着顺眼的 lib 目录里，索性干掉，换上 db-derby- 10.1.3.1-bin.tar.gz 解压出来以后 lib 目录下的 jar 文件，在 Build Path 里设置一下；
4，右键 Project，在 apache derby 里选择 start apache derby network server，控制台可以看到 derby 启动后打出的“服务器准备在端口 1527 上接受连接。”
5，右键 Project，在 apache derby 里选择 ij(Interactive SQL)，启动 SQL 控制台；
6，输入 connect jdbc:derby:testdb;create=true; 注意要有单引号，可以在工程跟目录下创建 testdb 数据库，可以看到一个新建的目录 testdb，那里的文件就是数据库咯；
7，用标准的 SQL 语句来建一个数据库试试：
create table test (a varchar(4) not null, b char(2) primary key);
居然可以用，太神奇了，呵呵
8，再插入一条语句试试呢，insert into test(a,b) values(a,11);，嗯，不错，可以用 select 查出来的哦。
9，再插一下：insert into test(a,b) values(a,11);，哦哦，报错了，“错误

23505:语句异常终止,因为它导致“TEST”上所定义的“SQL060710092132480”标识的唯一或主键约束或唯一索引中出现重复键值。” 呵呵。
10，好了，现在可以像你控制的其他数据库一样来控制 Derby 了。

如果上述方法不行，或者你习惯了在 eclipse 之外使用和管理数据库，那么可以很方便的把 Derby“装”在系统里。下面我说一下步骤：
1，把 db-derby-10.1.3.1-bin.tar.gz 解压到 c:\derby,使 lib 和 framework 两个目录在 c:\derby 下边即可
2，设置环境变量

设置一个 c:\derby\framework\embeded\bin 或 c:\derby\framework\NetworkServe\bin 到 Path 中，这样我们就可以直接执行上边介绍的 connect 这样的命令而不用每次钻到那个目录下去执行了
设置 c:\derby\lib\derby.jar;c:\derby\lib\derbytoos.jar 到 CLASSPATH 中,以便让这些 java 编成的命令能够正确执行；
3，打开 cmd
4,敲入 startNetworkServer,可以看到像在 eclisp 中提示的那样启动了 server
5，再打开一个 cmd,敲入 sysinfo,可以看到 derby 的环境信息了，注意在 java user dir 这一项，也许是 java 用户目录上和上边看到的会有所不同哦，这样在 connect jdbc:derby:testdb;create=true;的建的数据库目录就不一样咯。
6，敲入 ij，好了，进入到上边的交互界面，可以建一个数据库看看了。
7，最后在另外一个 cmd 中敲入 stopNetworkServer 就可以关闭数据库了。

如果你两种方法都试过了，那么需要注意的，还是上边步骤 5 的问题，这个问题是你可能随时会启动一个数据库或新建一个数据库，但如果你刚刚使用 derby，你可能还没有察觉。
derby 实际上有两种启动方式，一种是嵌入式的，一种是网络服务器的启动。
1，我们在 eclipse 中右键 start apache derby network server 那个，就是网络服务器的启动方式，在这种方式下可以用另外一台计算机在 ij 中以：
connect jdbc:derby://192.168.0.28:1527/testdb
的方式进行链接。
2，第二种启动方式是在 ij 里边就直接
connect jdbc:derby:testdb
这实际是在连当前配置环境下 java user dir 下那个目录的数据库。

看到这里可能有点糊涂了，这么就会出问题了那？
实际上 derby 的访问更像是一种使用 derby driver 对本地文件系统的访问，不管启动不启动网络服务器，都可以用 driver 访问本地的数据库。这样，在 ij 里边像第二种方式那样建立连接是完全可以的。启动了网络服务器，只不过是能够让其他主机访问罢了。

另外一个问题是，在 eclipse 中和在系统中连接服务器，在 connect 的时候这个当前配置环境是不一样的，eclipse 默认工程所在路径是数据库的所在路径，而

在系统中“装”derby 则会认为 c:\document and settings 下边那个用户目录是数据库的所在路径。
jdk1.7 新特性：

1，switch 中可以使用字串了

```
String s = "test";
switch (s) {
case "test" :
         System.out.println("test");
case "test1" :
        System.out.println("test1");
        break ;
default :
        System.out.println("break");
        break ;
}
```

2.运用 List<String> tempList = new ArrayList<>(); 即泛型实例化类型自动推断
3.语法上支持集合，而不一定是数组

```
final List<Integer> piDigits = [ 1,2,3,4,5,8 ];
```

4.新增一些取环境信息的工具方法

```
File System.getJavaIoTempDir() // IO 临时文件夹

File System.getJavaHomeDir() // JRE 的安装目录

File System.getUserHomeDir() // 当前用户目录

File System.getUserDir() // 启动 java 进程时所在的目录 5
```

5.Boolean 类型反转，空指针安全,参与位运算

```
Boolean Booleans.negate(Boolean booleanObj)

True => False , False => True, Null => Null

boolean Booleans.and(boolean[] array)

boolean Booleans.or(boolean[] array)

boolean Booleans.xor(boolean[] array)

boolean Booleans.and(Boolean[] array)
```

boolean Booleans.or(Boolean[] array)

boolean Booleans.xor(Boolean[] array)

6.两个 char 间的 equals

boolean Character.equalsIgnoreCase(char ch1, char ch2)

7.安全的加减乘除

int Math.safeToInt(long value)

int Math.safeNegate(int value)

long Math.safeSubtract(long value1, int value2)

long Math.safeSubtract(long value1, long value2)

int Math.safeMultiply(int value1, int value2)

long Math.safeMultiply(long value1, int value2)

long Math.safeMultiply(long value1, long value2)

long Math.safeNegate(long value)

int Math.safeAdd(int value1, int value2)

long Math.safeAdd(long value1, int value2)

long Math.safeAdd(long value1, long value2)

int Math.safeSubtract(int value1, int value2)

8.map 集合支持并发请求，且可以写成 Map map = {name:"xxx",age:18};

参考文章：

http://hi.baidu.com/haionlybing/blog/item/57953cedb705fc4378f055b8.html