Java SE 6 特性

Sun Microsystems

升级到 Java SE 6 的十大理由

- 简单的 Web Service 开发
- 脚本语言的集成
- JDBC 4.0 和内嵌 Java DB
- 新的桌面开发 API
- JMX 扩展和一系列管理监控工具
- 访问 Javac 编译器的编程接口
- Annotation 处理器插件接口
- 新的安装下载界面，更好的 Native Look&Feel 支持

升级到 Java SE 6 的十大理由（续）

- 提供 PKI，Java GSS，Kerberos，LDAP 服务支持
- 更好的性能和稳定性

Web Service 开发

- JAX-WS 2.0
  - JAXB 2.0
  - SAAJ 3.0

Web Service 开发

```
package hello;


public class CircleFunctions {

    public double getArea(double radius) {
        return java.lang.Math.PI * (r * r);
     }

    public double getCircumference(double radius) {
        return 2 * java.lang.Math.PI * r;
    }
}
```

Web Service开发

```
package hello;
import javax.jws.WebService;
@WebService
public class CircleFunctions {

    public double getArea(double radius) {
        return java.lang.Math.PI * (r * r);
     }

    public double getCircumference(double radius) {
        return 2 * java.lang.Math.PI * r;
    }
}
```

Web Service 开发

- 生成相关 Stub 等文件
  - wsgen 工具
- 部署创建的 Web Service

```
import javax.xml.ws.Endpoint;

public static void main(String[] args) {


Endpoint.publish(      "http://localhost:8080/WebServiceExam
ple/circlefunctions",
new CircleFunctions());


}
```

脚本语言的集成

- Java 不仅是一种语言，同时是一个平台
  - VM
  - 核心 Java 类库
- 对其它语言的支持 JSR-223
- 插件式的脚本语言引擎
- javax.script 包

脚本集成 API

- 脚本引擎
- ScriptContext, Bindings
- ScriptEngineFactory
- ScriptEngineManager

脚本引擎

- ScriptEngine 接口（必须）
  - 运行脚本文件 eval() 方法
  - 将 Java 对象映射到脚本变量（put() 方法）
- Invocable 接口（可选）
  - 调用脚本方法 / 函数
  - 通过脚本方法实现 Java 接口
- Compilable 接口（可选）
  - 把脚本编译成为中间结果，多次执行

ScriptContext, Bindings

- ScriptContext 是宿主程序界定的脚本环境
- ScriptContext 含有一个或者多个 Binding
- Binding 是 Map<String, Object>的子类型
- 引擎范围的 Binding
  - 脚本变量 =>宿主程序对象
- 全局 Binding
  - 在多个引擎间共享

ScriptEngineFactory

- 与 ScriptEngine 一一对应
- Factory 方法：getScriptEngine()
- 其它方法
  - 脚本文件扩展名
  - 脚本引擎版本
  - ……

ScriptEngineManager

- 实体类
- 脚本引擎查找
  - Jar 文件的 META-INF/services 机制
  - 文件后缀名
  - 脚本语言名称
- 指定类型装载器的查找
- 绑定全局 Binding

例程——HelloWorld

```
import javax.script.*;


public class Main {
  public static void main(String[] args) throws
ScriptException {
    // create a script engine manager
    ScriptEngineManager factory = new ScriptEngineManager();

    // create JavaScript engine
    ScriptEngine engine=factory.getEngineByName("JavaScript");

    // evaluate JavaScript code from String
    engine.eval("print('hello world')");
  }
}
```

例程——从文件运行脚本

```
// create script engine manager
ScriptEngineManager manager = new ScriptEngineManager();

// create JavaScript engine
ScriptEngine engine =
manager.getEngineByName("JavaScript");

// evaluate a file (or any java.io.Reader)
engine.eval(new FileReader("test.js"));
```

例程——脚本变量

```
// create script engine manager
ScriptEngineManager manager = new ScriptEngineManager();

// create JavaScript engine
ScriptEngine engine = manager.getEngineByName("JavaScript");
File f = new File("test.txt");

// expose File object as variable to script
engine.put("file", f);

// evaluate a script string
// script accesses "file" variable and calls method on it
engine.eval("print(file.getAbsolutePath())");
```

例程——调用脚本方法

```
// JavaScript code in a String
String script = "function hello(name) { print('Hello, ' +
name); }";

// evaluate script
engine.eval(script);

// JavaScript engine implements Invocable interface
Invocable inv = (Invocable) engine;

// invoke a global function called “hello”
inv.invoke("hello", new Object[] {"Scripting!!"} );
```

例程—脚本实现 Java 接口

```
// JavaScript code in a String
String script = "function run() { print('run
called\n'); }";

// evaluate script
engine.eval(script);

// JavaScript engine implements Invocable interface
Invocable inv = (Invocable) engine;
Runnable r = inv.getInterface(Runnable.class);
r.run();
```

JDBC 4.0 和内嵌 JavaDB

```
private Connection connect(String user, String passwd)
    throws SQLException
{
  String url = "jdbc:mysql://javadb.sfbay/jplan";
  String driver = "com.mysql.jdbc.Driver";
  try {
    Class.forName(driver);
    return DriverManager.getConnection(url,
                                       user,
                                       passwd);
  } catch (ClassNotFoundException x) {
    throw new SQLException("Can't load Driver " + x);
  }
}
```

JDBC 4.0 和内嵌 JavaDB

```
private Connection connect(String user, String passwd)
    throws SQLException
{
  String url = "jdbc:mysql://javadb.sfbay/jplan";
  String driver = "com.mysql.jdbc.Driver";
  try {
    Class.forName(driver);
    return DriverManager.getConnection(url,
                                       user,
                                       passwd);
  } catch (ClassNotFoundException x) {
    throw new SQLException("Can't load Driver " + x);
  }
}
```

新的桌面 API

- Splash 屏幕
- JTable 排序过滤
- SwingWorker
- Desktop API

Splash 屏幕

- 命令行参数
  - Java -splash:hello.png HelloWorld
- Jar 文件中参数设置
  - SplashScreen-Image: MyImage.png
- 在程序中通过 SplashScreen 类访问图形上下文
  - SplashScreen.getSplashScreen()
  - Graphics2D createGraphics()

JTable 排序过滤

- 根据 TableModel 创建 Sorter，然后将 Sorter 设置到 Table 中

```
//create JTable
JTable table = new Jtable(model);

//create RowSorter
RowSorter<TableModel> sorter =
    new TableRowSorter<TableModel>(model);

//set Sorter
table.setRowSorter(sorter);
```

JTable 排序过滤

- 通过向 TableRowSorter 设置 RowFilter 来实现对于 JTable 和 JList 的过滤
  - andFilter
  - orFilter
  - dateFilter
  - notFilter
  - numberFilter
  - regexFilter

SwingWorker

- Swing 的事件分派线程负责所有事件的调度和响应
- 费时的工作应该放在子线程中完成而不阻塞事件分派
  - doInBackground()：后台处理函数，负责处理较为复杂的计算
  - FirePropertyChange()：通知监听的线程状态变化
  - execute()：从 Worker 线程上启动任务

SwingWorker

```
final JLabel label;
class MeaningOfLifeFinder extends SwingWorker<String,
Object> {
    @Override
    public String doInBackground() {
        return findTheMeaningOfLife();
    }
    @Override
    protected void done() {
        try {
            label.setText(get());
        } catch (Exception ignore) {
        }
    }
}

(new MeaningOfLifeFinder()).execute();
```

Desktop API

- 在 java 程序中启动系统本地的应用程序
  - 浏览器，Email 客户端，打印

```
//get Desktop instance
Desktop desktop = Desktop.getDesktop();
//launch system browser with URI
desktop.browse(new URI("http://www.sun.com"));
//launch mail client with an email address filled
//in
desktop.mail(new
URI("mailto:joey.shen@sun.com"));
```

为应用程序创建 TrayIcon

- TrayIcon 创建时的三个参数

```
TrayIcon icon = new TrayIcon(img, "My App",
popmenu);
```

  - 图标
  - ToolTip 显示
  - 弹出菜单
- 创建一个 ActionListener

```
icon.addActionListener(myListener);
```

- 将创建的 TrayIcon 加入到系统的 Tray 中

```
SystemTray tray = SystemTray.getSystemTray();
tray.add(icon);
```

JMX 扩展和检测管理工具

- MXBean
  - CompositeData 对于复杂属性的自动包装
- Descriptors
  - 关于得到各属性的附加信息
- 各种工具用于监测管理 JVM 和应用程序的运行状况
  - 基于 JMX: Jconsole
  - 基于 JVMTI: jmp,hotspot provider

JConsole

- 通用的 JMX 客户端
- 连接本地和远程的 JVM 实例
- 监视和管理系统信息
  - 内存使用情况
  - 线程运行情况，死锁检测
  - 类的装载和卸载
  - VM 的参数和基本信息
  - 系统中各种 MBean 和 MXBean 提供的属性和操作

Jps, Jinfo, Jstat

- JPS：列出所有 Java 进程和 ID
  - > $jps -l | -m
  - > $jps -v | -V
- Jinfo： 得到或改变某一个 Java 进程运行参数和系统属性（在 Windows 上功能有限）
  - > $jps -flags [jid]
  - > $jps -sysprops [jid]
- Jstat： 统计 JVM 运行过程中的各种信息
  - > $jstat -gcutil [jid]

HPROF – Heap Profiler

- 作为 JVMTI 的一个演示程序，源代码在 <JAVA_HOME>/demo/jvmti/hprof 下
- 观测 Heap 内存分配
  - > $java -agentlib:hprof=heap=sites <Program>
  - > $java -agentlib:hprof=heap=dump <Program>
- 观测 CPU 时间
  - > $java -agentlib:hprof=cpu=samples <Program>
  - > $java -agentlib:hprof=cpu=times <Program>

Jmap

- 从运行的 JVM 进程，或者 Core 文件中提取内存使用的统计信息
  - $jmap -heap <jid>
  - $jmap -histo <jid>
  - $jmap -permstat <jid>
  - $jmap -dump:format=b,file=filename <jid>

Jhat & OQL

- 将 heap 中的对象通过 Web 的形式表现出来并且提供各种查询手段
  - 对于 Heap 上各个对象实例的统计显示
  - 显示活动对象实例到 root 对象的路径
  - 现实 Pending Finalization 对象
- OQL(Object Query Language)
  - 定制查询条件
  - 语法类似于 JavaScript

Dtrace

- hotspot Provider 提供各种观测 JVM 运行的探针
  - Vm 生命周期
  - 线程生命周期
  - 类装载
  - 垃圾回收
  - 方法编译
- hotspot_jni Provider 提供观测 JNI 代码的探针

访问 Javac 编译器的编程接口

```
import javax.compiler.*;

Compiler c = Compiler.newInstance();
c.setOutputDirectory(new File("build/classes"));
c.setSourcePath(new File("src"),
                new File("gensrc"));
c.setOption("Xlint", "all");
c.run(new File("src/Main.java"));
```

Annotation 处理器接口

```
public class ToDoProcessor
      implements AnnotationProcessor
{
  ...
  public void process() {
    AnnotationTypeDeclaration toDoDecl =
atds.iterator().next();
    for(Declaration decl :
      env.getDeclarationsAnnotatedWith(toDoDecl)) {
        System.out.format("%-16s %s%n",
                          decl.toString();
                          decl.getAnnotation(ToDo.Class);
    }
  }
}
```

资源和链接

- http://jdk.dev.java.net
- http://jcp.org
- http://java.sun.com/

Java SE 6 特性

Sun Microsystems