Fiese Sachen mit Objekten

Reflection
XML-RPC
JNI (Java Native Interface)
Profiling

ulrich.schaefer@dfki.de

Reflection

Reflection

- Motivation: get information about objects and their classes at runtime
- Sample application: dynamically create a plug-in class (object) at runtime from configuration file, GUI actions etc.
- Problem: Java compilation requires static class information (class names, method names etc.)
- Solution: `Class, Object, java.lang.reflect`

- Class Object – Methods inherited by all classes:
  - `equals()`, `toString()`, `clone()`, `hashCode()`
  - `notify()`, `notifyAll()`, `wait()`: for Threads
  - `getClass()` returns Class object which gives useful access to classes and their instances at runtime
- Class Class – the door to `java.lang.reflect`
  - Class.forName(String): get Class object from name
  - newInstance(): create instance of Class

Reflection – Example 1: getClass()


```
public class testGetClass {
  public static void main( String[] args ) {
    Class c1 = java.util.Date.class;
    Class c2 = new java.util.Date().getClass();
    Class c3 = null;
    try {
       c3 = Class.forName( "java.util.Date" );
    } catch ( ClassNotFoundException e )
              { e.printStackTrace(); }
  }
}
```

Reflection – Example 2: `Class.forName()`


```
try { Class c = Class.forName("Klasse");
      Object o = c.newInstance();
      ((Klasse)o).hello();
    } catch (ClassNotFoundException e) {
      System.out.println("Kann Klasse nicht finden");
    } catch (ClassCastException e) {
      System.out.println("Kann nicht Klasse casten");
    } catch (InstantiationException e) {
      System.out.println("Kann nicht instanziieren");
    } catch (IllegalAccessException e) {
      System.out.println("Kann nicht auf Methode zugreifen");
 }
```

Reflection – Example 3: Class.forName()


```
static void checkClassType( Class c )
 {
  if (c.isArray())
    System.out.println(c.getName() + " is an array.");
  else if (c.isPrimitive())
    System.out.println(c + " is a primitive datatype.");
  else if (c.isInterface())
    System.out.println(c.getName() + " is an
  interface.");
  else
    System.out.println(c.getName() + " is a class.");
 }
```

Reflection – Example 3: Class.forName()


```
class CheckClassType {
  public static void main( String[] args ) {
    Class observer = java.util.Observer.class;
    Class observable = java.util.Observable.class;
    Class array = (new int[2][3][4]).getClass();
    Class primitive = Integer.TYPE;
    checkClassType(observer);
    checkClassType(observable);
    checkClassType(array);
    checkClassType(primitive);
  }
```

```
Output:
java.util.Observer is an interface.
java.util.Observable is a class.
[[[I is an array.
int is a primitive data type.
```

Reflection with `java.lang.reflect`

- Object inspection: find out about public methods, fields etc. of other
  - getMethod(), getMethods() -> java.lang.reflect.Method
    - getName(), getParameterTypes(), getModifiers(), isPublic(), isPrivate(), isStatic(), isAbstract(), ...
    - invoke(Object obj, Object[] args)
  - getField(), getFields()
- Example: XML-RPC: call public Java methods via simple XML-based socket protocol

Reflection – A warning

- Warning: using reflection is normally slower than using the corresponding fixed, compilable code
- Hence, it is recommended to use reflection only where appropriate (e.g. plug-in architectures)

XML-RPC

XML-RPC

- RPC = Remote Procedure Call
- Based on standard HTTP for remote method invocation
- Uses XML to wrap data in method calls, method parameters and return values
- Fixed set of simple data types supported: string, bool, int, double, date/time, base64-encoded binary, array, struct

- Sample call via HTTP:

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param><value><int>41</int></value></param>
  </params>
</methodCall>
```

- Advantages:
  - Programming-language-independent
  - Small, lightweight
  - Implementations exist for many programming languages and platforms/operating systems
  - easy-to-implement simple web services

- Disadvantages:
  - Encoding issues (only ASCII guaranteed to work properly), especially between different implementations
  - No standardized object transport (-> SOAP), only simple data types

- Implementation easy in Java (but use existing Apache library!)
  - Based on simple HttpServer class in Java
  - Plus XML handling for data transport
  - Plus reflection: simply export all public methods of a specified set of objects using reflection

Sample XML-RPC Server


```
import org.apache.xmlrpc.server.PropertyHandlerMapping;
import org.apache.xmlrpc.server.XmlRpcServer;
import org.apache.xmlrpc.webserver.WebServer;

public class XmlRpcServerTest {

  private static final int port = 16384;

  public static void main(String[] args) throws Exception {
    WebServer webServer = new WebServer(port);
    XmlRpcServer xmlRpcServer = webServer.getXmlRpcServer();

    PropertyHandlerMapping phm = new PropertyHandlerMapping();
    phm.addHandler("servedObject1", PublishMe.class);
    xmlRpcServer.setHandlerMapping(phm);

    webServer.start();
  }
}
```

```
public class PublishMe
{
    public String sayHello (String toWhom) {
      return "Hello " + toWhom;
    }
}
```

Sample XML-RPC Client


```
import org.apache.xmlrpc.XmlRpcException;
import org.apache.xmlrpc.client.XmlRpcClient;
import org.apache.xmlrpc.client.XmlRpcClientConfigImpl;
import java.net.URL;
import java.net.MalformedURLException;

public class XmlRpcClientTest {

  public static void main (String[] args) {
    try {
   XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
   config.setServerURL(new URL("http://127.0.0.1:16384/xmlrpc"));
   XmlRpcClient client = new XmlRpcClient();
   client.setConfig(config);
   Object[] params = new Object[]{new String(args[0])};
   String result = (String) client.execute("servedObject1.sayHello", params);
   System.out.println (result);
    } catch (MalformedURLException mue) {
    System.err.println(mue.getLocalizedMessage());
    } catch (XmlRpcException xre) {
        System.err.println(xre.getLocalizedMessage());
    }
  }
}
```

Java Native Interface (JNI)

From reflection to JNI

- Class and reflection facilities are managed by the JVM (Java virtual machine) which is a C-implemented program available for the Java-supported platforms to load and execute Java programs that are compiled to class files.
- As the JVM methods are C functions, they can be called from other C programs or libraries and vice versa!
- The interface is called JNI = Java Native Interface

JNI – Java Native Interface

- integration of existing native (C/C++) libraries in Java programs (method calls)
- also vice versa: call Java (JVM) from C/C++
- platform-dependent and JVM-dependent !!
- avoid JNI whenever possible, e.g. through networking (Corba, SOAP, XML-RPC) or – on Unix only: process pipes

JNI – including a native library in Java

- in Java class MyClass calling a native library, add

```
System.loadLibrary("name");   // without .dll or .so
```

- declare external methods as native, e.g.,

```
public static native int getLength(String s);
```

- generate C/C++ header file using

```
javah -jni -o cheader.h MyClass
```

- implement C/C++ code that includes header file
- compile it and run MyClass.class with name.dll in path defined with `-Djava.library.path=path`

JNI header file generated with javah


```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class MyClass */

#ifndef _Included_MyClass
#define _Included_MyClass
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     MyClass
 * Method:    getLength
 * Signature: (Ljava/lang/String;)I
 */
JNIEXPORT jint JNICALL Java_MyClass_getLength
  (JNIEnv *, jclass, jstring);

#ifdef __cplusplus
}
#endif
#endif
```

full classname with package

Runtime Profiling with Sun JDK

Runtime Profiling with Sun JDK

- measure time and memory consumption
- detect flaws in code, optimization
- profiling options are passed to JVM (java executable) as parameters:
  - **cpu**: samples, times, old
  - **heap**: dump, sites, all
  - **file**: output file
  - **depth**: max size of stack trace

Runtime Profiling with Sun JDK: Times


```
java -Xint -Xrunhprof:cpu=samples,depth=10 Classname

CPU SAMPLES BEGIN (total = 508) Fri Feb  3 18:21:28 2006
rank   self  accum   count trace method
   1 17.32% 17.32%      88   136 java.lang.StrictMath.floor
   2  9.06% 26.38%      46    67 org.apache.xml.utils.XMLChar.<clinit>
   3  4.72% 31.10%      24    68 org.apache.xml.utils.XMLChar.<clinit>
   4  4.33% 35.43%      22   139 org.apache.xpath.objects.XString.substring
   5  3.35% 38.78%      17   137 org.apache.xpath.functions.FuncStringLength.execute
   6  3.35% 42.13%      17   162 java.lang.Object.clone
   7  2.56% 44.69%      13   147 java.lang.StrictMath.floor
   8  2.56% 47.24%      13   143 java.lang.String.substring
   9  2.36% 49.61%      12   149 org.apache.xpath.objects.XMLStringFactoryImpl.newstr
  10  1.57% 51.18%       8   155 java.lang.StrictMath.floor
  11  0.98% 52.17%       5   182 org.apache.xalan.transformer.TransformerImpl.transform
  12  0.98% 53.15%       5   160 java.lang.System.arraycopy
```

Runtime Profiling with Sun JDK: Memory


```
java -Xint -Xrunhprof:heap=sites,depth=10 Classname

SITES BEGIN (ordered by live bytes) Fri Feb  3 18:31:23 2006
          percent          live       alloc'ed  stack class
 rank   self  accum     bytes objs   bytes objs trace name
    1 27.15% 27.15%    640224   78  640224   78  3801 [I
    2  6.83% 33.98%    160992   78  160992   78  3833 [I
    3  2.78% 36.76%     65552    1   65552    1  1971 [B
    4  2.35% 39.11%     55400  376   57896  422     1 [C
    5  1.95% 41.07%     46080  218   46144  221     1 [B
    6  1.56% 42.63%     36872    1   36872    1  4017 [C
    7  1.39% 44.02%     32784    1   32784    1  3162 org.apache.xpath.objects.XObject
    8  1.39% 45.41%     32784    1   32784    1  4049 org.apache.xpath.objects.XObject
    9  1.36% 46.77%     32032   77   32032   77  3881 [I
   10  1.13% 47.90%     26544 1106  638184 26591  3864 java.lang.String
   11  1.07% 48.97%     25312    1   47280    6   623 [C
   12  0.91% 49.88%     21536 1346  486256 30391  3863 org.apache.xpath.objects.XString
```

Profiling with GUI

- JProfiler (commercial product, 10 days trial version)
- Netbeans Profiler (part of Netbeans IDE): http://netbeans.org/features/java/profiler.html
- List of other profilers for Java: http://java-source.net/open-source/profilers

Literature

- XML-RPC: http://www.xmlrpc.com/, http://ws.apache.org/xmlrpc/
- JNI – Chapter 26 in 'Java ist auch eine Insel': http://openbook.galileocomputing.de/javainsel8/
- JNI – Sun Tutorial: http://java.sun.com/developer/onlineTraining/Programming/JDCBook/jni.html
- Reflection – Sun Tutorial: http://java.sun.com/docs/books/tutorial/reflect/index.html
- Reflection, Profiling: Java-Handbuch & Java ist auch eine Insel