Önelemzés és a JavaBean komponensmodell

Kozsik Tamás
kto@elte.hu
http://kto.web.elte.hu/

Eötvös Loránd Tudományegyetem
Programozási Nyelvek és Fordítóprogramok Tanszék

2008.

Önelemzés

- Introspection, reflection
- Futás közben vizsgáljuk a kódot
- Aktiválhatjuk
- Sok nyelvben: akár kiegészíthetjük, módosíthatjuk

Dinamikus kötés

- Dynamic/late binding, dynamic lookup/dispatching
- Az OOP egyik alapgondolata
- Futási időben választjuk ki a végrehajtandó kódot
- Felüldefiniált műveletek közül
- Öröklődés mentén
- Nagyfokú rugalmasság, továbbfejleszthetőség, adaptálhatóság, újrafelhasználhatóság
- *Ez nem önelemzés*

Önelemzés Javában

- Egy típusleíró (.class) fájl információi alapján (JVM bájtkód)
  - osztályok
  - interfészek
  - annotációk
  - felsorolási típusok
  - *tömbök nem!*
  - *primitív típusok nem!*
- SDK eszközökkel elemezhető (pl. `javap`)
- Futás közben: API van hozzá
  `java.lang.reflect`

Példa

```
import java.lang.reflect.*;
class Example {
  public static void main(String[] args) throws Exception {
      Class c = Class.forName(args[0]);
      Class[] formalArgs = (args.length > 2)
                           ? new Class[] {String.class}
                           : new Class[] {};
      Object o = c.newInstance();
      Method m = c.getMethod(args[1],formalArgs);
      Object[] actualArgs = (args.length > 2)
                            ? new Object[] {args[2]}
                            : new Object[] {};
      System.out.println( m.invoke(o,actualArgs) );
  }
}
```

Típusok a JVM-ben

- Beépített típusok
  - Primitív típusok
  - Tömb típusok
- Betöltött típusok
  - Osztálybetöltés, bájtkód-ellenőrzés, dinamikus szerkesztés
  - Belső reprezentáció: `java.lang.Class<T>` objektumai
  - Tükrözés – reflection
  - Koncepcionálisan
    - statikus tagok
    - statikus metódusok szinkronizációja
  - Dinamikus típus lekérdezése
    `public final Class<?> Object.getClass()`
    - `instanceof`
    - dinamikus kötés
  - Önelemzés

Class objektumok

- Az önelemzés első lépése
- Minden típushoz tartozik
  - Primitív típusokhoz
  - Referencia típusokhoz
    - Tömbökhöz
    - Fájlból betöltött típusokhoz:
      - Egyéb (absztrakt és konkrét) osztályok
      - Interfészek
      - Felsorolási típusok
      - Annotáció típusok
  - Nem-típushoz: `void`
- Többféleképpen is szert lehet tenni rájuk

getClass

- Ha van egy objektumunk
- `public final Class<?> Object.getClass()`
- Például:
  - `"Hello".getClass()` → `String`
  - `System.out.getClass()` → `java.io.PrintStream`
  - Legyen `enum E {A,B,C,D}`. Ekkor:
    `A.getClass()` → `E`
  - `(new byte[1024]).getClass()` → `byte[]`
  - `(new HashSet<String>()).getClass()` → `HashSet`
  - `Serializable s = new Vector();`
    `... s.getClass() ...` → `Vector`

.class

Nem kell objektum: `.class`

- Például
  - `String.class`
  - `java.util.List.class`
  - `Class.class`
  - `int.class`
  - `double[][].class`
  - `void.class`
- A forráskódban benne van, a fordításhoz kell a típus

Csomagolóosztályok segítségével

Primitív típusokhoz

- `java.lang.Double.TYPE`
- `java.lang.Boolean.TYPE`
- ...
- `java.lang.Void.TYPE`

forName

Típusbetöltés paraméterezhetően, teljesen dinamikusan

`public static Class<?> Class.forName(String)`

- Például: `Class c = Class.forName(args[0]);`
- Fordításhoz nem kell a típus
- Futtatáskor a `classpath`-ban kell lennie
- Megadható
  - típusnév: `Class.forName("java.util.List")`
  - szignatúra: `Class.forName("[Ljava.lang.String")`

Betöltés során adódó hibák

- Ha nem található a `classpath` alapján a `class` fájl `java.lang.ClassNotFoundException`
- Szerkesztési hibák `java.lang.LinkageError`

Generikus Class

- A Java 5-től kezdve a `Class` osztály generikus
- Az `String` típus típusleíró objektuma például `Class<String>` típusú
- Csak fordítás során „létezik” a típusparaméter
- Típushelyettesítő

```
public final Class<?> Object.getClass()
public static Class<?> Class.forName(String)
```

Metódusok, amelyek Class-t adnak

- `Class.getSuperclass()`
- `Class.getInterfaces()`
- `Class.getClasses()`
- `Class.getDeclaredClasses()`
- `getDeclaringClass()`
  - `Class`
  - `java.lang.reflect.Field`
  - `java.lang.reflect.Method`
  - `java.lang.reflect.Constructor`
- `Class.getEnclosingClass()`
- `Class.getComponentType()`

Egy Class tulajdonságai

- Módosítók
  - `public`, `protected`, `private`
  - `abstract`
  - `static`
  - `final`
  - `strictfp`
  - annotációk
- Típusparaméterek
- Szülőosztály, kiterjesztett interfészek, generikus változatok
- Deklaráló/befoglaló `Class`/metódus/konstruktor
- Csomag
- Tömb esetén: komponens típus
- Konstruktorok
- Tagok

Módosítók lekérdezése

- Az annotációk erre a célra való műveletekkel
- A nem-annotáció módosítószók maszkolással
  - Minden módosítónak megfelel egy `int` konstans
  - Lekérdezés $\longrightarrow$ `int`
    `public int getModifiers()`
  - Az `int`-ből maszkolás
  - Egyszerűbben: speciális műveletekkel, pl.
    `public static boolean isStatic(int mod)`

Publikusság

- Publikus információk
  - `getMethods()`, `getFields()`, ...
  - Örököltek is
- Minden információ
  - `getDeclaredMethods()`, `getDeclaredFields()`, ...
  - Örököltek nem

Listázás és kikeresés

- Bizonyos információk alapján kikeres
  - `getMethod(String,Class<?>...)`
  - `getField(String)`
  - stb.
- Kilistázza az összeset
  - `getMethods()`
  - `getFields()`
  - stb.

Objektumok létrehozása

- `Class.newInstance`
  - Csak paraméter nélküli konstruktoron keresztül megy
  - A kivételellenőrzési mechanizmusnak keresztbevág (propagálja a kivételeket)
  - Probléma a hozzáféréssel
- `Constructor.newInstance`
  - Újabb lehetőség
  - Paraméterezhető – nem csak paraméter nélküli konstruktort lehet így hívni
  - A kivételeket csomagolja (`InvocationTargetException`)
  - `AccessibleObject` alapon kérhető privát hozzáférés

Mit lehet még csinálni a Class segítségével?

- típuskényszerítés: `public T cast(Object obj)`
- dinamikus típusellenőrzés: `public boolean isInstance(Object obj)`
- lekérdezések (`isArray()`, `isPrimitive()`, `isLocalClass`)
- név, egyszerű név kanonikus név
- osztálybetöltő

Típusok megadása szövegesen

- Primitív típusok és `void` – mint a kulcsszavak
- Referenciatípusok
  - tömbök: kódolva (Z,C,B,S,I,J,F,D,L)
    - `[I`
    - `[[C`
    - `[Ljava.lang.String;`
  - egyebek: minősített teljes névvel

Tagok és konstruktorok

- `java.lang.reflect.Member` interfész
  - `java.lang.reflect.Field`
  - `java.lang.reflect.Method`
  - `java.lang.reflect.Constructor`
- A `Class`-ból kiindulva megszerezhetők

Attribútumok

- Fajtái
  - Példányattribútum
  - Osztályszintű attribútum
  - Felsorolási típus típusértéke
- Alkotóelemei
  - Név
  - Típus
  - Módosítók, annotációk
- Szintetizált attribútumok

Érték lekérdezése és beállítása

- `Object get(Object)`
- `boolean getBoolean(Object)`
- `...`
- `void set(Object, Object)`
- `void setBoolean(Object, boolean)`
- `...`

Példa

```
import java.lang.reflect.*;

class Point { public int x, y; }

class SetField {
  public static void main( String[] args )
  throws Exception {
       Class<Point> c = Point.class;
       Object o = c.newInstance();
       Field x = c.getField("x");
       x.setInt(o,12);
       Point p = c.cast(o);
       System.out.println(p.x);
  }
}
```

Auto-(un)boxing

- Csak fordításkor
- Önelemzés során nem történik meg
- Ha `Integer` lenne `Point.x` típusa, az előző példa kivételt váltana ki (`IllegalArgumentException`)

Kompatibilitásvizsgálat

```
Integer.class.isAssignableFrom(int.class) == false
int.class.isAssignableFrom(Integer.class) == false
```

Metódusok

- Lekérdezhetők
- Végrehajthatók
  - Példánymetódusok objektumon
  - Statikus metódusok `null`-on
- Alkotóelemeik
  - Módosítók, annotációk
  - Kiváltható kivételek
  - Paramétertípusok
  - Visszatérési érték típusa
  - Típusparaméterek (ha sablon)
  - Alapértelmezett érték (ha annotációelem)
- Tulajdonságok
  - `isBridge()`
  - `isSynthetic()`
  - `isVarArgs()`

Típustörlés

```
import java.lang.reflect.Method;

public class Trouble<T>  {

  public void lookup(T t) {}
  public void find(Integer i) {}

  public static void main(String... args)
  throws Exception {
      Trouble<?> obj = new Trouble<Integer>();
      Class<?> c = obj.getClass();
      String mName = args[0];
      Class cArg = Class.forName(args[1]);
      Method m = c.getMethod(mName, cArg);
      System.out.println(m.toGenericString());
  }

}
```

Konstruktorok

- Lekérdezhetők
- Példányosításhoz használhatók
- Alkotóelemeik
  - Módosítók, annotációk
  - Kiváltható kivételek
  - Paramétertípusok
  - Típusparaméterek (ha sablon)
- Tulajdonságok
  - `isSynthetic()`
  - `isVarArgs()`

Tömbök

- `Class.isArray()`, `Class.getComponentType()`
- `java.lang.reflect.Array` osztály
  - `newInstance(Class<?>,int...)`
  - `getLength(Object)`
  - `get(Object,int)`, `getBoolean(Object,int)`, ...
  - `set(Object,int,Object)`, `setBoolean(Object,int,boolean)`, ...

Felsorolási típusok

- Ugyanúgy, mint az osztályok
- Típusértékek: attribútumok
- Vonatkozó műveletek
  - `Class.isEnum()`
  - `Class.getEnumConstants()`,
  - `java.lang.reflect.Field.isEnumConstant()`

Önelemzés tulajdonságai

- Pozitívum
  - Rugalmasság
  - Újrafelhasználhatóság
  - Továbbfejleszthetőség, adaptálhatóság
- Negatívum
  - Költséges (végrehajtási idő)
  - Futási idejű hibák
    - nem talál valamit
    - nincs joga valamihez (security manager)
  - Biztonsági problémák
    - privát tagokhoz hozzáférés

Mire használjuk

- Előre nem ismert kódot manipuláló kód írásánál
  - Keretrendszerek
  - JavaBeans
  - Enterprise JavaBeans
  - Vizuális szerkesztők
  - Nyomkövetők, log-rendszerek, profilozók
  - Dinamikus kódtranszformáció
- Generatív programozás
- Kóddal paraméterezhető kód

JDBC-s példa

- Adatbázist használó program
- Az adatbáziskezelő lelkét ismerő komponens
- Függetlenek egymástól

JDBC-driver betöltése a kódba

```
Class.forName(args[0]);
```

JDBC-driver nevének átadása

```
$ javac MyApp.java
$ java MyApp com.mysql.jdbc.Driver
```

JavaBeans

- Programozási konvencióknak megfelelő osztályok
  - setter/getter alapján „property”-k
  - tulajdonságszerkesztők, -konfigurálók
  - eseményvezérelt viselkedés, figyelők (listener)
- Grafikus felület, szerializálhatóság
- Vizuális szerkesztőkben
- JSP

Osztálybetöltő

- A dinamikus szerkesztés első lépése a betöltés
- Egy virtuális gépben több betöltő is lehet
- Konfigurálhatók, programozhatók
- Egy betöltött kód tulajdonságai függnek a használt betöltőtől (pl. security)