Javassist

Java-API zum Ändern von Java Bytecode

Markus Wichmann — June 2003

Allgemeines

- Javassist ist ein Open-Source-Projekt der JBoss Group
- Wurzeln reichen bis etwa 1997 zurück
- ergänzt das Java Reflection API um einige Funktionalität
- lässt sich für generative Verfahren einsetzen
- ist scheinbar kaum bekannt

Java Reflection API: Vorhandene Features

- Introspektion eines Programms zur Laufzeit
  - Welche Methoden hat eine Klasse?
  - Welche Konstruktoren hat eine Klasse?
  - Welche Attribute hat eine Klasse?
  - Von welchem Typ sind Attribute der Klasse?
  - Welche Parameter werden von Methoden/Konstruktoren erwartet?
  - Welche Interfaces implementiert eine Klasse?
  - Welche Superklasse hat eine Klasse?
- Erzeugung von Objekten
- Setzen der Werte von Feldern
- u.a.

Java Reflection API: Fehlende Features

- Ändern des Verhalten von Klassen zur Laufzeit
- Komfortables Hinzufügen von Klassen, Methoden und Attributen zur Laufzeit
- Ändern der Klassenhierarchie zur Laufzeit

Erweitern des Reflection APIs: Vorschlag 1

- Behavioural Reflection
  - Compiler oder Laufzeit-System einer derartigen Erweiterung fügen Hooks ein → Aufruf ausgewählter Methoden wird abgefangen, anstelle dessen wird eigene, neue, Methode aufgerufen
  - Vorteil: Geschwindigkeit leidet kaum, wenn dieses Verfahren zur Compile-Zeit eingesetzt wird
  - Nachteile:
    - Nur Verhalten lässt sich Ändern, nicht aber Struktur
    - Spezielle Compiler bzw. Laufzeit-Systeme notwendig

Erweitern des Reflection APIs: Vorschlag 2

- Structural Reflection
  - Definition (»Unterschieben«) von Klassen, Attributen und Methoden zur Laufzeit durch direkte Eingriffe ins Laufzeit-System
  - Nachteil: Laufzeit-System muss Eingriff unterstützen
    - Laufzeit-System plattform-spezifisch → Plattformunabhängigkeit nur schwer garantierbar
    - Laufzeit-System kann schlechter optimieren (Optimierung statischer Informationen?)
  - Wird eingesetzt von Smalltalk und CLOS (Common Lisp Object System)

Erweitern des Reflection APIs: Javassist

- Die JVM soll außerdem nicht verändert werden
  - Plattformunabhängig soll erhalten bleiben
- Es wird eine spezielle Structural Reflection eingesetzt
  - Bytecode wird im .class-File direkt verändert
  - Anschließend Neuladen des .class-Files
  - JVM bleibt unverändert
  - Eingriff ins Laufzeit-System nicht erforderlich
- »Structural Reflection at load time«

Grundlagen Vorgang Javassist

1. Reification: Anlegen eines CtClass Objekts (Compile time Class, repräsentiert den Bytecode)

```
CtClass neueKlasse =
pool.makeClass("Company");
```

```
neueKlasse.addInterface(pool.get
("java.io.Serializable"));
```

2. Modification: Introspektion und Ändern der Klasse

Weitere Methoden:

```
add{Method|Constructor|Field|...}
setName(...), setSuperclass(...),
setModifiers(...)
```

3. Translation: Schreiben des geänderten Bytecodes

```
neueKlasse.writeFile();
```

4. Reflection: Laden des neuen Bytecodes in die JVM

```
CtClass company =
pool.get("Company");
```

Möglichkeiten bei der Modifikation (Auszug)

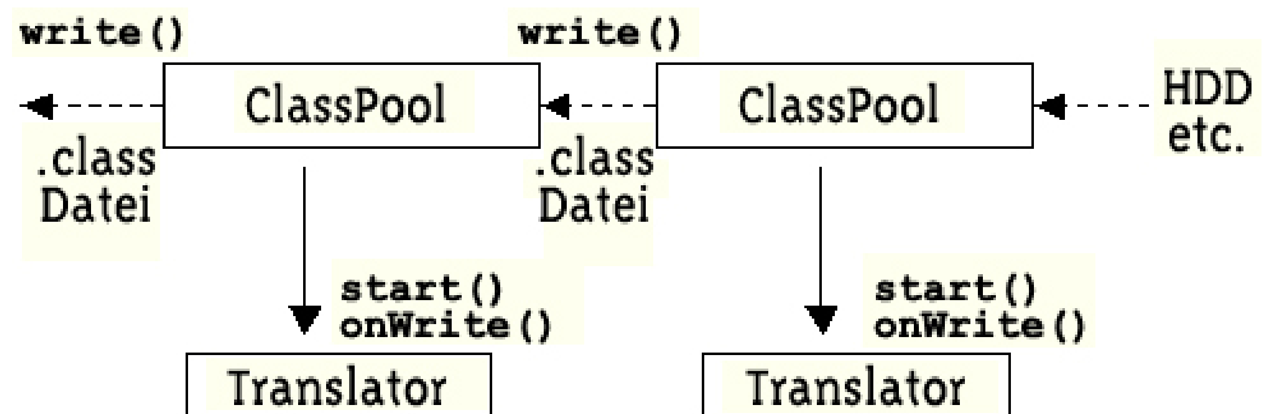
- Methoden für ein Klassen (CtClass Objekte):
  - `bePublic(), beAbstract(), notFinal(), setName(String name), setSuperclass(CtClass c), setInterface(CtClass[] i), addConstructor(…), addDefaultConstructor(), addAbstractMethod(…), addMethod(…), addWrapper(…), addField(…)`
- Methoden für Methoden (CtMethod Objekte):
  - `bePublic(), setBody(…), setWrapper()`
- Methoden für Felder (CtField Objekte):
  - `bePublic()`

Einsatzgebiete

- Unterstützung der Erweiterung von Software beim Einsatz von Software Design Patterns
  - Strategy Pattern (jede Klasse, deren Namen auf `Strat` endet, soll...)
  - Command Pattern (jede Command-Klasse soll Methode `do()` impl.)
- Beim Einsatz verteilter Systeme
  - strikte Anpassung von Client auf Server nicht immer gegeben
  - Client schickt „alte" Daten an Server
  - Server kann diese Anpassen
- Modifikation von Software, deren Sourcen nicht vorliegen ☺

Vorgang reloaded: Automatische Translation

- Translation kann beim Laden von Klassen automatisch erfolgen
- Implementieren des Interfaces Translator
  - `onWrite()`
    - wird gerufen, wenn Client mit der Modifikation fertig ist und `writeFile()` aufruft
    - definiert, was mit der Klasse geschehen soll, wenn der Client sie benutzen möchte
- Registrieren des Translators beim ClassPool
- Translatoren lassen sich hintereinander schalten



Auto-Translation: Beispiel Einsatzgebiete

- Automatische Anpassung von Legacy-Code an neue Interfaces

```
Class Calendar implements Writable {
  public void write(PrintStream s) {…}
}
```

```
[Translator W-to-P] // Pseudo-Code
  rename interface Writable Printable;
  add public void print() { write(System.out); }
```

```
Class Calendar implements Printable {
  public void write(PrintStream s) {…}
  public void print() { write(System.out); }
}
```

Kurzschreibweisen

- Eine Kurzschreibweise ermöglicht effizientere Translation
- Namen von Objekten, Klassen, Parametern müssen dadurch nicht explizit angegeben werden
- Beispiele bei Modifikation einer Methode:
  - $0, $1, $2,... entspricht den momentan verfügbaren Parametern
  - $$ entspricht allen Parametern
  - $class entspricht der momentan editierten Klasse

Kurzschreibweisen: Beispiel

- `$e` entspricht beim Arbeiten mit Exceptions immer der momentan verwendeten Exception

```
CtMethod m = /* hier eine Methode auswählen */;
CtClass etype = pool.get("java.io.IOException");
m.addCatch("{ System.out.println($e); throw $e; }", etype);
```

führt zu:

```
try {
    // ursprünglicher Methoden Rumpf der Methode m
} catch (java.io.IOException e) {
    System.out.println(e);
    throw e;
}
```

Javassist vs. andere Technologien

Punkt im L.-Zyklus	Technologie	Beschreibung	Fähigkeiten	Einschränkungen
Source Code	Reflective Java	Preprocessor	Abfangen von Methodenaufrufen	Sourcecode erforderlich
Compile Time	OpenJava	Compile Time Metaobject Protokoll (?)	Abfangen von Methodenaufrufen Erweitert die Sprachsyntax	Sourcecode erforderlich
Byte Code	Javassist Kava Dalang	Bytecode-Umschreiben zur Ladezeit	Kein Sourcecode erforderlich	Requires offline preprocessing
Runtime	MetaXa Java Reflection API	Reflective JVMs	Abfangen von Methodenaufrufen Introspektion zur Laufzeit	proprietäre JVM Nur Introspektion

Ressourcen

- Javassist Homepage:
  http://www.jboss.org/index.html?module=html&op=userdisplay&id=developers/projects/javassist
- Ideengeber für diese Präsentation:
  http://salmosa.kaist.ac.kr/LAB/RESEARCH/SEMINAR/GENERAL/year2002/material/20021105.ppt
- A Bytecode Translator for Distributed Execution of "Legacy" Java Software:
  http://www.csg.is.titech.ac.jp/mich/pub/200106_ecoop2001.pdf
- Load-Time Structural Reflection in Java:
  http://www.csg.is.titech.ac.jp/chiba/pub/chiba-ecoop00.pdf

Vielen Dank für Eure Aufmerksamkeit!