Verteilte Systeme – Übung

Java Reflection API

Sommersemester 2021

Michael Eischer, Laura Lawniczak, Tobias Distler

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.cs.fau.de

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Überblick

Java Reflection API

Java Reflection API

- Bietet die Möglichkeit, das Laufzeitverhalten von Applikationen zu analysieren und es gegebenenfalls sogar zu beeinflussen
- Tutorial: `http://docs.oracle.com/javase/tutorial/reflect/index.html`

> "[...] This is a relatively advanced feature and **should be used only by developers** who have a **strong grasp of the fundamentals of the language**. [...]"

- Ermöglicht zur Laufzeit
  - Analyse von Attributen, Konstruktoren, Methoden, ...
  - Erzeugung neuer Objekte
  - Modifikation bestehender Objekte
  - Dynamische Methodenaufrufe
  - ...

Class-Objekte

- Zentrale Klasse: `java.lang.Class`
  - Pro Objekttyp existiert ein unveränderliches `Class`-Objekt
  - Beispiel

```
String x = "x";
String y = "y";
boolean b = (x.getClass() == y.getClass());  // -> b == true
```

- Zugriff auf `Class`-Objekte
  - Allgemein: Per `class`-Attribut [Funktioniert auch bei primitiven Datentypen.]

```
Class c = <Klassenname>.class;
```

  - Über existierendes Objekt mit `getClass()`

```
Class c = <Objekt>.getClass();
```

  - Über Klassenname mit `forName()`

```
Class c = Class.forName(<Klassenname>);
```

- Analyse einer Klasse

```
public class Class<T> {
    public Class<? super T> getSuperclass();
    public Class<?>[] getInterfaces();
    public Method[] getMethods();
    [...]
}
```

`getSuperClass()` Zugriff auf `Class`-Objekt der Oberklasse

`getInterfaces()` Zugriff auf `Class`-Objekte der von dieser Klasse direkt implementierten Schnittstellen

`getMethods()` Rückgabe der öffentlichen Methoden dieser Klasse

- Beispiel: Ausgabe aller Methoden der implementierten Schnittstellen

```
Class<?> c = <Objekt>.getClass();
do {
    for(Class<?> intf: c.getInterfaces()) {
       for(Method m: intf.getMethods()) System.out.println(m);
    }
} while((c = c.getSuperclass()) != null);
```

- Analyse einer Methode: `java.lang.reflect.Method`

```
public class Method {
    public String getName();
    public Class<?>[] getParameterTypes();
    public Class<?> getReturnType();
    public Class<?>[] getExceptionTypes();
    public String toGenericString();
    [...]
}
```

`getName()`	Rückgabe des Methodennamens
`getParameterTypes()`	Zugriff auf `Class`-Objekte der Parameter
`getReturnType()`	Zugriff auf `Class`-Objekt des Rückgabewerts
`getExceptionTypes()`	Zugriff auf `Class`-Objekte der Exceptions
`toGenericString()`	Rückgabe der kompletten Methodensignatur

- Dynamischer Aufruf einer Methode

```
public class Method {
    public Object invoke(Object obj, Object... args);
}
```

- Beispiel: `registerAuction()`-Methodenaufruf am `VSAuctionService` aus Übungsaufgabe 1
  - Gewöhnlicher `registerAuction()`-Methodenaufruf

```
VSAuctionService service  = new VSAuctionServiceImpl();
service.registerAuction(new VSAuction("Testauktion", 1), 42, null);
```

  - `registerAuction()`-Methodenaufruf mit Java Reflection API

```
VSAuctionService service  = new VSAuctionServiceImpl();

// Holen des Methoden-Objekts fuer registerAuction()
Class<?> c = service.getClass();
Class<?>[] paramTypes = new Class<?>[]{ VSAuction.class, int.class, VSAuctionEventHandler.class };
Method m = c.getMethod("registerAuction", paramTypes);

// Zusammenstellung der Parameter und Aufruf der Methode
Object[] params = new Object[]{ new VSAuction("Testauktion", 1), 42, null };
m.invoke(service, params);
```

[Wie das Beispiel verdeutlicht, gibt es keinen Grund, für den Aufruf einer Methode die Java Reflection API zu verwenden, solange alles Mögliche unternommen wurde, dies zu verhindern.]