План лекции

- Понятие рефлексии
- Участники механизма рефлексии
- Получение информации о классе

Понятие рефлексии

- Рефлексия (от лат. reflexio – обращение назад) – обращение субъекта на себя самого, на свое знание или на свое собственное состояние

- Рефлексия в Java – возможность программы анализировать саму себя, взаимодействуя с виртуальной машиной Java (JVM)

Возможности механизма рефлексии

- Загрузка типов во время исполнения программы
- Исследование структуры типов и их элементов
- Создание экземпляров классов
- Вызов методов
- Загрузка классов из набора байтов

Участники механизма рефлексии

- Класс `java.lang.Class`
  - Класс является метаклассом по отношению к другим типам
  - Экземпляры класса `Class` описывают классы и интерфейсы выполняемого приложения
  - Методы класса `Class` позволяют исследовать содержимое описываемого класса и его свойства
- Класс `java.lang.ClassLoader`
  - Реализует механизмы загрузки классов

Участники механизма рефлексии

- Пакет `java.lang.reflect`
  - Содержит ряд дополнительных и вспомогательных классов
  - `Field`

    Описывает поле объекта
  - `Method`

    Описывает метод объекта
  - `Constructor`

    Описывает конструктор объекта
  - `Modifier`

    Инкапсулирует работу с модификаторами
  - `Array`

    Инкапсулирует работу с массивами

Получение представления класса

- Метод `Class Object.getClass()`
  Возвращает ссылку на представление класса, экземпляром которого является объект
- Псевдополе `Object.class`
  Ссылка на представление указанного класса
- Метод `static Class Class.forName(...)`
  Возвращает ссылку на представление класса, полное имя которого указывается параметром типа `String`

Получение представления класса

- Метод `Class[] Class.getClasses()`
  Возвращает ссылку на массив ссылок на объекты `Class` вложенных типов
- Метод `Class Class.getDeclaringClass()`
  Для вложенных типов возвращает ссылку на объект Class внешнего типа
- Метод `Class[] Class.getInterfaces()`
  Возвращает ссылки на описания интерфейсов, от которых наследует тип
- Метод `Class Class.getSuperclass()`
  Возвращает ссылку на описание родительского класса

Получение информации о классе

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

class ListMethods {
  public static void main(String[] args)
                throws ClassNotFoundException {
    Class c = Class.forName(args[0]);
    Constructor[] cons = c.getConstructors( );
    printList("Constructors", cons);
    Method[] meths = c.getMethods( );
    printList("Methods", meths);
    Field[] fields = c.getFields();
    printList("Fields", fields);
  }
  static void printList(String s, Object[] o) {
    System.out.println("*** " + s + " ***");
    for (int i = 0; i < o.length; i++)
      System.out.println(o[i].toString( ));
  }
}
```

Возможности класса Class

- Загрузка класса в JVM по его имени
  `static Class forName(String name)`
- Определение вида типа
  `boolean isInterface()`
  `boolean isLocalClass()`
- Получение родительских типов
  `Class getSuperclass()`
  `Class[] getInterfaces()`
- Получение вложенных типов
  `Class[] getClasses()`
- Создание объекта
  `Object newInstance()`

Возможности класса Class

- Получение списка всех полей и конкретного поля по имени
  ```
  Field[] getFields()
  Field getField(...)
  ```
- Получение списка всех методов и конкретного метода по имени и списку типов параметров
  ```
  Method[] getMethods()
  Method getMethod(...)
  ```
- Получение списка всех конструкторов и конкретного конструктора по списку типов параметров
  ```
  Constructor[] getConstructors()
  Constructor getConstructor(...)
  ```

Передача параметров в методы

- Поскольку на момент написания программы типы и даже количество параметров неизвестно, используется другой подход:
  - Ссылки на все параметры в порядке их следования помещаются в массив типа `Object`
  - Если параметр имеет примитивный тип, то в массив помещается ссылка на экземпляр класса-оболочки соответствующего типа, содержащий необходимое значение
- Возвращается всегда тип `Object`
  - Для ссылочного типа используется приведение типа или рефлексивное исследование
  - Для примитивных типов возвращается ссылка на экземпляр класса-оболочки, содержащий возвращенное значение

Создание экземпляров классов

- Метод `Object Class.newInstance()`
  Возвращает ссылку на новый экземпляр класса, используется конструктор по умолчанию

- Метод
  ```
  Object Constructor.newInstance(
      Object[] initArgs)
  ```
  Возвращает ссылку на новый экземпляр класса, с использованием конструктора и указанными параметрами конструктора

Вызов методов

- Прямой вызов
  Если на момент написания кода известен класс-предок загружаемого класса

- Вызов через экземпляр класса `Method`
  `Object Method.invoke(Object obj, Object[] args)`
  - `obj` – ссылка на объект, у которого должен быть вызван метод
  - принято передавать `null`, если метод статический
  - `args` – список параметров для вызова методов

Вызов статического метода

```
import java.lang.reflect.*;
public class Main {
  public static void main(String[] args) {
    if (args.length == 3) {
      try {
        Class c = Class.forName(args[0]);
        Method m = c.getMethod(args[1], new Class [] {Double.TYPE});
        Double val = Double.valueOf(args[2]);
        Object res = m.invoke(null, new Object [] {val});
        System.out.println(res.toString());
      } catch (ClassNotFoundException e) {
        System.out.println("Класс не найден");
      } catch (NoSuchMethodException e) {
        System.out.println("Метод не найден");
      } catch (IllegalAccessException e) {
        System.out.println("Метод недоступен");
      } catch (InvocationTargetException e) {
        System.out.println("При вызове возникло исключение");
      }
    }
}}
```

Аннотации (java 1.5)

Лекция 11.2

УНЦ «Инфоком»
Самара
2016

NetCracker®

Проблема

- Имеется:
  - вся информация о классе содержится непосредственно в нем
  - комментарии доступны только если доступен исходный текст
  - введение в класс методов, описывающих его семантику, приводит к существенному снижению понимания кода и его загромождению
- Хотелось бы:
  - иметь средство описания семантики и особенностей класса
  - это средство должно лежать за пределами самого класса

Метаданные

- В основе механизма метаданных лежат так называемые аннотации
- **Аннотация** – это «интерфейс» специфического вида, позволяющий задавать описания классов и их элементов
- Пример объявления аннотации:

```
@interface MyAnnotation {
  String str();
  int val();
}
```

Особенности аннотаций

- Члены-методы имеют, скорее, смысл полей
- Тела этих методов будут создаваться автоматически
- Аннотациями можно снабжать классы, методы, поля, параметры, константы перечислимых типов и аннотации
- Пример снабжения аннотацией:

```
@MyAnnotation(str = "Example", val= 100)
public static void myMeth() {...}
```

Особенности аннотаций

- В любом случае аннотация предшествует объявлению
- Все аннотации наследуют от интерфейса `java.lang.annotation.Annotation`
- Во время выполнения программы информация об аннотациях извлекается средствами рефлексии
- После получения ссылки на объект аннотации у него можно вызывать методы, возвращающие заданные при аннотировании значения

Особенности аннотаций

- Для методов допускаются значения по умолчанию

```
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnnotation {
  String str() default "Testing";
  int val() default 9000;
}
// @MyAnnotation()
// @MyAnnotation(str = "some string");
// @MyAnnotation(val = 100);
// @MyAnnotation(str = "Testing", val = 100);
```

Особенности аннотаций

- Бывают одночленные аннотации

  Содержат один член и имеют сокращенную форму записи

```
@interface MySingle {
  int value(); //Имя только такое!
}
// @MySingle(100)
```

- Бывают аннотации-маркеры

  Предназначены только для пометки элементов

```
@interface MyMarker {}
// @MyMarker()
```

Правила сохранения аннотаций

- Правила сохранения аннотаций определяют, в какой момент аннотации будут уничтожены
- Правила задаются с помощью перечислимого типа `java.lang.annotation.RetentionPolicy`
- Существует три правила:
  - `SOURCE`
    аннотации отбрасываются на этапе компиляции
  - `CLASS`
    сохраняются в байт-коде, но недоступны JVM во время выполнения программы
  - `RUNTIME`
    доступны JVM во время выполнения программы
- В зависимости от цели аннотации ей задается то или иное правило сохранения

Правила сохранения аннотаций

- Задание правило сохранения производится с помощью аннотации `java.lang.annotation.Retention`
- По умолчанию задается правило CLASS
- Пример задания правила сохранения:

```
@Retention(RetentionPolicy.RUNTIME)
@interface myAnnotation {
  String str();
  int val();
}
```

Стандартные аннотации
(работа с аннотациями)

- `@Retention`
  Применяется к аннотациям, позволяет задать правило сохранения
- `@Documented`
  Применяется к аннотациям, указывает, что она должна быть документирована
- `@Target`
  Применяется к аннотациям, позволяет указать типы объектов, к которым данная аннотация может применятся
- `@Inherited`
  Применяется к аннотациям классов, указывает, что данная аннотация будет унаследована потомками класса

Стандартные аннотации (инструкции компилятора)

- `@Override`
  Применяется к методам, указывает, что метод обязан переопределять метод родительского класса
- `@Deprecated`
  Указывает на то, что объявление является устаревшим или вышедшим из употребления
- `@SupressWarnings`
  Указывает на то, что указанные виды предупреждений компилятора не будут показываться

Особенности аннотаций

- Аннотация не может наследовать другую аннотацию
- Методы аннотаций не должны иметь параметров
- Возвращаемый тип методов:
  - примитивный тип
  - `String`
  - `Class`
  - перечислимый тип
  - другой тип аннотации
  - массив элементов одного из вышеперечисленных типов
- Аннотации не могут быть настраиваемыми
- Методы не могут объявлять исключения

Спасибо за внимание!

NetCracker®

Дополнительные источники

- Арнолд, К. Язык программирования Java [Текст] / Кен Арнолд, Джеймс Гослинг, Дэвид Холмс. – М. : Издательский дом «Вильямс», 2001. – 624 с.
- Вязовик, Н.А. Программирование на Java. Курс лекций [Текст] / Н.А. Вязовик. – М. : Интернет-университет информационных технологий, 2003. – 592 с.
- Эккель, Б. Философия Java [Текст] / Брюс Эккель. – СПб. : Питер, 2011. – 640 с.
- Шилдт, Г. Java 2, v5.0 (Tiger). Новые возможности [Текст] / Герберт Шилдт. – СПб. : БХВ-Петербург, 2005. – 206 с.
- JavaSE at a Glance [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/overview/index.html, дата доступа: 21.10.2011.
- JavaSE APIs & Documentation [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/documentation/api-jsp-136079.html, дата доступа: 21.10.2011.