Java Advanced

Reflection

Содержание

- Введение
- Структура класса
- Массивы
- Загрузчики классов
- Proxy
- Заключение

Reflection

- Библиотека, позволяющая оперировать информацией о типах во время выполнения
- Пакеты
  - java.lang
  - java.lang.reflect

Часть 1

Введение

Информация о типе

- Информация о типах классом Class
- Предоставляемая информация
  - Структура класса
  - Структура наследования
  - Проверки времени выполнения
  - …

Типы типов

- Для определения типов служат методы вида
  - is*()

Аннотация	Annotation
Массив	Array
Примитивный	Primitive
Перечисление	Enum
Интерфейс	Interface
Класс	*Class
Анонимный класс	AnonymousClass
Локальный класс	LocalClass
Класс-член	MemberClass

Получение информации о типе

- Во время исполнения
  object.getClass()
- Во время компиляции
  type.class
- Предопределенные
  Wrapper.TYPE

Общая информация о классе

- Имя класса
  - getCanonicalName() – каноническое имя
  - getName() – полное имя
  - getSimpleName() – простое имя
- Структура классов
  - getSuperClass() – предок
  - getInterfaces() – реализуемые интерфейсы
- Модификаторы
  - getModifiers() – модификаторы

Место определения класса

- Методы получения места, в котором определен класс

Тип класса	Метод
Верхнего уровня	getPackage()
Вложенный	getDeclaredClass()
в конструктор	getEnclosingConstructor()
в метод	getEnclosingMethod()

Приведение классов

- Приведение
  - cast(object) – привести ссылку к типу
- Определение возможности приведения
  - isAssignableFrom(class) – класса
  - isInstance(object) – объекта

Часть 2

Структура класса

Информация о члене класса

- Интерфейс Member
- Методы
  - getDeclaringClass() – класс, в котором определен
  - getName() – имя члена
  - getModifiers() – модификаторы

Модификаторы

- Класс Modifiers

Константа	Метод	Модификатор
ABSTRACT	isAbstract	abstract
FINAL	isFinal	final
INTERFACE	isInterface	interface
NATIVE	isNative	native
PRIVATE	isPrivate	private
PROTECTED	isProtected	protected
PUBLIC	isPublic	public
STATIC	isStatic	static
STRICT	isStrict	strictfp
SYNHRONIZED	isSynhronized	synhronized
TRANSIENT	isTransient	transient
VOLATILE	isVolatile	volatile

Поля

- Открытые
  - getFields() – все поля
  - getField(name) – конкретное поле
- Все
  - getDeclaredFields() – все поля
  - getDeclaredField(name) – конкретное поле
- Исключения
  - NoSuchFieldException

Свойства полей

- Класс Field
- Информация
  - getName() – имя поля
  - getType() – тип значения
- Чтение значения
  - get(object) – ссылки
  - get*(object) – значения примитивного типа
- Запись значения
  - set(object, value) – ссылки
  - set*(object, value) – значения примитивного типа

Методы

- Открытые
  - getMethods() – все методы
  - getMethod(name, Class… parameters) – конкретный метод
- Все
  - getDeclaredMethods() – все методы
  - getDeclaredMethod(name , Class… parameters) – конкретный метод
- Исключения
  - NoSuchMethodException

Свойства методов

- Класс Method
- Сигнатура метода
  - getName() – имя метода
  - getParameterTypes() – параметры метода
- Другая информация
  - getExceptionTypes() – возможные исключения
  - getReturnType() – тип возвращаемого значения
- Вызов метода
  - invoke(Object object, Object …args) – вызвать метод с указанными аргументами

Конструкторы

- Открытые
  - getConstructor() – все конструкторы
  - getConstructor(Class… parameters) – конкретный конструктор
- Все
  - getDeclaredConstructor() – все конструкторы
  - getDeclaredConstructor(name , Class… parameters) – конкретный конструктор
- Исключения
  - NoSuchMethodException

Свойства конструкторов

- Класс Constructor
- Информация о конструкторе
  - getParameterTypes() – параметры конструктора
  - getExceptionTypes() – возможные исключения
- Создание объекта
  - newInstance(Object … args) – создать новый объект
  - class.newInstance() – создать новый объект используя конструктор по умолчанию

Классы и интерфейсы

- Открытые
  - getClasses() – все классы и интерфейсы
- Все
  - getDeclaredClasses() – все классы и интерфейсы

Доступ к закрытым членам

- По умолчанию доступ к закрытым членам запрещен IllegalAccessException
- Все члены extends AccessibleObject
  - setAccessible(boolean) – запросить доступ
  - isAccessible() – проверить доступ

Пример: листинг класса

```
Class c = ArrayList.class;
for (Field m : c.getDeclaredFields()) {
    System.out.println(m);
}

for (Constructor m : c.getDeclaredConstructors()) {
    System.out.println(m);
}

for (Method m : c.getDeclaredMethods()) {
    System.out.println(m);
}
```

Пример: создание экземпляра

```
// Получение класса
Class clazz = Integer.class;
// Получение конструктора
Constructor c = clazz.getConstructor(int.class);
// Создание экземпляра
Integer i = (Integer) c.newInstance(100);
// Проверка
System.out.println(i);
```

Часть 3

Массивы

Операции с массивами

- Класс Array
- Создание массива заданного типа
  - newInstance(Class, length) – линейного
  - newInstance(Class, dims[]) – “кубического”
- Чтение значения из массива
  - get(array, index) – ссылки
  - get*(array, index) – значения примитивного типа
- Запись значения в массив
  - set(array, index, value) – ссылки
  - set*(array, index , value) – значения примитивного типа

Массивы как типы

- Имя типа массива

  [имя_типа_элемента
- Методы
  - isArray() – является ли массивом
  - getComponentType() – тип элемента массива

Имена для типов

- Имена классов типов в массиве кодируются специальным образом

class	L*class*;
boolean	Z
byte	B
char	C
double	D
float	F
int	I
long	J
short	S

Часть 4

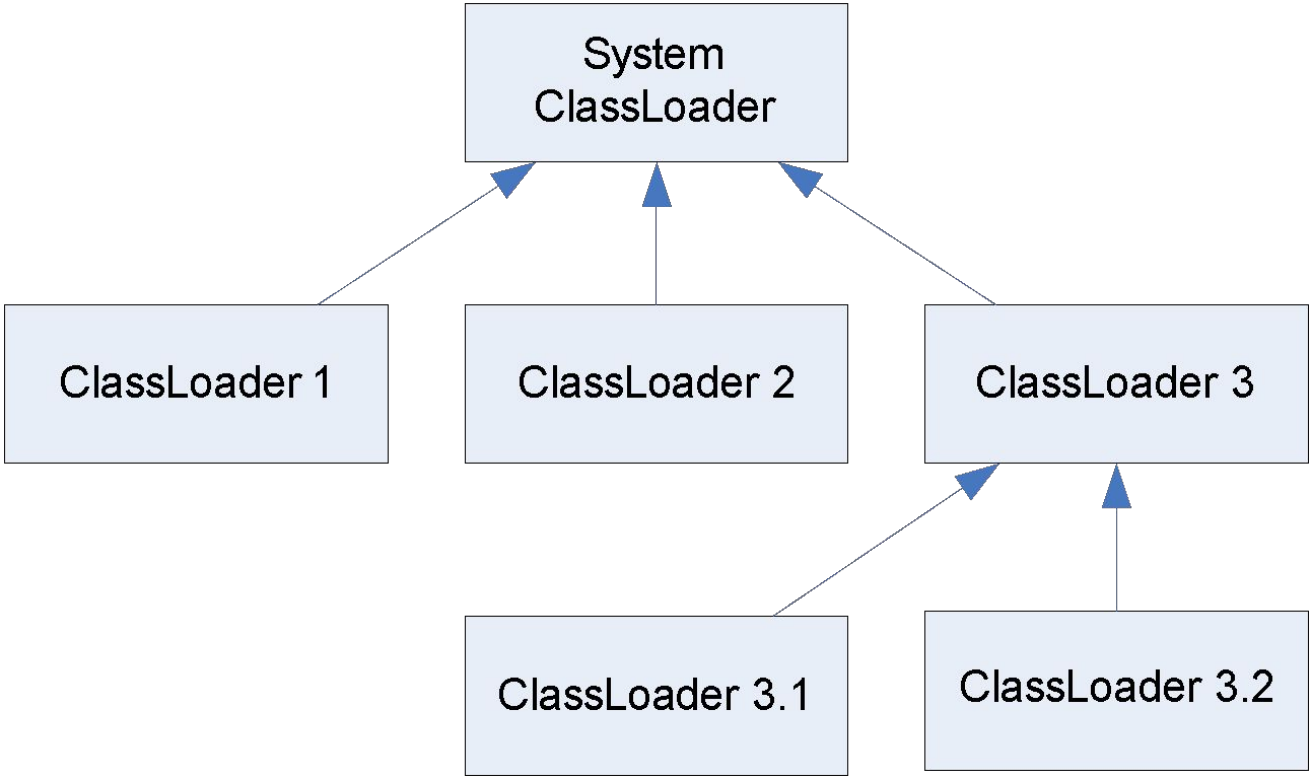
Загрузчики классов

Загрузчики классов

- Позволяют загружать и определять новые классы
- Класс ClassLoader
- Методы
  - loadClass(name, resolve?) – загружает класс по имени
  - findLoadedClass(name) – найти уже загруженный класс
  - resolveClass(class) – загружает библиотеки

Дерево загрузчиков

- Загрузчики образуют дерево
- Загрузчики в разных ветвях могут загрузить разные классы с одним полным именем



Дополнительные возможности

- Получения родителя
  - getParent()
- Загрузка ресурсов
  - URL getResouce(String name) – определение местоположения ресурса по имени
  - getResouceAsStream(String name) – чтение ресурса по имени

Загрузчики и классы

- Получение загрузчика
  - getClassLoader() – кто загрузил класс
  - Thread.getContextClassLoader() – контекстный загрузчик
- “Прямая” загрузка класса
  - Class.forName(name)

Реализации загрузчиков

- Класс URLClassLoader
  - Загружает классы из нескольких мест, заданных URL

Пример: загрузка класса

```
URL jar = new URL("file://.");
className = "Test";
ClassLoader cl = new URLClassLoader(new
    URL[]{jar});
Class c = cl.loadClass(className);


 Method m = c.getMethod("main", String[].class);
m.invoke(null, (Object) new String[]{"hello"});
```

Часть 5

Proxy

Proxy

- Механизм, позволяющий создавать фиктивные классы, реализующие требуемые интерфейсы
- Класс Proxy

Класс InvocationHandler

- Ему делегируются вызовы, совершенные для Proxy
- Методы
  - invoke(Object proxy, Method, Object[] args) – уведомляет о вызове метода

Методы Proxy

- Создание экземпляра Proxy
  - newProxyInstance(ClassLoader, Class[] interfaces, InvocationHandler)
- Получение класса Proxy
  - getProxyClass(ClassLoader, Class[] interfaces)
- Проверка класса
  - isProxyClass(Class)

Пример: профайлер (1)

- Класс

```
public class Profiler implements InvocationHandler {
    // Экземпляр Proxy
    private final Object instance;
    // Реальная реализация
    private final Object impl;

    …
}
```

Пример: профайлер (2)

- Конструктор

```
public Profiler(Class[] i8s, Object impl) {
    this.impl = impl;
    instance = Proxy.newProxyInstance(null,
      i8s, this);
}
```

- Создание экземпляра

```
public Object getInstance() {
    return instance;
}
```

Пример: профайлер (3)

- Основной метод

```
public Object invoke(
    Object proxy, Method method, Object[] args
) throws IllegalAccessException,
    InvocationTargetException
{
    System.out.println("Calling " + method +
            " on " + impl);
    return method.invoke(impl, args);
}
```

Пример: профайлер (4)

- Применение

```
public static void main(String[] args) {
  Integer i1 = new Integer(3);
  Profiler profiler = new Profiler(
      new Class[]{Comparable.class}, i1);
  Comparable i2 =
      (Comparable) profiler.getInstance();
  System.out.println(i2.compareTo(i1));
}
```

Часть 6

Заключение

Выводы

- Reflection позволяет
  - Анализировать классы по время исполнения
  - Загружать классы по имени
  - Создавать экземпляры классов по имени
  - Вызывать метод классов по имени
  - Оперировать значениями полей по имени
  - Создавать и оперировать с массивами по типу элемента
  - Создавать proxy для интерфейсов

Ссылки

- Reflection (Guide) // http://java.sun.com/j2se/1.5.0/docs/guide/reflection/index.html
- Reflection API Code Samples // http://java.sun.com/developer/codesamples/refl.html
- Using Java Reflection // http://java.sun.com/developer/technicalArticles/ALT/Reflection/index.html
- The Reflection API (tutorial) // http://java.sun.com/docs/books/tutorial/reflect/index.html

Вопросы