Введение в Java Reflection API

**Рефлексия** (от позднелат. reflexio - обращение назад) - это механизм исследования данных о программе во время её выполнения. Рефлексия позволяет исследовать информацию о полях, методах и конструкторах классов. Можно также выполнять операции над полями и методами которые исследуются. Рефлексия в Java осуществляется с помощью Java Reflection API. Этот интерфейс API состоит из классов пакетов java.lang и java.lang.reflect. С помощью интерфейса Java Reflection API можно делать следующее:

* Определить класс объекта.
* Получить информацию о модификаторах класса, полях, методах, конструкторах и суперклассах.
* Выяснить, какие константы и методы принадлежат интерфейсу.
* Создать экземпляр класса, имя которого неизвестно до момента выполнения программы.
* Получить и установить значение свойства объекта.
* Вызвать метод объекта.
* Создать новый массив, размер и тип компонентов которого неизвестны до момента выполнения программ.

Получение объекта типа Class

MyClass a = **new** MyClass();   
Class aclass = a.getClass();

Самое простое, что обычно делается в динамическом программировании, - это получают объект типа java.lang.Class. Если у нас есть экземпляр объекта Class мы можем получить всевозможную информацию об этом классе и даже осуществлять операции над ним. Вышеприведенный метод getClass() часто полезен тогда когда есть экземпляр объекта, но не известно какого класса этот экземпляр. Если у нас есть класс, для которого в момент компиляции известен тип, то получить экземпляр класса ещё проще.

Class aclass = MyClass.**class**;   
Class iclass = Integer.**class**;

Если имя класса не известно в момент компиляции, но становится известным во время выполнения программы, можно использовать метод forName(), чтобы получить объект Class.

Class c = Class.forName("com.mysql.jdbc.Driver");

Получение имени класса

Class c = myObject.getClass();   
String s = c.getName();

Объект типа String, возвращаемый методом getName(), будет содержать полностью уточненное имя класса, т.е. если типом объекта myObject будет Integer, то результат будет вида java.lang.Integer .

Исследование модификаторов класса

Class c =  obj.getClass();   
**int** mods = c.getModifiers();   
**if** (Modifier.isPublic(mods)) {   
    System.out.println("public");   
}   
**if** (Modifier.isAbstract(mods)) {   
    System.out.println("abstract");   
}   
**if** (Modifier.isFinal(mods)) {   
    System.out.println("final");   
}

Чтобы узнать, какие модификаторы были применены к заданному классу, сначала нужно с помощью метода getClass получить объект типа Class, представляющий данный класс. Затем нужно вызвать метод getModifiers() для объекта типа Class, чтобы определить значение типа int, биты которого представляют модификаторы класса. После этого можно использовать статические методы класса java.lang.reflect.Modifier, чтобы определить, какие именно модификаторы были применены к классу.

Нахождение суперклассов

Class c = myObj.getClass();   
Class superclass = c.getSuperclass();

Можно также использовать метод getSuperclass() для объекта Class, чтобы получить объект типа Class, представляющий суперкласс рефлексированного класса. Нужно не забывать учитывать, что в Java отсутствует множественное наследование и класс java.lang.Object является базовым классом для всех классов, вследствие чего если у класса нет родителя то метод getSuperclass вернет null. Для того чтобы получить все родительские суперклассы, нужно рекурсивно вызывать метод getSuperclass().

Определение интерфейсов, реализуемых классом

Class c =  LinkedList.**class**;   
Class[] interfaces = c.getInterfaces();   
**for**(Class cInterface : interfaces) {   
    System.out.println( cInterface.getName() );   
}

С помощью рефлексии можно также определить, какие интерфейсы реализованы в заданном классе. Метод getInterfaces() вернет массив объектов типа Class. Каждый объект в массиве представляет один интерфейс, реализованный в заданном классе.

Исследование, получение и установка значений полей класса.

Class c = obj.getClass();   
Field[] publicFields = c.getFields();   
**for** (Field field : publicFields) {   
    Class fieldType = field.getType();   
    System.out.println("Имя: " + field.getName());   
    System.out.println("Тип: " + fieldType.getName());   
}

Чтобы исследовать поля принадлежащие классу, можно воспользоваться методом getFields() для объекта типа Class. Метод getFields() возвращает массив объектов типа java.lang.reflect.Field, соответствующих всем открытым полям объекта. Эти открытые поля необязательно должны содержаться непосредственно внутри класса, с которым вы работаете, они также могут содержатся в суперклассе, интерфейсе или интерфейсе представляющем собой расширение интерфейса, реализованного классом. С помощью класса Field можно получить имя поля, тип и модификаторы. Если известно имя поля, то можно получить о нем информацию с помощью метода getField()

Class c = obj.getClass();   
Field nameField = c.getField("name");

Методы getField() и getFields() возвращают только открытые члены данных класса. Если требуется получить все поля некоторого класса нужно использовать методы getDeclaredField() и getDeclaredFields(). Эти методы работают точно также как их аналоги getField() и getFields(), за исключением того, что они возвращают все поля, включая закрытые и защищенные. Чтобы получить значение поля, нужно сначала получить для этого поля объект типа Field затем использовать метод get(). Метод принимает входным параметром ссылку на объект класса.

Class c = obj.getClass();   
Field field = c.getField("name");   
String nameValue = (String) field.get(obj)

Так же у класса Field имеются специализированные методы для получения значений примитивных типов: getInt(), getFloat(), getByte() и др.. Для установки значения поля, используется метод set().

Class c = obj.getClass();   
Field field = c.getField("name");   
field.set(obj, "New name");

Для примитивных типов имеются методы setInt(), setFloat(), setByte() и др. Вопрос к читателю, можно ли изменить значение private поля? А private final? Ответ узнаем чуть позже.

Исследование конструкторов класса

Class c = obj.getClass();   
Constructor[] constructors = c.getConstructors();   
**for** (Constructor constructor : constructors) {   
    Class[] paramTypes = constructor.getParameterTypes();   
    **for** (Class paramType : paramTypes) {   
        System.out.print(paramType.getName() + " ");   
    }   
    System.out.println();   
}

Чтобы получить информацию об открытых конструкторах класса, нужно вызвать метод getConstructors() для объекта Class. Этот метод возвращает массив объектов типа java.lang.reflect.Constructor. С помощью объекта Constructor можно затем получить имя конструктора, модификаторы, типы параметров и генерируемые исключения. Можно также получить по отдельному открытому конструктору, если известны типы его параметров.

Class[] paramTypes = **new** Class[] { String.**class**, **int**.**class** };   
Constructor aConstrct = c.getConstructor(paramTypes);

Методы getConstructor() и getConstructors() возвращают только открытые конструкторы. Если требуется получить все конструкторы класса, включая закрытые можно использовать методы getDeclaredConstructor() и getDeclaredConstructors() эти методы работают точно также, как их аналоги getConstructor() и getConstructors().

Исследование информации о методе, вызов метода.

Class c = obj.getClass();   
Method[] methods = c.getMethods();   
**for** (Method method : methods) {   
    System.out.println("Имя: " + method.getName());   
    System.out.println("Возвращаемый тип: " + method.getReturnType().getName());   
   
    Class[] paramTypes = method.getParameterTypes();   
    System.out.print("Типы параметров: ");   
    **for** (Class paramType : paramTypes) {   
        System.out.print(" " + paramType.getName());   
    }   
    System.out.println();   
}

Чтобы получить информацию об открытых методах класса, нужно вызвать метод getMethods() для объекта Class. Этот метод возвращает массив объектов типа java.lang.reflect.Method. Затем с помощью объекта Method можно получить имя метода, тип возвращаемого им значения, типы параметров, модификаторы и генерируемые исключения. Также можно получить информацию по отдельному методу если известны имя метода и типы параметров.

Class c = obj.getClass();   
Class[] paramTypes = **new** Class[] { **int**.**class**, String.**class**};   
Method m = c.getMethod("methodA", paramTypes);

Методы getMethod() и getMethods() возвращают только открытые методы, для того чтобы получить все методы класса не зависимо от типа доступа, нужно воспользоватся методами getDeclaredMethod() и getDeclaredMethods(), которые работают точно также как и их аналоги (getMethod() и getMethods()). Интерфейс Java Reflection Api позволяет динамически вызвать метод, даже если во время компиляции имя этого метода неизвестно (Имена методов класса можно получить методом getMethods() или getDeclaredMethods()). В следующем примере рассмотрим вызов метода зная его имя. Например метод getCalculateRating:

Class c = obj.getClass();   
Class[] paramTypes = **new** Class[] { String.**class**, **int**.**class** };   
Method method = c.getMethod("getCalculateRating", paramTypes);   
Object[] args = **new** Object[] { **new** String("First Calculate"), **new** Integer(10) };   
Double d = (Double) method.invoke(obj, args);

В данном примере сначала получаем объект Method по имени метода getCalculateRating, затем вызываем метод invoke() объекта Method, и получаем результат работы метода. Метод invoke принимает два параметра, первый - это объект, класс которого объявляет или наследует данный метод, а второй - массив значений параметров, которые передаются вызываемому методу. Если метод имеет модификатор доступа private, тогда выше приведённый код нужно модифицировать таким образом, для объекта Method вместо метода getMethod() вызываем getDeclaredMethod(), затем для получения доступа вызываем setAccessible(true).

Method method = c.getDeclaredMethod("getCalculateRating", paramTypes);   
method.setAccessible(**true**);

Загрузка и динамическое создание экземпляра класса

Class c = Class.forName("Test");   
Object obj = c.newInstance();   
Test test = (Test) obj;

С помощью методов Class.forName() и newInstance() объекта Class можно динамически загружать и создавать экземпляры класса в случае, когда имя класса неизвестно до момента выполнения программы. В приведенном коде мы загружаем класс с помощью метода Class.forName(), передавая имя этого класса. В результате возвращается объект типа Class. Затем мы вызываем метод newInstance() для объекта типа Class, чтобы создать экземпляры объекта исходного класса. Метод newInstance() возвращает объет обобщенного типа Object, поэтому в последней строке мы приводим возвращенный объект к тому типу, который нам нужен.

И напоследок, пример модификации private полей.

**import** java.lang.reflect.Field;   
   
**class** WithPrivateFinalField {   
    **private** **int** i = 1;   
    **private** **final** String s = "String S";   
    **private** String s2 = "String S2";   
   
    **public** String toString() {   
        **return** "i = " + i + ", " + s + ", " + s2;   
    }   
}   
   
**public** **class** ModifyngPrivateFields {   
   
    **public** **static** **void** main(String[] args) **throws** Exception {   
        WithPrivateFinalField pf = **new** WithPrivateFinalField();   
           
        Field f = pf.getClass().getDeclaredField("i");   
        f.setAccessible(**true**);   
        f.setInt(pf, 47);   
        System.out.println(pf);   
   
        f = pf.getClass().getDeclaredField("s");   
        f.setAccessible(**true**);   
        f.set(pf, "MODIFY S");   
        System.out.println(pf);   
   
   
        f = pf.getClass().getDeclaredField("s2");   
        f.setAccessible(**true**);   
        f.set(pf, "MODIFY S2");   
        System.out.println(pf);   
    }   
}

Из приведённого кода видно что private поля можно изменять. Для этого требуется получить объект типа java.lang.reflect.Field с помощью метода getDeclaredField(), вызвать метод setAccessible(true) и с помощью метода set() устанавливаем значение поля. Учтите что поле final при выполнении данной процедуры не выдаёт предупреждений, а значение поля остаётся прежним, т.е. final поля остаются неизменные.