Reflexe

RTTI – Runtime Type Identification

Rudolf PECINOVSKÝ
rudolf@pecinovsky.cz

Co je to reflexe/RTTI

- **Termínem Reflexe označujeme schopnost získat za běhu informace o typu objektu, s nímž program pracuje**
  - Tato schopnost je velice potřebná pro vzdálené zpracování, kdy na jeden počítač přicházejí ke zpracování objekty z jiného počítače a zpracovávající počítač musí mít možnost zjistit o těchto objektech potřebné údaje
- **Všechny potřebné informace o každém typu získáme z jeho class-objektu – instance třídy `java.lang.Class`**
  - Je-li v OOP vše objekt, je objektem o třída či libovolný jiný datový typ
  - Class-objekt můžeme považovat za objekt představující v programu daný datový typ
- **Svůj class-objekt mají nejenom klasické třídy, ale i rozhraní a primitivní typy**
- **Class-objekt je vytvářen instancí třídy `ClassLoader` při zavádění daného typu do paměti**

Základní charakteristika

- **Java je schopná získat za běhu programu od objektu kompletní informaci o jeho typu**
  - Název typu
  - Druh typu (rozhraní – třída – výčtový typ)
  - Atributy, konstruktory, metody a vnořené datové typy
  - Anotace
  - …
- **S informacemi zjištěnými za běhu se dá pracovat**
  - Volat konstruktory (vytvářet instance) a metody
  - Zjišťovat a přiřazovat hodnoty atributů
  - …
- **Lze zjistit i informace o nedostupných členech, ale za běžné situace k nim nelze přistupovat**

Využití

- Nutné při vzdáleném zpracování, protože metoda musí umět zjistit, zda obdržela po síti parametr požadovaného typu
- Na této schopnosti je založená řada aplikací, které za běhu přizpůsobují své chování objektům s nimiž pracují
- Potřebujete-li přizpůsobit chování programu verzi virtuálního stroje, na kterém program aktuálně běží
  - Jednotlivé verze se poněkud liší svými knihovnami
  - Může nastat případ, kdy neexistuje společná použitelná třída ve všech podporovaných verzích knihoven
- Chcete-li dosáhnout dynamického chování programu
  - JVM nenabízí (zatím) nic, co by Java přímo neuměla
  - Překladače dynamických jazyků nad JVM (Groovy, Jython, JRuby, ...) řeší dynamičnost prostřednictvím reflexe

Získání existujícího class-objektu

- **Literál *Typ*`.class`**
  - Vrací class-objekt zadaného typu
  - Typ se zapisuje přímo svým názvem,
    a musí být proto znám již v době překladu (`String.class`)
  - Takto lze získat i class-objekt primitivních typů (`int.class`)
- **Metoda *instance*`.getClass()`**
  - Nepřekrytelná instanční metoda zděděné od třídy `Object`
  - Vrací class-objekt vlastní rodičovské třídy dané instance
  - Překladač zná pouze deklarovaný typ dané instance,
    nezná však její skutečný typ, tj. její vlastní rodičovskou třídu

Získání (i) nového class-objektu

- **Statická metoda `Class.forName(String)`**
  - Parametr definuje název typu, jehož class-objekt chceme získat
  - Název typu musí být úplný – např. `"java.lang.String"`
  - Metoda umožňuje zjistit název potřebného typu až za běhu
  - Umožňuje získávat class-objekty typů vytvořených až po spuštění programu
- **Stat. m. `Class.forName(String, boolean, ClassLoader)`**
  - 2. parametr umožní zavést, ale neinicializovat typ se zadaným názvem
    - Zrychlí zavedení typu / vytvoření class-objektu
    - Výhodné, pokud pouze zjišťujeme přítomnost typu, ale nechceme s ním dále pracovat
  - 3. parametr umožní definovat, odkud a jak daný typ získám
    - Zadaný `ClassLoader` může načítat podklady např. z databáze či bajtového pole
  - Výraz `Class.forName("Foo")` je ekvivalentní zápisu `Class.forName("Foo",true,this.getClass().getClassLoader())`
- **Je-li typ zadaného názvu již zaveden, je vrácen jeho class-objekt (tj. nevytváří se nový)**

Třída `Class` a její metody

Metody

- Třída definuje 57 metod,
  z toho pouze 2 statické (metody `forName(???)`)
- `ClassLoader getClassLoader()`
  - Vrátí `ClassLoader`, který zaváděl danou třídu
- `URL getResource(String name)`
  - Vrátí URL zadané relativně vůči umístění třídy
  - Lze použít i uvnitř JAR-souborů, avšak pouze v jedné úrovni
- `InputStream getResourceAsStream(String name)`
  - Vrátí vstupní proud čtoucí ze „souboru“ zadaného URL relativní vůči umístění třídy
- Zjištění názvu třídy
  - `String getSimpleName()`
  - `String getName()`
  - `String toString()`

Zjišťování druhu datového typu

- **Class-objekt nereprezentuje pouze třídy, ale i jiné druhy datových typů**
- **Získání informací o druhu typu reprezentovaného class-objektem**
  - `isAnnotation()`
  - `isAnonymousClass()`
  - `isArray()`
  - `isEnum()`
  - `isInterface()`
  - `isLocalClass()`
  - `isMemberClass()`
  - `isPrimitive()`
  - `isSynthetic()`

Další informace o typu

- `Package getPackage()`
  - Objekt reprezentující balíček
- `int getModifiers()`
  - Každému modifikátoru je přiřazen bit,
    přítomnost modifikátoru je reprezentována nahozením bitu
- `Class<? super T> getSuperclass()`
  - Získání class-objektu rodičovské třídy
- `Class<?> getComponentType()`
  - Získání class-objektu prvků pole

Informace o členech daného typu

- **Metody vracející reprezentanty veřejných členů**
  - Vrací i zděděné členy –
    vracené odvozuje od rozhraní
  - `Constructor<?>[] getConstructors()`
  - `Field[] getFields()`
  - `Method[] getMethods()`
  - `Class<?>[] getClasses()`
  - `Annotation[] getAnnotations()`
- **Metody vracející reprezentanty deklarovaných členů**
  - Vrací i soukromé členy, ale nevrací zděděné členy –
    vracené odvozuje od obsahu class-souboru
  - `Constructor<?>[] getDeclaredConstructors()`
  - `Field[] getDeclaredFields()`
  - `Method[] getDeclaredMethods()`
  - `Class<?>[] getDeclaredClasses()`
  - `Annotation[] getDeclaredAnnotations()`

Získání konkrétního členu

- **U konstruktorů je třeba zadat typy parametrů**
  - Constructor<T> getConstructor(Class<?>... parameterTypes)
  - Constructor<T> getDeclaredConstructor(Class<?>... parameterTypes)
  - Statický konstruktor nelze získat, protože by nám stejně k ničemu nebyl
- **U metod je třeba zadat název a typy parametrů**
  - Method getMethod(String name, Class<?>... parameterTypes)
  - Method getDeclaredMethod(String name, Class<?>... parameterTypes)
  - Nelze použít názvy <init> a <clinit> – jsou to sice metody, ale musí se volat jako konstruktory, aby se před nimi zavolal `new`
- **U polí se zadává pouze název**
  - `Field getField(String name)`
  - `Field getDeclaredField(String name)`

Práce se získanými členy

- **Vytvoření instance pomocí konstruktoru**
  - `T newInstance(Object... initargs)`
  - V poli předávaném jako parametr se zadávají hodnoty parametrů
  - Metodu `newInstance()` má i class-objekt a volá implicitní konstruktor => vystačíme-li s implicitním konstruktorem, nepotřebujeme `Constructor`
- **Zavolání metody**
  - `Object invoke(Object obj, Object... args)`
  - V poli předávaném jako parametr se zadávají hodnoty parametrů
  - Parametr `obj` je `this` = instance, jejíž metodu voláme (u statických metod se ignoruje – může být `null`)
- **Získání a nastavení hodnoty pole**
  - `Object get(Object obj)`
  - `Object set(Object obj, Object value)`
  - `Boolean getBoolean(Object obj)`
  - `Boolean setBoolean(Object obj, boolean value)`
  - `...` a další metody pro zbylé primitivní datové typy
  - Zadávaný parametr `obj` představuje objekt, s jehož polem pracujeme

Soukromé a konstantní členy

- Třídy `Constructor`, `Method` a `Field` jsou definovány v balíčku `java.lang.reflect`
- Všechny jsou potomky třídy `java.lang.reflect.AccessibleObject`
- Od ní dědí (mimo jiné) metody
  - `boolean isAccessible()`
  - `void setAccessible(boolean flag)`
- Nastavením přístupnosti lze zařídit:
  - Použitelnost soukromých členů
  - Modifikovatelnost konstant
    (nevztahuje se na konstanty vyhodnotitelné v době překladu)

Přetypování & spol.

- `<U> Class<? extends U> asSubclass(Class<U> clazz)`
  - Přetypuje class-objekt aby reprezentoval svého zadaného potomka
- `T cast(Object obj)`
  - Přetypuje zadaný objekt na typ reprezentovaný class-objektem
- `boolean isAssignableFrom(Class<?> cls)`
  - Zjistí, lze li do instancí class-objektu přiřazovat instance zadaného typu, tj. lze-li typ class-objektu považovat za rodič zadaného typu
- `boolean isInstance(Object obj)`
  - Zjistí, je-li zadaný objekt instancí typu class-objektu či jeho potomka

Vlastnosti interních tříd

- **Je interní?**
  - `boolean isLocalClass()`
  - `boolean isMemberClass()`
- **Zjištění definičního bloku lokální třídy**
  - `Constructor<?> getEnclosingConstructor()`
  - `Method getEnclosingMethod()`
- **Zjištění vlastníka dané třídy**
  - `Class<?> getDeclaringClass()`
    Pouze pro globální (vnořené a vnitřní) třídy
  - `Class<?> getEnclosingClass()`
    I pro lokální třídy

Práce s anotacemi

- Datové typy mohou být označeny anotacemi – kvůli nim třída implementuje rozhraní `AnnotatedElement`, které deklaruje metody:
  - `boolean isAnnotationPresent (Class<? extends Annotation> annotationClass)`
  - `<A extends Annotation> A getAnnotation (Class<A> annotationClass)`
  - `Annotation[] getAnnotations()`
  - `public Annotation[] getDeclaredAnnotations()`

Děkuji za pozornost

Rudolf Pecinovský
mail: rudolf@pecinovsky.cz
ICQ: 158 156 600

Katedra informačních technologií
Kit
VŠE Praha