# Reflection в языке Java

Виктор Яковлев
Кафедра АТП МФТИ 2017

# Интроспекция объектов - пример на Python

```
def func():
    print("Hello, World!");

func_name = input("Please enter func name: ")

func_ref = globals()[func_name]

func_ref()  # Hello, World!
```

# Что такое reflection

1. Возможность обратиться к полям любого класса по имени во время выполнения, минуя этап компиляции

2. Возможность вызвать произвольный метод по его имени

# Предпосылки рефлексии

Необходимо во время выполнения иметь список пар:

имя (строка) -> указетель на код/свойство

Выполнение кода на Python - интерпретация (все тексты заведомо известны)

Выполнение кода на Java - JIT-интрепретация подробно аннотированного байт-кода (все public-названия известны)

C/C++ -- ????

# Механизм связывания на C/C++

**Linux/Mac:**

```
#include <dlfcn.h>
void some_func() {
    void* lib =
       dlopen(
          "libSDL.so",
          RTLD_LAZY
       ); // Check for NULL!!!

    void* func_ptr =
       dlsym(lib, "SDL_Init");
          // Check for NULL!!!

    (*func_ptr)(); // Call
}
```

**Windows:**

```
#include <Windows.h>
void some_func() {
    HMODULE lib =
       LoadLibraryA(
          "winhttp.dll"
       ); // Check for NULL!!!

    FARPROC func_ptr =
       GetProcAddress(
         lib, "WinHttpConnect"
       ); // Check for NULL!!!

    (*func_ptr)(....); // Call
}
```

# Механизм связывания на C/C++

**Linux/Mac:**

```
#include <dlfcn.h>

void callable() {}

void some_func() {
    void* lib =
       dlopen(NULL, 0);

    void* func_ptr =
       dlsym(lib, "callable");
          // Check for NULL!!!

    (*func_ptr)(); // Call
}
```

**Windows:**

```
#include <Windows.h>

__declspec(dllexport)
void callable() {}

void some_func() {
    HMODULE lib =
       LoadLibrary(NULL);

    FARPROC func_ptr =
       GetProcAddress(
         lib, "callable"
       ); // Check for NULL!!!

    (*func_ptr)(); // Call
}
```

# Механизм связывания на C/C++

**Linux/Mac:**

```
#include <dlfcn.h>

void callable() {}

void some_func() {
    void* lib =
       dlopen(NULL, 0);

    void* func_ptr =
       dlsym(lib, "callable");
          // Check for NULL!!!

    (*func_ptr)(); // Call
}
```

**Ограничения:**

- Код должен быть скомпилирован в качестве библиотеки (все имена функций хранятся в таблице объектного файла)
- Известны только имена, но не сигнатуры функций*

*** функции и методы, с точки зрения реализации - это одно и то же**

# C++ - Runtime Type Information (RTTI)

- оператор dynamic_cast
- стандартная функция typeid, возвращающая std::type_info

Часто отключаются на этапе компиляции *(опция -fno-rtti)*, т.к. радикально снижают производительность

# Рефлексия в Java

**`java.lang.Class`** - тип данных, описывающий Runtime-информацию о конкретном классе

Для каждого класса (не базового типа!) определено поле **`.class`**, хранящее ссылку на **`java.lang.Class`**

Для каждого объекта определен метод **`.getClass()`**, возвращающий ссылку на **`java.lang.Class`**

# Пример использования

```
// Без рефлексии
Foo foo = new Foo();
foo.hello();


// Рефлекисия
Class fooClass = Class.forName("полное.имя.Foo");
Object foo = fooClass.newInstance();
Method m = foo.getClass().getDeclaredMethod("hello", new Class<?>[0]);
m.invoke(foo);
```

# Что можно выяснить про объект

```
String getCanonicalName()

// Обращение к public полю или методу объекта, включая
// классы-родители
Field getField(String name)
Method getMethod(String name, Class<?>... parameterTypes)

// Обращение к public полю или методу объекта конкретного класса
Field getDeclaredField(String name)
Method getDeclaredMethod(String name, Class<?>... parameterTypes)
```

# Что можно выяснить про объект

```
// Обращение к public полю или методу объекта, включая
// классы-родители
Field[] getFields()
Method[] getMethod()

// Обращение к public полю или методу объекта конкретного класса
Field[] getDeclaredFields()
Method[] getDeclaredMethods()
```

# Ресурсные файлы внутри .jar

```
// Загрузка произвольных данных из classpath

URL getResource(String name)
InputStream getResourceAsStream(String name)
```

# Загрузка классов

java.lang.ClassLoader - абстрактный класс для загрузки произвольных классов

java.lang.URLClassLoader - его стандартная реализация в JVM

```
public static ClassLoader ClassLoader.getSystemClassLoader()
```

# Динамическая загрузка классов из внешних файлов

1. Создать новый загрузчик классов (их может быть несколько)

```
static URLClassLoader URLClassLoader.newInstance(URL[] urls)
```

2. Найти и загрузить произвольный класс по имени

```
Class<?> ClassLoader.loadClass(String name)
                    throws ClassNotFoundException
```

# Ключевой вопрос - зачем это нужно?

Загрузка backend'ов различных фреймворков:

slf4j -> logback

JDBC -> mysql, sqlite и т.д.

Загрузка plugin'ов в зависимости от сценариев использования.