NeRd Talk vol 1

Reflection

Introduction

What is reflection in computer programming?

In computer science, reflection is the ability of a process to examine, introspect, and modify its own structure and behavior.
[J.Malenfant]

Historical Background

According to the definition what's the first programming language allowing reflection?

- Assembler is the first programming language using functions as argument for other functions

Historical Background

Assembler had the reflection capability as requirement by design

First compiled languages hadn’t this capability included in first place

Reflection appeared again in 3-Lisp in 1982

Historical Background

Reflection is a key component for Metaprogramming

Most of the modern languages provide structures to implement Metaprogramming

Main Concepts

Meta Functions

Functions that allows the inspection of other function

Main Concepts

Meta Functions

- Functions to obtain info about a class
- Functions to obtain info about a method
- Functions to instantiate an object a run time
- Functions to call a static or dynamic method

Main Concepts

- All these functions have classes or methods as input and operate as what is called a Meta Object
- A Meta Object allows us to obtain info about the class or object we want to inspect at runtime
- A Meta Object implements all the functions showed previously

Implementation Java

```
import java.lang.reflect.Method;

// Without reflection
Foo foo = new Foo();
foo.hello();

// With reflection
try{

    // Alternatively: Object foo = Foo.class.newInstance();
    Object foo = Class.forName("complete.classpath.and.Foo").newInstance();

    Method m = foo.getClass().getDeclaredMethod("hello", new Class<?>[0]);
    m.invoke(foo);

}catch(Exception e){

    //Catching ClassNotFoundException, NoSuchMethodException
    //InstantiationException, IllegalAccessException

}
```

Implementation PHP

```
// Without reflection
$foo = new Foo();
$foo->hello();


// With reflection, using Reflections API
$reflector = new ReflectionClass('Foo');
$foo = $reflector->newInstance();

$hello = $reflector->getMethod('hello');
$hello->invoke($foo);
```

Implementation Qt/C++

```
#include <QMetaType>

// Without reflection
Foo *foo = new Foo();
foo->hello();


// With reflection
QMetaObject metaObj = Foo::staticMetaObject;

Foo * foo = qobject_cast<Foo*>(metaObj.newInstance());
metaObj.invokeMethod(foo,"hello");
```

Use case: Atena

Scenario:

- The Bot API resolver
- Centralized on a method with a switch (100+ codelines)

Use case: Atena

```
NRHttpResponse * resolveRequest(NRHttpRequest * request)
{
     NRHttpResponse *response;

     switch (request->name())
     {
          case NRHttpRequest_Login:
               response = new NRHttpLoginResponse();
          ...
          break;

          case NRHttpRequest_ListFeatures:
               response = new NRHttpListFeaturesResponse();
          ...
          break;

          case NRHttpRequest_DetailFeature:
               response = new NRHttpDetailFeatureResponse();
          ...
          break;

          ...

          default:
          response = new NRHttpErrorResponse();
          ...
          }

     return response;
}
```

Use case: Atena

Problems:

- The switch will grow as we had new API support

Use case: Atena

```
Qmap<NRHttpRequestType, QmetaObject> NRHttpResolver::mapping()
{
    if(_mapping.isEmpty())
    {
        _mapping.insert(NRHttpRequest_Login,
                    NRHttpLoginResponse::staticMetaObject);

        _mapping.insert(NRHttpRequest_ListFeatures,
                NRHttpListFeaturesResponse::staticMetaObject);

        _mapping.insert(NRHttpRequest_DetailFeature,
            NRHttpDetailFeatureResponse::staticMetaObject);

        _mapping.insert(NRHttpRequest_BadRequest,
            NRHttpErrorResponse::staticMetaObject);
    }
    return _mapping;
}
```

Use case: Atena

```
NRHttpResponse * NRHttpResolver::resolveRequest(NRHttpRequest * request)
{

    QMetaObject metaObj = this->mapping().value(request->name())
    if(this->mapping().contains(request->name()))
    {
        metaObj = this->mapping().value(request->name())
    }else
    {
        metaObj = this->mapping().value(NRHttpRequest_BadRequest)
    }

    NRHttpResponse *response;

    QgenericArgument requestArg = Q_ARG(NRHttpRequest*, request);

response = qobject_cast<NRHttpResponse*>(metaObj.newInstance(request,
                                                             arg));


    return response;
}
```

Conclusions

- Pro
  - Makes your code mode readable
  - Makes your code more flexible
- Cons
  - Stong language dependency
- Side effects
  - Can make you bypass access modificators

Meta Question Time