C++反射技术实现

0. 引言

反射是现代程序设计中的重要机制，通过反射，可以在运行时获得程序中每一个类型（包括类、结构、委托、接口和枚举等）的成员，包括方法、属性、事件,以及构造函数等。还可以获得每个成员的名称、限定符和参数等。有了反射，即可对每一个类型了如指掌。如果获得了构造函数的信息，即可直接创建对象，即使这个对象的类型在编译时还不知道。在许多高级语言当中，例如 java 等，语言本身就支持反射。本文将使用 C++语言实现一个类似 java 当中的反射框架。

1. 相关工作

C++语言作为精密紧密接触与操作系统的语言，因为其高效率所以一直被广泛使用，但是其语言特性并不支持反射技术，所以只能通过运用一些设计模式实现一些比较简单的反射技术框架。通过参考一些博客，目前通过 C++实现反射主要通过函数回调的方式来实现，即每一个类自己定义一个能够创建自己实例的函数，将该函数注册到工厂中完成反射。

2. 通过类名实例化对象的反射技术

通过类名实例化对象看上去的使用方式应该是这样子的：

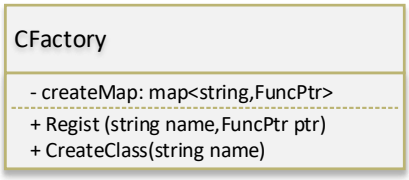
void* ptr = CreateObject("Test");

通过 CreateObject 传入参数，返回一个 Test 类型的对象。

这里肯定就要用到工厂模式，建立一个简单工厂，使用 Switch 控制语句就可以完成这个功能：

```
class CFactory {
public:
    void* CreateClass(string name) {
        switch (name)
        {
        case "Test":
            return new Test();
        default:
            break;
        }
        return NULL;
    }
};
```

虽然完成了我们的目标但是这个存在很大的问题，每次添加新类都要添加一个 case，这里可能想到用抽象工厂，这样我们就可以为每个类实现一个生成自己的工厂类，为了简化这个过程，我们使用函数在工厂上注册生成自己的函数来完成这个过程，那么工厂当中就只需要记录不同类名对应的生成函数就可以了。如下图 Factory：



通过每个类调用 Regist 来注册对应的名字和创建函数。

那么下一个问题就是类何时来注册自己呢，如果全部写在 main 函数当中对于每一个添加的类都得有一条语句加入到 main 函数当中。这里我们可以使用全局静态对象的构造函数来完成这个注册，通过在类下声明一个全局的静态对象，对象将在程序被加载的时候构造，通过构造函数就可以注册了。(注：由于全局静态对象构造顺序的不确定性，可能导致全局静态对象构造时候工厂构造尚未开始，所以需要使用 lazySingleton)。

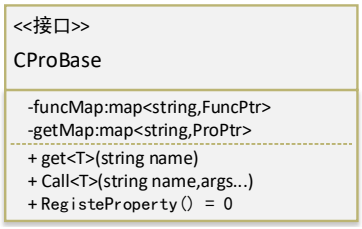
到这里基本通过类名生成指定对象就完成了，为了使得代码能够重用，减少程序员工作量，可以使用宏定义来替换每个类的实例化函数定义。

3. 通过属性名或方法名反射对象属性或者对象方法

通过类名创建的对象一般情况我们没有办法调用该对象的属性或者方法，除非强制转换为对应的类型，但是要强制转换的化就失去了反射的意义，因为我们不得不在使用该类的地方引入该类的头文件，并且使用也与类型相关了，所以还得完成属性与方法的反射才能更好的使用反射技术。
这种反射使用如下：

```
CProBase* ptr = FactoryCreate("Test"); // 根据类名创建类型
ptr->get<string>("str") = "666"; // 反射获取Test对象的str (string)属性
ptr->get<int>("a") = 15;          // 反射设置Test对象的a (int)属性
cout << ptr->get<string>("str") << " " << ptr->get<int>("a") << endl;  // 输出：666 15
ptr->Call<void>("fun",123);   // 反射调用 Test 对象的 fun 方法 返回为 void 传入参数 123
```

其中 CProBase 是反射对象的接口类型：
根据上面的使用场景来看，所有需要被反射的对象都需要继承于一个反射接口，那么这样一个反射接口定义如下：



首先是属性反射，根据类名反射的经验，我们在基类当中定义一个名字到属性指针的反射，因为类型的不确定性，所以我们统一采用 void*指针方便管理，在调用的时候在通过 c++模板方法转换为对应的类型。纯虚方法用于子类实现，来注册属性和方法。

这里有一个主要的问题就是类成员方法指针无法确定类型（返回类型和参数类型），而且类成员方法地址没办法直接转变为 void*指针，所以这里我们使用与类名注册相似的办法，如下：

```
static auto funPtr = &Test::fun;
funcMap.insert(std::pair<string, CallFun>("fun", &funPtr));
```

我们通过声明一个全局的静态变量指针，并且使用 C++11 auto 根据具体的类型来推导指针类型，然后我们的 map 里面存放该指针的地址（二重指针），指针的地址就很容易转变为 void*指针了。在调用的时候我们再通过将 void*地址转换为指向类成员方法的指针的指针，然后解引用调用，传入 this 指针即可。
关于参数我是直接使用的 C++11 的变长参数模板来传递调用的参数，具体实现如下：

```
template<typename ReturnType, typename ClassType, typename ...Args>
ReturnType CProBase::Call(string func_name, Args ...args)
{
    if (funcMap.count(func_name)) {
        ReturnType(ClassType::*func)(Args...) =
*(ReturnType(ClassType::**)(Args...))funcMap[func_name];
        ClassType* _this = static_cast<ClassType*>(this);
        return (_this->*func)(args...);
    }
    throw Error_Function_Name();
}
```

代码：见附录

参考文献：
[1].反射技术，百度百科 https://baike.baidu.com/item/%E5%8F%8D%E5%B0%84%E6%8A%80%E6%9C%AF/4483623?fr=aladdin
[2].C++反射机制的实现：https://blog.csdn.net/cen616899547/article/details/9317323