**C++ 反射：从设计到实现**

**介绍**

**概念**

反射机制允许程序在运行时借助Reflection API取得任何类的内部信息，并能直接操作对象的内部属性和方法

**问题**

1. C++ 不支持反射
2. 很多业务场景需要依赖反射机制，比如：RPC、WEB MVC、对象序列化等

**目标**

1. 类对象反射
2. 类成员数据反射
3. 类成员函数反射

**演示**

类定义：

|  |
| --- |
| C++ class A : public Object { public:  A() {}  ~A() {}   void f1()  {  std::cout << "f1" << std::endl;  }   void f2()  {  std::cout << "f2" << std::endl;  }  public:  string m\_name;  int m\_age; };  REGISTER\_CLASS(A); REGISTER\_CLASS\_FIELD(A, m\_name, string); REGISTER\_CLASS\_FIELD(A, m\_age, int); REGISTER\_CLASS\_METHOD(A, f1, void, void); REGISTER\_CLASS\_METHOD(A, f2, void, void); |

三种类型的宏

1、类对象注册宏：

|  |
| --- |
| C++ REGISTER\_CLASS(A); |

2、类成员数据注册宏：

|  |
| --- |
| C++ REGISTER\_CLASS\_FIELD(A, m\_name, string); |

3、类成员函数注册宏：

|  |
| --- |
| C++ REGISTER\_CLASS\_METHOD(A, f1, void, void); |

反射用法：

|  |
| --- |
| C++ int main() {  // 类对象反射  ClassFactory \* factory = Singleton<ClassFactory>::instance();  Object \* a = factory->create\_class("A");    // 类成员数据反射  string name;  a->get\_field\_value("m\_name", name);  a->set\_field\_value("m\_name", "kitty");    int age;  a->get\_field\_value("m\_age", age);  a->set\_field\_value("m\_age", 30);   // 类成员函数反射  a->call("f1");  a->call("f2");   return 0; } |

**类对象反射**

简单地说就是程序运行时读进来一个字符串，然后程序就会自动创建出字符串对应名字的类对象

|  |
| --- |
| C++ class A {}; class B {};  void \* create\_class(const string & className) {  if (className == "A")  {  return new A();  }  else if (className == "B")  {  return new B();  }  else  {  return nullptr;  } } |

**类成员数据反射**

关键问题：如何获取类成员数据的内存地址偏移量（offset）

黑科技1：

|  |
| --- |
| C++ #define OFFSET(className, fieldName) \  ((size\_t)(&((className \*)0)->fieldName)) |

黑科技2：

|  |
| --- |
| C++ Test t; size\_t offset = (size\_t)(&(t.member)) - (size\_t)(&t); |

**类成员函数反射**

类成员函数指针：

|  |
| --- |
| C++ typedef std::function<void(Test \*)> test\_method; method method = &Test::f1;  Test \* t = new Test(); method(t);  uintptr\_t ptr = (uintptr\_t)&method; (\*(test\_method \*)(ptr))(t); |