C++编译器优化技术：RVO、NRVO和复制省略

现代编译器缺省会使用RVO（return value optimization，返回值优化）、NRVO（named return value optimization、命名返回值优化）和复制省略（Copy elision）技术，来减少拷贝次数来提升代码的运行效率

注1：vc6、vs没有提供编译选项来关闭该优化，无论是debug还是release都会进行RVO和复制省略优化

注2：vc6、vs2005以下及vs2005+ Debug上不支持NRVO优化，vs2005+ Release支持NRVO优化

注3：g++支持这三种优化，并且可通过编译选项：-fno-elide-constructors来关闭优化

RVO

```
#include <stdio.h>
class A
{
public:
    A()
    {
        printf("%p construct\n", this);
    }
    A(const A& cp)
    {
        printf("%p copy construct\n", this);
    }
    ~A()
    {
        printf("%p destruct\n", this);
    }
};

A GetA()
{
    return A();
}

int main()
{
    {
        A a = GetA();
    }

    return 0;
}
```

在g++和vc6、vs中，上述代码仅仅只会调用一次构造函数和析构函数，输出结果如下：

```
0x7ffe9d1edd0f construct
0x7ffe9d1edd0f destruct
```

在g++中，加上-fno-elide-constructors选项关闭优化后，输出结果如下：

```
0x7ffc46947d4f construct  // 在函数GetA中，调用无参构造函数A()构造出一个临时变量temp
0x7ffc46947d7f copy construct // 函数GetA return语句处，把临时变量temp做为参数传入并调用拷贝构造函数A(const A& cp)将返回值ret构造出来
0x7ffc46947d4f destruct // 函数GetA执行完return语句后，临时变量temp生命周期结束，调用其析构函数~A()
0x7ffc46947d7e copy construct // 函数GetA调用结束，返回上层main函数后，把返回值变量ret做为参数传入并调用拷贝构造函数A(const A& cp)将变量A a构造出来
0x7ffc46947d7f destruct // A a = GetA()语句结束后，返回值ret生命周期结束，调用其析构函数~A()
0x7ffc46947d7e destruct // A a要离开作用域，生命周期结束，调用其析构函数~A()
```

注：临时变量temp、返回值ret均为匿名变量

下面用c++代码模拟一下其优化行为：

```
#include <new>
A& GetA(void* p)
{
    //由于p的内存是从外部传入的，函数返回后仍然有效，因此返回值可为A&
    //vs中，以下代码还可以写成：
    // A& o = *((A*)p);
    // o.A::A();
    // return o;
    return *new (p) A(); // placement new
}

int main()
{
    {
        char buf[sizeof(A)];
        A& a = GetA(buf);
        a.~A();
    }

    return 0;
}
```

NRVO

g++编译器、vs2005+ Release（开启/O2及以上优化开关）

修改上述代码，将GetA的实现修改成：

```
A GetA()
{
    A o;
    return o;
}
```

在g++、vs2005+ Release中，上述代码也仅仅只会调用一次构造函数和析构函数，输出结果如下：

```
0x7ffe9d1edd0f construct
0x7ffe9d1edd0f destruct
```

g++加上-fno-elide-constructors选项关闭优化后，和上述结果一样

```
0x7ffc46947d4f construct
0x7ffc46947d7f copy construct
0x7ffc46947d4f destruct
0x7ffc46947d7e copy construct
0x7ffc46947d7f destruct
0x7ffc46947d7e destruct
```

但在vc6、vs2005以下、vs2005+ Debug中，没有进行NRVO优化，输出结果为：

```
18fec4 construct  // 在函数GetA中，调用无参构造函数A()构造出一个临时变量o
18ff44 copy construct  // 函数GetA return语句处，把临时变量o做为参数传入并调用拷贝构造函数A(const A& cp)将返回值ret构造出来
18fec4 destruct  // 函数GetA执行完return语句后，临时变量o生命周期结束，调用其析构函数~A()
18ff44 destruct // A a要离开作用域，生命周期结束，调用其析构函数~A()
```

下面用c++代码模拟一下vc6、vs2005以下、vs2005+ Debug上的行为：

```
#include <new>
A& GetA(void* p)
{
    A o;
    //由于p的内存是从外部传入的，函数返回后仍然有效，因此返回值可为A&
    //vs中，以下代码还可以写成：
    // A& t = *((A*)p);
    // t.A::A(o);
    // return t;
    return *new (p) A(o); // placement new
}

int main()
{
    {
        char buf[sizeof(A)];
        A& a = GetA(buf);
        a.~A();
    }

    return 0;
}
```

注：与g++、vs2005+ Release相比，vc6、vs2005以下、vs2005+ Debug只优化掉了返回值到变量a的拷贝，命名局部变量o没有被优化掉，所以最后一共有2次构造和析构的调用

复制省略

典型情况是：调用构造函数进行值类型传参

```
void Func(A a)
{
}

int main()
{
    {
        Func(A());
    }

    return 0;
}
```

在g++和vc6、vs中，上述代码仅仅只会调用一次构造函数和析构函数，输出结果如下：

```
0x7ffeb5148d0f construct
0x7ffeb5148d0f destruct
```

在g++中，加上-fno-elide-constructors选项关闭优化后，输出结果如下：

```
0x7ffc53c141ef construct   // 在main函数中，调用无参构造函数构造实参变量o
0x7ffc53c141ee copy construct // 调用Func函数后，将实参变量o做为参数传入并调用拷贝构造函数A(const A& cp)将形参变量a构造出来
0x7ffc53c141ee destruct // 函数Func执行完后，形参变量a生命周期结束，调用其析构函数~A()
0x7ffc53c141ef destruct // 返回main函数后，实参变量o要离开作用域，生命周期结束，调用其析构函数~A()
```

下面用c++代码模拟一下其优化行为：

```
void Func(const A& a)
{
}

int main()
{
    {
        Func(A());
    }

    return 0;
}
```

优化失效的情况

开启g++优化，得到以下各种失效情况的输出结果：

（1）根据不同的条件分支，返回不同变量

```
A GetA(bool bflag)
{
    A a1, a2;
    if (bflag)
        return a1;
    return a2;
}

int main()
{
    A a = GetA(true);

    return 0;
}
```

```
0x7ffc3cca324f construct
0x7ffc3cca324e construct
0x7ffc3cca327f copy construct
0x7ffc3cca324e destruct
0x7ffc3cca324f destruct
0x7ffc3cca327f destruct
```

注1：2次缺省构造函数调用：用于构造a1、a2

注2：1次拷贝构造函数调用：用于拷贝构造返回值

注3：这儿仍然用右值引用优化掉了一次拷贝函数调用：返回值赋值给a

（2）返回参数变量

（3）返回全局变量

（4）返回复合数据类型中的成员变量

（5）返回值赋值给已构造好的变量（此时会调用operator==赋值运算符）

参考

Return Value Optimization

What are copy elision and return value optimization?

Copy elision（wiki）

C++ 命名返回值优化（NRVO）

Named Return Value Optimization in Visual C++ 2005