临时性对象(Temporary Objects)

何时生成临时对象

对于一个下面这样的程序片段：

```
T a, b;T c=a+b;
```

死板一点来讲，它应当产生一个临时对象用来存储 a+b 的结果，然后以临时对 象作为初值调用拷贝构造函数初始化对象 c。而实际上编译器更愿意直接调用 拷贝构造函数的方式将 a+b 的值放到 c 中，这样就不需要临时对象，和它的构造 函数和拷贝构造函数的调用了。

更进一步，如果 operator +的定义符合 NRV 优化的条件，那么 NRV 优化的开启， 将使得拷贝构造函数的调用和 named object 的析构函数都免了。期间详情可 以参见"NRV 优化"。也就是说对于上面那种情形在我们的代码中是不产生 临时对象的。但是对于一个情况非常类似的赋值操作语句 `c = a+b`，却有很 大的差别，那个临时变量是不能省的

不能忽略临时对象，反而导致如下过程：

```
// Pseudo C++ code  // T temp = a + b;  T temp; a.operator+( temp, b ); //
@1 [^注 1]// c = temp  c.operator =( temp ); // @2  temp.T::~T();
```

在代码@1 处，表明以拷贝构造函数或 NRV 方式将结果保存的临时对象中。为什 么不能省略那个临时对象，比如直接这样：

```
c.T::~T();c.T::T(a+b);
```

这不是更高效，更简洁的方式吗？不行，其原因在于，拷贝构造函数、析构 函数以及赋值操作符都可以由使用者提供，没有人能保证，析构函数加拷贝 构造函数的组合和赋值操作符具有相同的含义。所以：`T c=a+b` 总是比 `c = a + b` 更有效率。

对于一个没有出现目标对象的表达式 `a + b`,那么产生一个临时对象来存储 运算结果，则是非常必要的。

临时对象的生命周期

很多时候，产生临时对象是必不可少的，但是何时摧毁一个临时对象才是最 佳行为呢？过早或过晚都不太适合，过早有可能使得程序错误，过晚的话又 使得资源没有得到及时回收。对于下面的程序：

```
string s1("hello "), s2("world "),s3("by Adoo");std::cout<<s1+s2+s3<<std::endl;
```

显然保存 `s1+s2` 结果的临时对象，如果在与 s3 进行加法之前析构，将会带来 大麻烦。于是 C++标准中有一条：

- 临时性对象的摧毁应当作为造成产生这个临时对象的完整表达式的最后 一个步骤。

完整的表达式，是指涵括的表达式中最外围的那个。我们再看上面那个字符 串相加的表达式，当计算完成，而 cout 还未调用，此时我们析构掉存储最终 结果的临时对象，岂不悲剧。其实上面的规定还有两个例外：

1. 凡含有表达式执行结果的临时性对象，应该保存到 Object 的初始化操作 完成为止。

2. 如果临时性对象被绑定与一个引用，临时对象将残留，直至被初始化的 引用的生命结束，或直到临时对象的生命周期结束——视哪一种情况先达 到，对应于这种情况：

```
::string s1("hello ");::string &s=s1+"world";
```

侯捷认为此处为 Lippman 的错误，他认为应该为 `temp.operator + ( a, b )`但我以为是侯捷并没有理解 Lippman 的意思，回 顾一下,《深度探索对象模型》2.3 讲到的返回值初始化(Return Value Initialization)——返回值将作为一个额外的参数提供给函数，来传回函数内 部的值，也就是说对于一个 operator + 操作符 `T T::operator+ (const T& right)`将转化为 `void T::operator+ (T &result ,const T& right)`所以 `temp=a+b` 是 `a.operator+( temp, b )`还是 `temp.operator+( a, b )`自然不言而喻。