模板二事

模板的实例化

一个模板只有被使用到，才会被实例化，否则不会被实例化。对于一个实例化 后的模板来说，未被调用的成员函数将不会被实例化，只有成员函数被使用时， C++标准才要求实例化他们。其原因，有两点：

- 空间和时间效率的考虑，如果模板类中有 100 个成员函数，对某个特定类型 只有 2 个函数会被使用，针对另一个特定类型只会使用 3 个，那么如果将剩 余的 195 个函数实例化将浪费大量的时间和空间。
- 使模板有最大的适用性。并不是实例化出来的每个类型都支持所有模板的全 部成员函数所需要的运算符。如果只实例化那些真正被使用的成员函数的话， 那么原本在编译期有错误的类型也能够得到支持。

可以明确的要求在一个文件中将整个类模板实例化：

```
template class Point3d<float>;
```

也可以显示指定实例化一个模板类的成员函数：

```
template float Point3d<float>::X() const;
```

或是针对一个模板函数：

```
template Point3d<float> operator+(
    const Point3d<float>&, const Point3d<float>& );
```

模板的错误报告，使用模板并遇到错误的大概都深有体会，那就是一个灾难。

模板的名称决议

一开始先要区分两种意义,一种是 C++ 标准所谓的“scope of the template definition”，直译就是“定义模板的范围”。另一种是 C++标准所谓的“scope of the temlate instantiation”，可以直译为“实例化模板的范围”。

第一种情况：

```
// scope of the template definitionextern double foo ( double );template < class type > class ScopeRules { public:
void invariant() {
_member = foo( _val );
}
type type_dependent() {
return foo( _member );
}
// ... private:
int _val;
type _member; };
```

第二种情况:

```
//scope of the template instantiation extern int foo( int ); // ... ScopeRules< int > sr0; sr0.invariant();sr0.type_dependent();
```

在"scope of the template instantiation "中 两个 foo()都声明在此 scope 中。猜猜 sr0.invariant() 中调用的是哪个 foo()函数，出乎意料，实际调用的 是:

```
extern double foo ( double );
```

看上去，应该调用:

```
extern int foo( int );
```

毕竟，_val 的类型是 int 类型，它们才完全匹配。而 sr0.type_dependent() 中调用的却在我们意料之中，调用的是:

```
extern int foo( int );
```

诸上所述,看上去或合理或不合理的选择，原因在于:

> *template 之中， 对于一个非成员名字的决议结果是根据这个 name 的使用是否与"用以实例化该模板的参数类型"有关来决定 name。如果其使用互 不相干，那么就以"scope of the template dclaration"来决定 name。如果其 使用的互相关联，那么就以"scope of the template instantiation"来决定 name.*

对于上面这一段话我的理解比较粗鲁且直接：在模板中，一个非成员名字的决议 在于它适不适合在当前决议，当它完全与实例化模板的参数的类型无关的时候， 就可以在当前决议下来；如果有关的话，则认为不适合在当前决议下来，将被推 迟到实例化这个模板实例化的时候来决议。为什么以与实例化的类型相关不相关 来区别适不适合当前决议？一个与实例化类型无关的名字，如果推迟到实例化的 时候来决议，将使模板的设计者无所适从，一个模板的设计者能容忍一个与实例 化类型无关的名字在他的模板中表现出当前不具有的含义吗？当然不行，那种场 面，估计没有一个模板设计者能够 hold 住。相反，对于一个与实例化类型有关的 名字，天生就应该可以根据实例化模板的不同类型表现出不同含义，如果其名字 早在模板定义时被决议出来，那就该轮到模板的使用者 hold 不住了。当然所上完 全属一家之言，呸，连一家之言都不算，怎么敢自称"家"。如有不同理解，可当 我一派胡言，如果你聊发善心，可以对我赐教一二，当聆听受教。

参考：深度探索 C++对象模型