auto_ptr指针介绍

auto_ptr是这样一种指针：它是“它所指向的对象”的拥有者。这种拥有具有唯一性，即一个对象只能有一个拥有者，严禁一物二主。当auto_ptr指针被摧毁时，它所指向的对象也将被隐式销毁，即使程序中有异常发生，auto_ptr所指向的对象也将被销毁。

设计动机

在函数中通常要获得一些资源，执行完动作后，然后释放所获得的资源，当程序员忘记释放所申请的到的资源，或者由于异常发生而没有正常释放资源时，这就将产生一系列的内存泄漏问题。因此，你可能将函数写成如下形式：

```
void fun()
{
	try {
		T *ptr = new T;
		......
	}catch(...){
		delete ptr;
		throw;
	}
	delete ptr;
}
```

这样使程序变得太过复杂。使用auto_ptr可以使上述程序简化为：

```
void fun()
{
	auto_ptr<T>ptr(new T);
	......
}
```

不再需要delete，也不再需要catch了。与上面的程序相比，这个非常简单。

auto_ptr是一个模板类，适合于任何类型，其接口行为与普通指针相似，opeator *用来提取其所指向对象的值。operator->用来指向对象中的成员。但是，他没有定义任何的算术运算（包括++）。

由于auto_ptr定义中“用一般指针构造一个auto_ptr”的构造函数被声明为explicit（拒绝隐式变换），所以一下的方式是错误的：

```
auto_ptr<int> ptr = new int(0); // 错
```

必须这样：

```
auto_ptr<int> ptr(new int(0)); //正确
```

auto_ptr拥有权的转移

由于auto_ptr指针唯一性，即一个对象只能有一个auto_ptr指针所指向它。因此，当auto_ptr以传值方式被作为参数传递给某函数时，这时对象的原来拥有者（实参）就放弃了对象的拥有权，把它转移到被调用函数中的参数（形参）上，如果函数不再将拥有权传递出去，由于形参的作用域仅仅为函数内部，当函数退出时，它所指向的对象将被销毁。当函数返回一个auto_ptr时，其拥有权又被转移到了调用端。因此，应该尽量不要在参数中使用auto_ptr指针，我们也不要通过引用传递auto_ptr。

当然一个方法是我们可以通过使用常量引用来实现auto_ptr的传递，我们也可以声明一个const auto_ptr指针，这样将使auto_ptr不能转移它的拥有权。例如：

```
void print(auto_pt<int> ptr);
const auto_ptr<int> p(new int(0));
*p = 10;
print(p); //编译时发生错误
```

当auto_ptr被使用最为一个类成员时，由于其可能发生拥有权转移的问题，所以你必须亲自写复制构造函数和复制运算操作符。

auto_ptr的错误运用

- auto_ptr之间不能共享一个对象。一个auto_ptr不能指向另一个auto_ptr所指向的的对象。否则，当第一个指针删除该对象时，林一个指针马上就指向了一个已经被销毁的对象，那么，如果在使用该指针，就会引发某些错误。
- 并不存在针对array而设计的auto_ptr。auto_ptr不可以指向一个array，因为auto_ptr是通过delete而不是delete[]来释放其所拥有的资源的。

- auto_ptr绝非是一个通用的智能指针。并非任何适用智能型指针地方都可以适用auto_ptr指针。
- auto_ptr不满足STL容器对其元素的要求。因为在拷贝和赋值动作以后，原本的auto_ptr和新产生的auto_ptr并不相等，在拷贝和赋值之后，原本的auto_ptr会交出拥有权，而不是拷贝给新的auto_ptr。因此，决不要将auto_ptr作为标准容器的元素。