# 头文件

## #include <netdb.h>

//Unix和Linux特有的头文件，主要定义了与网络有关的结构、变量类型、宏、函数等

## #include <sys/types.h>

//是Unix/Linux系统的基本系统数据类型的头文件，含有size\_t，time\_t，pid\_t等类型

## #include <ev.h>

## #include <boost/mpl/xxx>

MPL（Meta-Programming Library）是由David Abrahams和Aleksey Gurtovoy为方便模板元编程而开发的库，2003年被Boost吸纳为其中的一员，此后又历经一些大幅度修改，目前已经相当完善，其最新版本于2004年11月发布。MPL的出现是C++模板元编程发展中的一大创举，它提供了一个通用、高层次的编程框架，其中包括了序列（Sequence）、迭代器（Iterator）、算法（Algorithm）、元函数（Metafunction）等多种组件，具有高度的可重用性，不但提高了模板元编程的效率，而且使模板元编程的应用范围得到相当的扩展。

这里也要用到智能指针

## #include <cassert>

头文件assert.h（C的表达）的唯一目的是提供宏assert的定义。可以在程序中关键的地方使用这个宏来进行断言。如果一处断言被证明非真，我们希望程序在标准错误流输出一条适当的提示信息，并且使执行异常终止。

## #include <functional>

<functional>是C++标准库中的一个头文件，定义了C++标准中多个用于表示函数对象(function object)的类模板，包括算法操作、比较操作、逻辑操作；以及用于绑定函数对象的实参值的绑定器(binder)。这些类模板的实例是具有函数调用运算符(function call operator)的C++类，这些类的实例可以如同函数一样调用。不必写新的函数对象，而仅是组合预定义的函数对象与函数对象适配器(function object adaptor)，就可以执行非常复杂的操作

<https://blog.csdn.net/fengbingchun/article/details/78006735?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.channel_param&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.channel_param>

## #include <type\_traits>

（1）type\_traits可以在一定程度上消除 switch-case 或者 if-else语句，降低程序的复杂度

（2）可以在编译期就检查出是否是正确类型

## #include <typeinfo>

1.typeid是C++的关键字之一，等同于sizeof这类的操作符。

2.typeid操作符的返回结果是名为type\_info的标准库类型的对象的引用（在头文件typeinfo中定义）

3.C++并没有规定typeid实现标准，各个编译器可能会不一样。

4.编译器会为每一种typeid操作的类型生成一份保存在数据段的type\_info数据。

5.每种类型的type\_info数据长度依赖于类型名称，至少9个字节。

## #include <memory>

结合动态内存的运用，要用到智能指针std::shared\_ptr

## #include <cassert>

# 类

## Private\protected\public

私有成员变量或函数在类的外部是不可访问的，甚至是不可查看的。只有类和友元函数可以访问私有成员。

保护成员变量或函数与私有成员十分相似，但有一点不同，保护成员在派生类（即子类）中是可访问的。

公有成员在程序中类的外部是可访问的。

# 网络

# 其他

## 智能指针shared\_ptr

shared\_ptr是一个引用计数智能指针，用于共享对象的所有权也就是说它允许多个指针指向同一个对象。

shared\_ ptr是一种智能指针(smart pointer)，作用有如同指针，但会记录有多少个shared\_ ptrs共同指向一个对象。这便是所谓的引用计数(reference counting)，比如我们把只能指针赋值给另外一个对象，那么对象多了一个智能指针指向它，所以这个时候引用计数会增加一个，我们可以用shared\_ ptr.use\_ count(函数查看这个智能指针的引用计数，一但最后一个这样的指针被销毁，也就是一旦某个对象的引用计数变为0，这个对象会被自动删除当我们程序结束进行return的时候，智能指针的弓用计数会减1。

<https://blog.csdn.net/Xiejingfa/article/details/50750037?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.channel_param&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.channel_param>

## 名称空间namespace boost

## Typedef typename关键字

举例说明，vector::size\_type是vector的嵌套类型定义，其实际等价于 size\_t类型。 也就是说：

vector<int>::size\_type ssize;

*//就等价于*

size\_t ssize;

实际上，模板类型在实例化之前，编译器并不知道vector<T>::size\_type是什么东西，事实上一共有三种可能：静态数据成员、静态成员函数、嵌套类型。那么此时typename的作用就在此时体现出来了——定义就不再模棱两可。

所以根据上述两条分析，

typedef typename std::vector<T>::size\_type size\_type;

语句的真是面目是：   
typedef创建了存在类型的别名，而typename告诉编译器std::vector<T>::size\_type是一个类型而不是一个成员。