相比于0.3版本0.2版本下做了以下修改：

将所有的裸指针用智能指针替代 – shared\_ptr

利用weak\_ptr解决时间节点和HTTP类互指的问题（request）-- 及requestData中包含mtimer计时器，mtimer计时器中又包含了requestData请求

任务队列的任务结构从函数指针+参数指针转换为c++11的function

1. static关键字

静态成员变量：

静态数据成员被当作是类的成员。无论这个类的对象被定义了多少个，静态数据成员在程序中也只有一份拷贝，由该类型的所有对象共享访问。也就是说，静态数据成员是该类的所有对象所共有的。对该类的多个对象来说，静态数据成员只分配一次内存，供所有对象共用。所以，静态数据成员的值对每个对象都是一样的，它的值可以更新；

静态数据成员存储在全局数据区。静态数据成员定义时要分配空间，所以不能在类声明中定义。

静态成员函数：

但是与普通函数相比，[静态成员](https://baike.baidu.com/item/%E9%9D%99%E6%80%81%E6%88%90%E5%91%98)函数由于不是与任何的对象相联系，因此它不具有this指针。从这个意义上讲，它无法访问属于类对象的非静态数据成员，也无法访问非静态成员函数，它只能调用其余的静态成员函数。下面举个静态成员函数的例子。

1. shared\_ptr 和 weak\_ptr的区别与联系

shared\_ptr 控制对象的生命周期，shared\_ptr是强引用(想象成用铁丝绑住堆上的对象)，只要有一个指向x对象的shared\_ptr存在，该x对象就不会被析构。当指向对象x的最后一个shared\_ptr析构或者reset()时，x保证会被销毁。

weak\_ptr 不控制对象的生命周期，但是它知道对象是否还活着(想象成用棉线轻轻拴住堆上的对象)。如果对象还活着，那么它可以提升为有效的shared\_ptr；如果对象已经死了，提升会失败，返回一个空的shared\_ptr。提升/lock() 行为是线程安全的

shared\_ptr/weak\_ptr 的计数在主流平台上是原子操作，没有用锁，性能不俗

shared\_ptr/weak\_ptr 的线程安全级别与std::string 和 STL容器一样

1. enable\_shared\_from\_this（继承这个类 this指针就能变身为shared\_ptr）

struct requestData : public *std*::*enable\_shared\_from\_this*<requestData>

我们知道使用裸指针和使用shared\_ptr的最大区别就是方便我们对对象资源的管

理，具体体现在当我们在不同地方使用同一对象时不需要重复的new和delete，我们只需要使用shared\_ptr将对象引用次数记录即可，请注意这里的引用是强引用；因此当该对象生命周期结束时，析构函数只需要释放一次即可，有效的避免了一个对象的重复释放。

1. shared\_ptr<> lock

weak\_ptr是为了配合shared\_ptr而引入的一种智能指针，它更像是shared\_ptr的一个助手而不是智能指针，因为它不具有普通指针的行为，没有重载operator\*和->,它的最大作用在于协助shared\_ptr工作，像旁观者那样观测资源的使用情况.

用法:

 weak\_ptr被设计为与shared\_ptr共同工作，可以从一个shared\_ptr或者另一个weak\_ptr对象构造，获得资源的观测权。但weak\_ptr没有共享资源，它的构造不会引起指针引用计数的增加。

 使用weak\_ptr的成员函数use\_count()可以观测资源的引用计数，另一个成员函数expired()的功能等价于use\_count()==0,但更快，表示被观测的资源(也就是shared\_ptr的管理的资源)已经不复存在。

 weak\_ptr可以使用一个非常重要的成员函数lock()从被观测的shared\_ptr获得一个可用的shared\_ptr对象， 从而操作资源。但当expired()==true的时候，lock()函数将返回一个存储空指针的shared\_ptr.