1. 线程安全的对象生命期管理
2. Observer（观察者）设计模式

观察者模式定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主体对象。这个主体对象在发生状态变化时，会通知所有观察者对象，使他们能够自动更新自己。  
一个软件系统常常要求某一个对象的状态发生变化的时候，某些其他的对象作出相应的改变。做到这一点的设计方案有很多，但是为了使系统能够易于复用，应该选择低耦合度的设计方案。减少对象之间的耦合有利于系统的复用，但是同时设计师需要使这些低耦合度的对象之间能够维持行动的协调一致，保证高度的协作（Collaboration）。观察者模式是满足这一要求的各种设计方案中最重要的一种。



从上面的解释可以看出，观察者模式是非常适合客户-服务器模式的编程的。

1. 对象线程安全的定义：

多个线程同时访问时，都表现出正确的行为，无论操作系统以何种顺序调度线程，调用段无需额外的同步或其他协调操作。

1. 对象构造线程安全的要求只有一点：不要在创建对象的时候泄露this指针——不要在构造函数中注册任何回调，不要在构造函数的任何地方把this指针传给跨线程的对象。
2. 析构函数遇到多线程会存在的问题：

在一个对象被多个线程使用时，对象的销毁会出现会存在问题：产生多种竞态条件，有的线程可能在使用对象，有些线程可能却在销毁对象。即出现线程不安全问题。重点应关注析构函数的线程安全。

1. 作为对象成员的mutex并不能完全保证线程安全，如下代码：

Foo::~Foo(){ //释放对象状态

MutextLockGuard lock(mutex\_); //对Foo对象的临界资源加锁过程

//free internal state

}

void Foo::updata(){ //使用对象状态

MutextLockGuard lock(mutex\_);

//make use of internal state

}

有下面代码可能产生竞态条件：

//线程1 //线程2

extern Foo \*x; extern Foo \*x;

delete x; if (x){

x = NULL; x->update();

}

当线程1执行到析构函数并获得锁准备继续执行，线程2执行if(x)判断x不为空后阻塞准备或得x，当线程1执行完，线程2的结果是不可预见的。

另外如果同时读写同一个类的两个对象时也可能发生死锁。解决的办法有，保持相同的加锁顺序，即先加锁地址较小的mutex。

1. 面向对象程序设计中，对象的关系主要有三种：组合（composition）、聚合（aggregation）、关联（association）。（其实还有个依赖（dependency）），其中，聚合和关联都会存在线程不安全问题。
2. 上面的问题很大一部分原因是因为使用了原始指针，因为原始指针不能确保它指向的对象是否还存活着。
3. 使用shared\_ptr（强引用）和weak\_ptr（弱引用）来解决上面提到的问题。weak\_ptr虽然是弱引用，并不能增加引用计数，但是他可以知道一个对象是否还是活着的：如果还活着，那么他可以提升(使用weak\_ptr的lock()函数，该函数是线程安全的)成一个shared\_ptr，如果对象已经死了，提升会失败，返回一个空的shared\_ptr。
4. C++可能的内存问题有以下几大类：

a.缓存区溢出 b.空悬指针 c.重复释放 d.内存泄露 e.不配对的new[]/delete f.内存碎片

其中可以使用只能指针解决上面前五个问题。

1. shared\_ptr本身不是100%线程安全的，所以要在多线程中使用同一个shared\_ptr，正确的做法用mutex保护。

第二章 线程同步精要

第三章 多线程服务器的使用场合与常用编程模型

第四章 C++多线程系统编程精要

第五章 高效的多线程日志