https://www.cnblogs.com/LUO77/p/5771237.html

https://blog.csdn.net/Solstice/article/details/6206154

**1、智能指针**：

智能指针是在堆栈上声明的类模板，是一个栈对象，并可通过使用指向某个堆分配的对象的原始指针进行初始化。 在初始化智能指针后，它将拥有原始的指针。 这意味着智能指针负责删除原始指针指定的内存。 智能指针析构函数包括要删除的调用，并且由于在堆栈上声明了智能指针，当智能指针超出范围时将调用其析构函数；智能指针的设计原则是在内存和性能上尽可能高效；智能指针是一个可以像指针一样工作的对象，但是当它不再被使用时，可以自动删除动态分配的内存；智能指针背后的核心概念是动态分配内存的所有权；所有智能指针都重载了“operator->”操作符，直接返回对象的引用，用以操作对象。访问智能指针原来的方法则使用“.”操作符；

**C++ 标准库智能指针**

使用这些智能指针作为将指针封装为纯旧 C++ 对象 (POCO) 的首选项。

- unique_ptr
  只允许基础指针的一个所有者。 除非你确信需要 shared_ptr，否则请将该指针用作 POCO 的默认选项。 可以移到新所有者，但不会复制或共享。 替换已弃用的 auto_ptr。 与 boost::scoped_ptr 比较。 unique_ptr 小巧高效；大小等同于一个指针且支持 rvalue 引用，从而可实现快速插入和对 STL 集合的检索。 头文件： <memory>。 有关更多信息，请参见如何：创建和使用 unique_ptr 实例和unique_ptr 类。

- shared_ptr
  采用引用计数的智能指针。 如果你想要将一个原始指针分配给多个所有者（例如，从容器返回了指针副本又想保留原始指针时），请使用该指针。 直至所有 shared_ptr 所有者超出了范围或放弃所有权，才会删除原始指针。 大小为两个指针；一个用于对象，另一个用于包含引用计数的共享控制块。 头文件： <memory>。 有关更多信息，请参见如何：创建和使用 shared_ptr 实例和shared_ptr 类。

- weak_ptr
  结合 shared_ptr 使用的特例智能指针。 weak_ptr 提供对一个或多个 shared_ptr 实例拥有的对象的访问，但不参与引用计数。 如果你想要观察某个对象但不需要其保持活动状态，请使用该实例。 在某些情况下，需要断开 shared_ptr 实例间的循环引用。 头文件： <memory>。 有关更多信息，请参见如何：创建和使用共享 weak_ptr 实例和weak_ptr 类。

头文件：

```
#include <memory>
```

unique_ptr：独占式智能指针，保证同一时间内只有一个智能指针可以指向该对象；

shared_ptr：共享式智能指针，多个智能指针可以指向相同对象，该对象和其相关资源会在“最后一个引用被销毁”时候释放；

**2、std::bind()和std::function()**

https://blog.csdn.net/myan/article/details/5928531

https://blog.kedixa.top/2017/cpp-std-bind/comment-page-1/

c++11的特性，std::bind和std::function均为模板类；我理解的function是申明一种函数指针类型，bind函数是生成一种函数指针类型，bind和function配合使用，用来做回调函数；

a、bind预先绑定的参数需要传具体的变量或值进去，对于预先绑定的参数，是pass-by-value的，值传递；

b、对于不事先绑定的参数，需要传std::placeholders进去，从_1开始，依次递增。placeholder是pass-by-reference的，引用传递；

c、bind的返回值是可调用实体，可以直接赋给std::function对象；

d、对于绑定的指针、引用类型的参数，使用者需要保证在调用实体调用之前，这些参数是可用的；

e、类的this可以通过对象或者指针来绑定，绑定类成员函数时，需要传递类指针；

实现原理：

http://www.cnblogs.com/qicosmos/p/3723388.html

头文件：

```
#include <functional>
```

### 3：std::thread()

c++11特性，用于创建线程，适用于全局函数，类的静态函数，类的public函数和类的private函数；类的非静态函数时，需要传递类指针；

头文件：

```
#include <thread>
```

接口：

```
//构造函数
default thread() noexcept; //默认构造函数
initialization template <class Fn, class... Args>; //初始化构造函数
 explicit thread (Fn&& fn, Args&&... args);
//不允许拷贝构造，fn为类成员函数时，需要传递类指针

//获取线程ID
id get_id() const noexcept;

 //判断线程是否是可连接的
 bool joinable() const noexcept;

 //阻塞等待线程结束，释放线程资源
 void join();

 //分离线程,线程结束时,马上释放系统资源
 void detach();
```