参考文档：

<https://blog.csdn.net/xujiangdong1992/article/details/102857063>

底层原理：

Synchronized：底层使用指令码方式来控制锁的，映射成字节码指令就是增加来两个指令：monitorenter和monitorexit。当线程执行遇到monitorenter指令时会尝试获取内置锁，如果获取锁则锁计数器+1，如果没有获取锁则阻塞；当遇到monitorexit指令时锁计数器-1，如果计数器为0则释放锁

Lock：底层是CAS乐观锁，依赖AbstractQueuedSynchronizer类，把所有的请求线程构成一个CLH队列。而对该队列的操作均通过Lock-Free（CAS）操作。

**Synchronized和Lock比较**

* Synchronized是关键字，内置语言实现，Lock是接口。
* Synchronized在线程发生异常时会自动释放锁，因此不会发生异常死锁。Lock异常时不会自动释放锁，所以需要在finally中实现释放锁。
* Lock是可以中断锁，Synchronized是非中断锁，必须等待线程执行完成释放锁。
* Lock可以使用读锁提高多线程读效率。

参考文档

<https://www.cnblogs.com/handsomeye/p/5999362.html>

synchronized 非公平锁。不能实现读写锁（读读可以，读写、写写不行）。没有获取锁超时功能。不能中断。

lock 非公平锁，公平锁都可以。可以实现读写锁功能。可以设置超时。可以中断

　总结来说，Lock和synchronized有以下几点不同：

　　1）Lock是一个接口，而synchronized是Java中的关键字，synchronized是内置的语言实现；

　　2）synchronized在发生异常时，会自动释放线程占有的锁，因此不会导致死锁现象发生；而Lock在发生异常时，如果没有主动通过unLock()去释放锁，则很可能造成死锁现象，因此使用Lock时需要在finally块中释放锁；

　　3）Lock可以让等待锁的线程响应中断，而synchronized却不行，使用synchronized时，等待的线程会一直等待下去，不能够响应中断；

　　4）通过Lock可以知道有没有成功获取锁，而synchronized却无法办到。

　　5）Lock可以提高多个线程进行读操作的效率。

　　在性能上来说，如果竞争资源不激烈，两者的性能是差不多的，而当竞争资源非常激烈时（即有大量线程同时竞争），此时Lock的性能要远远优于synchronized。所以说，在具体使用时要根据适当情况选择。

**Synchronized原理**

moniter对象，每个对象都有一个monitor锁

方法ACC\_SYCHronized标志

指令：

moniterenter

moniterexit

锁计数器

Synchronized是通过对象内部的一个叫做监视器锁（monitor）来实现的，监视器锁本质又是依赖于底层的操作系统的Mutex Lock（互斥锁）来实现的。

而操作系统实现线程之间的切换需要从用户态转换到核心态，这个成本非常高，状态之间的转换需要相对比较长的时间，这就是为什么Synchronized效率低的原因。

因此，这种依赖于操作系统Mutex Lock所实现的锁我们称之为“重量级锁”。

Java SE 1.6为了减少获得锁和释放锁带来的性能消耗，引入了“偏向锁”和“轻量级锁”：锁一共有4种状态，级别从低到高依次是：无锁状态、偏向锁状态、轻量级锁状态和重量级锁状态。锁可以升级但不能降级。

阻塞线程，唤醒线程是，需要切换上下文，用户态核心态切换，是非常耗时的操作。

**java lock原理**

依赖java.util.concurrent.AbstractQueuedSynchronizer类，传说中的AQS

AbstractQueuedSynchronizer中抽象了绝大多数Lock的功能，而只把tryAcquire方法延迟到子类中实现。

CLH CLH(Craig, Landin, and Hagersten locks): 是一个自旋锁，能确保无饥饿性，提供先来先服务的公平性。

<https://www.jianshu.com/p/4682a6b0802d>