java锁有哪些种类，以及区别

1.公平锁/非公平锁

公平锁是指多个线程按照申请锁的顺序来获取锁

非公平锁是指多个线程获取锁的顺序并不是按照申请锁的顺序，有可能是申请的线程比先申请的线程优先获取锁。有可能，会造成优先级反转或者饥饿问题。

2.可重入锁

可重入锁又名递归锁，是指在同一个线程在外层获取锁的时候，在进入内层方法会自动优先获取锁。有可能，会造成优先级反转或者饥饿现象。Synchronized和ReentrantLock都是可重入锁。

对于Synchronized而言,也是一个可重入锁。可重入锁的一个好处是可一定程度避免死锁。

```
synchronized void setA() throws Exception{
    Thread.sleep(1000);
    setB();
}

synchronized void setB() throws Exception{
    Thread.sleep(1000);
}
```

上面的代码就是一个可重入锁的一个特点，如果不是可重入锁的话，setB可能不会被当前线程执行，可能造成死锁。

3.独享锁/共享锁（互斥锁/读写锁）

独享锁是指锁一次只能被一个线程所持有

共享锁是指该锁可被多个线程持有。

ReentrantLock是独享锁，ReadWriteLock，其读锁是共享锁，其写锁是独享锁

读锁的共享锁可保证并发读是非常高效的，读写、写读、写写的过程是互斥的。

Synchronized是独享锁。

分段锁：

分段锁其实是一种锁的设计，并不是具体的一种锁，对于concurrentHashMap而言，其并发的实现就是通过分段锁的形式来实现高效的并发操作。

concurrentHashMap中的分段锁称为segment，它类似于HashMap的结构，即内部是数组链表的形式，同时又是一个ReentrantLock。

当需要put元素的时候，并不是对整个hashmap进行加锁，而是先通过hashcode来知道他要放在哪一个分段里面，然后对这个分段进行加锁，所以当多线程put的时候，只要不是放在一个分段中，就实现了真正的并行插入。但在统计size的时候，可就是获取hashmap的全局信息，就需要获取所有的分段锁才能统计。

分段锁的设计目的是细化锁的粒度，当操作不需要更新整个数组的时候，就仅仅针对数组中的一项进行加锁操作。

偏向锁/轻量级锁/重量级锁：

这三种锁是指锁的状态，并且是针对Synchronized。在Java 5通过引入锁升级的机制来实现高效Synchronized。这三种锁的状态是通过对象监视器在对象头中的字段来表明的。

偏向锁是指一段同步代码一直被一个线程所访问，那么该线程会自动获取锁。降低获取锁的代价。

轻量级锁是指当锁是偏向锁的时候，被另一个线程所访问，偏向锁就会升级为轻量级锁，其他线程会通过自旋的形式尝试获取锁，不会阻塞，提高性能。
重量级锁是指当锁为轻量级锁的时候，另一个线程虽然是自旋，但自旋不会一直持续下去，当自旋一定次数的时候，还没有获取到锁，就会进入阻塞，该锁膨胀为重量级锁。重量级锁会让其他申请的线程进入阻塞，性能降低。

乐观锁和悲观锁（在mysql文件夹里面）

自旋锁：
在java中，自旋锁是指尝试获取锁的线程不会立即阻塞，而是采用循环的方法去尝试获取锁，这样的好处是减少线程上下文切换的消耗，缺点是循环会消耗cpu。