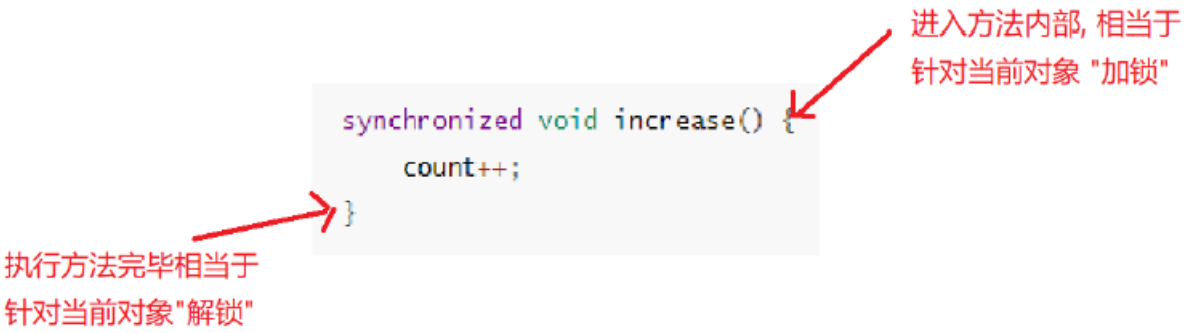
1、synchronized的特性

1, 互斥

synchronized 会起到互斥效果, 某个线程执行到某个对象的 synchronized 中时, 其他线程如果也执行到同一个对象 synchronized 就会阻塞等待.

- 进入 synchronized 修饰的代码块, 相当于~~加锁~~
- 退出 synchronized 修饰的代码块, 相当于 解锁

如果当前是已经加锁的状态,其他的线程就无法执行这里的逻辑,就只能阻塞等待.



synchronized的功能本质就是,把"并发"变成"串行".

synchtonized除了可以用来修饰方法之外,还可以修饰一个"代码块".

```
synchronized(this){
 count ++;
}
```

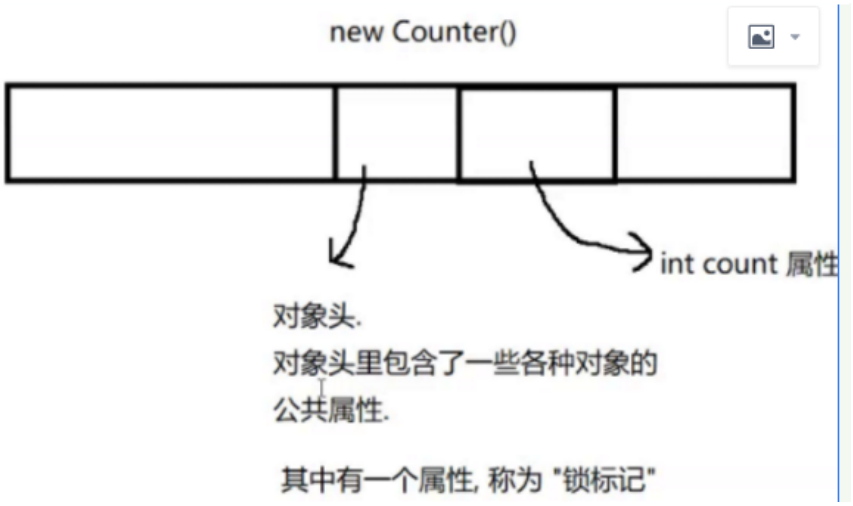
synchronized 如果是修饰代码块的时候,需要显式的在()中指定一个要加锁的对象.如果是 synchronized 直接修饰的非静态方法,相当于加锁的对象就是this

Java中, 一个对象的内存布局分析:

```
public void increase(){
 synchronized (this){
 count ++;
 }
}

Counter counter = new Counter();
```

在内存中的布局的样子



所谓的"加锁操作",其实就是把一个指定的锁对象中的锁标记 设为 true.

所谓的"解锁操作",其实就是把一个指定的锁对象中的锁标记设为 false.

如果两个线程,尝试针对同一个锁对象进行加锁,此时一个线程会先获取到锁,另外一个线程就会阻塞等待.

如果两个线程,尝试针对两个不同的锁对象进行加锁,此时两个线程都能获取到各自的锁,互不冲突.

Java中,任意的对象,多可以作为"锁对象'.

理解"阻塞等待"

针对每一把锁, 操作系统内部都维护了一个等待队列. 当这个锁被某个线程占有的时候, 其他线程尝试进行加锁, 就加不上了, 就会阻塞等待, 一直等到之前的线程解锁之后, 由操作系统唤醒一个新的线程, 再来获取到这个锁.

注意:

- 上一个线程解锁之后, 下一个线程并不是立即就能获取到锁. 而是要靠操作系统来 "唤醒". 这也就是操作系统线程调度的一部分工作.
- 假设有 A B C 三个线程, 线程 A 先获取到锁, 然后 B 尝试获取锁, 然后 C 再尝试获取锁, 此时 B和 C 都在阻塞队列中排队等待. 但是当 A 释放锁之后, 虽然 B 比 C 先来的, 但是 B 不一定就能获取到锁, 而是和 C 重新竞争, 并不遵守先来后到的规则.

synchronized的底层是使用操作系统的mutex lock实现的.

2. 刷新内存

synchronized 不光能起到互斥的效果,还能够刷新内存(解决内存可见性的问题)

synchronized 的工作过程:

1.获得互斥锁

2.从主内存拷贝变量的最新副本到工作的内存

3.执行代码

4.将更改后的共享变量的值刷新到主内存

5.释放互斥锁

所以 synchronized 也能保证内存可见性.

一旦代码中使用了 synchronized ,此时程序就很可能与"高性能"无缘了.

3. 可重入

synchronized 同步块对同一条线程来说是可重入的，不会出现自己把自己锁死的问题;

```
synchronized public void increase(){
synchronized (this){
count ++;
}
}
```

进入 increase 方法,就加了一次锁,在进入代码块,又加了一次锁.这种操作对于 synchronized 来说没有问题,这里进行了特殊处理(**可重入**).

在可重入锁的内部, 包含了 "线程持有者" 和 "计数器" 两个信息.

1. 如果某个线程加锁的时候, 发现锁已经被人占用, 但是恰好占用的正是自己, 那么仍然可以继续获取到锁, 并让计数器自增.

2. 解锁的时候计数器递减为 0 的时候, 才真正释放锁. (才能被别的线程获取到)

2、synchronized使用示例

synchronized 本质上要修改指定对象的 "对象头". 从使用角度来看, synchronized 也势必要搭配一个具体的对象来使用.

1. 直接修饰普通方法: 锁的 SynchronizedDemo 对象

```
public class SynchronizedDemo{
 public synchronized void methond(){
 }
}
```

2. 修饰静态方法: 锁的 SynchronizedDemo 类的对象

```
public class SynchronizedDemo{
 public synchronized stati void methond(){
 }
}
```

3. 修饰代码块:明确指定锁哪个对象.

3.1 锁当前对象:

```
public class SynchronizedDemo {
 public void method() {
 synchronized (this) {
 }
 }
}
```

3.2 锁类对象

```
public class SynchronizedDemo {
 public void method() {
 synchronized (SynchronizedDemo.class) {
 }
 }
}
```

我们重点要理解，synchronized 锁的是什么. 两个线程竞争同一把锁, 才会产生阻塞等待.

3、Java标准库里的线程安全类

Java 标准库中很多都是线程不安全的. 这些类可能会涉及到多线程修改共享数据, 又没有任何加锁措施.

```
ArrayList/LinkedList/HashMap/TreeMap/HashSet/TreeSet/StringBuilder
```

但是还有一些是线程安全的. 使用了一些锁机制来控制.

```
Vector (不推荐使用)/HashTable (不推荐使用)/ConcurrentHashMap/StringBuffer
```

StringBuffer 的核心方法都带有 synchronized .

还有的虽然没有加锁, 但是不涉及 "修改", 仍然是线程安全的

```
String(不可变对象)
```