获取该锁。如果锁不具备重入性，那么当同一个线程两次获取锁的时候就会发生死锁。Java 提供了 java.util.concurrent.ReentrantLock 来解决重入锁问题。ReentrantLock 是唯一实现了 Lock 接口的类，并且 ReentrantLock 提供了更多的方法。示例代码如下：

```
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class Demo
{
    private Lock lock = new ReentrantLock();

    public void method1()
    {
        lock.lock();
        System.out.println("method1 is called" );
        method2();
        lock.unlock();
    }

    public void method2()
    {
        lock.lock();
        System.out.println("method2 is called" );
        lock.unlock();
    }
}

public class Test
{
    public static void main(String[] args)
    {
            Demo demo = new Demo();
            demo.method1();
    }
}
```

程序运行结果为：

```
method1 is called
method2 is called
```

从上的例子可以看出，在 method1 中已经通过 lock 方法获取到锁了。然后在调用 method2 的时候通过 lock 方法仍然能获取到锁，这就充分体现了重入锁的特性。下面在通过另外一个例子来说明 ReentrantLock 具备的锁的特性：

```
import java.util.Date;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class Demo
{
    private Lock lock = new ReentrantLock();

    public void method1()
    {
        try
        {
            lock.lock();
            System.out.println("method1 is called, timestamp= " + new Date().toString());
            Thread.sleep(5000);
            System.out.println("method1 is called, timestamp= " + new Date().toString());
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        finally
```