JDK源码学习

1. transient关键词：

描述类的成员变量，该成员变量不会由于类实现了Serializable而被自动序列化，比如User中的password，可以加上transient修饰，User在序列化后，不会带有password的数据才。

\*类的static不论是否被transient修饰，都不会被自动序列化。

2. ArrayList初始化：

类中有一个共享的static的空数组Object[]，在用户创建一个大小为0的ArrayList时，返回给用户。

3. Thread.setDaemon：

设置该线程是用户线程还是守护线程，守护线程与JVM生命周期相同（比如垃圾收集线程，内存管理线程），优先级较低，用于辅助用户线程执行任务；用户线程就是用户定义的，执行特定任务的线程，当所有用户线程退出后，守护线程也都会退出。

4. @CallerSensitive注释：

这个注解是为了堵住漏洞用的。

曾经有黑客通过构造双重反射来提升权限，原理是当时反射只检查固定深度的调用者的类，看它有没有特权，例如固定看两层的调用者（getCallerClass(2)）。如果我的类本来没足够权限群访问某些信息，那我就可以通过双重反射去达到目的：反射相关的类是有很高权限的，而在 我->反射1->反射2 这样的调用链上，反射2检查权限时看到的是反射1的类，这就被欺骗了，导致安全漏洞。使用CallerSensitive后，getCallerClass不再用固定深度去寻找actual caller（“我”），而是把所有跟反射相关的接口方法都标注上CallerSensitive，搜索时凡看到该注解都直接跳过，这样就有效解决了前面举例的问题

5. JDK提供的动态代理

使用反射的机制，不过调用对象一定要实现某个接口，将接口与扩展的方法作为参数，Java会通过反射，生成代理实例，并调用方法。

6. 写文件：FileWriter

FileWriter本身自带一个比较小的缓冲区，但是由于缓冲区太小了并且无法手动设置大小，所以使用的时候一般会在外面套一层BufferWriter。

比如：new BufferWrite(new FileWriter())加入缓冲区的功能

在外面套一层是因为JAVA.IO使用了装饰器的设计模式。

7. CountDownLatch：多线程计数器

Java.concurrent类库提供的线程计数器，用于等待估计数目的线程执行完毕后，在开始下面的任务。

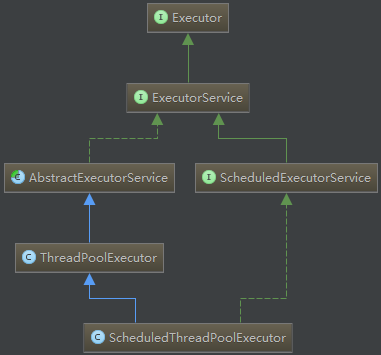
CountDownLatch latch = new CountDownLatch(2);

latch.countDown();

latch.await();

8. Executor接口

Java有关线程池的类的最上层的接口，使用生产者-消费者的模式，提交Runnable任务，执行Runnable任务



9. 匿名类

可以在函数内，通过代码，动态创建某一个接口的实现类，匿名类中可以看到该函数的本地变量，以及外层类的所有成员变量。

10. 闭包

是一个可调用的对象，它记录了一些信息，这些信息来自于创建它的作用域。

具体体现：匿名内部类，匿名函数等。

闭包是函数式编程语言的标配，在面向对象编程的世界里，数据和对象绑定在一起的，而我们在写代码实现一个业务流程的时候，理想的情况是，数据是跟着代码走的，拿来就用，用完即走，而不是需要通过一个对象来取数据，用完之后还要手动去释放对象。闭包正是基于这种理念，数据和函数绑定在一起，写代码的时候拿来即用，用完即走，这样的代码写起来行云流水，但是看起来可能不是那么结构清晰。