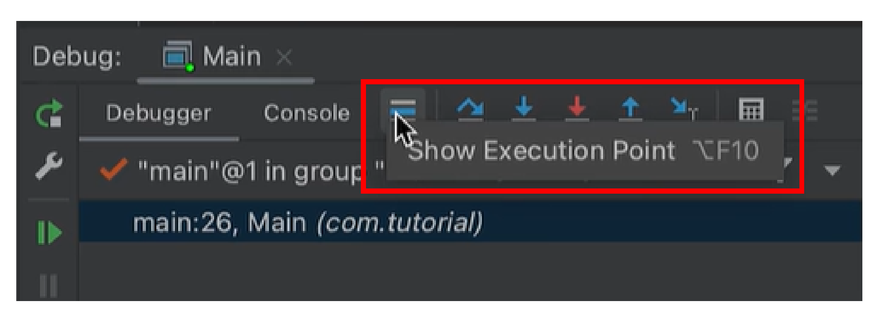
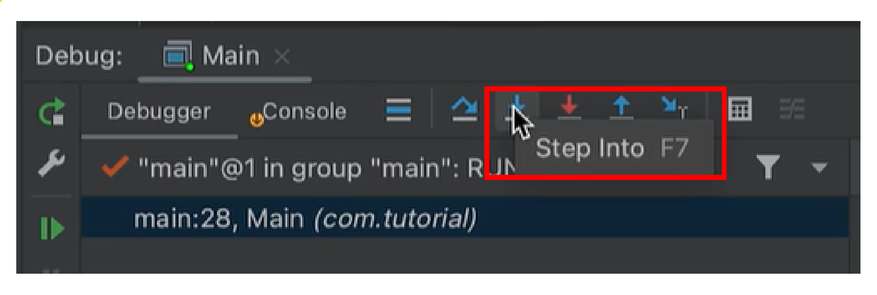
IdeaDebug

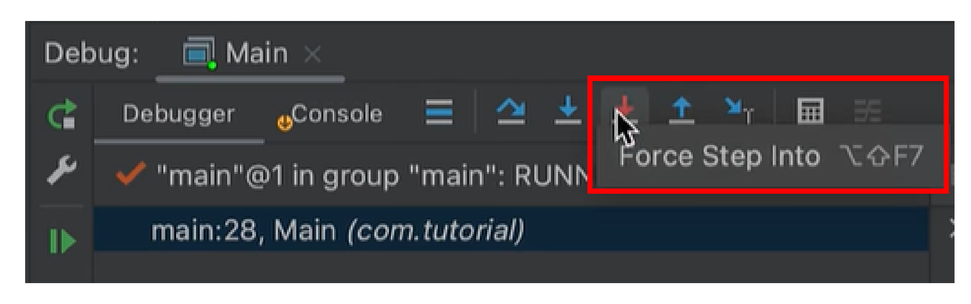
Debug界面按钮



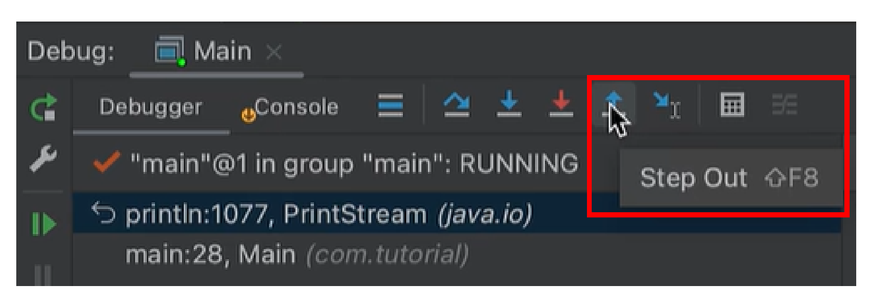
"Show Execution Point"按键：不管当前光标在哪，按下后光标立刻回到应用程序当前运行的那一行。



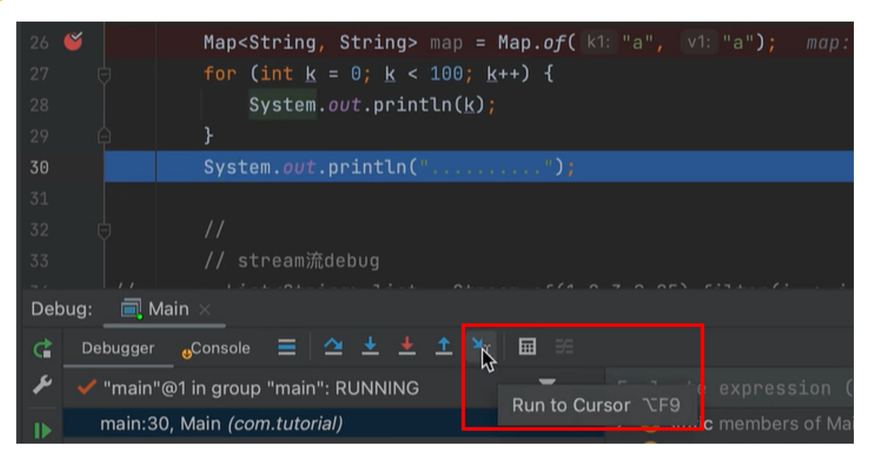
"Step Into"按键：进入方法内部，但默认不会进入JDK类的方法，比如System.out.println是JDK自带类的方法。



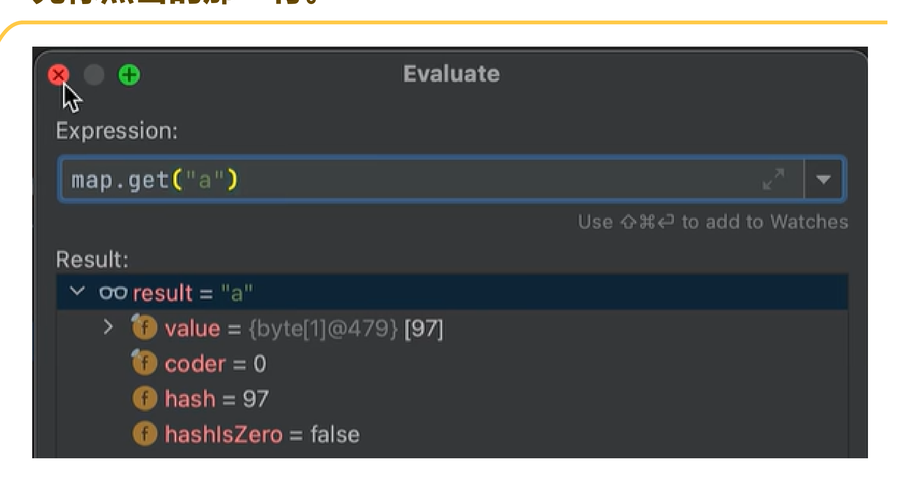
碰到这种JDK类的方法，可以用这个"Force Step Into"按键。



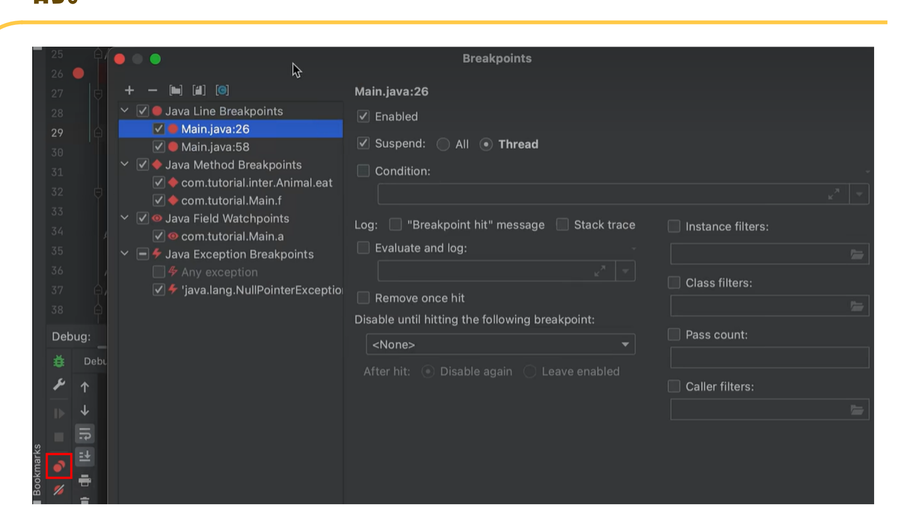
"Step Out"按键：顾名思义，跳出当前方法。



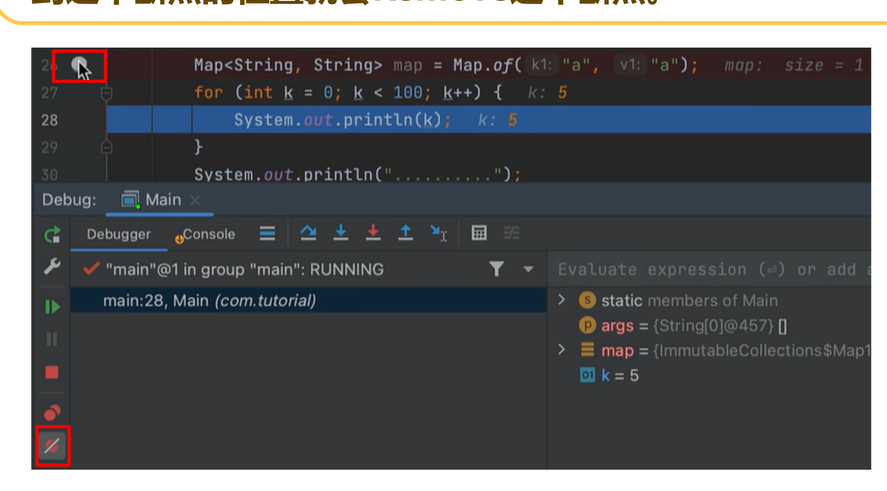
"Run to Cursor"按键：将当前Debug的Step跳到鼠标光标点击的那一行。



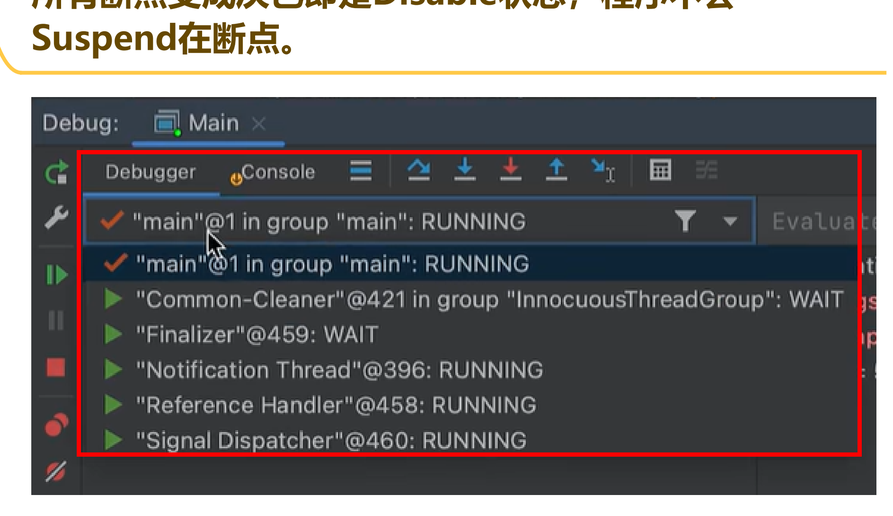
"Evaluate Expression"按键：可以获取当前程序运行的所有变量的值，或任何操作/Java语句运行都是可以的。



"View Breakpoints"按键：可以看到程序中的所有断点，可以看到这里断点分为四种(Java Line Breakpoints, Java Method Breakpoints, Java Field Watchpoints, Java Exception Breakpoints)。里面这个"Enabled"就是是否启用这个断点。然后这个Condition可以设置断点停下来时变量的值是多少等条件。Log这里勾选上的话，就是代码执行到断点这里的话就打印一个日志或输出当前的堆栈。然后这个"Remove once hit"的话就是"临时断点"，一旦运行到这个断点的位置就会Remove这个断点。

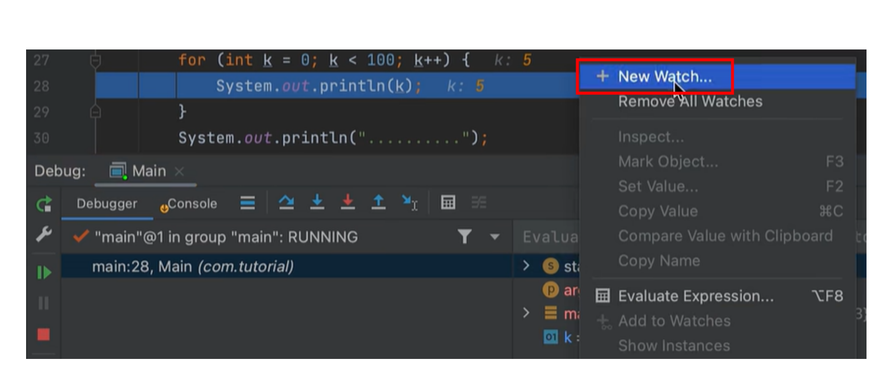


"Mute Breakpoints"按键：点上Mute这个按钮以后，所有断点变成灰色即是Disable状态，程序不会Suspend在断点。

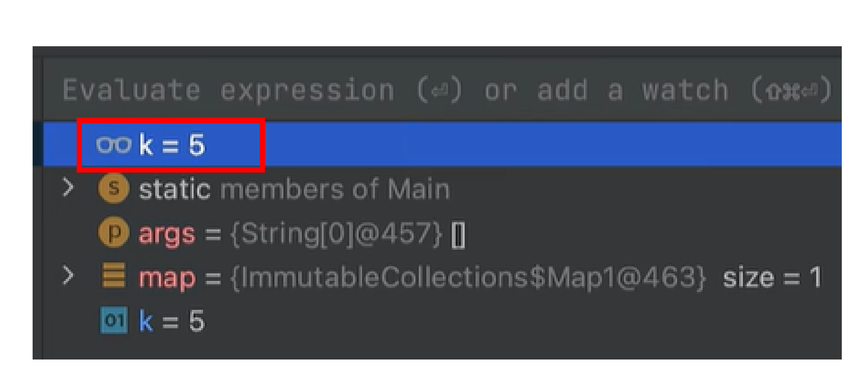


"线程堆栈区"：这里可以展示所有线程的堆栈。

"变量区"：展示当前的变量和它们的值。然后这里上方的文本框"Evaluate Expression"和之前的那个功能是一样的。

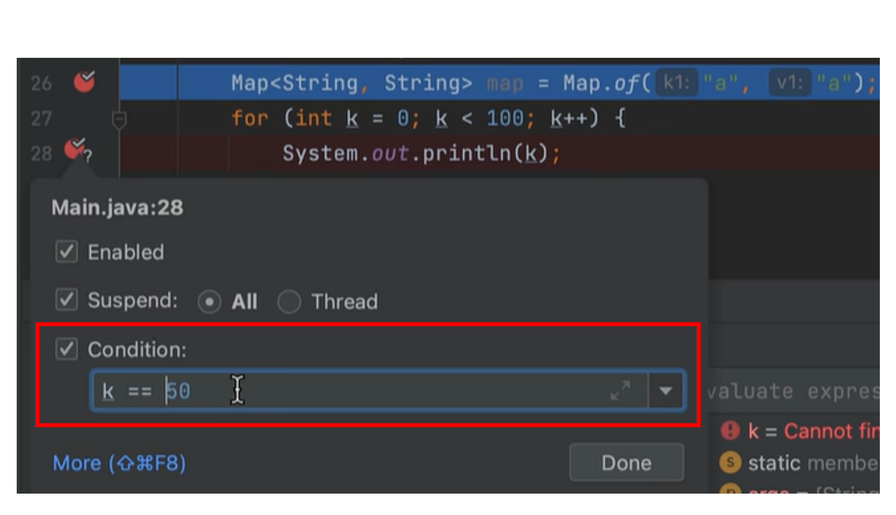


"Watch功能"。

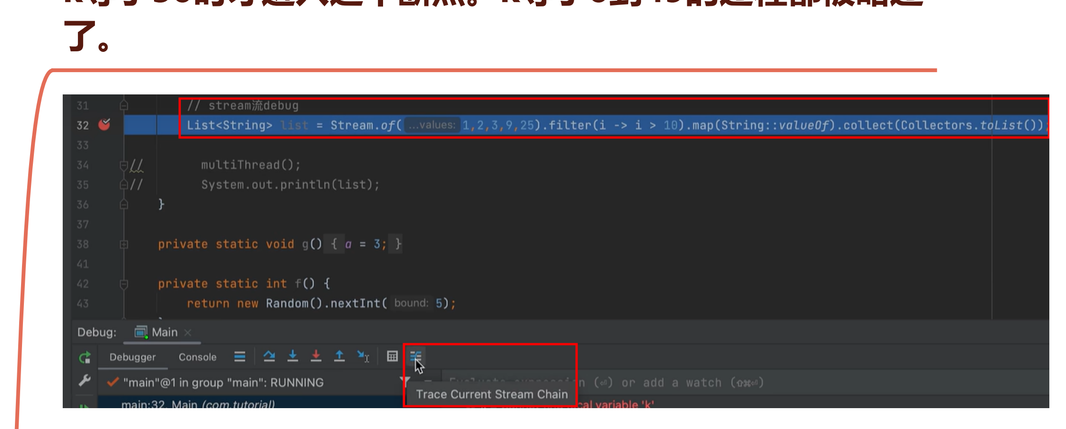


比如说在这加一个k=5，每次运行Step Over，这个k的值都会变化，监控k的值的变化。

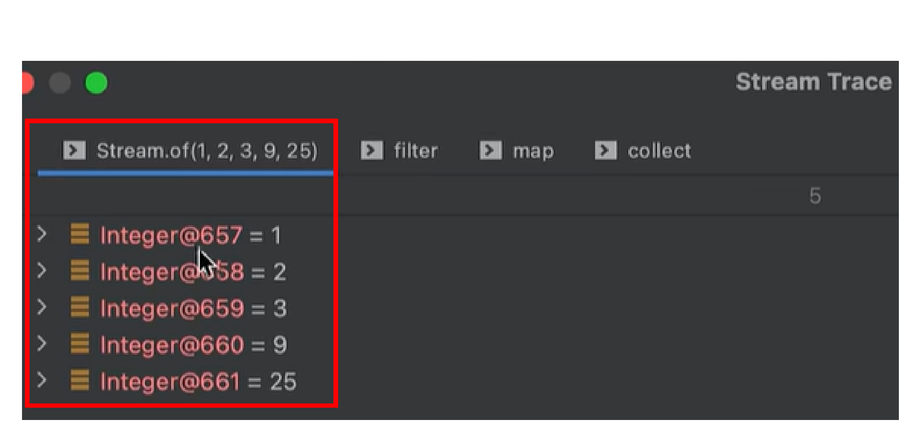
行断点高级特性



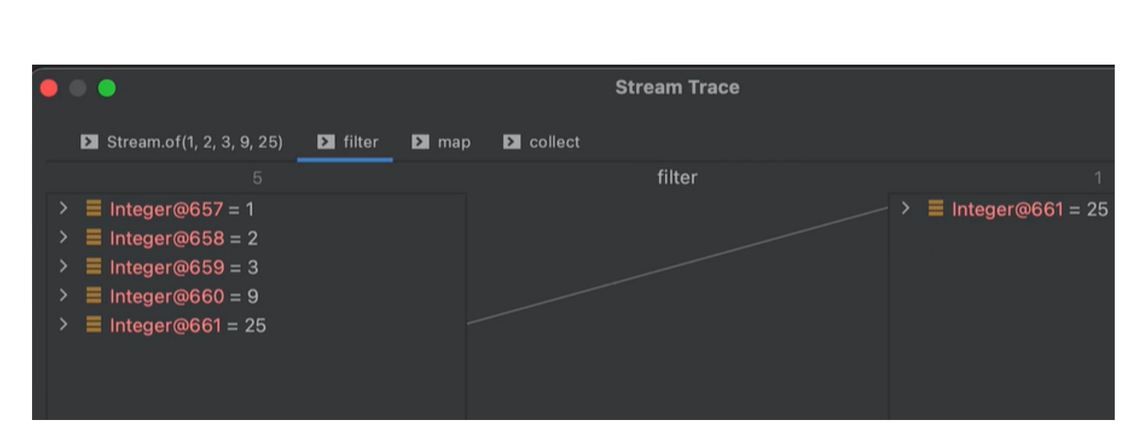
"条件断点"：右击断点，在这里写上一个判断表达式，这个表达式就必须返回的是"布尔值"！比如这个例子里只有k等于50时才进入这个断点。k等于0到49的过程都被略过了。



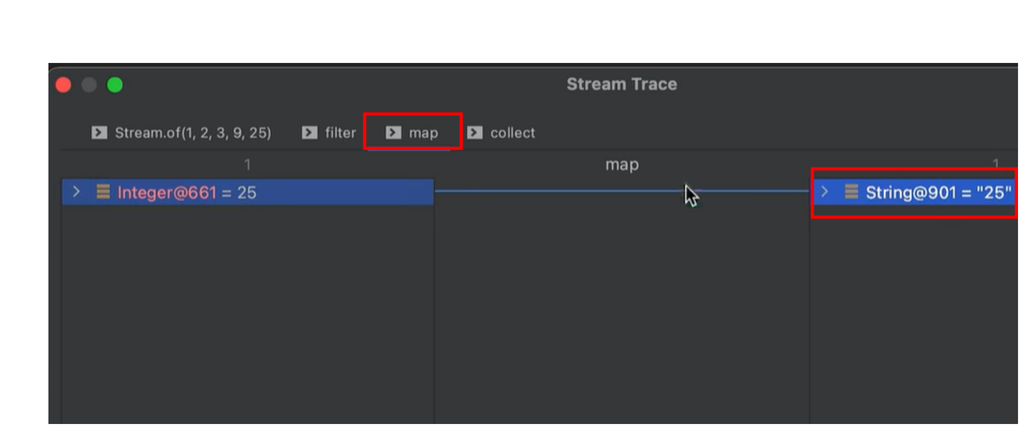
"Stream流断点"：因为我们是没法进入"filter"、"map"这些方法的，然后我们就可以点这个"Trace Current Stream Chain"这个按钮，可以看到流在每一个方法处理过程中的一些数据：



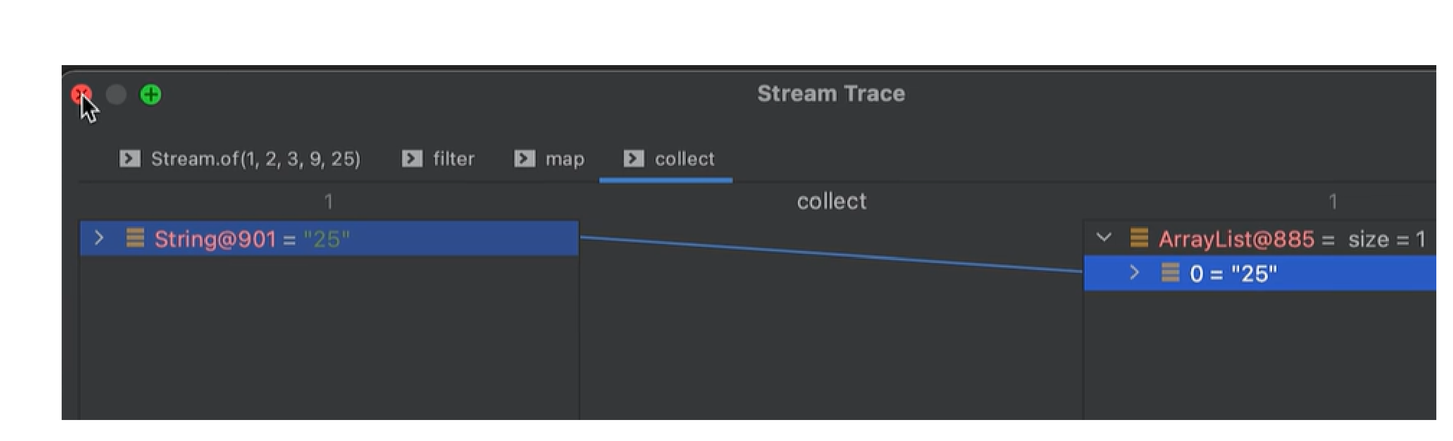
比如说一开始是1,2,3,9,25这些数据。



到filter之后就只有一个数据了。



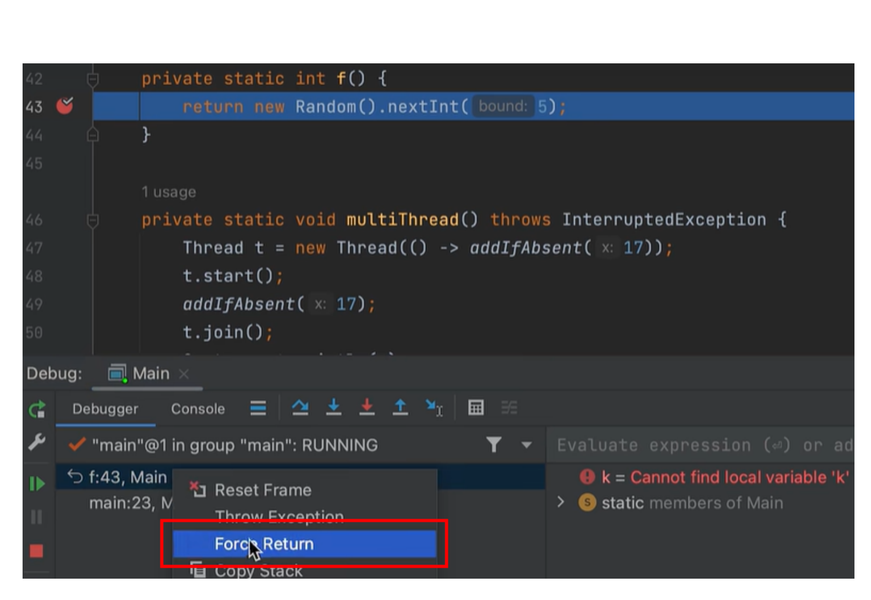
到map之后就把这个整数转化为字符串类型了。



到collect之后就是现在这个数据。

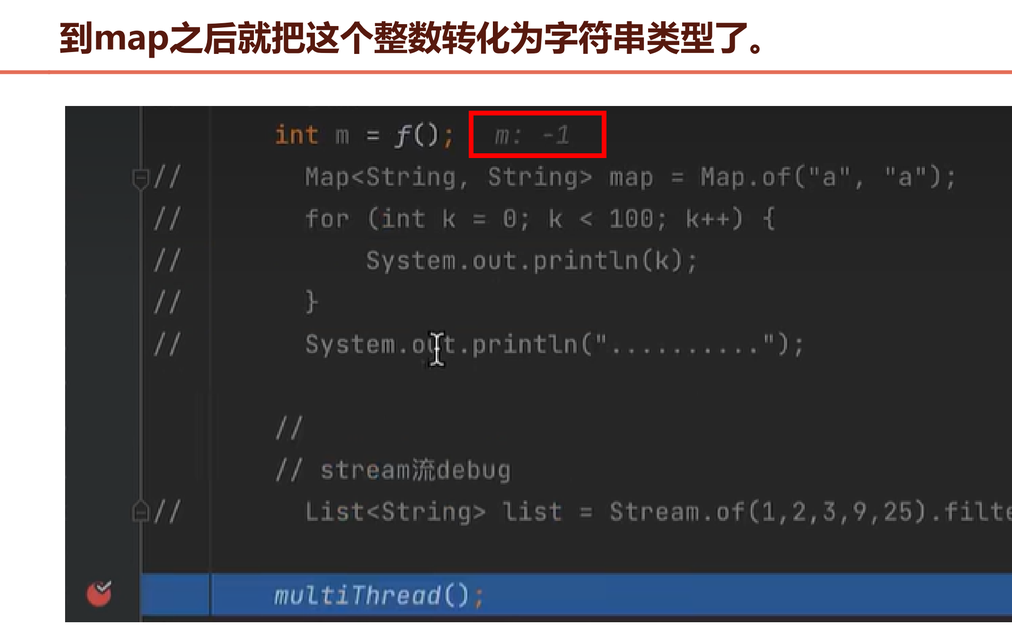
总结：就可以详细地看到Stream流处理过程中的每一步数据具体的数值，可以帮我们Debug这个Sream代码。

"Force Return"：应用场景——在Debug时执行到一个方法的开头，不希望后面继续执行，直接想提前返回这个方法，因为后面可能会删除数据或有其它误操作，所以必须跳过后面的流程提前返回。

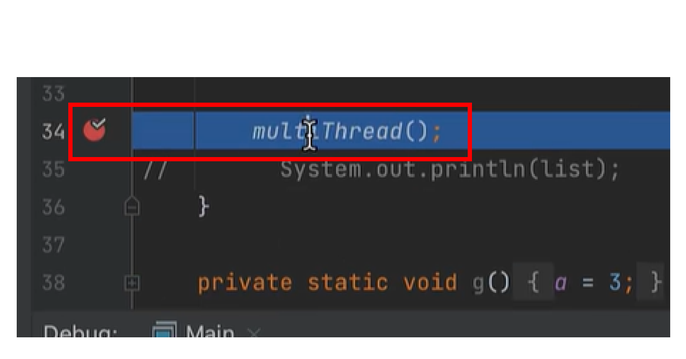


右击这个堆栈方法这里。

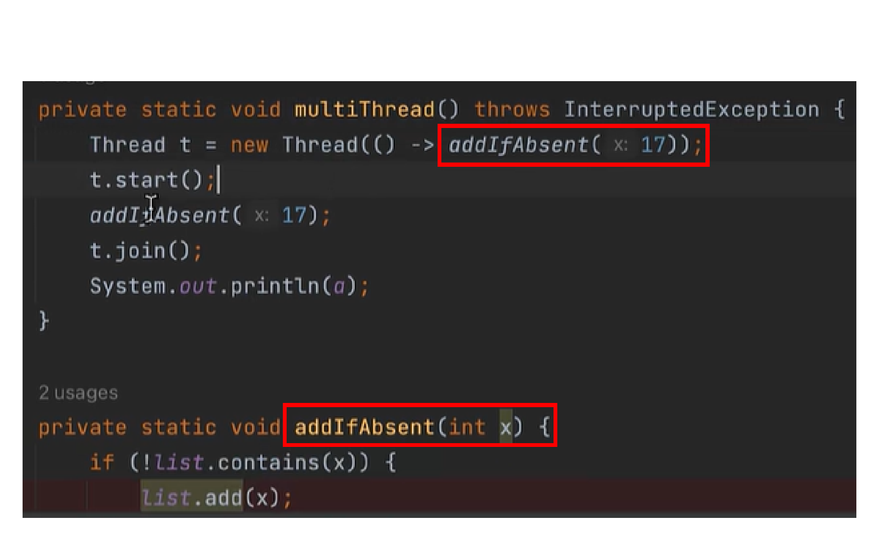
这里直接输入-1，就不执行这段Random方法了。



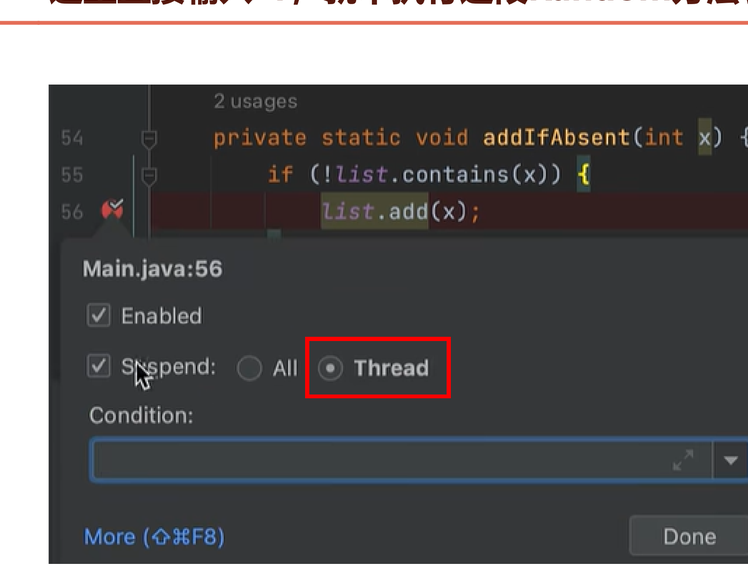
继续往下执行，可以看到m的值已经等于-1了，没有执行Random那个方法去生成一个随机数。



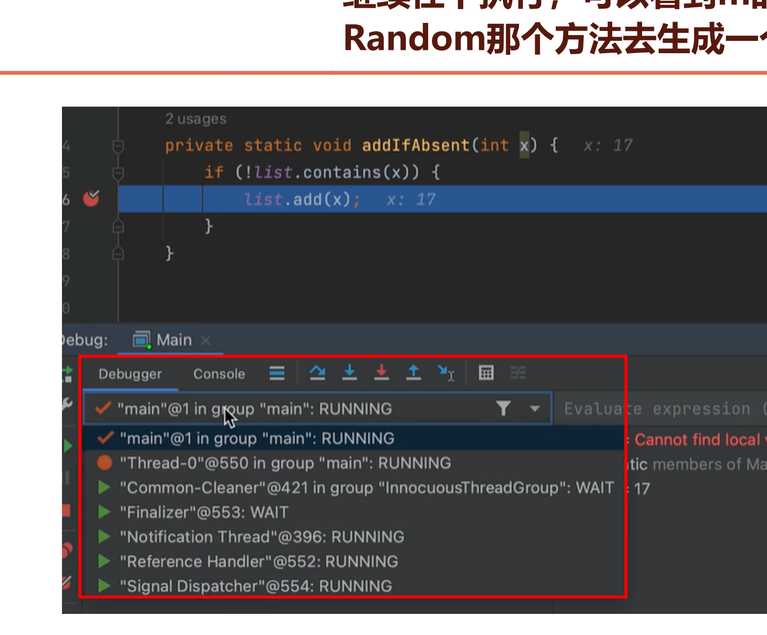
"多线程Multi-thread"：比如这里有个多线程的方法。



这个Main线程之外new了一个线程去对容器进行新增元素的操作。

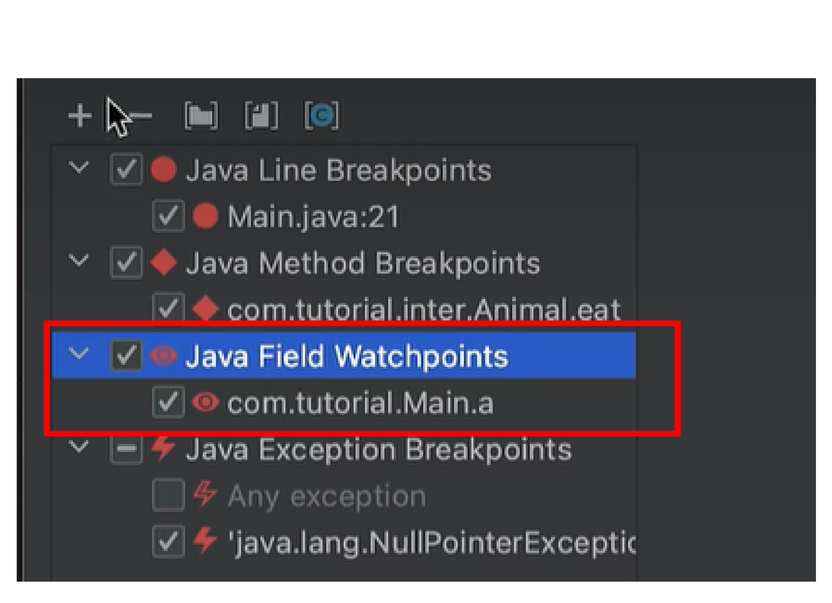


在方法里加了断点并右击，这个Suspend这里默认是All，这里选上Thread。

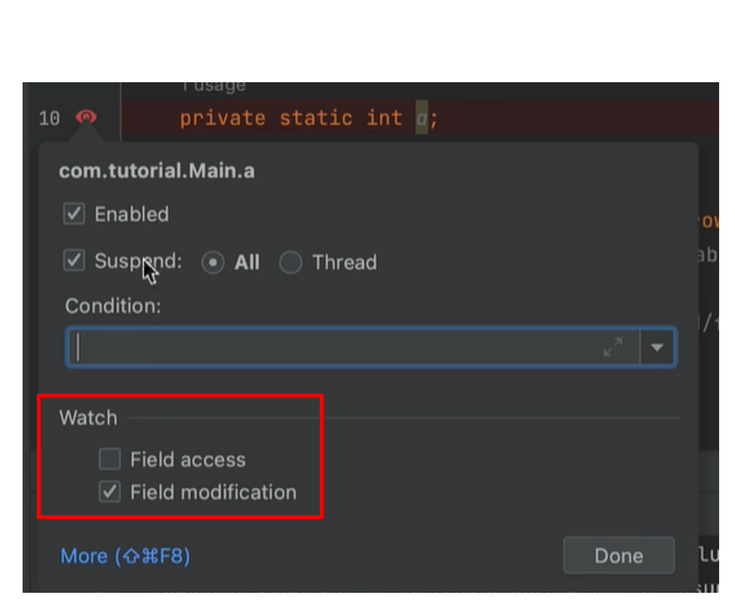


线程堆栈区的话，这里可以切换线程，当前是Main线程，我们可以切换Thread线程去先执行它的一些操作。

属性断点



眼睛的标志就是属性断点。



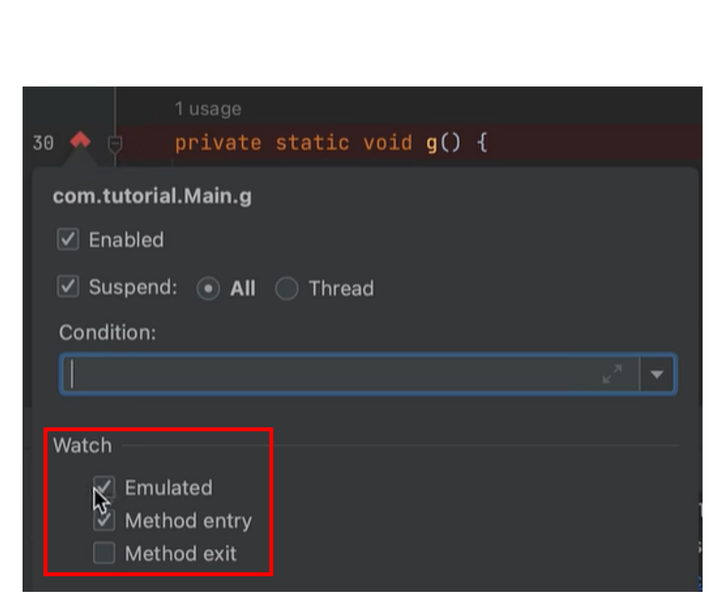
右键可以选择是"字段访问"或"字段修改"时去激活这个断点。用的比较多的就是"字段修改"，在字段处打断点，在字段值被修改时激活这个断点，可以找到修改某字段数值的代码部分。

可以看到，上一步中的"private static int a"那里有一个"Field modification"的断点，而g方法处并没打断点，但g方法确实是修改a数值的函数代码。

可以看到，虽然g方法原本没打断点，但运行Debug代码后，代码确实Suspend在了g方法这里。

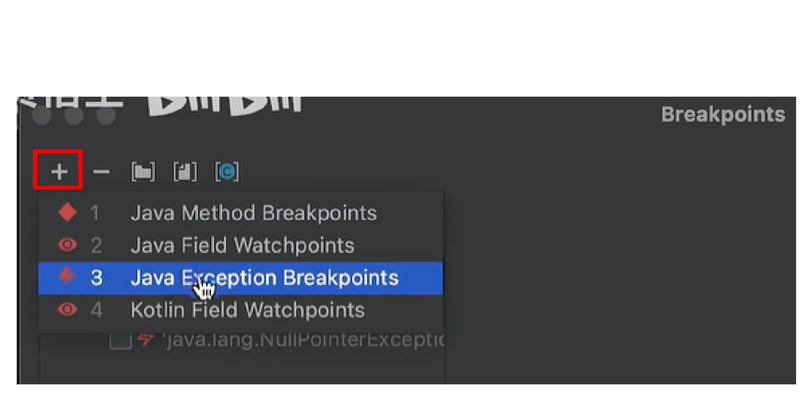
方法断点

顾名思义，就是可以打在方法上的。其主要用处是在接口方法加上方法断点，然后所有的实现类就会进入这个断点代码。

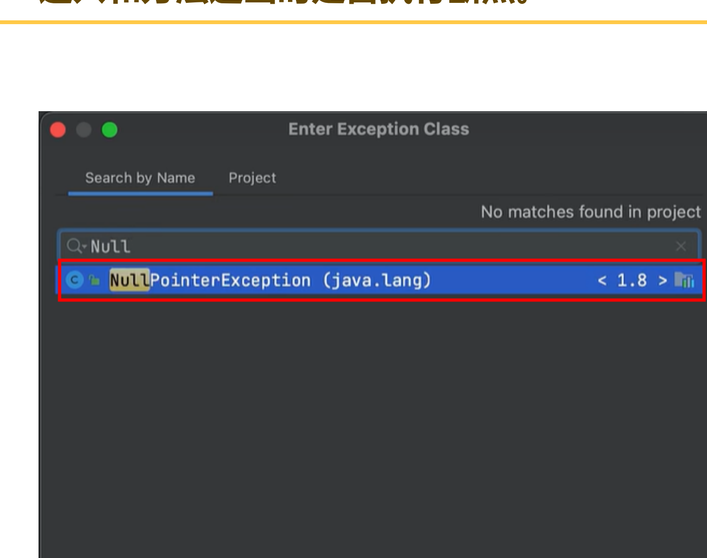


右击可以看到，"Method entry"和"Method exit"就是方法进入和方法退出时是否执行断点。

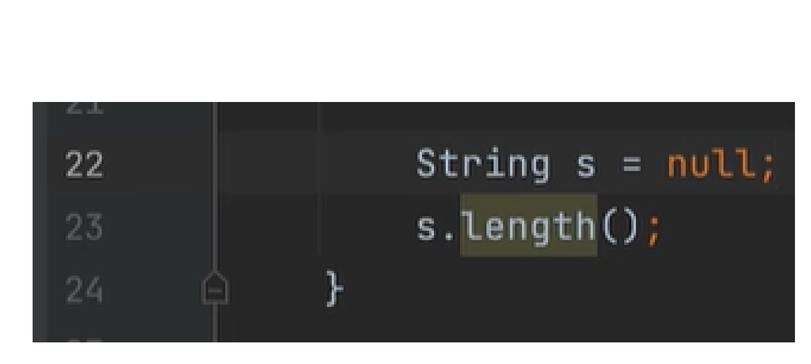
异常断点



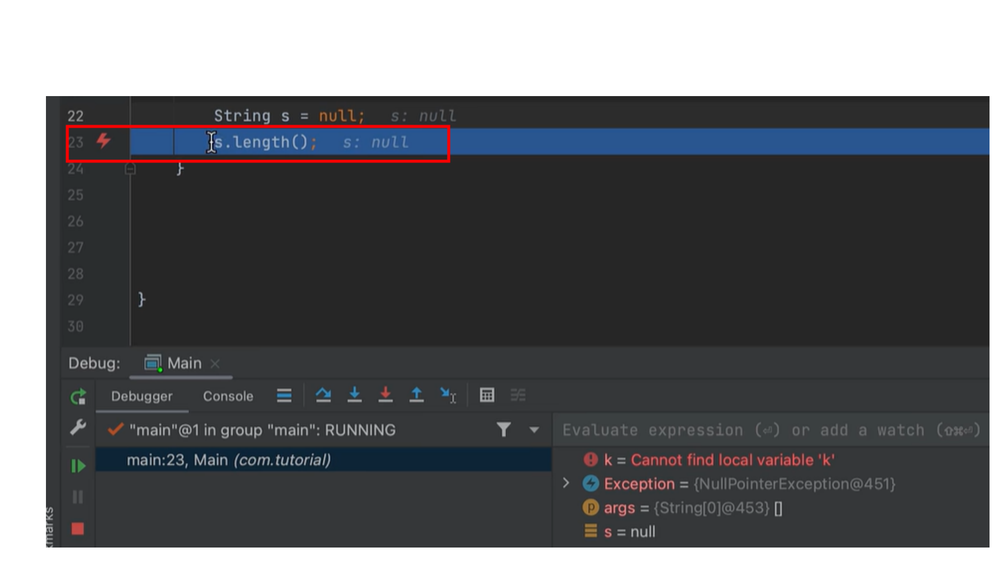
加一个Java Exception Breakpoints。



比如说加一个空指针异常，即任何代码里抛出空指针异常它就会停在断点处。



比如举例写一个空指针代码，这里不加任何断点。



然后运行Debug模式，可以看到代码自动停在会报空指针的代码处，左边这里就自动出现了一个异常断点的标志，就会报一个空指针了。