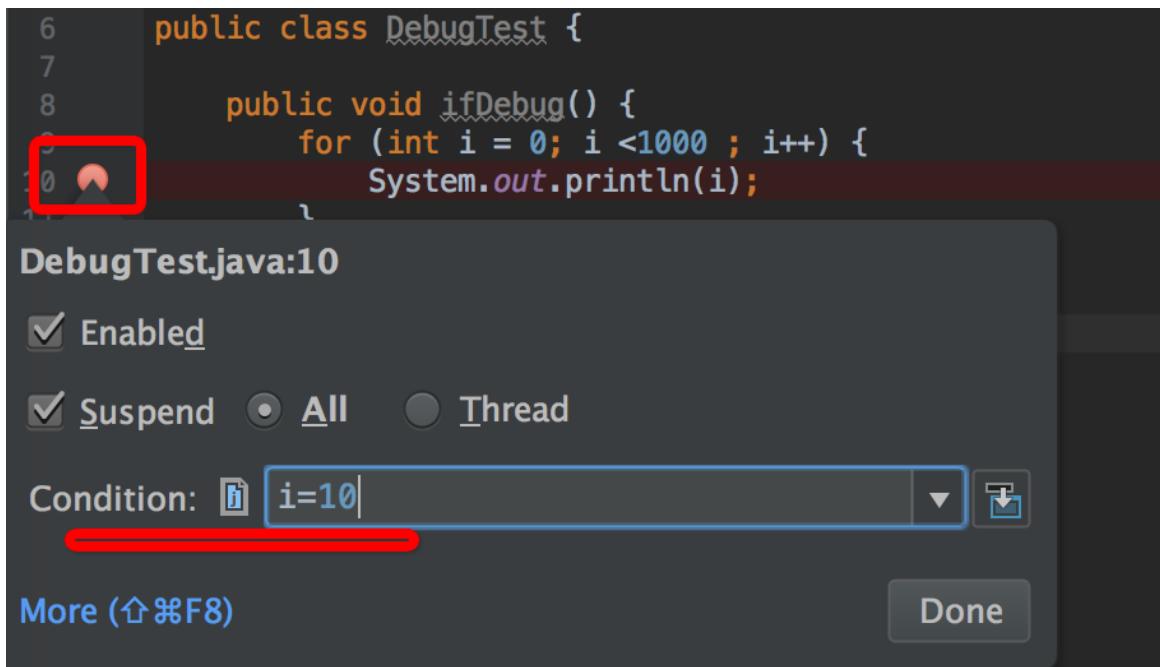


IntelliJ IDEA 调试技巧，比 Eclipse 强太多了！

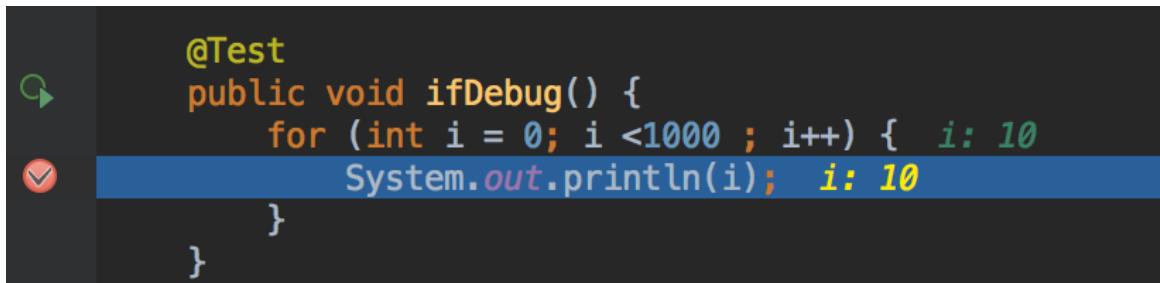
今天给大家分享一下 IntelliJ IDEA 的高级调试技巧，来看下有多骚，确实要比 Eclipse 强太多了！

一、条件断点

循环中经常用到这个技巧，比如：遍历1个大List的过程中，想让断点停在某个特定值。



参考上图，在断点的位置，右击断点旁边的小红点，会出来一个界面，在 Condition这里填入断点条件即可，这样调试时，就会自动停在i=10的位置



二、回到“上一步”

该技巧最适合特别复杂的方法套方法的场景，好不容易跑起来，一不小心手一抖，断点过去了，想回过头看看刚才的变量值，如果不知道该技巧，只能再跑一遍。

```
18     @Test
19     public void dropFrameDebug() {
20         int i = 99;
21         method1(i);
22     }
23
24     private void method1(int i) {
25         System.out.println("method1:" + i);
26         method2(i);
27     }
28
29     private void method2(int j) { j: 100
30         j++;
31         System.out.println("method2:" + j); j: 100
32     }
33 }
34
35
```

Debug DebugTest.dropFrameDebug

Console Debugger Variables

Breakpoint icon (red circle with a dot) → Step Into icon (blue arrow) → Drop Frame icon (red square with a white arrow)

Variables:

- this = {DebugTest@803}
- j = 100

参考上图，method1方法调用method2，当前断点的位置j=100，点击上图红色箭头位置的Drop Frame图标后，时间穿越了

The screenshot shows a Java code editor and a debugger interface. The code is as follows:

```
19     public void dropFrameDebug() {  
20         int i = 99;  
21         method1(i);  
22     }  
23  
24     private void method1(int i) { i: 99  
25         System.out.println("method1:" + i);  
26         method2(i); i: 99  
27     }  
28  
29     private void method2(int j) {  
30         j++;  
31         System.out.println("method2:" + j);  
32     }  
33 }  
34  
35 }
```

The debugger window at the bottom shows the current state:

- Stack Trace: Debug Test.dropFrameDebug
- Variables:
 - this = {DebugTest@803}
 - i = 99

回到了method1刚开始调用的时候，变量i变成了99，没毛病吧，老铁们，是不是很6 :)

注：好奇心是人类进步的阶梯，如果想知道为啥这个功能叫Drop Frame，而不是类似Back To Previous 之类的，可以去翻翻JVM的书，JVM内部以栈帧为单位保存线程的运行状态，drop frame即扔掉当前运行的栈帧，这样当前“指针”的位置，就自然到了上一帧的位置。

三、多线程调试

多线程同时运行时，谁先执行，谁后执行，完全是看CPU心情的，无法控制先后，运行时可能没什么问题，但是调试时就比较麻烦了，最明显的就是断点乱跳，一会儿停这个线程，一会儿停在另一个线程，比如下图：

```
@Test
public void multiThreadTest() {
    new Thread(() -> {
        System.out.println("1.卧枝商恨低");
    }, "菩提树下的杨过").start();

    new Thread(() -> {
        System.out.println("2.卧梅又闻花");
    }, "天空中的飞鸟").start();

    System.out.println("3.要问卧似水");
    System.out.println("4.倚头答春绿");
}
```

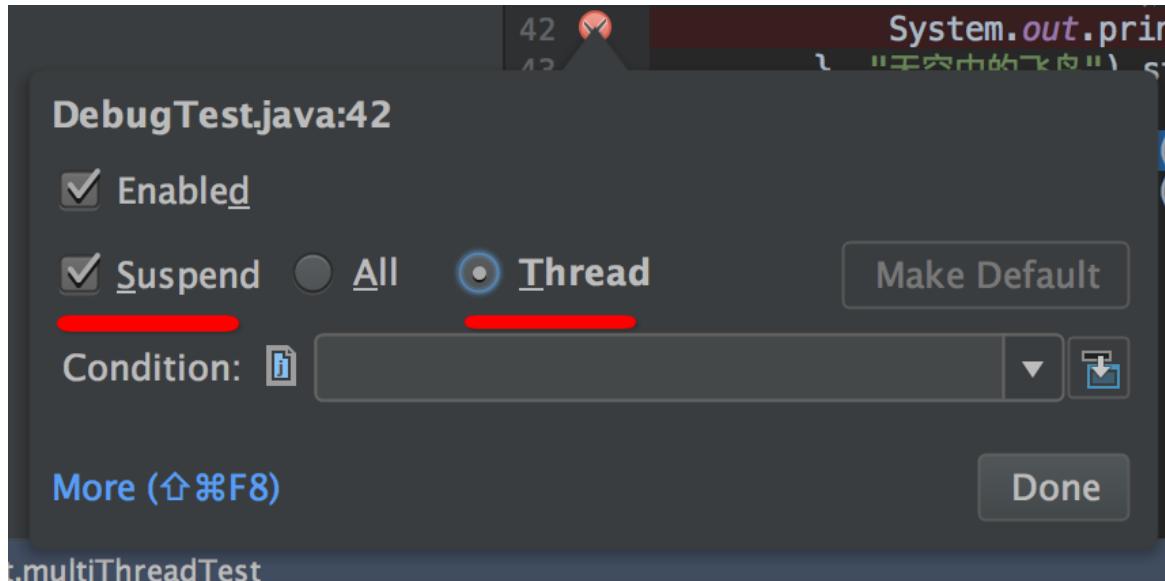
如果想希望下一个断点位置是第2句诗句，可能要失望了：

```
@Test
public void multiThreadTest() {
    new Thread(() -> {
        System.out.println("1.卧枝商恨低");
    }, "菩提树下的杨过").start();

    new Thread(() -> {
        System.out.println("2.卧梅又闻花");
    }, "天空中的飞鸟").start();

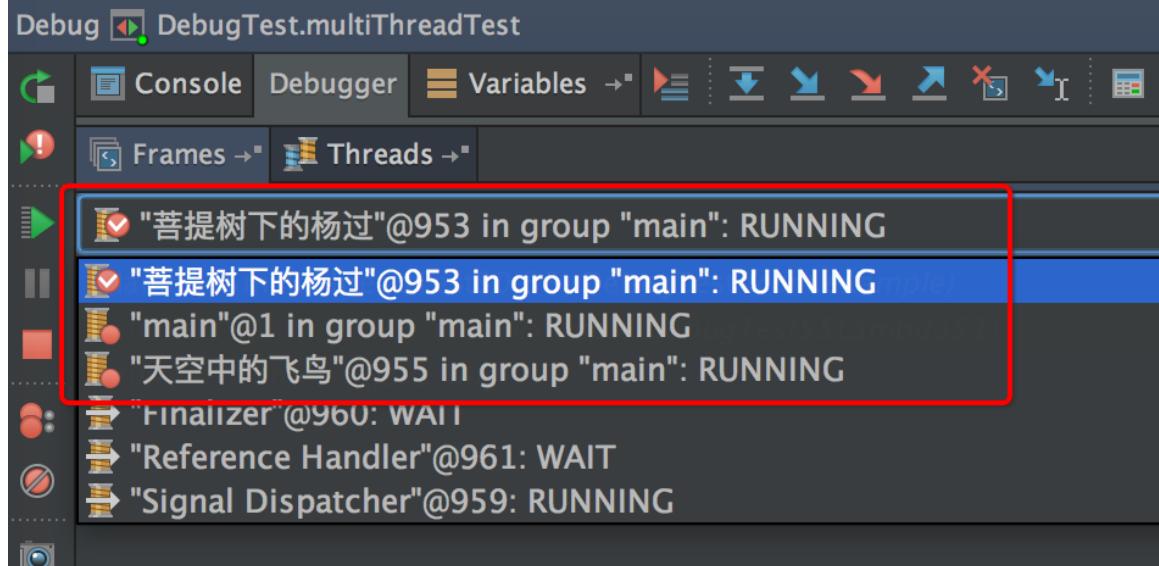
    System.out.println("3.要问卧似水");
    System.out.println("4.倚头答春绿");
}
```

如果想让线程在调试时，想按自己的愿意来，让它停在哪个线程就停在哪个线程，可以在图中3个断点的小红点上右击，



即： Suspend挂起的条件是按每个线程来，而非All。把这3个断点都这么设置后，再来一发试试

```
34
35
36     @Test
37     public void multiThreadTest() {
38         new Thread(() -> {
39             System.out.println("1.卧枝商恨低");
40             }, "菩提树下的杨过").start();
41
42         new Thread(() -> {
43             System.out.println("2.卧梅又闻花");
44             }, "天空中的飞鸟").start();
45
46         System.out.println("3.要问卧似水");
47         System.out.println("4.倚头答春绿");
48     }
49 }
```



注意上图中的红框位置，断点停下来时，这个下拉框可以看到各个线程（注：给线程起个容易识别的名字是个好习惯！），我们可以选择线程“天空中的飞鸟”断点如愿停在了第2句诗。

四、远程调试

这也是一个装B的利器，本机不用启动项目，只要有源代码，可以在本机直接远程调试服务器上的代码，打开姿势如下：

4.1 项目启动时，先允许远程调试

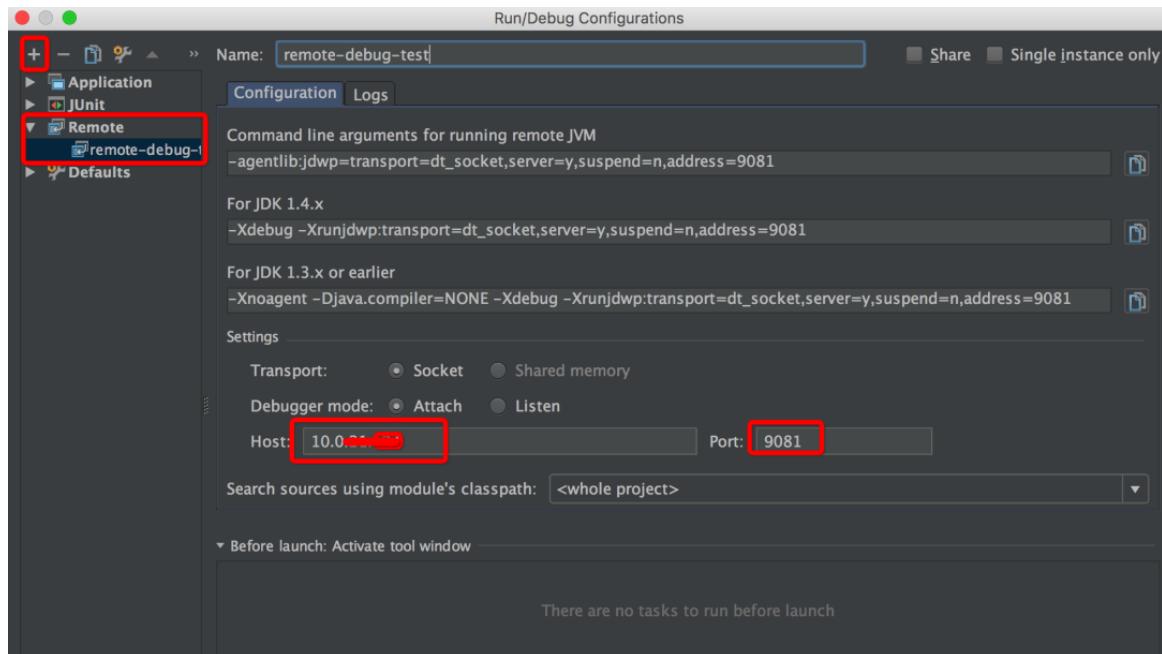
```
1  java -server -Xms512m -Xmx512m -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt\_\_socket,server=y,suspend=n,address=9081 -Djava.ext.dirs=\${main\_class}
```

起作用的就是

```
1 -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt\_socket,server=y,suspend=n,address=9081
```

注意：远程调试从技术上讲，就是在本机与远程建立scoket通讯，所以端口不要冲突，而且本机要允许访问远程端口，另外这一段参数，放要在-jar 或 \${main_class}的前面

4.2 idea中设置远程调试

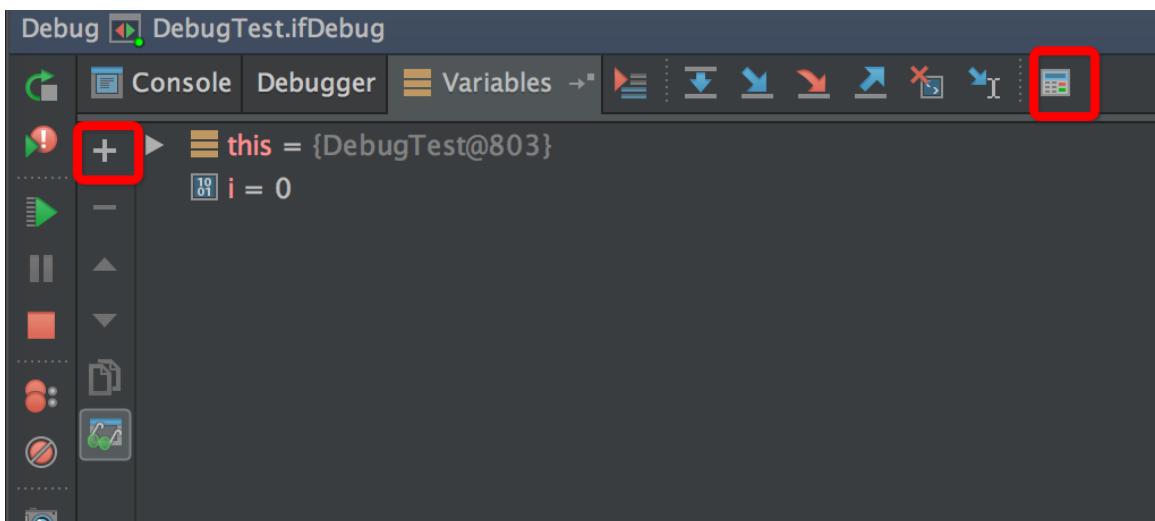


然后就可以调试了

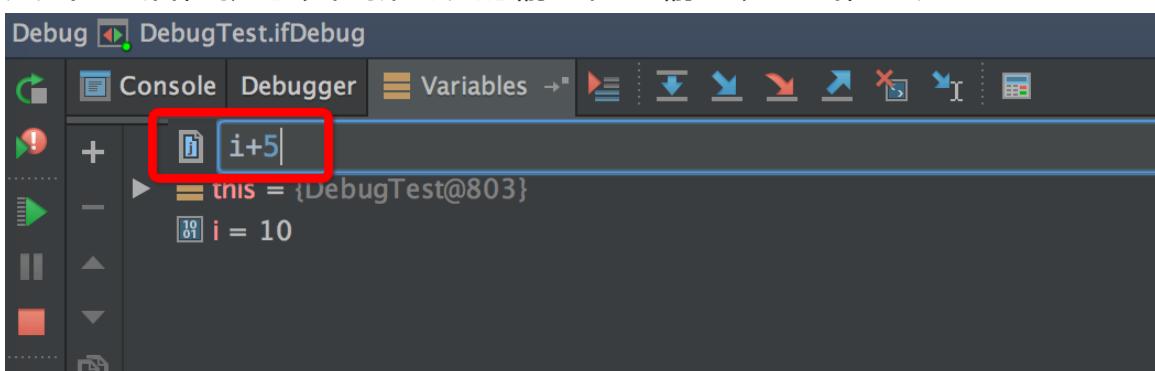
前提是本机有项目的源代码，在需要的地方打个断点，然后访问一个远程的url试试，断点就会停下来。

五、临时执行表达式/修改变量的运行值

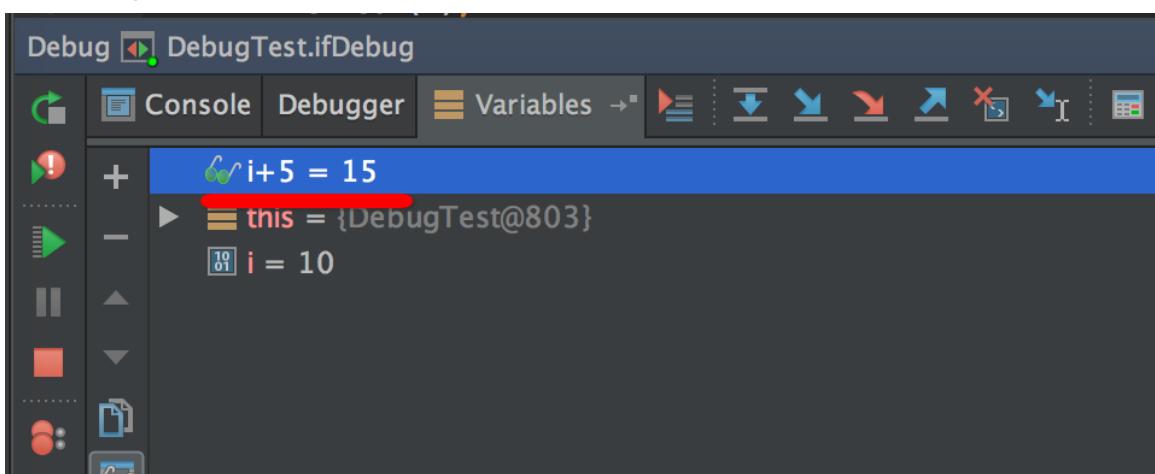
调试时，可以临时执行一些表达式，参考下图：点击这两个图标中的任何一个都可以



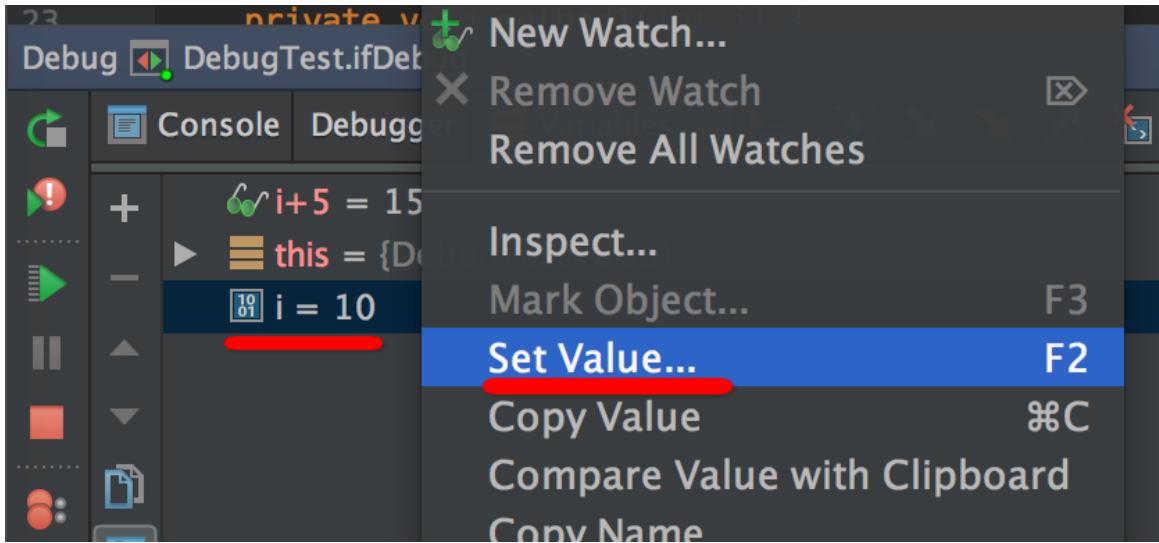
点击+号后，就可以在新出现的输入框里输入表达式，比如 $i+5$



然后回车，马上就能看到结果



当然，如果调试时，想动态修改变量的值，也很容易，在变量上右击，然后选择 Set Value，剩下的事，地球人都知道。



善用上述调试技巧，相信大家撸起代码来会更有感觉，祝大家使用愉快