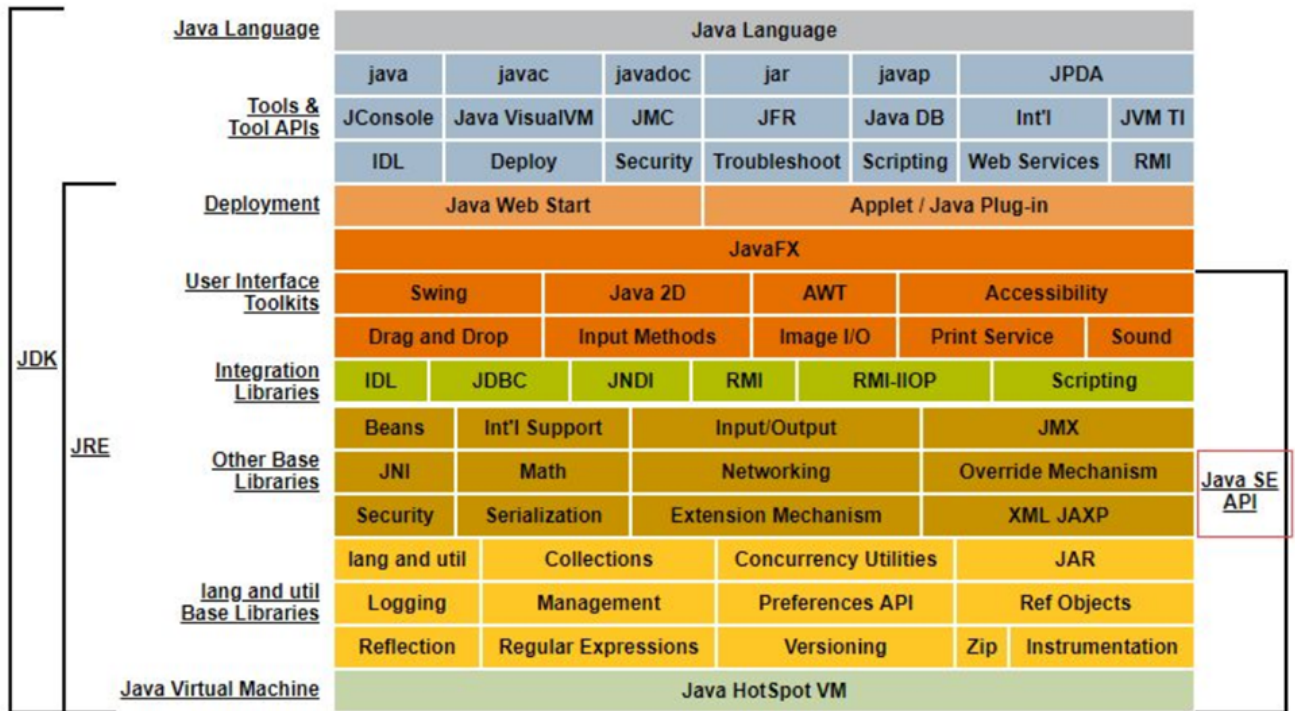
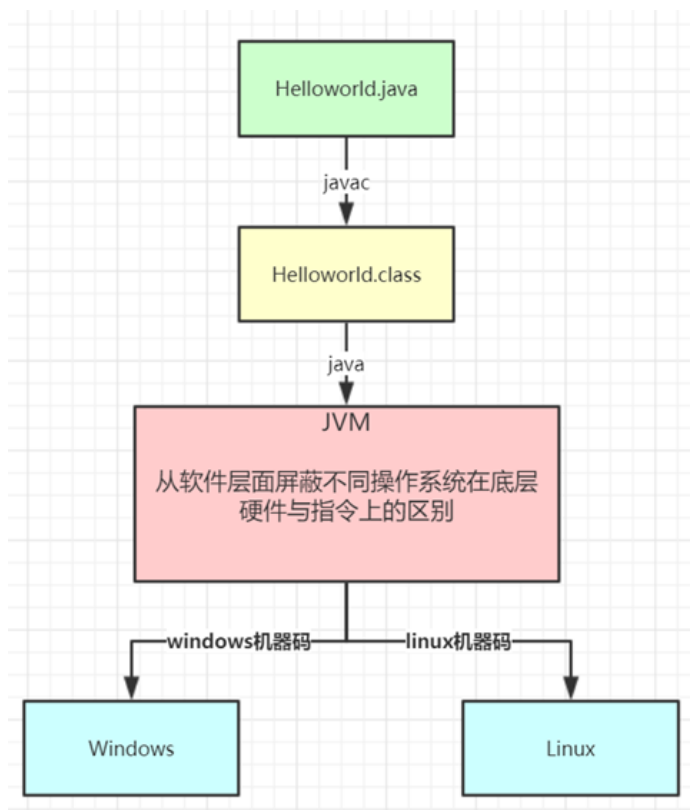


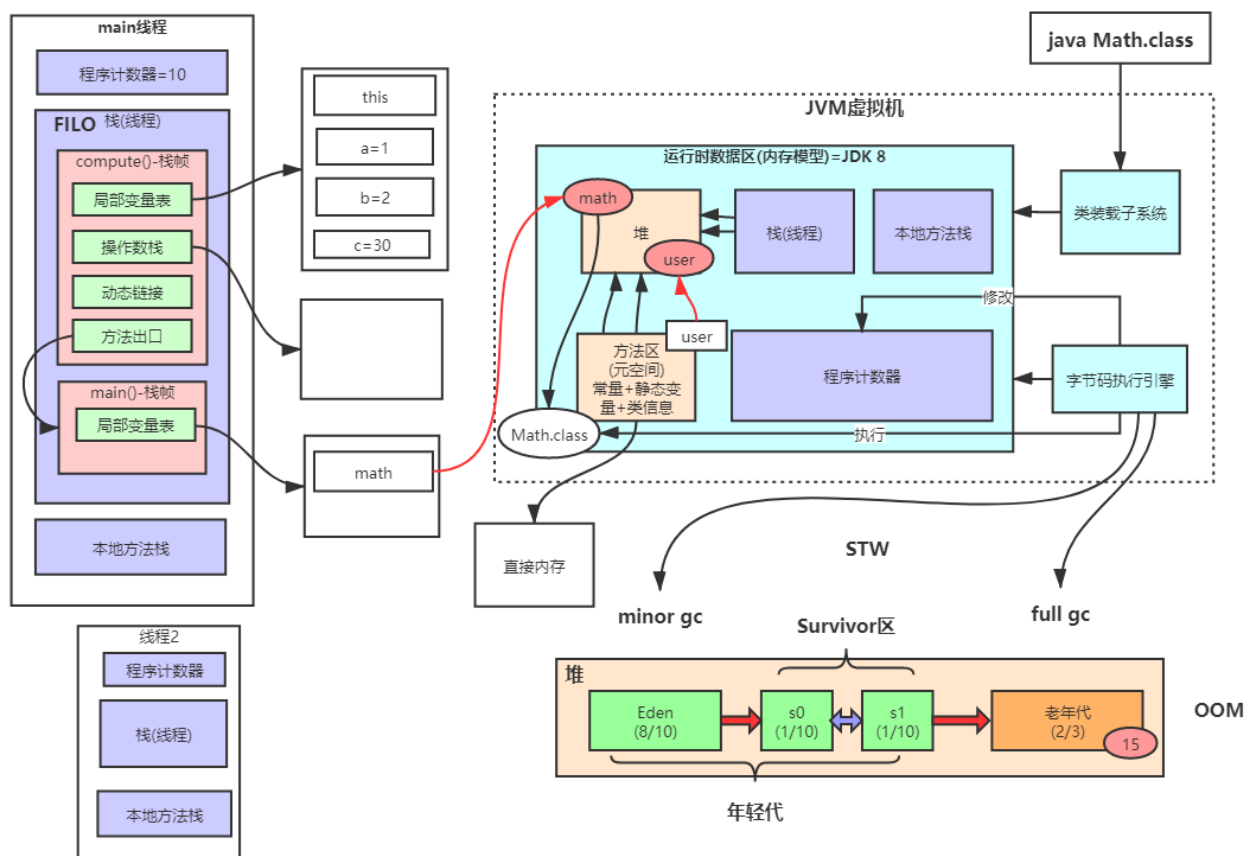
## JDK体系结构



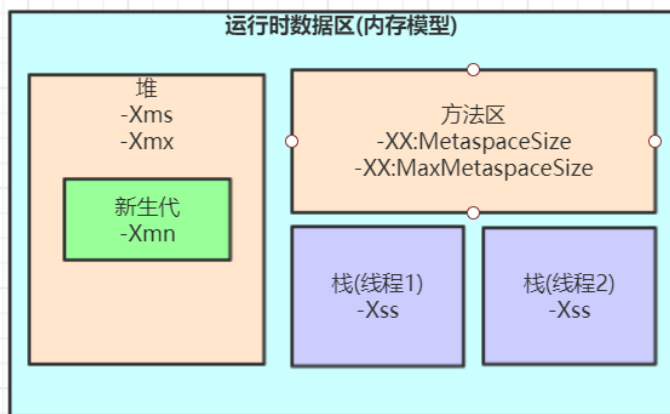
## Java语言的跨平台特性



## JVM整体结构及内存模型



## 二、JVM内存参数设置



Spring Boot程序的JVM参数设置格式(Tomcat启动直接加在bin目录下catalina.sh文件里):

```
1 java -Xms2048M -Xmx2048M -Xmn1024M -Xss512K -XX:MetaspaceSize=256M -XX:MaxMetaspaceSize=256M -jar microservice-eureka-server.jar
```

关于元空间的JVM参数有两个: `-XX:MetaspaceSize=N`和 `-XX:MaxMetaspaceSize=N`

**-XX: MaxMetaspaceSize:** 设置元空间最大值, 默认是-1, 即不限制, 或者说只受限于本地内存大小。

**-XX: MetaspaceSize:** 指定元空间触发Fullgc的初始阈值(元空间无固定初始大小), 以字节为单位, 默认是21M, 达到该值就会触发full gc进行类型卸载, 同时收集器会对该值进行调整: 如果释放了大量的空间, 就适当降低该值; 如果释放了很少的空间, 那么在不超过-XX: MaxMetaspaceSize (如果设置了的话) 的情况下, 适当提高该值。这个跟早期jdk版本的**-XX:PermSize**参数意思不一样, **-XX:PermSize**代表永久代的初始容量。

由于调整元空间的大小需要Full GC, 这是非常昂贵的操作, 如果应用在启动的时候发生大量Full GC, 通常都是由于永久代或元空间发生了大小调整, 基于这种情况, 一般建议在JVM参数中将MetaspaceSize和MaxMetaspaceSize设置成一样的值, 并设置得比初始值要大, 对于8G物理内存的机器来说, 一般我会将这两个值都设置为256M。

**StackOverflowError**示例:

```
1 // JVM设置 -Xss128k(默认1M)
2 public class StackOverflowTest {
3
4     static int count = 0;
5
6     static void redo() {
7         count++;
8         redo();
9     }
10
11     public static void main(String[] args) {
12         try {
13             redo();
14         } catch (Throwable t) {
15             t.printStackTrace();
16             System.out.println(count);
17         }
18     }
19 }
20
21 运行结果:
22 java.lang.StackOverflowError
23   at com.tuling.jvm.StackOverflowTest.redo(StackOverflowTest.java:12)
24   at com.tuling.jvm.StackOverflowTest.redo(StackOverflowTest.java:13)
25   at com.tuling.jvm.StackOverflowTest.redo(StackOverflowTest.java:13)
26   .....
```

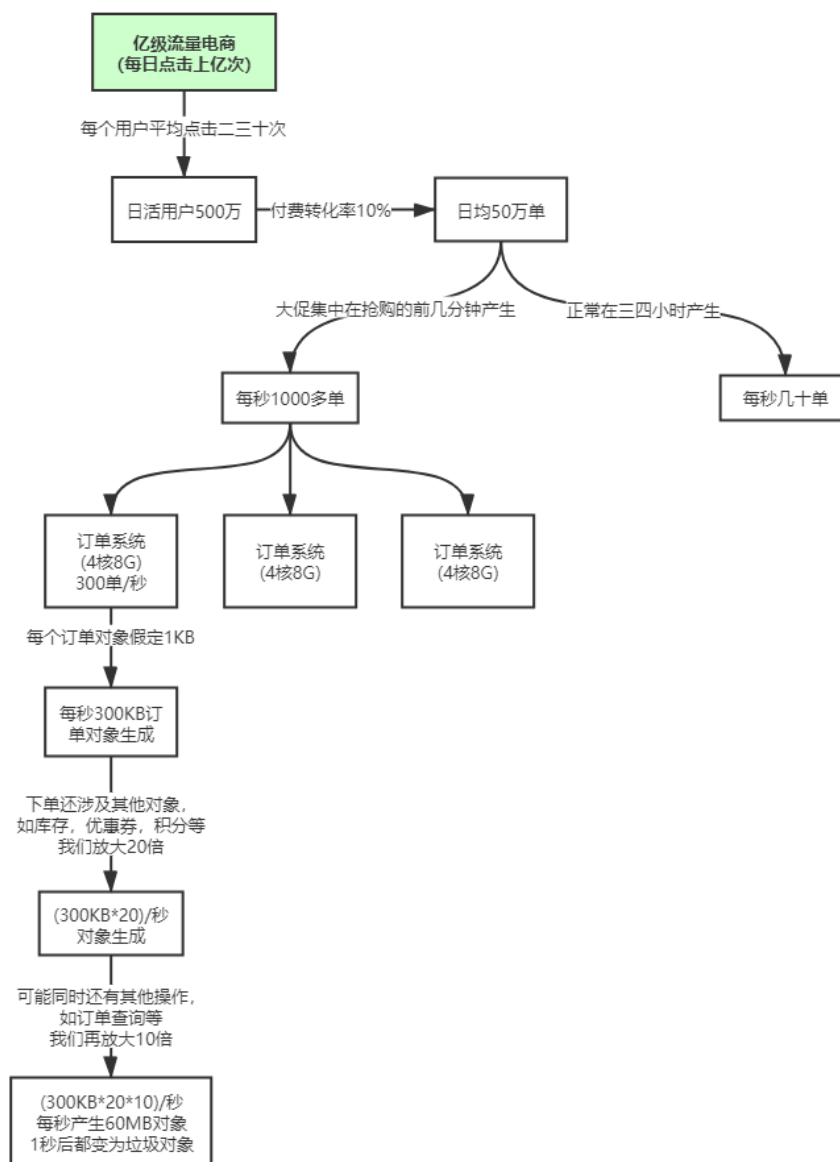
#### 结论:

-Xss设置越小count值越小, 说明一个线程栈里能分配的栈帧就越少, 但是对JVM整体来说能开启的线程数会更多

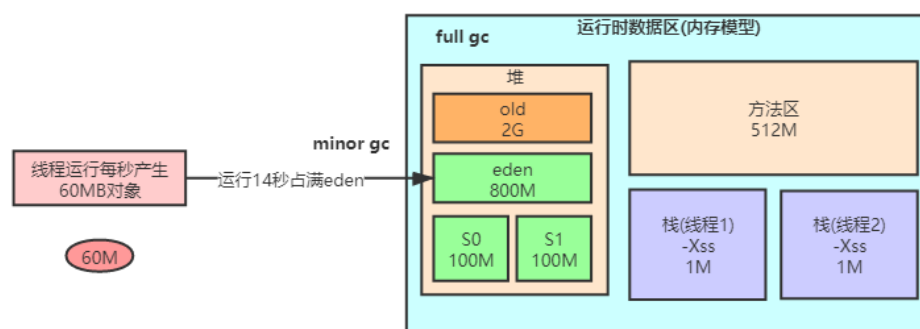
#### JVM内存参数大小该如何设置?

JVM参数大小设置并没有固定标准, 需要根据实际项目情况分析, 给大家举个例子

#### 日均百万级订单交易系统如何设置JVM参数



```
java -Xms3072M -Xmx3072M -Xss1M -XX:MetaspaceSize=512M -XX:MaxMetaspaceSize=512M -jar microservice-eureka-server.jar
```



阿里面试题：能否对JVM调优，让其几乎不发生Full GC

```
java -Xms3072M -Xmx3072M -Xmn2048M -Xss1M -XX:MetaspaceSize=256M -XX:MaxMetaspaceSize=256M -jar microservice-eureka-server.jar
```

**结论：**通过上面这些内容介绍，大家应该对JVM优化有些概念了，就是尽可能让对象都在新生代里分配和回收，尽量别让太多对象频繁进入老年代，避免频繁对老年代进行垃圾回收，同时给系统充足的内存大小，避免新生代频繁的进行垃圾回收。

文档：02-VIP-JVM内存模型深度剖析与优化

1 <http://note.youdao.com/noteshare?id=ad3d29fc27ff8bd44e9a2448d3e2706d&sub=AC12369487BB46F2B3006BB4F3148D01>