1. Classloader

三类类加载器：

Bootstrap ClassLoader ：最顶层的加载类，主要加载核心类库，%JRE\_HOME%\lib下的rt.jar、resources.jar、charsets.jar和class等。另外需要注意的是可以通过启动jvm时指定-Xbootclasspath和路径来改变Bootstrap ClassLoader的加载目录。比如java -Xbootclasspath/a:path被指定的文件追加到默认的bootstrap路径中。我们可以打开我的电脑，在上面的目录下查看，看看这些jar包是不是存在于这个目录。

Extention ClassLoader 扩展的类加载器，加载目录%JRE\_HOME%\lib\ext目录下的jar包和class文件。还可以加载-D java.ext.dirs选项指定的目录。

Appclass Loader也称为SystemAppClass 加载当前应用的classpath的所有类。

加载顺序：

Bootstrap CLassloder

Extention ClassLoader

AppClassLoader

BootstrapClassLoader、ExtClassLoader、AppClassLoader实际是查阅相应的环境属性sun.boot.class.path、java.ext.dirs和java.class.path来加载资源文件

AppClassLoader的parent为ExtClassLoader

Bootstrap ClassLoader是由C/C++编写的，它本身是虚拟机的一部分，所以它并不是一个JAVA类。

一个类加载器查找class和resource时，是通过“委托模式”进行的，它首先判断这个class是不是已经加载成功，如果没有的话它并不是自己进行查找，而是先通过父加载器，然后递归下去，直到Bootstrap ClassLoader，如果Bootstrap classloader找到了，直接返回，如果没有找到，则一级一级返回，最后到达自身去查找这些对象。这种机制就叫做双亲委托

ExtClassLoader的parent为null

自定义ClassLoader步骤

编写一个类继承自ClassLoader抽象类。

复写它的findClass()方法。

在findClass()方法中调用defineClass()。

1. Jvm内存模型

这几个存储区最主要的就是栈区和堆区，那么什么是栈什么是堆呢？说的简单点，栈里面存放的是基本的数据类型和引用，而堆里面则是存放各种对象实例

| **名称** | **特征** | **作用** | **配置** | **异常** |
| --- | --- | --- | --- | --- |
| 栈区 | 线程私有，使用一段连续的内存空间 | 存放局部变量表、操作栈、动态链接、方法出口 | -XSs | StackOverflowError OutOfMemoryError |
| 堆 | 线程共享，生命周期与虚拟机相同 | 保存对象实例 | -Xms -Xmx -Xmn | OutOfMemoryError |
| 程序计数器 | 线程私有、占用内存小 | 字节码行号 | 无 | 无 |
| 方法区 | 线程共享 | 存储类加载信息、常量、静态变量等 | -XX:PermSize -XX:MaxPermSize | OutOfMemoryError |

1. 线程池

ThreadPoolExecutor：

A.ThreadPoolExecutor的重要参数  
    a、corePoolSize：核心线程数  
        \* 核心线程会一直存活，及时没有任务需要执行  
        \* 当线程数小于核心线程数时，即使有线程空闲，线程池也会优先创建新线程处理  
        \* 设置allowCoreThreadTimeout=true（默认false）时，核心线程会超时关闭

    b、queueCapacity：任务队列容量（阻塞队列）  
        \* 当核心线程数达到最大时，新任务会放在队列中排队等待执行

    c、maxPoolSize：最大线程数  
        \* 当线程数>=corePoolSize，且任务队列已满时。线程池会创建新线程来处理任务  
        \* 当线程数=maxPoolSize，且任务队列已满时，线程池会拒绝处理任务而抛出异常

    d、 keepAliveTime：线程空闲时间  
        \* 当线程空闲时间达到keepAliveTime时，线程会退出，直到线程数量=corePoolSize  
        \* 如果allowCoreThreadTimeout=true，则会直到线程数量=0

    e、allowCoreThreadTimeout：允许核心线程超时  
    f、rejectedExecutionHandler：任务拒绝处理器  
        \* 两种情况会拒绝处理任务：  
            - 当线程数已经达到maxPoolSize，切队列已满，会拒绝新任务  
            - 当线程池被调用shutdown()后，会等待线程池里的任务执行完毕，再shutdown。如果在调用shutdown()和线程池真正shutdown之间提交任务，会拒绝新任务  
        \* 线程池会调用rejectedExecutionHandler来处理这个任务。如果没有设置默认是AbortPolicy，会抛出异常  
        \* ThreadPoolExecutor类有几个内部实现类来处理这类情况：  
            - AbortPolicy 丢弃任务，抛运行时异常  
            - CallerRunsPolicy 执行任务  
            - DiscardPolicy 忽视，什么都不会发生  
            - DiscardOldestPolicy 从队列中踢出最先进入队列（最后一个执行）的任务  
        \* 实现RejectedExecutionHandler接口，可自定义处理器

B、ThreadPoolExecutor执行顺序  
        线程池按以下行为执行任务  
    a. 当线程数小于核心线程数时，创建线程。  
    b. 当线程数大于等于核心线程数，且任务队列未满时，将任务放入任务队列。  
    c. 当线程数大于等于核心线程数，且任务队列已满  
        - 若线程数小于最大线程数，创建线程  
        - 若线程数等于最大线程数，抛出异常，拒绝任务

1. JAVA 垃圾回事机制
2. redis 持久化策略
3. Java中的原子性类
4. Java中的锁
5. Eureka集群
6. 设计模式
7. Java中的流
8. Java中的socket
9. http协议
10. dubbo
11. ActiveMQ
12. Spring mvc流程
13. Jsp 九大内置对象
14. Struts2流程
15. Mybatis分页
16. Jdbc实现
17. Spring注入
18. Spring控制反转
19. Spring AOP
20. Redis的session共享
21. 分布式锁的实现
22. kafka
23. DFDS
24. FDS
25. FDS
26. FDS
27. FDSF
28. DSF
29. DSF
30. DSF
31. DSF
32. D
33. d