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

---------rdb:快照形式是直接把内存中的数据保存到一个dump文件中，定时保存，保存策略，配置：

save 900 1

save 300 10

save 60 10000

默认是如上配置：900秒之内，如果超过1个key被修改，则发起快照保存；

300秒内，如果超过10个key被修改，则发起快照保存

1分钟之内，如果1万个key被修改，则发起快照保存

---------aof：每一个写命令都通过write函数追加到appendonly.aof中.Java中的原子性类,配置：appendonly yes

1. lock锁

Lock和ReadWriteLock是两大锁根接口，Lock代表实现类是ReentrantLock（可重入锁），ReadWriteLock（读写锁）的代表实现类是ReentrantReadWriteLock。

1. Eureka集群
2. Eureka元数据

Eureka的元数据有两种：标准元数据和自定义元数据。

标准元数据：主机名、IP地址、端口号、状态页和健康检查等信息，这些信息都会被发布在服务注册表中，用于服务之间的调用。

自定义元数据：可以使用eureka.instance.metadata-map配置

1. 设计模式
2. Java中的流
3. Java中的socket
4. http协议
5. dubbo

调用流程

a.服务容器负责启动，加载，运行服务提供者。

b.服务提供者在启动时，向注册中心注册自己提供的服务。

c.服务消费者在启动时，向注册中心订阅自己所需的服务。

d.注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。

e.服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。

f.服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心

1. ActiveMQ
2. Spring mvc流程

具体步骤：

第一步：发起请求到前端控制器(DispatcherServlet)

第二步：前端控制器请求HandlerMapping查找 Handler （可以根据xml配置、注解进行查找）

第三步：处理器映射器HandlerMapping向前端控制器返回Handler，HandlerMapping会把请求映射为HandlerExecutionChain对象（包含一个Handler处理器（页面控制器）对象，多个HandlerInterceptor拦截器对象），通过这种策略模式，很容易添加新的映射策略

第四步：前端控制器调用处理器适配器去执行Handler

第五步：处理器适配器HandlerAdapter将会根据适配的结果去执行Handler

第六步：Handler执行完成给适配器返回ModelAndView

第七步：处理器适配器向前端控制器返回ModelAndView （ModelAndView是springmvc框架的一个底层对象，包括 Model和view）

第八步：前端控制器请求视图解析器去进行视图解析 （根据逻辑视图名解析成真正的视图(jsp)），通过这种策略很容易更换其他视图技术，只需要更改视图解析器即可

第九步：视图解析器向前端控制器返回View

第十步：前端控制器进行视图渲染 （视图渲染将模型数据(在ModelAndView对象中)填充到request域）

第十一步：前端控制器向用户响应结果

1. Jsp 九大内置对象
2. Struts2流程
3. Mybatis分页
4. Jdbc实现
5. Spring注入
6. Spring控制反转
7. Spring AOP
8. Redis的session共享
9. 分布式锁的实现
10. Threadlocal
11. @Resource、@Autowired、@Qualifier的注解注入及区别

@Autowired和@Qualifier是Spring项目中比较常用的两个注解。一般在同一个实现Bean可以在不同的平台使用，Controller调用Service,Service调用DAO。其中很多实例都是使用@Autowired自动实现，但是如果同一份Bean实例有不同的构造器来创建的时候，使用Autowired自动创建Bean实例IOC容器就分不清了，这时候需要使用@Qualifier来指定特定的bean实例构造方法。

@Autowired可以标注类中引用属性，构造方法，和普通方法。用Autowired标记的属性和方法，都会自动创建实体，当然对于方法中的参数意思是IOC容器自动创建bean实体，并负责给参数列表中的引用了。

@Autowired标记的方法都会在实体创建的时候，自动调用。

@Resource（这个注解属于J2EE的），默认按照名称进行装配，名称可以通过name属性进行指定，如果没有指定name属性，当注解写在字段上时，默认取字段名进行安装名称查找，如果注解写在setter方法上默认取属性名进行装配。当找不到与名称匹配的bean时才按照类型进行装配。但是需要注意的是，如果name属性一旦指定，就只会按照名称进行装配

1. kafka
2. java 原子性
3. FDS
4. FDS
5. FDS
6. FDSF
7. DSF
8. DSF
9. DSF
10. DSF
11. D
12. d