# 总结

## Jvm

Jvm原理：

  java程序经过一次编译之后，将java代码编译为字节码也就是class文件，然后在不同的操作系统上依靠不同的java虚拟机进行解释，最后再转换为不同平台的机器码，最终得到执行。

类加载过程：

加载、链接（验证、准备、解析）、初始化、使用、卸载

加载：通过包名、类名获取定义这个类的二进制字节流；把二进制字节流的静态存储结构转化为方法区可运行的数据结构；在堆中生成一个class对象，作为方法区这些数据的访问入口。

链接：jvm验证.class文件，确保符合jvm的需求；为类变量分配内存并设置初始值；将符号引用转换成直接引用。

初始化：赋值

使用：

卸载：gc垃圾回收

什么时候进行垃圾回收：

利用可达性分析算法，寻找对应的引用节点，找到这个节点以后，继续寻找这个节点的引用节点，当所有的引用节点寻找完毕之后，剩余的节点则被认为是没有被引用到的节点，即无用的节点，无用的节点将会被判定为是可回收的对象。

在java语言中，可作为GC Roots的对象包括下面几种：

a) 虚拟机栈中引用的对象（栈帧中的本地变量表）；

b) 方法区中类静态属性引用的对象；

c) 方法区中常量引用的对象；

d) 本地方法栈中JNI（Native方法）引用的对象。

对象死亡被回收：

**第一次标记**：如果对象在进行可达性分析后发现没有与GC Roots相连接的引用链，那它将会被第一次标记；

**第二次标记**：第一次标记后接着会进行一次筛选，筛选的条件是此对象是否有必要执行finalize()方法。在finalize()方法中没有重新与引用链建立关联关系的，将被进行第二次标记。

第二次标记成功的对象将真的会被回收，如果对象在finalize()方法中重新与引用链建立了关联关系，那么将会逃离本次回收，继续存活。

Jvm内存区域：

程序计数器、虚拟机栈、本地方法栈、堆、方法区

程序计数器是一块很小的内存空间，它是线程私有的，可以认作为当前线程的行号指示器。

虚拟机栈：同计数器也为线程私有，生命周期与相同，就是我们平时说的栈。存储基本数据类型与对象引用的地址。

本地方法栈：执行native方法，可能调用底层的c或者c++，与底层进行交互。

堆：为对象实例以及数组分配内存地址。线程共享

方法区：用于存储已被虚拟机加载的类信息、常量、静态变量，线程共享。

Jvm调优：调整内存大小（jvm初始内存和最大内存）；GC策略

## Spring

对于spring的理解：

Spring是一个轻量级的IoC和AOP容器框架。

Springboot是怎么注入bean的？

由Spring容器(比如 ApplicationContext)来实现,Spring启动时会把所有需要的bean扫描并注册到容器里，在这个过程当中Spring会根据我们定义的bean之间的依赖关系来进行注入，Spring启动时会把这些依赖关系转化成Spring能够识别的数据结构，并根据它来进行bean的初始化，依赖注入等操作。

SpringIOC的三种注入方式：

构造器注入、setter|getter注入、注解注入

构造器注入可能会产生循环依赖问题，比如在Car的构造方法中依赖了Wheel类，而Wheel类的构造方法中也依赖了Car类，此时启动spring容器就会有问题。

AOP：

一般称为面向切面，用于将那些与业务无关，但却对多个对象产生影响的公共行为和逻辑，抽取并封装为一个可重用的模块,减少系统中的重复代码，

降低了模块间的耦合度,可用于权限认证、日志、事务处理。

一般分为静态代理和动态代理：

静态代理：会在编译阶段将AspectJ(切面)织入到Java字节码中，运行的时候就是增强之后的AOP对象。

动态代理：AOP框架不会去修改字节码，而是每次运行时在内存中临时为方法生成一个AOP对象，这个AOP对象包含了目标对象的全部方法，并且在特定的切点做了增强处理，

并回调原对象的方法。

@Autowired和@Resource之间的区别：

(1)@Autowired默认是按照类型装配注入的，默认情况下它要求依赖对象必须存在。

(2)@Resource默认是按照名称装配注入的，只有当找不到与名称匹配的bean才会按照类型来装配注入。

## Redis

Redis的存储类型：String,List,hash,set,sort set

Redis的优势：单线程，执行速度快

Redis的持久化：

Rdb: 保存redis某个时间点上的数据集。 缺点：可能会丢失一些实时数据。（将Reids在内存中的数据库记录定时 dump到磁盘上的RDB持久化）

Aof: 设置fsync策略，可以每秒一次；保证实时数据。(原理是将Reids的操作日志—主要是写操作以追加的方式写入文件)

Redis设置过期时间，过了过期时间，key会被删除吗？

aof是以记录命令（写操作）方式，所以如果还没过期，那么不会有变更的命令，如果过期，会在aof日志插入一条del命令。

rdb的话是内存快照方式，如果持久化时，key已经过期，那么不会持久化，如果在过期之前就已经持久化了，那么在恢复数据时，会判断key是否过期，如果过期不会导入。

Redis与ehcache的区别：

**redis是通过socket访问到缓存服务，效率比ecache低，比数据库要快很多，处理集群和分布式缓存方便，有成熟的方案。**

**ehcache直接在jvm虚拟机中缓存，速度快，效率高；但是缓存共享麻烦，集群分布式应用不方便。**

Redis的删除策略：

被动删除：key过期后不会立即删除，再次访问此key的时候，会检查是否过期，过期即删除。

主动删除：key过期后立即删除，这样对cpu压力比较大。

定时删除：定时统一删除过期的key。这个是折中方案。

## Ehcache

CacheManager：是缓存管理器，可以通过单例或者多例的方式创建，也是Ehcache的入口类。  
Cache：每个CacheManager可以管理多个Cache，每个Cache可以采用hash的方式管理多个Element。  
Element：用于存放真正缓存内容的。

Ehcache的缓存数据淘汰策略:

LFU：最少被使用，缓存的元素有一个hit属性，hit值最小的将会被清出缓存。

LRU：最近最少使用，缓存的元素有一个时间戳，当缓存容量满了，而又需要腾出地方来缓存新的元素的时候，那么现有缓存元素中时间戳离当前时间最远的元素将被清出缓存。

Ehcache的缓存数据过期策略:

Ehcache采用的是懒淘汰机制，每次往缓存放入数据的时候，都会存一个时间，在读取的时候要和设置的时间做TTL比较来判断是否过期。

存储类型：序列化的对象

## HashMap

本质：

jdk1.7中底层是由数组（也有叫做“位桶”的）+链表实现；jdk1.8中底层是由数组+链表/红黑树实现。

初始size为16，扩容：newsize = oldsize\*2，size一定为2的n次幂

可**以存储null键和null值**，线程**不安全**

可以看作是数组和链表结合组成的复合结构，数组被分为一个个桶（bucket），每个桶存储有一个或多个Entry对象(每个Entry对象包含三部分key、value，next)，通过哈希值决定了Entry对象（键值对）在这个数组的寻址；哈希值相同的Entry对象（键值对），则以链表形式存储。

hashMap怎么put、get：

put:首先通过key的hashCode值计算出一个位置，这个位置是数组的下标，把entry元素放到数组对应的位置，这个entry元素包括key、value、还有一个指针，指向下一个entry元素。（如果下标一致，在对应的位置向下插入，指针指向原有位置的entry）

get: 通过key的hashCode值计算出一个位置，找到数组对应下标的位置，再根据key找到对应的元素。

HashMap在jdk1.8中为什么要树化:

**本质上这是个安全问题**。因为在元素放置过程中，如果一个对象哈希冲突，都被放置到同一个桶里，则会形成一个链表，我们知道链表查询是线性的，会严重影响存取的性能。

## Hihernate

一级缓存与二级缓存：

 一级缓存：基于Session的缓存，缓存内容只在当前session有效，session关闭，缓存内容失效。作用范围较小！ 缓存的事件短。

二级缓存：

Hibernate提供了基于应用程序级别的缓存即为二级缓存，可以跨多个session，即不同的session都可以访问缓存数据。

## 线程

线程状态：新建、就绪、运行、阻塞、死亡；

阻塞状态分为：等待阻塞、同步阻塞、其他阻塞

了解的线程池：ExecutorService pool = Executors.newFixedThreadPool(10);

线程池中主线程与子线程的执行顺序：

先执行完子线程，再执行主线程。全部的子线程都执行完，主线程再关闭。

线程开启的是start方法，实际上执行的是run方法。

## Shiro

Springboot中通过ShiroFilterFactoryBean来创建shiroFilter ,对访问的url 的访问进行控制。

Shiro集成单点：

搭建Cas-server服务器，在cas-client(实际上就是shiro项目)，配置登录路径，直接跳转到cas-server中登录，返回一个带有user信息的ticket，cas-client根据ticket信息进行授权。

## 数据库

乐观锁、悲观锁

## 中间件

RabbitMQ:

怎么实现多人重复消费同一个信息：

将消息存到多个队列中去，不同用户去消费不同的队列。

Kafka：

怎么实现多人重复消费同一个信息：

## MySql

Mysql的执行引擎：

innoDB：

优点：InnoDB是一个事务型的存储引擎, 提供了对数据库ACID事务的支持，并且实现了SQL标准的四种隔离级别。

缺点：不支持全文索引，而且它没有保存表的行数，当SELECT COUNT(\*) FROM TABLE时需要扫描全表。

Mysql事务的隔离级别：

Mysql的事务特性：

原子性：

一致性：

隔离性：

持久性：