1. 内存模型

类加载过程，classloader，字节码， .class文件，二进制文件，双亲委派机制。

线程共享部分，堆（字符串常量池），方法区（元空间）存放类信息，

线程不共享部分，虚拟机栈，本地方法栈（native方法），程序计数器，栈帧（局部变量表，操作数栈，方法返回地址，动态链接）

垃圾回收，新生代，老年代，伊甸区，s0，s1，minorgc和fullgc。对象来到eden，存不下就minorgc，还存不下就进老年代，还存不下就oom。被minorgc后进s0或s1，在其中每活过一次gc年龄加1。超过15进老年代。

死亡判断，引用计数（两对象相互引用）和可达性分析（gcroot），强软弱虚

垃圾收集算法，标记清除（内存碎片），标记复制（内存缩小到一半，适合新生代，其他适合老年代），标记整理

1. mysql
2. 基础

Mysql分为service层和存储引擎层，连接器、数据库连接（TCP长连接），查询缓存，解析sql（解析器），执行器（预处理，优化，执行），索引下推（组合索引a，b，当查询条件a＞1 and b=2 ，索引遇到＞会失效，b的判断交给存储引擎而不是执行器，减少回表次数）。

1. 行，页，区，段，表空间，行格式（变长字段长度列表，null值列表，记录头信息，隐藏的row\_id，trx\_id事务id，roll\_ptr记录上一个版本的指针，undolog，MVCC）
2. Mysql主键如果时自增，分表时会造成主键重复。如果使用uuid，一是uuid16位太长，二是uuid是无序的，每次新增数据的时候都会造成索引重排
3. MVCC针对快照读，在可重复读隔离级别下，每次开启事务，会生成快照ReadView，里面包含了当前活跃事务id列表，活跃事务id最小值，活跃事务id最大值的下一个事务id，创建该ReadView的事务id。读取一条记录时会判断该记录trx\_id小于min\_id，则该记录可见，如果大于max\_id，则不可见，如果位于这之间，则判读是否存在于ids中，存在则不可见(不能完全解决幻读)，沿着记录指向的undolog找下一条记录，不存在则可见
4. 脏读，不可重复读，幻读，原子性(undo log)，一致性，隔离性(MVCC)，持久性（redo\_log）
5. 全局锁，表锁，行锁（记录锁（x，s型），间隙锁，next-key lock），updata语句如果条件没索引，进行全表扫描，就会锁全表。Sql语句后面加/g可以分析加的什么锁
6. Bufferpool（数据页，索引页，undo页），undolog，redolog，binlog。读取数据先查bufferpool，没有再读磁盘，读完放bufferpool，修改数据先记录undolog（写到undo页），修改bufferpool，标记脏页，记录相应redolog（redolog buffer）。至此更新完成（WAL）。

redolog刷盘（每隔一秒，或者事务提交），redolog写满了会擦除掉已经刷入磁盘的脏页的记录，数据不完整，不能做备份，主从复制（写入binlog，同步binlog（同步复制，异步复制，半同步复制），回放binlog），只能用于掉电等故障恢复。

Binlog刷盘，先写到binlogcache，等事务提交的时候刷盘

1. mybatis二级缓存(一级基于sqlsession默认开启，二级基于mapper)
2. SpringBoot原理：构造springbootApplication类（添加初始化器和监听器），运行run方法。
3. 创建事件发布器SpringApplicationRunListeners
4. 发布starting事件
5. 准备环境prepareEnvironment（发布environmentPreparedevent使某个监听器加载配置文件）
6. 打印banner
7. 创建容器createApplicationContext（构造方法初始化sacnner和reader）
8. 准备容器prepareContext---->使用初始化器对容器增强，发布listeners.contextPrepared事件，读取source并调用load，然后发布listeners.contextLoaded(context);
9. 刷新容器
10. 刷新容器后（空方法）
11. 发布listeners.started(context);
12. 执行callRunners接口
13. 发布listeners.running(context);
14. Bean生命周期：实例化（反射），属性注入,初始化前(主要处理postConstruct注解)，初始化（处理InitializingBean接口），初始化后（AOP在这里生成代理对象）
15. Spring xml整合第三方框架：引入自定义命名空间，在META-INF里的spring.handlers文件中指定处理类实现NamespaceHandlerSupport接口，在init方法里注册不同标签的处理器，处理器实现BeanDefinitionParser（）接口,实现parse方法，一般在这个方法里直接注册一个bean或者注册一个beanPostProcess进行特点bean的增强（比如AOP就是在这里注册了后置处理器，在after方法里进行判断，并生成代理对象）
16. Sql调优
17. 服务变慢排查
18. JavaWeb 三大组件servlet(包含一个doget和一个dopost方法)，lisener，filter。Web容器包含request域，session域和servletcontext域。Web容器与spring结合时应当在web容器启动时在servletContext域中设置applicationContext对象
19. SpringMVC流程。消除了JavaWeb中众多的servlet，请求统一交给dispatcherServelet处理。过程包含三大组件
20. handlerMapping
21. HandlerAdapter
22. viewResolver
23. 雪花算法时钟回拨
24. Mq订阅方式
25. 策略模式，装饰器模式，代理模式
26. Mq队列(点对点，一个消息消费一次)和发布订阅（一个消息消费多次）
27. topicexchange，根据路由键规则将消息路由到队列（#匹配0个或多个，\*匹配一个）
28. Fanoutexchange，将消息路由到所有与之绑定的队列（广播，无视路由键规则）
29. Direct，点对点，完全匹配路由规则

18、阿萨



1. Feign调用丢失请求头
2. 延时队列。一个topic交换机（a）绑定两个队列（路由键不同b，c），先发送消息到死信队列（b），死信队列指定过期时间，过期交换机（a）和过期路由键(c)，监听c做过期处理
3. 消息可靠性保证：消息到达broker回调，消息到达队列失败回调，消费者开启手动ack，保证消息消费失败时会重新回到队列。try消息发送catch记录消息到mysql，定时任务扫描表，将表中消息重新投递
4. 消息重复
5. Redis持久化，AOF日志（文件过大时会重写，基于当前内存数据生成命令重写生成一个AOF文件，完成后替换老的，如果有写时复制，将这些数据记录到AOF重写缓冲区，等待AOF重写完后，再将重写缓冲区的数据追加到AOF，再替换）和RDB快照（默认开子线程写，也可手动在主线程写
6. Redis混合持久化 使用了混合持久化，AOF 文件的前半部分是 RDB 格式的全量数据，后半部分是 AOF 格式的增量数据。
7. Redis过期策略：惰性删除（每次请求来了先看key是否过期，过期删除，否则正常返回）+定期删除（每次取一定数量的key，检查是否过期，删除过期的，超过25%，则继续循环。可以设定总流程时限）
8. Redis内存淘汰：不进行数据淘汰（报错返回）和进行数据淘汰（设置了过期时间的数据中淘汰（最早过期、最少使用、最久未使用）或者所有数据中进行淘汰（随机、最少使用，最久未使用））
9. Redis集群：一主多从，主写从读，给主节点设置从时，会发生全量同步(RDB),往后都是增量同步。哨兵（sentinel，也需要集群）在哨兵配置文件设置心跳间隔，超时时间(用来选主时排除slave),master地址(设置完了后客户端直接连sentinel就可以访问redis)，主观过期转客观过期数目。
10. 锁（重量级锁）：将锁对象（MarkWord）绑定一Monitor，线程尝试成为Monitor的onwer，成不了就去排队
11. 偏向锁：尝试在MarkWord中设置自己的线程id
12. synchronized原理：偏向锁，如果有其他线程竞争，升级为轻量级锁（CAS尝试获锁），如果自选过多，升级为重量级锁
13. 乐观锁：常见的实现为CAS和版本号，CAS即，预期值，修改值（A1，B1，A2解决ABA问题），非阻塞，对应的悲观锁时阻塞的
14. Wait和sleep：wait放锁，sleep不放，wait需要Object，sleep需要Thread
15. Reentrantlock：相比于synchronized，它可以实现公平锁（即放锁后，谁能拿锁取决于谁先排队的），默认是非公平锁（线程尝试获锁，正好锁被释放，线程无需等待队列其他人，直接拿锁。如果锁没被释放，则进入队列等待）。它可以实现读写锁。它可以尝试获锁（tryLock方法返回Boolean），拿不到就去做其他事
16. HashMap原理，扩容
17. Concurrenthashmap原理
18. 动态线程池实现
19. 同步异步（异步没有阻塞的），阻塞非阻塞（都是同步），多路复用（同步）
20. 零拷贝（一次内核态和用户态的切换，且不会将数据拷贝到内存）