## Java

### 1，String s=new String("abc")创建了几个对象?

答：2个，一个是栈中的s，一个是堆中的“abc”。引申：String s="a"+"b"+"c"创建了几个？答：1个。因为连接的都是常量，变动的只是对象的引用。

### MyBatis中占位符#{}与${}的区别？

答：#将传入的数据都当成一个字符串，会对传入的数据自动加上引号,在sql解析时会变成 ？；$将传入的数据直接显示生成在SQL中。注意：使用$占位符可能会导致SQL注入攻击，能用#的地方就不要使用$，写order by子句的时候应该用$而不是#。

### MyBatis常用动态sql标签

答：trim标签，例<trim prefix="(" suffix=")" suffixOverrides=",">

<if test="name != null"> NAME, </if>

</trim>

<foreach>，例<foreach item="item" index="index"

collection="list" open="(" separator="," close=")">

#{item}

</foreach>

<If>,<choose>...<when><otherwise>

### Spring中Bean的作用域有哪些？

1，Singleton（单例模式）：使用该属性定义Bean时，I0C 容器仅创建一个Bean实例，I0C 容器每次返回的是同一个 Bean实例。

2，Prototype（多例模式）：使用该属性定义Bean时，I0C 容器可以创建多个Bean实例，每次返回的都是一个新的实例。

3，Request：该属性仅对HTTP请求产生作用，使用该属性定义Bean时，每次HTTP请求都会创建一个新的Bean，适用于WebApplicationContext环境

4，Session：该属性仅用于HTTP Session， 同一个Session共享一个Bean实例。不同Session使用不同的实例。

5，global-session（全局会话）：该属性仅用于HTTP Session， 同session 作用域不同的是，所有的Session共享一个Bean实例。

### 5，Spring框架中Bean的生命周期？

答：实例化--属性赋值--初始化--销毁。

### 6，Spring事务的7大传播机制

1，PROPAGATION\_REQUIRED：（默认）若当前存在事务，则加入该事务，若不存在事务，则新建一个事务。

2，PAOPAGATION\_REQUIRE\_NEW：若当前没有事务，则新建一个事务。若当前存在事务，则新建一个事务，新老事务相互独立。外部事务抛出异常回滚不会影响内部事务的正常提交。

3，PROPAGATION\_NESTED：如果当前存在事务，则嵌套在当前事务中执行。如果当前没有事务，则新建一个事务，类似于REQUIRE\_NEW。

4，PROPAGATION\_SUPPORTS：支持当前事务，若当前不存在事务，以非事务的方式执行。

5，PROPAGATION\_NOT\_SUPPORTED：以非事务的方式执行，若当前存在事务，则把当前事务挂起。

6，PROPAGATION\_MANDATORY：强制事务执行，若当前不存在事务，则抛出异常。

7，PROPAGATION\_NEVER：以非事务的方式执行，如果当前存在事务，则抛出异常。

思考：A方法调用B方法，AB都有事务，并且传播特性都不同，那么A有异常B怎么办，B有异常A又会怎么样？

核心逻辑：判断内外方法是否是同一个事务。是，异常统一在外层方法处理。不是，内层方法有可能影响到外层方法，但外层方法是不会影响到内层方法的。

### 7，Spring事务实现方式有哪些？

答：声明式事务：声明式事务也有两种实现方式，基于 xml 配置文件的方式和注解方式（在类上添加 @Transaction 注解）。

编码方式：提供编码的形式管理和维护事务。就是JDBC时代那样写。

### 8，Spring如何解决循环依赖？

答：要点--三级缓存，提前暴露对象，aop等等。

现象--A依赖B，B依赖A。

先创建A对象，实例化A，此时A中的b属性为空，填充属性b

从容器中查找B对象，如果找到了直接赋值，找不到直接创建B对象

实例化B对象，此时B对象中的a属性为空，填充属性a

从容器中查找A对象，找不到，直接创建

形成闭环的原因：

以上4步仔细看的话，会发现第4步时A对象是存在的，只不过A对象不是一个完整的状态，只完成了实例化未完成初始化，所以只要可以提前暴露某个不完整对象的引用，将实例化和初始化分开操作，成为了解决循环依赖的关键。

具体解决：

三级缓存--3个Map，一级存放完整的对象，二级存放是非完整的对象，而三级缓存存放代理的Bean，存在的意义是为了保证在整个容器的运行过程中同名的bean只能有一个。

有了第3级缓存，在第4步时，B就可以从3级缓存中得到A对象的代理，从而完成自身的创建，有了B对象A也就可以完成创建了。

### 9，HashMap的实现原理

答：HashMap 基于 Hash 算法实现的，我们通过 put(key,value)存储，get(key)来获取。当传入 key 时，HashMap 会根据 key. hashCode() 计算出 hash 值，根据 hash 值将 value 保存在 bucket 里。当计算出的 hash 值相同时，我们称之为 hash 冲突，HashMap 的做法是用链表和红黑树存储相同 hash 值的 value。当 hash 冲突的个数比较少时，使用链表否则使用红黑树。

### 10，Mybatis一级缓存和二级缓存

答：一级缓存的作用域是SqlSession范围的，当在同一个sqlSession中执行两次相同的sql语句时，第一次执行完毕会将数据库中查询的数据写到缓存，第二次查询时会从缓存中获取数据，不再去底层数据库查询。（默认开启）。

二级缓存的作用域是mapper的同一个namespace。不同的sqlSession两次执行相同的namespace下的sql语句，且向sql中传递的参数也相同，即最终执行相同的sql语句，则第一次执行完毕会将数据库中查询的数据写到缓存，第二次查询会从缓存中获取数据，不再去底层数据库查询。

### 11，1.8以后线程池的要点

答：新增newWorkStealingPool了。参考（https://blog.csdn.net/tjbsl/article/details/98480843）

### 12，BIO，NIO，AIO的区别

答：BIO：同步并阻塞，服务实现模式为一个连接对应一个线程，即客户端发送一个连接，服务端要有一个线程来处理。如果连接多了，线程数量不够，就只能等待，即会发生阻塞。

NIO：同步非阻塞，服务实现模式是一个线程可以处理多个连接，即客户端发送的连接都会注册到多路复用器上，然后进行轮询连接，有I/O请求就处理。

AIO：异步非阻塞，引入了异步通道，采用的是proactor模式，特点是：有效的请求才启动线程，先有操作系统完成在通知服务端。

### 13，SpringBoot2.0的变化

答：1，使用了Spring WebFlux异步非阻塞的响应式编程框架，能够充分利用多核CPU的硬件资源去处理大量的并发请求。

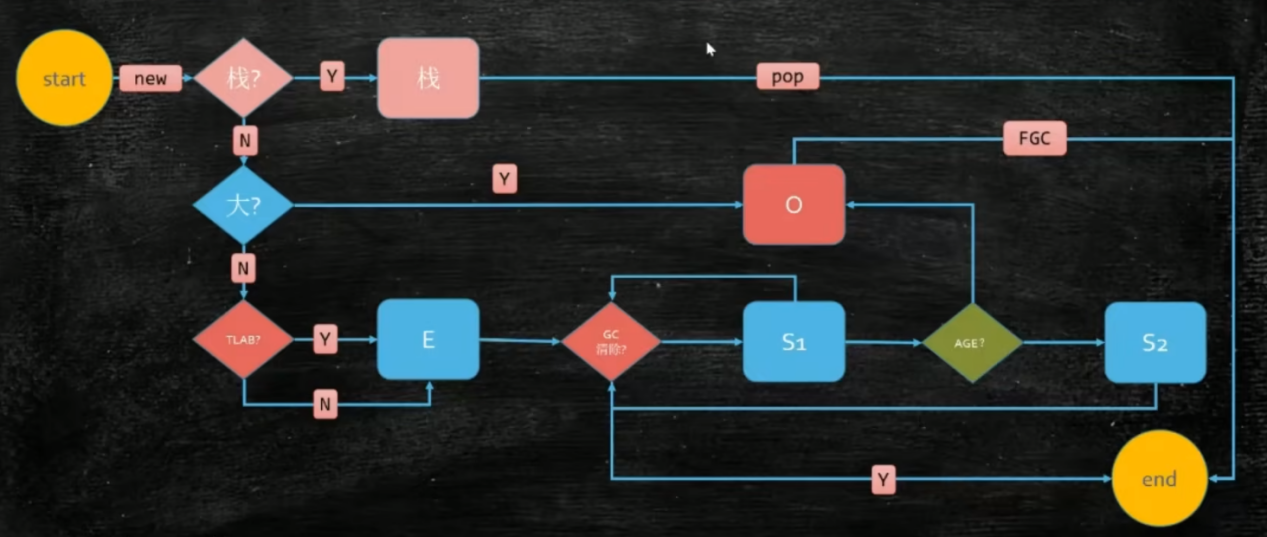
2，支持 Quartz。3，支持Security。4，Hikari数据库连接池。等等

### 14，MyBatis与Hibernate的区别

答：1，Hibernate是全自动，Mybatis是半自动。2，Hibernate数据库移植性大于Mybatis。

3，Hibernate日志功能比Mybatis更详细。4，sql优化上，Mybatis要比Hibernate强很多。

### 15，对象在JVM中如何分配？



### 16，高并发中的集合类

答：第一代：Vector、Hashtable。内部的方法都用synchronized修饰，效率低。

第二代：ArrayList、HashMap。自身线程不安全，但效率高。可使用Collections.synchronizedList(list)/Collections.synchronizedList(map)/Collections.synchronizedXXX(xxx)转为线程安全，底层使用synchronized在方法内部上锁。

第三代：ConcurrentHashMap。在大量并发情况下提高效率和安全。

### 17，CAP定理

答：C：一致性。在多个副本中要保证数据一致。A：高可用。在任何故障下，都能获得服务端非错误的响应。P：分区容错性。不会因为某个分区挂掉而导致整个系统瘫痪。

### 18，JVM常用命令

答：jinfo == 查看JVM配置参数

-- jinfo -flag 属性名 PID

-- jinfo -flags PID == 查看所有JVM参数

jstat == 查看JVM性能信息

-- jstat -class PID 500 10 == 打印类加载信息，每隔500毫秒打印1次，打10次

-- jstat -gc PID 500 10 == 打印gc信息

jstack == 查看线程堆栈情况（可分析JVM死锁） -- jstack PID

jmap == 生成堆内存快照

## MySQL

### 1，常用函数

答：CONCAT(s1,s2...,sn)，TRIM(str)，SUM(col)，LENGTH(s)，SUBSTR(s, start, length)，REPLACE(str,from\_str,to\_str)，AVG(col)，COUNT(col)，MIN(col)，MAX(col)，ROUND(x,y)，TRUNCATE(x,y)...

### 行列互转

答：可使用case...when....then 进行行转列、使用UNION ALL进行列转行。

### 什么情况下应不建或少建索引？

答：表记录太少；经常插入、删除、修改的表；数据重复且分布平均的表字段，假如一个表有10万行记录，有一个字段A只有T和F两种值，且每个值的分布概率大约为50%，那么对这种表A字段建索引一般不会提高数据库的查询速度；经常和主字段一块查询但主字段索引值比较多的表字段。

### MySQL索引类型

答：普通、唯一、主键、联合、全文。Tips：索引是保存在磁盘上的。

### MVCC解决的问题

答：MVCC--多版本并发控制。1，解决了数据库脏读，幻读，不可重复读等事务隔离问题。2，在并发读写数据库时，读操作不用阻塞写操作，写也不阻塞读，提高了性能。

### 6，事务的四大特性（ACID）

原子性，一致性，隔离性，持久性

### 7，事务的隔离级别

1. 读未提交（read uncommitted）：事务尚未提交，其他事务即可以看到该事务的修改结果。隔离级别最差，脏读、不可重复读、幻读都不能避免。
2. 读提交（read committed）：事务只能看到其他事务提交之后的数据。可避免脏读。不可重复读、幻读无法避免。大部分数据库的默认隔离级别（MySQL不是！）。
3. 可重复读（repeatable read）：MySQL innodb默认隔离级别。一个事务多次查询，无论其他事务对数据如何修改，看到的数据都是一致的。可避免脏读、不可重复读，幻读无法避免。（MySQL利用MVCC已经解决了幻读的问题）。
4. 序列化（serializable）：事务顺序执行，可避免脏读、不可重复读、幻读，但效率最差。因为X事务执行时，其他事务必须等待。

Tips：

脏读：读到了事务还未提交的数据。

不可重复读：一个事务执行过程中同语句多次查询可能返回不能的结果。

以上两者的区别：前者是一个事务读取了另一未完成的事务执行过程中的数据，后者是一个事务执行过程中，另一个事务提交并修改了当前事务正在读取的数据。

幻读：比如事务T1批量对一个表中某一列列值为1的数据修改为2，但在此同时，事务T2插入了一条值为1的数据，并完成提交。当事务T1查看刚刚完成修改操作的数据，会发现还有一条值为1的数据没有被修改，而其实这是T2插入进去的，这就是幻读。

幻读和不可重复读的区别：后者查询的都是同一个数据项，而幻读针对的是一批数据整体。

### 执行计划

答system>const>eq\_ref>ref>fulltext>ref\_or\_null>index\_merge>unique\_subquery>index\_subquery>range>index>ALL。一般情况下，查询至少要保证达到range级别，最好能达到ref。

## Redis

### Redis的数据类型

答：String、List、Set、ZSet、Hash。

### Redis的淘汰策略

答：noeviction(默认策略)：对于写请求不再提供服务，直接返回错误（DEL请求和部分特殊请求除外）；

allkeys-lru：从所有key中使用LRU算法进行淘汰；

volatile-lru：从设置了过期时间的key中使用LRU算法进行淘汰；

allkeys-random：从所有key中随机淘汰数据；

volatile-random：从设置了过期时间的key中随机淘汰；

volatile-ttl：在设置了过期时间的key中，根据key的过期时间进行淘汰，越早过期的越优先被淘汰；

volatile-lfu：从所有配置了过期时间的key中驱逐使用频率最少的key；

allkeys-lfu：从所有key中驱逐使用频率最少的key；

当使用volatile-lru、volatile-random、volatile-ttl这三种策略时，如果没有key可以被淘汰，则和noeviction一样返回错误。

### Redis分布式锁

答：先拿 setnx 来争抢锁，抢到之后，再用 expire 给锁加一个过期时间防止锁忘记了释放。如果在 setnx 之后执行 expire 之前进程意外 crash 或者要重启维护了，那会怎么样？set 指令有非常复杂的参数，利用这些入参同时把 setnx 和 expire 合成一条指令来执行。

### Redis持久化方式

答：RDB --根据指定的规则“定时”将内存中的数据存储在硬盘上，生成的快照，

适合用于进行备份和灾难恢复，在恢复大数据集时的速度比AOF的恢复速度要快。缺点：因为是“定时”的，可能8点存储了数据，8点59分机器宕机，那么这59分钟的数据是无法恢复的。

AOF --默认不开启。AOF采用日志的形式来记录每个写操作，并追加到文件中。开启后，执行更改Redis数据的命令时，就会把命令写入到AOF文件中。

### Redis集群模式

* 主从模式：读写分离，主节点挂掉需要手动切换新的主节点。
* 哨兵模式：多了一个哨兵服务来监控主从节点，当主节点挂掉后，可自动选出主节点。
* Cluster 集群模式

### 穿透、击穿、雪崩

穿透：缓存不存在，数据库不存在，高并发，少量key。解决方案：缓存空值或者默认值；非法请求校验。

击穿：缓存不存在，数据库存在，高并发，少量key。解决方案：互斥锁。

雪崩：缓存不存在，数据库存在，高并发，大量key。解决方案：过期时间随机；互斥锁，保证同一时间只有一个请求来构建缓存，后面的请求继续从缓存读取。

## MongoDB

### 1，常用语句

答：查询--db.collectionName.find({key:value})

更新--db.collectionName.update({key:value},{$set:{newkey:newValue}})

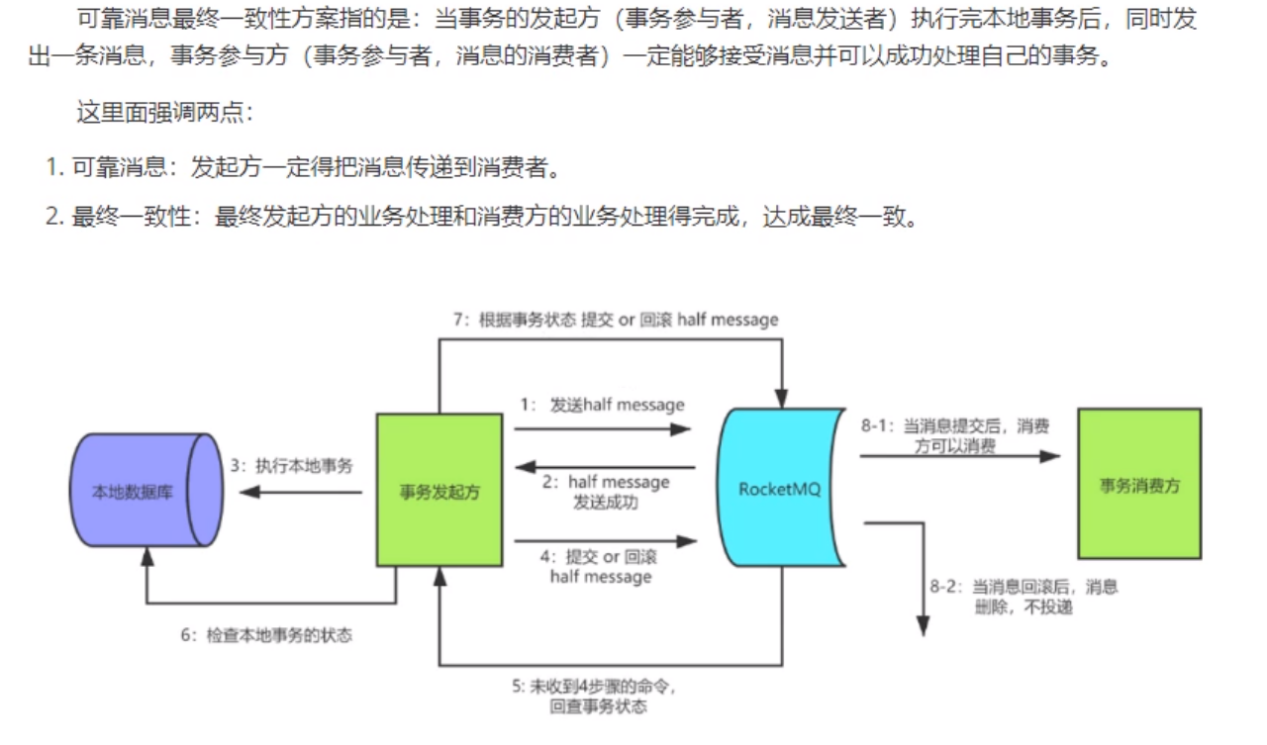
删除--db.collectionName.remove({key:value})

排序--使用 1 和 -1 来指定排序方式，其中1表示升序，而-1表示降序。

db.connectionName.find({key:value}).sort({columnName:1})

## MQ

### 可靠消息服务方案



## Dubbo

### 1，什么是 RPC？

答：RPC 就是 Remote Procedure Call，远程过程调用，它相对应的是本地过程调用。

### 2，RPC 和HTTP协议的区别？

答：RPC是传输层协议(4层)，而HTTP协议是应用层协议(7层)，所以RPC速度更快；RPC协议是长链接，HTTP协议一般采用短连接需要3次握手；RPC协议传递数据是加密压缩传输；RPC协议一般都有注册中心，有丰富的监控机制；

### 3，Dubbo简介

答：首先服务提供者 Provider 启动然后向注册中心注册自己所能提供的服务，服务消费者 Consumer 启动向注册中心订阅自己所需的服务。然后注册中心将Provider元信息通知给 Consumer， 之后 Consumer 因为已经从注册中心获取到了Provider的地址，因此可以通过负载均衡选择一个 Provider 直接调用。

如果Provider元数据变更的话注册中心会把变更推送给Consumer。Provider和Consumer都会在内存中记录调用的次数和时长，然后定时的发送到监控中心。

### 4，Dubbo启动时如果依赖的服务不可用会怎样？

答：Dubbo 缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，阻止 Spring 初始化完成，默认 check="true"，可以通过 check="false" 关闭检查。

### 6，Dubbo有哪几种集群容错方案，默认是哪种？



### 7，Dubbo有哪几种负载均衡策略，默认是哪种？



### 8，注册了多个同一样的服务，如果测试指定的某一个服务呢？

答：可以配置环境点对点直连，绕过注册中心，将以服务接口为单位，忽略注册中心的提供者列表。

### 9，当一个服务接口有多种实现时怎么做？

答：当一个接口有多种实现时，可以用 group 属性来分组，服务提供方和消费方都指定同一个 group 即可。

### 10，Dubbo的注册中心集群挂掉，发布者和订阅者之间还能通信么？

答：可以通讯。启动 Dubbo 时，Consumer会从 Zookeeper 拉取注册的Provider的地址接口等数据，缓存在本地。每次调用时，按照本地存储的地址进行调用。

### 11，Dubbo支持服务降级吗？

答：以通过 dubbo:reference 中设置 mock=“return null”。mock 的值也可以修改为 true，然后再跟接口同一个路径下实现一个 Mock 类，命名规则是 “接口名称+Mock” 后缀。然后在 Mock 类里实现自己的降级逻辑。

## Docker

### 常用命令

* 拉取/更新镜像：docker pull
* 推镜像：docker push
* 列出所有镜像：docker images
* 删除镜像：docker rmi
* 列出所有容器：docker ps
* 删除容器：docker rm

### 使用流程

答：创建Dockerfile后，docker build创建容器的镜像； 推送或拉取镜像。

### Dockerfile常见指令

答：FROM/RUN/CMD/ENTRYPOINT/WORKDIR/ENV/VOLUME/CPOY/ADD（COPY:直接拷贝; ADD:拷贝+解压功能）

## K8S

### 1，常用命令

* 创建资源：kubectl create|apply -f file.yaml
* 查看资源列表：kubectl get pod ... -n namespace [-o wide] [-w]
* 查看特定资源详情：kubectl describe pod podname -n namespace
* 查看资源定义：kubectl get deploy deployname -n namespace -o yaml [>file.yaml] 重定向到文件里，可以再次apply -f yaml
* 删除资源（慎用）：kubectl delete pod podname -n namespace [--force --grace-period=0] 强制删除grace-period表示过渡存活期，默认30s，在删除POD之前允许POD慢慢终止其上的容器进程，从而优雅退出，0表示立即终止POD
* 查看pod log：kubectl logs [-f] podname -n namespace
* 查看node节点标签：kubectl get node --show-labels
* 查看指定标签的节点： kubectl get node -l key=value
* 节点增加标签：kubectl label nodes nodename key=value
* 删除节点标签：kubectl label nodes nodename key-
* 修改节点标签： kubectl label nodes nodename key=newvalue --overwrite

### 2，常用组件

* Etcd(Master节点)：存放数据以及保存整个集群的状态
* apiserver(Master节点)：提供了资源操作的唯一入口，并提供认证、授权、访问控制、API注册和发现等机制
* controller manager(Master节点)：负责维护集群的状态，比如故障检测、自动扩展、滚动更新等
* scheduler(Master节点)：负责资源的调度，按照预定的调度策略将Pod调度到相应的机器上
* Kubelet(Node节点)：负责维护容器的生命周期，同时也负责Volume（CVI）和网络（CNI）的管理
* kube-proxy(Node节点)：负责为Service提供cluster内部的服务发现和负载均衡
* Pod(Node节点)

### 3，常用控制器

* Deployment：部署无状态应用
* DaemonSet：每个节点上都部署一个，用于收集、监控日志
* StatufluSet：部署有状态应用

### 4，Service是什么？

答：Pod每次重启或者重新部署，其IP地址都会产生变化，这使得pod间通信和pod与外部通信变得困难，这时候，就需要Service为pod提供一个固定的入口。

### 5，Service的类型有哪些？

* ClusterIP：（默认类型）分配一个仅Cluster内部可以访问的虚拟IP
* NodePort：在ClusterIP基础上为Service在每台机器上绑定一个端口，这样就可以通过<NodeIP>:NodePort来访问该服务
* LoadBalancer：在NodePort的基础上，借助cloud provider创建一个外部负载均衡器，并将请求转发到<NodeIP>:NodePort。（收费，NodePort也可以自己在外面套个Nginx来达到目的）
* ExtennalName：将外部的一个服务定义一个集群内部的别名

## Web

### 常见的jQuery 选择器

答：元素选择器---$("html标签名")；id选择器---$("#id")；class选择器---$(".class")