1、JVM内存结构

本地方法栈、Java虚拟机栈、程序计数器、堆、元空间、直接内存

2、字符串常量在哪个区域

字符串常量池，jdk8之前是perm区，jdk8及以后是在堆上面。

3、类文件常量池在哪个区域

类文件常量池，是位于方法区，也就是元空间。

4、类加载有哪些过程

加载

验证

准备（为一些类变量分配内存，并将其初始化为默认值）

解析（将符号引用替换为直接引用）

初始化

5、类加载器

Bootstrap ClassLoader（启动类加载器）

Extention ClassLoader（扩展类加载器）

App ClassLoader（应用类加载器）

6、软引用和弱引用的区别

软引用一般用于维护一些可有可无的对象。在内存足够的时候，软引用对象不会被回收，只有在内存不足时，系统则会回收软引用对象，如果回收了软引用对象之后仍然没有足够的内存，才会抛出内存溢出异常。

弱引用对象相比较软引用，要更加无用一些，它拥有更短的生命周期。当JVM进行垃圾回收时，无论内存是否充足，都会回收被弱引用关联的对象

7、垃圾收集算法

标记清除（碎片化）

复制算法（浪费空间）

标记整理算法（效率比前两者差）

分代收集算法（老年代一般使用“标记-清除”、“标记-整理”算法，年轻代一般用复制算法）

8、JVM怎么判断哪些对象需要删除哪些不用删除

根据GC Root进行查找标记

9、CMS垃圾收集器收集的过程？

CMS 工作机制相比其他的垃圾收集器来说更复杂，整个过程分为以下 4 个阶段：

初始标记

只是标记一下 GC Roots 能直接关联的对象，速度很快，仍然需要暂停所有的工作线程。

并发标记

进行 GC Roots 跟踪的过程，和用户线程一起工作，不需要暂停工作线程。

重新标记

为了修正在并发标记期间，因用户程序继续运行而导致标记产生变动的那一部分对象的标记记录，仍然需要暂停所有的工作线程。

并发清除

清除 GC Roots 不可达对象，和用户线程一起工作，不需要暂停工作线程。由于耗时最长的并发标记和并发清除过程中，垃圾收集线程可以和用户线程一起并发工作， 所以总体上来看CMS 收集器的内存回收和用户线程是一起并发地执行。

10、G1垃圾收集器收集的过程？

Garbage first 垃圾收集器是目前垃圾收集器理论发展的最前沿成果，相比与 CMS 收集器， G1 收集器两个最突出的改进是：

1. 基于标记-整理算法，不产生内存碎片。

2. 可以非常精确控制停顿时间，在不牺牲吞吐量前提下，实现低停顿垃圾回收。

G1 收集器避免全区域垃圾收集，它把堆内存划分为大小固定的几个独立区域，并且跟踪这些区域的垃圾收集进度，同时在后台维护一个优先级列表，每次根据所允许的收集时间， 优先回收垃圾最多的区域。区域划分和优先级区域回收机制，确保 G1 收集器可以在有限时间获得最高的垃圾收集效率。

11、ThreadLocal原理

可以看这篇：

https://www.jianshu.com/p/98b68c97df9b

12、synchronized锁升级

偏向锁

在 JDK1.8 中，其实默认是轻量级锁，但如果设定了 -XX:BiasedLockingStartupDelay = 0 ，那在对一个 Object 做 syncronized 的时候，会立即上一把偏向锁。当处于偏向锁状态时， markwork 会记录当前线程 ID 。

升级到轻量级锁

当下一个线程参与到偏向锁竞争时，会先判断 markword 中保存的线程 ID 是否与这个线程 ID 相等，如果不相等，会立即撤销偏向锁，升级为轻量级锁。每个线程在自己的线程栈中生成一个 LockRecord ( LR )，然后每个线程通过 CAS (自旋)的操作将锁对象头中的 markwork 设置为指向自己的 LR 的指针，哪个线程设置成功，就意味着获得锁。关于 synchronized 中此时执行的 CAS 操作是通过 native 的调用 HotSpot 中 bytecodeInterpreter.cpp 文件 C++ 代码实现的，有兴趣的可以继续深挖。

升级到重量级锁

如果锁竞争加剧(如线程自旋次数或者自旋的线程数超过某阈值， JDK1.6 之后，由 JVM 自己控制该规则)，就会升级为重量级锁。此时就会向操作系统申请资源，线程挂起，进入到操作系统内核态的等待队列中，等待操作系统调度，然后映射回用户态。在重量级锁中，由于需要做内核态到用户态的转换，而这个过程中需要消耗较多时间，也就是"重"的原因之一。

13、CAS的问题

ABA问题，可以用版本号来避免这个问题。

具体可以看这篇：

https://blog.csdn.net/summerZBH123/article/details/80642467

14、LongAdder原理

LongAdder类与AtomicLong类的区别在于高并发时前者将对单一变量的CAS操作分散为对数组cells中多个元素的CAS操作，取值时进行求和；而在并发较低时仅对base变量进行CAS操作，与AtomicLong类原理相同。

具体可以看这篇：

https://www.jianshu.com/p/ec045c38ef0c

15、动态代理

具体可以看这篇：

https://mp.weixin.qq.com/s/lR2pJTy5cbX43YvaQ8uUgQ

16、Java单例模式

可以看这篇：

https://blog.csdn.net/fd2025/article/details/79711198

17、JWT

可以看这篇：

https://baijiahao.baidu.com/s?id=1608021814182894637

18、加密和签名的区别

加密是为了防泄露

签名是为了防伪造、防篡改

19、对称加密与非对称加密的区别

对称加密: 加密和解密的秘钥使用的是同一个.

非对称加密: 与对称加密算法不同，非对称加密算法需要两个密钥：公开密钥（publickey）和私有密钥（privatekey）。

20、JWT的缺点

1、JWT默认不加密，但可以加密。生成原始令牌后，可以使用改令牌再次对其进行加密。

2、当JWT未加密方法是，一些私密数据无法通过JWT传输。

3、JWT不仅可用于认证，还可用于信息交换。善用JWT有助于减少服务器请求数据库的次数。

4、JWT的最大缺点是服务器不保存会话状态，所以在使用期间不可能取消令牌或更改令牌的权限。也就是说，一旦JWT签发，在有效期内将会一直有效。

5、JWT本身包含认证信息，因此一旦信息泄露，任何人都可以获得令牌的所有权限。为了减少盗用，JWT的有效期不宜设置太长。对于某些重要操作，用户在使用时应该每次都进行进行身份验证。

6、为了减少盗用和窃取，JWT不建议使用HTTP协议来传输代码，而是使用加密的HTTPS协议进行传输。

21、分布式session

可以看这篇：

https://www.cnblogs.com/daofaziran/p/10933221.html

22、微服务有什么好处

独立的可扩展性，每个微服务都可以独立进行横向或纵向扩展，根据业务实际增长情况来进行快速扩展；

独立的可升级性，每个微服务都可以独立进行服务升级、更新，不用依赖于其它服务，结合持续集成工具可以进行持续发布，开发人员就可以独立快速完成服务升级发布流程；

易维护性，每个微服务的代码均只专注于完成该单个业务范畴的事情，因此微服务项目代码数量将减少至IDE可以快速加载的大小，这样可以提高了代码的可读性，进而可以提高研发人员的生产效率；

语言无关性，研发人员可以选用自己最为熟悉的语言和框架来完成他们的微服务项目（当然，一般根据每个公司的实际技术栈需要来了），这样在面对新技术或新框架的选用时，微服务能够更好地进行快速响应；

故障和资源的隔离性，在系统中出现不好的资源操作行为时，例如内存泄露、数据库连接未关闭等情况，将仅仅只会影响单个微服务；

优化跨团队沟通，如果要完全实践微服务架构设计风格，研发团队势必会按照新的原则来进行划分，由之前的按照技能、职能划分的方式变为按照业务（单个微服务）来进行划分，如此这般团队里将有各个方向技能的研发人员，沟通效率上来说要优于之前按照技能进行划分的组织架构；

原生基于“云”的系统架构设计，基于微服务架构设计风格，我们能构建出来原生对于“云”具备超高友好度的系统，与常用容器工具如Docker能够很方便地结合，构建持续发布系统与IaaS、PaaS平台对接，使其能够方便的部署于各类“云”上，如公用云、私有云以及混合云。

23、微服务有什么缺点

增加了系统复杂性

运维难度增加

本地调用变成RPC调用，有些操作会比较耗时

可能会引入分布式事务

24、spring cloud有哪些组件

Spring Cloud Netflix：核心组件，可以对多个Netflix OSS开源套件进行整合，包括以下几个组件：

Eureka：服务治理组件，包含服务注册与发现

Hystrix：容错管理组件，实现了熔断器

Ribbon：客户端负载均衡的服务调用组件

Feign：基于Ribbon和Hystrix的声明式服务调用组件

Zuul：网关组件，提供智能路由、访问过滤等功能

Archaius：外部化配置组件

Spring Cloud Config：配置管理工具，实现应用配置的外部化存储，支持客户端配置信息刷新、加密/解密配置内容等。

Spring Cloud Bus：事件、消息总线，用于传播集群中的状态变化或事件，以及触发后续的处理

Spring Cloud Security：基于spring security的安全工具包，为我们的应用程序添加安全控制

Spring Cloud Consul : 封装了Consul操作，Consul是一个服务发现与配置工具（与Eureka作用类似），与Docker容器可以无缝集成

25、微服务网关的好处

（1）统一入口

为全部微服务提供唯一入口点，网关起到内部和外部隔离，保障了后台服务的安全性。

（2）鉴权校验

识别每个请求的 权限，拒绝不符合要求的请求。

（3）动态路由

动态的将请求 路由 到不同的后端集群中。

（4）降低耦合度

减少客户端与服务的 耦合 ，服务可以独立发展。通过网关层来做映射。

26、MySQL引擎

27、MySQL索引优化

最左匹配原则、不在列上做函数运算、不在列上加表达式运算

具体可以看这篇：

https://monkeysayhi.github.io/2018/03/06/%E6%B5%85%E8%B0%88MySQL%E7%9A%84B%E6%A0%91%E7%B4%A2%E5%BC%95%E4%B8%8E%E7%B4%A2%E5%BC%95%E4%BC%98%E5%8C%96/

28、MySQL回表

具体可以看这篇：

https://www.jianshu.com/p/8991cbca3854

29、覆盖索引

具体可以看这篇：

https://www.cnblogs.com/happyflyingpig/p/7662881.html

30、索引下推

具体可以看这篇：

https://blog.csdn.net/mccand1234/article/details/95799942

31、Redis事务

Redis事务的概念：

　　Redis 事务的本质是一组命令的集合。事务支持一次执行多个命令，一个事务中所有命令都会被序列化。在事务执行过程，会按照顺序串行化执行队列中的命令，其他客户端提交的命令请求不会插入到事务执行命令序列中。

　　总结说：redis事务就是一次性、顺序性、排他性的执行一个队列中的一系列命令。

Redis事务没有隔离级别的概念：

　　批量操作在发送 EXEC 命令前被放入队列缓存，并不会被实际执行，也就不存在事务内的查询要看到事务里的更新，事务外查询不能看到。

Redis不保证原子性：

　　Redis中，单条命令是原子性执行的，但事务不保证原子性，且没有回滚。事务中任意命令执行失败，其余的命令仍会被执行。

可以看这篇：

https://www.cnblogs.com/DeepInThought/p/10720132.html

32、缓存击穿/穿透

可以看这篇：

https://baijiahao.baidu.com/s?id=1655304940308056733

33、布隆过滤器原理

原理可以看这篇：

https://www.cnblogs.com/heihaozi/p/12174478.html

34、Redis持久化

RDB、AOF

具体可以看这篇：

https://baijiahao.baidu.com/s?id=1654694618189745916

25、Redis分布式锁

可以看这篇：

https://www.cnblogs.com/wukongbubai/p/12393370.html

26、TCP中的time\_wait状态

具体可以看这篇：

https://www.cnblogs.com/cenglinjinran/p/8482412.html