1、JVM内存区域

本地方法栈、Java虚拟机栈、程序计数器、堆、元空间、直接内存

2、程序计数器的作用  
程序计数器是一块较小的内存空间，它的作用可以看作是当前线程所执行的字节码的行号指示器。这里面存的，就是当前线程执行的进度。

3、字符串常量位于哪个区域

字符串常量池，jdk8之前是perm区，jdk8及以后是在堆上面。

类文件常量池，是位于方法区，也就是元空间。

4、类加载的过程

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、双亲委派是可以被违背的吗？

可以。需要重写ClassLoader#loadClass和ClassLoader#findClass实现。

7、Tomcat类加载

可以看这篇：

https://www.cnblogs.com/aspirant/p/8991830.html

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

8、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）

9、为什么年轻代用复制算法

因为年轻代存活的对象比较少，用复制算法只需要复制少量对象。

10、年轻代区域

默认比例 eden:from survivor:to survivor=8:1:1

11、新创建的对象都是首先分配在年轻代吗？

不是的。

新生代内存不够时，老年代分配担保。

大对象直接在老年代分配。

12、垃圾收集器

年轻代

Serial 垃圾收集器（单线程，通常用在客户端应用上。因为客户端应用不会频繁创建很多对象，用户也不会感觉出明显的卡顿。相反，它使用的资源更少，也更轻量级。）

ParNew 垃圾收集器（多线程，追求降低用户停顿时间，适合交互式应用。）

Parallel Scavenge 垃圾收集器（追求 CPU 吞吐量，能够在较短时间内完成指定任务，适合没有交互的后台计算。）

老年代

Serial Old 垃圾收集器

Parallel Old垃圾收集器

CMS 垃圾收集器（以获取最短 GC 停顿时间为目标的收集器，它在垃圾收集时使得用户线程和 GC 线程能够并发执行，因此在垃圾收集过程中用户也不会感到明显的卡顿。）

13、客户端应用一般用哪个收集器

Serial 垃圾收集器（单线程，通常用在客户端应用上。因为客户端应用不会频繁创建很多对象，用户也不会感觉出明显的卡顿。相反，它使用的资源更少，也更轻量级。）

14、微服务有什么好处

1. 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；
2. 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；
3. 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；
4. 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；
5. 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；
6. 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；
7. 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。

15、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应用，有些操作会比较耗时

可能会引入分布式事务

16、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

17、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。

18、Eureka原理

1、Eureka Server 启动成功，等待服务端注册。在启动过程中如果配置了集群，集群之间定时通过 Replicate 同步注册表，每个 Eureka Server 都存在独立完整的服务注册表信息

2、Eureka Client 启动时根据配置的 Eureka Server 地址去注册中心注册服务

3、Eureka Client 会每 30s 向 Eureka Server 发送一次心跳请求，证明客户端服务正常

4、当 Eureka Server 90s 内没有收到 Eureka Client 的心跳，注册中心则认为该节点失效，会注销该实例

5、单位时间内 Eureka Server 统计到有大量的 Eureka Client 没有上送心跳，则认为可能为网络异常，进入自我保护机制，不再剔除没有上送心跳的客户端

6、当 Eureka Client 心跳请求恢复正常之后，Eureka Server 自动退出自我保护模式

7、Eureka Client 定时全量或者增量从注册中心获取服务注册表，并且将获取到的信息缓存到本地

8、服务调用时，Eureka Client 会先从本地缓存找寻调取的服务。如果获取不到，先从注册中心刷新注册表，再同步到本地缓存

9、Eureka Client 获取到目标服务器信息，发起服务调用

10、Eureka Client 程序关闭时向 Eureka Server 发送取消请求，Eureka Server 将实例从注册表中删除

具体可以看这篇：

https://blog.csdn.net/qwe86314/article/details/94552801

19、zokeeper服务发现

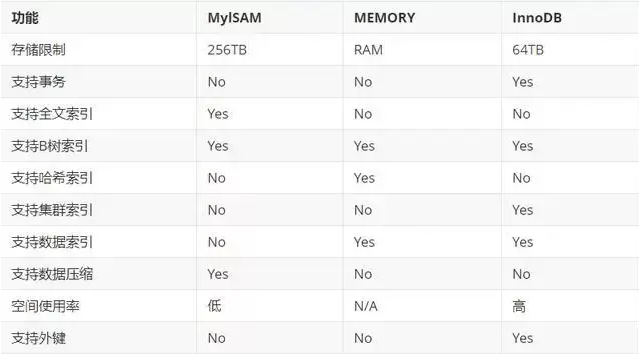
可以看这篇：

https://segmentfault.com/a/1190000019670015

20、zookeeper节点类型

永久节点、临时节点、有序永久节点、有序临时节点

21、MySQL引擎及区别



22、索引索引为什么要用B+树不用二叉树

MySQL索引是按页存储的，相同数据量二叉树的高度比B+树高，高度越高，磁盘IO次数越多，效率也就越低。所以用B+树是为了减少磁盘IO次数。

23、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/

24、MySQL回表

具体可以看这篇：

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

25、索引下推

具体可以看这篇：

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

26、like查询什么情况可以用到索引，什么情况用不到

like ‘aaa%’  可以用到索引

like '%aaa' 不能用到索引

27、缓存穿透/击穿

可以看这篇：  
https://baijiahao.baidu.com/s?id=1655304940308056733

28、布隆过滤器

原理可以看这篇：

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

29、Redis分布式锁

可以看这篇：

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

30、Redis是单线程还是多线程

Redis在执行命令的过程中是单线程的

Redis 6.0引进了多线程IO，不过执行命令依旧还是单线程的。

31、Redis高性能原因

可以看这篇：  
https://www.cnblogs.com/xlecho/archive/2019/11/10/11832118.html

32、Redis协议是文本还是二进制？

文本协议

33、IO模型

可以看这篇：

https://blog.csdn.net/ZWE7616175/article/details/80591587

34、TCP协议

具体可以看这篇：

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