《基 Dubbox 的微服实战 - 就业技能》

章节	Ch01 - 分布式、微服、集群概念介绍及 docker 环境搭建
题目 1)	什么是集群
	集群是一组相互独立的、通过高速网络互联的计算机，它们构成了一个组，并以单一系统的模式加以管理。一个客户与集群相互作用时，集群像是一个独立的服务器。集群配置是用于提高可用性和可缩放性。 和传统的高性能计算机技术相比，集群技术可以利用各档次的服务器作为节点，系统造价低，可以实现很高的运算速度，完成大运算量的计算，具有较高的响应能力，能够满足当今日益增长的信息服务的需求。 而集群技术是一种通用的技术，其目的是为了解决单机运算能力的不足、IO 能力的不足、提高服务的可靠性、获得规模可扩展能力，降低整体方案的运维成本（运行、升级、维护成本）。只要在其他技术不能达到以上的目的，或者虽然能够达到以上的目的，但是成本过高的情况下，就可以考虑采用集群技术
题目 2)	什么是分布式应用开发
	分布式应用开发简单的说，是指将用户界面、控制台服务、数据库管理三个层次部署在不同的位置上。其中用户界面是客户端实现的功能，控制台服务是一个专门的服务器，数据管理是在一个专门的数据库服务器上实现的
题目 3)	分布式系统技术
	集群管理技术 Zookeeper 远程调用 Dubbo 序列化技术 Thrift（它是一个高性能、跨语言的 RPC 服务框架，适合用来实现内部服务的 RPC 调用） 消息中间件 Kafka RabbitMQ ActiveMQ 分布式文件系统 Ceph（提供了块存储和对象存储的功能） GridFS（GridFS 是 MongoDB 的一部分。用来存储超过 BSON 大小限制（16MB）的文件。GridFS 将文件分成一个个部分，分开存储） 分布式内存 Memcached Redis 分布式数据库 NoSQL 虚拟化技术 Docker（Docker 能够提供给使用者一种轻量化的虚拟机的使用体验）
题目 4)	Docker 架构
	Docker 使用客户端-服务器（C/S）架构模式，使用远程 API 来管理和创建 Docker 容器。 Docker 容器通过 Docker 镜像来创建。容器与镜像的关系类似于面向对象编程中的对象与类 Docker 采用 C/S 架构 Docker daemon 作为服务端接受来自客户的请求，并处理这些请求（创建、运行、分发容器）。 客户端和服务端既可以运行在一个机器上，也可通过 socket 或者 RESTful API 来进行通信。 Docker 并非适合所有应用场景，Docker 只能虚拟基于 Linux 的服务。Windows Azure 服务能够运行 Docker 实例，但到目前为止 Windows 服务还不能被虚拟化
题目 5)	什么是微服
	微服务架构强调的第一个重点就是业务系统需要彻底的组件化和服务化，原有的单个业务系统会拆分为多个可以独立开发、设计、运行和运维的小应用。这些小应用之间通过服务完成交互和集成。每个小应用从前端 web ui，到控制层，逻辑层，数据库访问，数 据库都完全是独立的一套。每个应用除了完成自身业务功能外还需要消费外部其他应用暴露的服务，同时自身

	也将自身的能力对外发布为服务 1.首先对于应用本身暴露出来的服务，是和应用一起部署的，即服务本身并不单独部署，服务本身就是业务组件已有的接口能力发布和暴露出来的 2.其次，微服务架构本身来源于互联网的思路，因此组件对外发布的服务强调了采用 HTTP Rest API 的方式来进行 3.微服务的基本思想在于考虑围绕着业务领域组件来创建应用，这些就应用可独立地进行开发、管理和加速。在分散的组件中使用微服务云架构和平台使部署、管理和服务功能交付变得更加简单。

章节	**Ch02 - Springboot（一）**
题目 1)	**SpringMVC 和 Springboot 的区别**
	Spring MVC 的功能 1.Spring MVC 提供了一种轻度耦合的方式来开发 web 应用。 2.Spring MVC 是 Spring 的一个模块，式一个 web 框架。通过 Dispatcher Servlet, ModelAndView 和 View Resolver，开发 web 应用变得很容易。解决的问题领域是网站应用程序或者服务开发——URL 路由、Session、模板引擎、静态 Web 资源等等。 Spring Boot 的功能 1.Spring Boot 实现了自动配置，降低了项目搭建的复杂度 2.Spring Boot 只是承载者，辅助你简化项目搭建过程的。如果承载的是 WEB 项目，使用 Spring MVC 作为 MVC 框架，那么工作流程和你上面描述的是完全一样的，因为这部分工作是 Spring MVC 做的而不是 Spring Boot
题目 2)	**Springboot 常用的 starter 有哪些**
	spring-boot-starter-web 嵌入 tomcat 和 web 开发需要 servlet 与 jsp 支持 spring-boot-starter-data-jpa 数据库支持 spring-boot-starter-data-redis redis 数据库支持 spring-boot-starter-data-solr solr 支持 mybatis-spring-boot-starter 第三方的 mybatis 集成 starter
题目 3)	**Springboot 自动配置的原理**
	在spring程序main方法中 添加@SpringBootApplication或@EnableAutoConfiguration会自动去 maven 中读取每个 starter 中的 spring.factories 文件 该文件里配置了所有需要被创建 spring 容器中的 bean
题目 4)	**Springboot 读取配置文件的方式**
	springboot 默认读取配置文件为 application.properties 或者是 application.yml
题目 5)	**Springboot 集成 mybatis 的过程**
	添加 mybatis 的 starter maven 依赖 <dependency> <groupId>org.mybatis.spring.boot</groupId> <artifactId>mybatis-spring-boot-starter</artifactId> <version>1.2.0</version> </dependency> 在 mybatis 的接口中 添加@Mapper 注解 在 application.yml 配置数据源信息

章节	Ch03 - Springboot（二）
题目 1)	**SpringBoot 的事务管理**
	Springboot 内部提供的事务管理器是根据 autoconfigure 来进行决定的。比如当使用 jpa 的时候，也就是 pom 中加入了 spring-boot-starter-data-jpa 这个 starter 之后 Springboot 会构造一个 JpaTransactionManager 这个事务管理器。而当我们使 spring-boot-starter-jdbc 的时候，构造的事务管理器则是 DataSourceTransactionManager。 这 2 个事务管理器都实现了 spring 中提供的 PlatformTransactionManager 接口，这个接口是 spring 的事务核心接口。 这个核心接口有以下这几个常用的实现策略： HibernateTransactionManager DataSourceTransactionManager JtaTransactionManager JpaTransactionManager
题目 2)	**SpringBoot 核心功能**
	1.独立运行 Spring 项目 Spring boot 可以以 jar 包形式独立运行，运行一个 Spring Boot 项目只需要通过 java -jar xx.jar 来运行。 2.内嵌 servlet 容器 Spring Boot 可以选择内嵌 Tomcat、jetty 或者 Undertow,这样我们无须以 war 包形式部署项目。 3.提供 starter 简化 Maven 配置 spring 提供了一系列的 start pom 来简化 Maven 的依赖加载，例如，当你使用了 spring-boot-starter-web，会自动加入如图 5-1 所示的依赖包。 4.自动装配 Spring SpringBoot 会根据在类路径中的 jar 包，类、为 jar 包里面的类自动配置 Bean，这样会极大地减少我们要使用的配置。当然，SpringBoot 只考虑大多数的开发场景，并不是所有的场景，若在实际开发中我们需要配置 Bean，而 SpringBoot 灭有提供 支持，则可以自定义自动配置。 5.准生产的应用监控 SpringBoot 提供基于 http ssh telnet 对运行时的项目进行监控。 6.无代码生产和 xml 配置 SpringBoot 不是借助与代码生成来实现的，而是通过条件注解来实现的，这是 Spring4.x 提供的新特性
题目 3)	**SpringBoot 优缺点**
	优点： 1.快速构建项目。 2.对主流开发框架的无配置集成。 3.项目可独立运行，无须外部依赖 Servlet 容器。 4.提供运行时的应用监控。 5.极大的提高了开发、部署效率。 6.与云计算的天然集成。 缺点：

	1.如果你不认同 spring 框架，也许这就是缺点。
题目 4)	**SpringBoot 几个常用的注解**
	（1）@RestController 和@Controller 指定一个类，作为控制器的注解 （2）@RequestMapping 方法级别的映射注解，这一个用过 Spring MVC 的小伙伴相信都很熟悉 (3)@EnableAutoConfiguration 和@SpringBootApplication 是类级别的注解，根据 maven 依赖的 jar 来自动猜测完成正确的 spring 的对应配置，只要引入了 spring-boot-starter-web 的依赖，默认会自动配置 Spring MVC 和 tomcat 容器 （4）@Configuration 类级别的注解，一般这个注解，我们用来标识 main 方法所在的类,完成元数据 bean 的初始化。 （5）@ComponentScan 类级别的注解，自动扫描加载所有的 Spring 组件包括 Bean 注入，一般用在 main 方法所在的类上 （6）@ImportResource 类级别注解，当我们必须使用一个 xml 的配置时，使用 @ImportResource 和@Configuration 来标识这个文件资源的类。 （7）@Autowired 注解，一般结合@ComponentScan 注解，来自动注入一个 Service 或 Dao 级别的 Bean （8）@Component 类级别注解，用来标识一个组件，比如我自定了一个 filter，则需要此注解标识之后，Spring Boot 才会正确识别。
题目 5)	**SpringBoot maven 构建项目**
	spring-boot-starter-parent：是一个特殊 Start，它用来提供相关的 Maven 依赖项，使用它之后，常用的包依赖可以省去 version 标签。

章节	**Ch04 - 基于 Dubbox 的微服架构搭建（一）**
题目 1)	**dubbox 与 spring cloud 区别**
	dubbox 一个 Java 高性能优秀的服务框架，使得应用可通过高性能的 RPC 实现服务的输出和输入功能，可以和 Spring 框架无缝集成 支持 REST 风格远程调用（HTTP + JSON/XML)； 支持基于 Kryo 和 FST 的 Java 高效序列化实现； 支持基于 Jackson 的 JSON 序列化； 支持基于嵌入式 Tomcat 的 HTTP remoting 体系； 升级 Spring 至 3.x； 升级 ZooKeeper 客户端； 支持完全基于 Java 代码的 Dubbo 配置 spring cloud 完全基于 Spring Boot ，是一个非常新的项目 但是相比 Dubbo 等 RPC 框架, Spring Cloud 提供的全套的分布式系统解决方案。Spring Cloud 为开发者提供了在分布式系统（配置管理，服务发现，熔断，路由，微代理，控制总线，一次性 token，全局琐，leader 选举，分布式 session，集群状态） 中快速构建的工具，使用 Spring Cloud 的开发者可以快速的启动服务或构建应用．它们将在任何分布式环境中工作
题目 2)	**Dubbo 中 zookeeper 做注册中心，如果注册中心集群都挂掉，发布者和订阅者之间还能通信么**

	可以的，启动 dubbo 时，消费者会从 zk 拉取注册的生产者的地址接口等数据，缓存在本地。每次调用时，按照本地存储的地址进行调用 注册中心对等集群，任意一台宕掉后，会自动切换到另一台 注册中心全部宕掉，服务提供者和消费者仍可以通过本地缓存通讯 服务提供者无状态，任一台 宕机后，不影响使用 服务提供者全部宕机，服务消费者会无法使用，并无限次重连等待服务者恢复
题目 3)	**dubbo 连接注册中心和直连的区别**
	在开发及测试环境下，经常需要绕过注册中心，只测试指定服务提供者，这时候可能需要点对点直连，点对点直联方式，将以服务接口为单位，忽略注册中心的提供者列表， 服务注册中心，动态的注册和发现服务，使服务的位置透明，并通过在消费方获取服务提供方地址列表，实现软负载均衡和 Failover， 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时与注册中心交互，注册中心不转发请求，服务消费者 向注册中心获取服务提供者地址列表，并根据负载算法直接调用提供者，注册中心，服务提供者，服务消费者三者之间均为长连接，监控中心除外，注册中心通过长连接感知服务提供者的存在，服务提供者宕机，注册中心将立即推送事件通知消费者 注册中心和监控中心全部宕机，不影响已运行的提供者和消费者，消费者在本地缓存了提供者列表 注册中心和监控中心都是可选的，服务消费者可以直连服务提供者
题目 4)	**dubbo 在安全机制方面是如何解决的**
	Dubbo 通过 Token 令牌防止用户绕过注册中心直连，然后在注册中心上管理授权。Dubbo 还提供服务黑白名单，来控制服务所允许的调用方
题目 5)	**dubbo 服务负载均衡策略?**
	Random LoadBalance 随机，按权重设置随机概率。在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重。(权重可以在 dubbo 管控台配置) RoundRobin LoadBalance 轮循，按公约后的权重设置轮循比率。存在慢的提供者累积请求问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡在那，久而久之，所有请求都卡在调到第二台上。 LeastActive LoadBalance 最少活跃调用数，相同活跃数的随机，活跃数指调用前后计数差。使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。 ConsistentHash LoadBalance 一致性 Hash，相同参数的请求总是发到同一提供者。当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。缺省只对第一个参数 Hash，如果要修改，请配置

章节	**Ch05 - 基于 Dubbox 的微服架构搭建（二）**
题目 1)	**dubbo 通信协议 dubbo 协议为什么要消费者比提供者个数多**
	因 dubbo 协议采用单一长连接，假设网络为千兆网卡(1024Mbit=128MByte)，根据测试经验数据每条连接最多只能压满 7MByte(不同的环境可能不一样，供参考)，理论上 1 个服务

	提供者需要 20 个服务消费者才能压满网卡
题目 2)	**dubbo 通信协议 dubbo 协议为什么不能传大包**
	因 dubbo 协议采用单一长连接， 如果每次请求的数据包大小为 500KByte，假设网络为千兆网卡(1024Mbit=128MByte)，每条连接最大 7MByte(不同的环境可能不一样，供参考)，单个服务提供者的 TPS(每秒处理事务数)最大为：128MByte / 500KByte = 262。单个消费者调用单个服务提供者的 TPS(每秒处理事务数)最大为：7MByte / 500KByte = 14。如果能接受，可以考虑使用，否则网络将成为瓶颈
题目 3)	**dubbo 通信协议 dubbo 协议为什么采用异步单一长连接**
	因为服务的现状大都是服务提供者少，通常只有几台机器，而服务的消费者多，可能整个网站都在访问该服务，比如 Morgan 的提供者只有 6 台提供者，却有上百台消费者，每天有 1.5 亿次调用，如果采用常规的 hessian 服务，服务提供者很容易就被压跨， 通过单一连接，保证单一消费者不会压死提供者，长连接，减少连接握手验证等，并使用异步 IO，复用线程池，防止 C10K 问题
题目 4)	**dubbo 通信协议 dubbo 协议适用范围和适用场景**
	适用范围：传入传出参数数据包较小（建议小于 100K），消费者比提供者个数多，单一消费者无法压满提供者，尽量不要用 dubbo 协议传输大文件或超大字符串。 适用场景：常规远程服务方法调用
题目 5)	**RMI 协议**
	RMI 协议采用 JDK 标准的 java.rmi.*实现，采用阻塞式短连接和 JDK 标准序列化方式，Java 标准的远程调用协议。 连接个数：多连接 连接方式：短连接 传输协议：TCP 传输方式：同步传输 序列化：Java 标准二进制序列化 适用范围：传入传出参数数据包大小混合，消费者与提供者个数差不多，可传文件。 适用场景：常规远程服务方法调用，与原生 RMI 服务互操作

章节	**Ch06 - 基于 Dubbox 的微服架构搭建（三）**
题目 1)	**Zookeeper 对节点的 watch 监听通知是永久的吗?**
	不是。官方声明：一个 Watch 事件是一个一次性的触发器，当被设置了 Watch 的数据发生了改变的时候，则服务器将这个改变发送给设置了 Watch 的客户端，以便通知它们
题目 2)	**集群支持动态添加机器吗?**
	其实就是水平扩容了，Zookeeper 在这方面不太好。两种方式： 全部重启：关闭所有 Zookeeper 服务，修改配置之后启动。不影响之前客户端的会话。 逐个重启：顾名思义。这是比较常用的方式
题目 3)	**Zookeeper 原理**
	1.配置管理 Zookeeper 提供了这样的一种服务：一种集中管理配置的方法，我们在这个集中的地方修改了配置，所有对这个配置感兴趣的都可以获得变更。这样就省去手动拷贝配置了，还保证了可靠和一致性 2.名字服务

	分布式环境下，经常需要对应用/服务进行统一命名，便于识别不同服务； 3.分布式锁 单机程序的各个进程需要对互斥资源进行访问时需要加锁，那分布式程序分布在各个主机上的进程对互斥资源进行访问时也需要加锁。很多分布式系统有多个可服务的窗口，但是在某个时刻只让一个服务去干活，当这台服务出问题的时候锁释放，立即 fail over 到另外的服务。 4.集群管理 集群中有些机器（比如 Master 节点）需要感知到这种变化，然后根据这种变化做出对应的决策
题目 4)	**Zookeeper 是如何保证事务的顺序一致性的**
	zookeeper 采用了递增的事务 Id 来标识，所有的 proposal 都在被提出的时候加上了 zxid，zxid 实际上是一个 64 位的数字，高 32 位是 epoch 用来标识 leader 是否发生改变，如果有新的 leader 产生出来，epoch 会自增，低 32 位用来递增计数。 当新产生 proposal 的时候，会依据数据库的两阶段过程，首先会向其他的 server 发出事务执行请求，如果超过半数的机器都能执行并且能够成功，那么就会开始执行
题目 5)	**Zookeeper 有哪几种节点类型**
	持久：创建之后一直存在，除非有删除操作，创建节点的客户端会话失效也不影响此节点。 持久顺序：跟持久一样，就是父节点在创建下一级子节点的时候，记录每个子节点创建的先后顺序，会给每个子节点名加上一个数字后缀。 临时：创建客户端会话失效（注意是会话失效，不是连接断了），节点也就没了。不能建子节点。 临时顺序：不用解释了吧。

章节	**Ch07 - MQ+Redis（一）**
题目 1)	**ActiveMQ 服务器宕机怎么办?**
	这得从 ActiveMQ 的储存机制说起。在通常的情况下，非持久化消息是存储在内存中的，持久化消息是存储在文件中的，它们的最大限制在配置文件的<systemUsage>节点中配置。但是，在非持久化消息堆积到一定程度，内存告急的时候， ActiveMQ 会将内存中的非持久化消息写入临时文件中，以腾出内存。虽然都保存到了文件里，但它和持久化消息的区别是，重启后持久化消息会从文件中恢复，非持久化的临时文件会直接删除 解决方案：尽量不要用非持久化消息，非要用的话，将临时文件限制尽可能的调大。
题目 2)	**ActiveMQ 丢消息怎么办?**
	解决方案：用持久化消息，或者非持久化消息及时处理不要堆积，或者启动事务，启动事务后，commit()方法会负责任的等待服务器的返回，也就不会关闭连接导致消息丢失了。
题目 3)	**持久化消息非常慢怎么办?**
	默认的情况下，非持久化的消息是异步发送的，持久化的消息是同步发送的，遇到慢一点的硬盘，发送消息的速度是无法忍受的。但是在开启事务的情况下，消息都是异步发送的，效率会有 2 个数量级的提升。所以在发送持久化消息时，请务必开启事务模式。其 实发送非持久化消息时也建议开启事务，因为根本不会影响性能。
题目 4)	**消息的不均匀消费怎么办?**
	解决方案：将 prefetch 设为 1，每次处理 1 条消息，处理完再去取，这样也慢不了多少。
题目 5)	**死信队列怎么处理?**

	如果你想在消息处理失败后，不被服务器删除，还能被其他消费者处理或重试，可以关闭 AUTO_ACKNOWLEDGE，将 ack 交由程序自己处理。

章节	**Ch08 -MQ+Redis（二）**
题目 1)	**使用 redis 有哪些好处?**
	(1) 速度快，因为数据存在内存中，类似于 HashMap，HashMap 的优势就是查找和操作的时间复杂度都是 O(1) (2) 支持丰富数据类型，支持 string，list，set，sorted set，hash (3) 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行 (4) 丰富的特性：可用于缓存，消息，按 key 设置过期时间，过期后将会自动删除
题目 2)	**redis 相比 memcached 有哪些优势?**
	(1) memcached 所有的值均是简单的字符串，redis 作为其替代者，支持更为丰富的数据类型 (2) redis 的速度比 memcached 快很多 (3) redis 可以持久化其数据
题目 3)	**redis 常见性能问题和解决方案:**
	(1) Master 最好不要做任何持久化工作，如 RDB 内存快照和 AOF 日志文件 (2) 如果数据比较重要，某个 Slave 开启 AOF 备份数据，策略设置为每秒同步一次 (3) 为了主从复制的速度和连接的稳定性，Master 和 Slave 最好在同一个局域网内 (4) 尽量避免在压力很大的主库上增加从库 (5) 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3... 这样的结构方便解决单点故障问题，实现 Slave 对 Master 的替换。如果 Master 挂了，可以立刻启用 Slave1 做 Master，其他不变
题目 4)	**什么是 redis?**
	Redis 是一个基于内存的高性能 key-value 数据库。
题目 5)	**Reids 的特点**
	1.Redis 本质上是一个 Key-Value 类型的内存数据库，很像 memcached，整个数据库统统加载在内存当中进行操作，定期通过异步操作把数据库数据 flush 到硬盘上进行保存。因为是纯内存操作，Redis 的性能非常出色，每秒可以处理超过 10 万次读写操作，是已知性能最快的 Key-Value DB 2.支持保存多种数据结构，此外单个 value 的最大限制是 1GB 3.息队列服务，用他的 Set 可以做高性能的 tag 系统等等 4.Redis 的主要缺点是数据库容量受到物理内存的限制，不能用作海量数据的高性能读写

章节	**Ch09 - MQ+Redis（三）**
题目 1)	**MySQL 里有 2000w 数据，redis 中只存 20w 的数据，如何保证 redis 中的数据都是热点数据**
	redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略（回收策略）。redis 提供 6 种数据淘汰策略： volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少

	使用的数据淘汰 volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰 volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰 allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰 allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰 no-enviction（驱逐）：禁止驱逐数据
题目 2)	**为什么 redis 需要把所有数据放到内存中?**
	Redis 为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以 redis 具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘 I/O 速度为严重影响 redis 的性能
题目 3)	**Redis 是单进程单线程的吗**
	redis 利用队列技术将并发访问变为串行访问，消除了传统数据库串行控制的开销
题目 4)	**Redis 的并发竞争问题如何解决?**
	Redis 为单进程单线程模式，采用队列模式将并发访问变为串行访问。Redis 本身没有锁的概念，Redis 对于多个客户端连接并不存在竞争，但是在 Jedis 客户端对 Redis 进行并发访问时会发生连接超时、数据转换错误、阻塞、客户端关闭连接等问题，这些问题均是由于客户端连接混乱造成。对此有 2 种解决方法 1.客户端角度，为保证每个客户端间正常有序与 Redis 进行通信，对连接进行池化，同时对客户端读写 Redis 操作采用内部锁 synchronized。 2.服务器角度，利用 setnx 实现锁。
题目 5)	**Redis 持久化的几种方式**
	1、快照（snapshots） 缺省情况情况下，Redis 把数据快照存放在磁盘上的二进制文件中，文件名为 dump.rdb。你可以配置 Redis 的持久化策略，例如数据集中每 N 秒钟有超过 M 次更新，就将数据写入磁盘；或者你可以手工调用命令 SAVE 或 BGSAVE。 工作原理 Redis forks. 子进程开始将数据写到临时 RDB 文件中。 当子进程完成写 RDB 文件，用新文件替换老文件。 这种方式可以使 Redis 使用 copy-on-write 技术。 2、AOF 快照模式并不十分健壮，当系统停止，或者无意中 Redis 被 kill 掉，最后写入 Redis 的数据就会丢失。这对某些应用也许不是大问题，但对于要求高可靠性的应用来说，Redis 就不是一个合适的选择。Append-only 文件模式是另一种选择。 你可以在配置文件中打开 AOF 模式 3、虚拟内存方式 当你的 key 很小而 value 很大时,使用 VM 的效果会比较好.因为这样节约的内存比较大.当你的 key 不小时,可以考虑使用一些非常方法将很大的 key 变成很大的 value,比如你可以考虑将 key,value 组合成一个新的 value.vm-max-threads 这个参数,可以设置访问 swap 文件的线程数,设置最好不要超过机器的核数,如果设置为 0,那么所有对 swap 文件的操作都是串行的.可能会造成比较长时间的延迟,但是对数据完整性有很好的保证.

章节	Ch10 - 分布式锁及分布式事务处理、高并发测试（一）
题目 1)	**分布式锁有几种实现方式**
	1.基于数据库实现分布式锁 2.基于缓存实现分布式锁 3.基于 Zookeeper 实现分布式锁
题目 2)	**基于数据库实现分布式锁**
	基于数据库表 要实现分布式锁，最简单的方式可能就是直接创建一张锁表，然后通过操作该表中的数据来实现了。当我们要锁住某个方法或资源时，我们就在该表中增加一条记录，想要释放锁的时候就删除这条记录。
题目 3)	**基于缓存实现分布式锁**
	相比较于基于数据库实现分布式锁的方案来说，基于缓存来实现在性能方面会表现的更好一点。而且很多缓存是可以集群部署的，可以解决单点问题。 目前有很多成熟的缓存产品，包括 Redis，memcached 以及我们公司内部的 Tair。 这里以 Tair 为例来分析下使用缓存实现分布式锁的方案。关于 Redis 和 memcached 在网络上有很多相关的文章，并且也有一些成熟的框架及算法可以直接使用。 基于 Tair 的实现分布式锁其实和 Redis 类似，其中主要的实现方式是使用 TairManager.put 方法来实现
题目 4)	**基于 Zookeeper 实现分布式锁**
	每个客户端对某个方法加锁时，在 zookeeper 上的与该方法对应的指定节点的目录下，生成一个唯一的瞬时有序节点。 判断是否获取锁的方式很简单，只需要判断有序节点中序号最小的一个。 当释放锁的时候，只需将这个瞬时节点删除即可。同时，其可以避免 服务宕机导致的锁无法释放，而产生的死锁问题
题目 5)	**三种分布锁的比较**
	上面几种方式，哪种方式都无法做到完美。就像 CAP 一样，在复杂性、可靠性、性能等方面无法同时满足，所以，根据不同的应用场景选择最适合自己的才是王道。 从理解的难易程度角度（从低到高） 数据库 > 缓存 > Zookeeper 从实现的复杂性角度（从低到高） Zookeeper >= 缓存 > 数据库 从性能角度（从高到低） 缓存 > Zookeeper >= 数据库 从可靠性角度（从高到低） Zookeeper > 缓存 > 数据库

章节	Ch11 - 分布式锁及分布式事务处理、高并发测试（二）
题目 1)	**什么是分布式事务**
	分布式事务就是指事务的参与者、支持事务的服务器、资源服务器以及事务管理器分别位于不同的分布式系统的不同节点之上
题目 2)	**分布式事务的处理方案**
	1.两阶段提交协议

	两阶段提交协议可以很好得解决分布式事务问题，它可以使用 XA 来实现，XA 它包含两个部分：事务管理器和本地资 源管理器。其中本地资源管理器往往由数据库实现，比如 Oracle、DB2 这些商业数据库都实现了 XA 接口，而事务管 理器作为全局的协调者，负责各个本地资源的提交和回滚 2.消息中间件 消息中间件也可称作消息系统（MQ），它本质上是一个暂存转发消息的一个中间件。在分布式应用当中，我们可以把一 个业务操作转换成一个消息，比如支付宝的余额转如余额宝操作，支付宝系统执行减少余额操作之后向消息系统发一个 消息，余额宝系统订阅这条消息然后进行增加账户金额操作
题目 3)	**说说项目中使用分布式的场景**
	1.解决 java 集群的 session 共享的解决方案： 1.客户端 cookie 加密。(一般用于内网中企业级的系统中，要求用户浏览器端 cookie 不能禁用，禁用的话，该方案会失效)。 2.集群中，各个应用服务器提供了 session 复制的功能，tomcat 和 jboss 都实现了这样 的功能。特点：性能随着服务器增加急剧下降，容易引起广播风暴；session 数据需要序列化，影响性能。 3.session 的持久化，使用数据库来保存 session。就算服务器宕机也没事儿，数据库中的 session 照样存在。特点：每次请求 session 都要读写数据库，会带来性能开销。使用内存数据库，会提高性能，但是宕机会丢失数据(像支付宝的宕机，有同城灾备、异地灾备)。 4.使用共享存储来保存 session。和数据库类似，就算宕机了也没有事儿。其实就是专门搞一台服务器，全部对 session 落地。特点：频繁的进行序列化和反序列化会影响性能。 5.使用 memcached 来保存 session。本质上是内存数据库的解决方案。特点：存入 memcached 的数据需要序列化，效率极低。 2.分布式事务的解决方案: 1.所谓的 TCC 编程模式，也是两阶段提交的一个变种。TCC 提供了一个编程框架，将整个业务逻辑分为三块：Try、Confirm 和 Cancel 三个操作
题目 4)	**ThreadLocal,以及死锁分析**
	ThreadLocal 为每个线程维护一个本地变量。 采用空间换时间，它用于线程间的数据隔离，为每一个使用该变量的线程提供一个副本，每个线程都可以独立地改变自己的副本，而不会和其他线程的副本冲突。 ThreadLocal 类中维护一个 Map，用于存储每一个线程的变量副本，Map 中元素的键为线程对象，而值为对应线程的变量副本。
题目 5)	**CachedThreadPool 、 FixedThreadPool、SingleThreadPool 、 newScheduledThreadPool**
	newSingleThreadExecutor :创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务， 保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行 适用场景：任务少 ，并且不需要并发执行 **newCachedThreadPool** :创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程. 线程没有任务要执行时，便处于空闲状态，处于空闲状态的线程并不会被立即销毁（会被缓存住），只有当空闲时间超出一段时间(默认为 60s)后，线程池才会销毁该线程（相当于清除过时的缓存）。新任务到达后，线程池首先会让被缓存住的线程（空闲状态） 去执行任务，

	如果没有可用线程（无空闲线程），便会创建新的线程。 适用场景：处理任务速度 > 提交任务速度,耗时少的任务(避免无限新增线程) **newFixedThreadPool** :创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。 **newScheduledThreadPool:**创建一个定长线程池，支持定时及周期性任务执行