|  |
| --- |
| 技术问题(至少50个) |
| 问题：**JAVA 线程实现/创建方式**  回答：**继承 Thread 类**  Thread 类本质上是实现了 Runnable 接口的一个实例，代表一个线程的实例。启动线程的唯一方 法就是通过 Thread 类的 start()实例方法。start()方法是一个 native 方法，它将启动一个新线 程，并执行 run()方法。  如果自己的类已经 extends 另一个类，就无法直接 extends Thread，此时，可以实现一个 Runnable 接口。 有返回值的任务必须实现 Callable 接口，类似的，无返回值的任务必须 Runnable 接口。执行 Callable 任务后，可以获取一个 Future 的对象，在该对象上调用 get 就可以获取到 Callable 任务 返回的 Object 了，再结合线程池接口 ExecutorService 就可以实现传说中有返回结果的多线程了。  问题：**Spring IOC 原理**  回答：**概念**  Spring 通过一个配置文件描述 Bean 及 Bean 之间的依赖关系，利用 Java 语言的反射功能实例化 Bean 并建立 Bean 之间的依赖关系。 Spring 的 IoC 容器在完成这些底层工作的基础上，还提供 了 Bean 实例缓存、生命周期管理、 Bean 实例代理、事件发布、资源装载等高级服务。  问题：**Spring APO 原理**  回答：**概念**  "横切"的技术，剖解开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块， 并将其命名为"Aspect"，即切面。所谓"切面"，简单说就是那些与业务无关，却为业务模块所共 同调用的逻辑或责任封装起来，便于减少系统的重复代码，降低模块之间的耦合度，并有利于未 来的可操作性和可维护性。  使用"横切"技术，AOP 把软件系统分为两个部分：核心关注点和横切关注点。业务处理的主要流 程是核心关注点，与之关系不大的部分是横切关注点。横切关注点的一个特点是，他们经常发生 在核心关注点的多处，而各处基本相似，比如权限认证、日志、事物。AOP 的作用在于分离系统 中的各种关注点，将核心关注点和横切关注点分离开来。  **AOP 核心概念**  1、切面（aspect）：类是对物体特征的抽象，切面就是对横切关注点的抽象  2、横切关注点：对哪些方法进行拦截，拦截后怎么处理，这些关注点称之为横切关注点。  3、连接点（joinpoint）：被拦截到的点，因为 Spring 只支持方法类型的连接点，所以在 Spring 中连接点指的就是被拦截到的方法，实际上连接点还可以是字段或者构造器。  4、切入点（pointcut）：对连接点进行拦截的定义  5、通知（advice）：所谓通知指的就是指拦截到连接点之后要执行的代码，通知分为前置、后置、 异常、最终、环绕通知五类。  6、目标对象：代理的目标对象  7、织入（weave）：将切面应用到目标对象并导致代理对象创建的过程  8、引入（introduction）：在不修改代码的前提下，引入可以在运行期为类动态地添加一些方法或字段。  **AOP 两种代理方式**  Spring 提供了两种方式来生成代理对象: JDKProxy 和 Cglib，具体使用哪种方式生成由  AopProxyFactory 根据 AdvisedSupport 对象的配置来决定。默认的策略是如果目标类是接口， 则使用 JDK 动态代理技术，否则使用 Cglib 来生成代理。  ***JDK 动态接口代理***  1. JDK 动态代理主要涉及到 java.lang.reflect 包中的两个类：Proxy 和 InvocationHandler。 InvocationHandler是一个接口，通过实现该接口定义横切逻辑，并通过反射机制调用目标类 的代码，动态将横切逻辑和业务逻辑编制在一起。Proxy 利用 InvocationHandler 动态创建 一个符合某一接口的实例，生成目标类的代理对象。  ***CGLib 动态代理***   1. ：CGLib 全称为 Code Generation Library，是一个强大的高性能，高质量的代码生成类库， 可以在运行期扩展 Java 类与实现 Java 接口，CGLib 封装了 asm，可以再运行期动态生成新 的 class。和 JDK 动态代理相比较：JDK 创建代理有一个限制，就是只能为接口创建代理实例， 而对于没有通过接口定义业务方法的类，则可以通过 CGLib 创建动态代理。     问题：**Spring MVC 原理**  回答：Spring 的模型-视图-控制器（MVC）框架是围绕一个 DispatcherServlet 来设计的，这个 Servlet 会把请求分发给各个处理器，并支持可配置的处理器映射、视图渲染、本地化、时区与主题渲染等，甚至还能支持文件上传。  ***Http 请求到 DispatcherServlet***  (1) 客户端请求提交到 DispatcherServlet。  ***HandlerMapping 寻找处理器***  (2) 由 DispatcherServlet 控制器查询一个或多个 HandlerMapping，找到处理请求的  Controller。  ***调用处理器 Controller***  (3) DispatcherServlet 将请求提交到 Controller。  ***Controller 调用业务逻辑处理后，返回 ModelAndView***  (4)(5)调用业务处理和返回结果：Controller 调用业务逻辑处理后，返回 ModelAndView。  ***DispatcherServlet 查询 ModelAndView***  (6)(7)处理视图映射并返回模型： DispatcherServlet 查询一个或多个 ViewResoler 视图解析器， 找到 ModelAndView 指定的视图。  ***ModelAndView 反馈浏览器 HTTP***  (8) Http 响应：视图负责将结果显示到客户端。  问题：**Spring Boot 原理**  回答：Spring Boot 是由 Pivotal 团队提供的全新框架，其设计目的是用来简化新 Spring 应用的初始搭 建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的 配置。通过这种方式，Spring Boot 致力于在蓬勃发展的快速应用开发领域(rapid application development)成为领导者。其特点如下：  **1. 创建独立的 Spring 应用程序**  **2. 嵌入的 Tomcat，无需部署 WAR 文件**  **3. 简化 Maven 配置**  **4. 自动配置 Spring**  **5. 提供生产就绪型功能，如指标，健康检查和外部配置**  **6. 绝对没有代码生成和对 XML 没有要求配置**  问题：**事务**  回答：事务是计算机应用中不可或缺的组件模型，它保证了用户操作的原子性 ( Atomicity )、一致性 ( Consistency )、隔离性 ( Isolation ) 和持久性 ( Durabilily )。    **本地事务**  紧密依赖于底层资源管理器（例如数据库连接 )，事务处理局限在当前事务资源内。此种事务处理 方式不存在对应用服务器的依赖，因而部署灵活却无法支持多数据源的分布式事务。  **分布式事务**  Java 事务编程接口（JTA：Java Transaction API）和 Java 事务服务 (JTS；Java Transaction Service) 为 J2EE 平台提供了分布式事务服务。分布式事务（Distributed Transaction）包括事务管理器（Transaction Manager）和一个或多个支持 XA 协议的资源管理器 ( Resource Manager )。我们可以将资源管理器看做任意类型的持久化数据存储；事务管理器承担着所有事务参与单元的协调与控制。  **两阶段提交**  两阶段提交主要保证了分布式事务的原子性：即所有结点要么全做要么全不做，所谓的两个阶段 是指：第一阶段：准备阶段；第二阶段：提交阶段。  ***1 准备阶段***  事务协调者(事务管理器)给每个参与者(资源管理器)发送 Prepare 消息，每个参与者要么直接返回 失败(如权限验证失败)，要么在本地执行事务，写本地的 redo 和 undo 日志，但不提交，到达一 种“万事俱备，只欠东风”的状态。  ***2 提交阶段：***  如果协调者收到了参与者的失败消息或者超时，直接给每个参与者发送回滚(Rollback)消息；否则， 发送提交(Commit)消息；参与者根据协调者的指令执行提交或者回滚操作，释放所有事务处理过 程中使用的锁资源。(注意:必须在最后阶段释放锁资源)将提交分成两阶段进行的目的很明确，就是尽可能晚地提交事务，让事务在提交前尽可能地完成 所有能完成的工作。  问题：**Mybatis 缓存**  回答：Mybatis 中有一级缓存和二级缓存，默认情况下一级缓存是开启的，而且是不能关闭的。一级缓存 是指 SqlSession 级别的缓存，当在同一个 SqlSession 中进行相同的 SQL 语句查询时，第二次以 后的查询不会从数据库查询，而是直接从缓存中获取，一级缓存最多缓存 1024 条 SQL。二级缓存 是指可以跨 SqlSession 的缓存。是 mapper 级别的缓存，对于 mapper 级别的缓存不同的 sqlsession 是可以共享的。  **Mybatis 的一级缓存原理（sqlsession 级别）**  第一次发出一个查询 sql，sql 查询结果写入 sqlsession 的一级缓存中，缓存使用的数据结构是一 个 map。 key：MapperID+offset+limit+Sql+所有的入参 value：用户信息  同一个 sqlsession 再次发出相同的 sql，就从缓存中取出数据。如果两次中间出现commit 操作 （修改、添加、删除），本 sqlsession 中的一级缓存区域全部清空，下次再去缓存中查询不到所 以要从数据库查询，从数据库查询到再写入缓存。  **二级缓存原理（mapper 基本）**  二级缓存的范围是 mapper 级别（mapper 同一个命名空间），mapper 以命名空间为单位创建缓 存数据结构，结构是 map。mybatis 的二级缓存是通过 CacheExecutor 实现的。CacheExecutor13/04/2018 Page 139 of 283 其实是 Executor 的代理对象。所有的查询操作，在 CacheExecutor 中都会先匹配缓存中是否存 在，不存在则查询数据库。  key：MapperID+offset+limit+Sql+所有的入参  ***具体使用需要配置：***  1. Mybatis 全局配置中启用二级缓存配置  2. 在对应的 Mapper.xml 中配置 cache 节点  3. 在对应的 select 查询节点中添加 useCache=true  问题：微服务  回答：**服务注册发现**  服务注册就是维护一个登记簿，它管理系统内所有的服务地址。当新的服务启动后，它会向登记 簿交待自己的地址信息。服务的依赖方直接向登记簿要 Service Provider 地址就行了。当下用于服 务注册的工具非常多 ZooKeeper，Consul，Etcd, 还有 Netflix 家的 eureka 等。服务注册有两种  形式：客户端注册和第三方注册。  **客户端注册（zookeeper）**  客户端注册是服务自身要负责注册与注销的工作。当服务启动后向注册中心注册自身，当服务下 线时注销自己。期间还需要和注册中心保持心跳。心跳不一定要客户端来做，也可以由注册中心 负责（这个过程叫探活）。这种方式的缺点是注册工作与服务耦合在一起，不同语言都要实现一套注册逻辑。  **第三方注册（独立的服务 Registrar）**  第三方注册由一个独立的服务Registrar负责注册与注销。当服务启动后以某种方式通知Registrar， 然后 Registrar 负责向注册中心发起注册工作。同时注册中心要维护与服务之间的心跳，当服务不 可用时，向注册中心注销服务。这种方式的缺点是 Registrar 必须是一个高可用的系统，否则注册工作没法进展。  **客户端发现**  客户端发现是指客户端负责查询可用服务地址，以及负载均衡的工作。这种方式最方便直接，而 且也方便做负载均衡。再者一旦发现某个服务不可用立即换另外一个，非常直接。缺点也在于多语言时的重复工作，每个语言实现相同的逻辑。  **服务端发现**  服务端发现需要额外的 Router 服务，请求先打到 Router，然后 Router 负责查询服务与负载均衡。这种方式虽然没有客户端发现的缺点，但是它的缺点是保证 Router 的高可用。  **API 网关**  API Gateway 是一个服务器，也可以说是进入系统的唯一节点。API Gateway 封装内部系统的架构，并且提供 API 给各个客户端。它还可能有其他功能，如授权、监控、负载均衡、缓存、请求分片和管理、静态响应处理等。API Gateway 负责请求转发、合成和协议转换。所有来自客户端的请求都要先经过 API Gateway， 然后路由这些请求到对应的微服务。API Gateway 将经常通过调用多个微服务来处理一个请求以 及聚合多个服务的结果。它可以在 web 协议与内部使用的非 Web 友好型协议间进行转换，如 HTTP 协议、WebSocket 协议。  **请求转发 ：**服务转发主要是对客户端的请求安装微服务的负载转发到不同的服务上  **响应合并 ：**把业务上需要调用多个服务接口才能完成的工作合并成一次调用对外统一提供服务。  **协议转换 ：**重点是支持 SOAP，JMS，Rest 间的协议转换。  **数据转换 ：**重点是支持 XML 和 Json 之间的报文格式转换能力（可选）  **安全认证**  1. 基于 Token 的客户端访问控制和安全策略  2. 传输数据和报文加密，到服务端解密，需要在客户端有独立的 SDK 代理包  3. 基于 Https 的传输加密，客户端和服务端数字证书支持  4. 基于 OAuth2.0 的服务安全认证(授权码，客户端，密码模式等）  **配置中心**  配置中心一般用作系统的参数配置，它需要满足如下几个要求：高效获取、实时感知、分布式访问。  **事件调度（kafka）**  消息服务和事件的统一调度，常用用 kafka ，activemq 等。  **服务跟踪（starter-sleuth）**  随着微服务数量不断增长，需要跟踪一个请求从一个微服务到下一个微服务的传播过程， Spring Cloud Sleuth 正是解决这个问题，它在日志中引入唯一 ID，以保证微服务调用之间的一致性，这样你就能跟踪某个请求是如何从一个微服务传递到下一个。  **服务熔断（Hystrix）**  在微服务架构中通常会有多个服务层调用，基础服务的故障可能会导致级联故障，进而造成整个 系统不可用的情况，这种现象被称为服务雪崩效应。服务雪崩效应是一种因“服务提供者”的不 可用导致“服务消费者”的不可用,并将不可用逐渐放大的过程。熔断器的原理很简单，如同电力过载保护器。它可以实现快速失败，如果它在一段时间内侦测到许多类似的错误，会强迫其以后的多个调用快速失败，不再访问远程服务器，从而防止应用程序不断地尝试执行可能会失败的操作，使得应用程序继续执行而不用等待修正错误，或者浪费 CPU  时间去等到长时间的超时产生。熔断器也可以使应用程序能够诊断错误是否已经修正，如果已经修正，应用程序会再次尝试调用操作。  **Hystrix 断路器机制**  断路器很好理解, 当 Hystrix Command 请求后端服务失败数量超过一定比例(默认 50%), 断路器会切换到开路状态(Open). 这时所有请求会直接失败而不会发送到后端服务. 断路器保持在开路状态一段时间后(默认 5 秒), 自动切换到半开路状态(HALF-OPEN). 这时会判断下一次请求的返回情况, 如果请求成功, 断路器切回闭路状态(CLOSED), 否则重新切换到开路状态(OPEN). Hystrix 的断路器就像我们家庭电路中的保险丝, 一旦后端服务不可用, 断路器会直接切断请求链, 避免发送大量无效请求影响系统吞吐量, 并且断路器有自我检测并恢复的能力。  **Kafka**  **Kafka 概念**  Kafka 是一种高吞吐量、分布式、基于发布/订阅的消息系统，最初由 LinkedIn 公司开发，使用Scala 语言编写，目前是 Apache 的开源项目。  1. broker：Kafka 服务器，负责消息存储和转发  2. topic：消息类别，Kafka 按照 topic 来分类消息  3. partition：topic 的分区，一个 topic 可以包含多个 partition，topic 消息保存在各个partition 上  4. offset：消息在日志中的位置，可以理解是消息在 partition 上的偏移量，也是代表该消息的唯一序号  5. Producer：消息生产者  6. Consumer：消息消费者  7. Consumer Group：消费者分组，每个 Consumer 必须属于一个 group  8. Zookeeper：保存着集群 broker、topic、partition 等 meta 数据；另外，还负责 broker 故障发现，partition leader 选举，负载均衡等功能  **负载均衡（partition 会均衡分布到不同 broker 上）**  由于消息 topic 由多个 partition 组成，且 partition 会均衡分布到不同 broker 上，因此，为了有效利用 broker 集群的性能，提高消息的吞吐量，producer 可以通过随机或者 hash 等方式，将消息平均发送到多个 partition 上，以实现负载均衡。  **批量发送**  是提高消息吞吐量重要的方式，Producer 端可以在内存中合并多条消息后，以一次请求的方式发送了批量的消息给 broker，从而大大减少 broker 存储消息的 IO 操作次数。但也一定程度上影响了消息的实时性，相当于以时延代价，换取更好的吞吐量。    **消费者设计**  **Consumer Group**  同一 Consumer Group 中的多个 Consumer 实例，不同时消费同一个 partition，等效于队列模 式。partition 内消息是有序的，Consumer 通过 pull 方式消费消息。Kafka 不删除已消费的消息  对于 partition，顺序读写磁盘数据，以时间复杂度 O(1)方式提供消息持久化能力。  **RabbitMQ**  **概念**  RabbitMQ 是一个由 Erlang 语言开发的 AMQP 的开源实现。  AMQP ：Advanced Message Queue，高级消息队列协议。它是应用层协议的一个开放标准，为 面向消息的中间件设计，基于此协议的客户端与消息中间件可传递消息，并不受产品、开发语言等条件的限制。  RabbitMQ 最初起源于金融系统，用于在分布式系统中存储转发消息，在易用性、扩展性、高可  用性等方面表现不俗。具体特点包括：  1. 可靠性（Reliability）：RabbitMQ 使用一些机制来保证可靠性，如持久化、传输确认、发布确认。  2. 灵活的路由（Flexible Routing）：在消息进入队列之前，通过 Exchange 来路由消息的。对于典型的路由功能，RabbitMQ 已经提供了一些内置的 Exchange 来实现。针对更复杂的路由功能，可以将多个 Exchange 绑定在一起，也通过插件机制实现自己的 Exchange 。  3. 消息集群（Clustering）：多个 RabbitMQ 服务器可以组成一个集群，形成一个逻辑 Broker 。  4. 高可用（Highly Available Queues）：队列可以在集群中的机器上进行镜像，使得在部分节点出问题的情况下队列仍然可用。  5. 多种协议（Multi-protocol）：RabbitMQ 支持多种消息队列协议，比如 STOMP、MQTT等等。  6. 多语言客户端（Many Clients）：RabbitMQ 几乎支持所有常用语言，比如 Java、.NET、  Ruby 等等。  7. 管理界面（Management UI）:RabbitMQ 提供了一个易用的用户界面，使得用户可以监控和管理消息 Broker 的许多方面。  8. 跟踪机制（Tracing）:如果消息异常，RabbitMQ 提供了消息跟踪机制，使用者可以找出发生了什么。  9. 插件机制（Plugin System）:RabbitMQ 提供了许多插件，来从多方面进行扩展，也可以编写自己的插件。  **分布式缓存**  **缓存雪崩**  缓存雪崩我们可以简单的理解为：由于原有缓存失效，新缓存未到期间所有原本应该访问缓存的请求都去查询数据库了，而对数据库 CPU 和内存造成巨大压力，严重的会造成数据库宕机。从而形成一系列连锁反应，造成整个系统崩溃。一般有三种处理办法：  1. 一般并发量不是特别多的时候，使用最多的解决方案是加锁排队。  2. 给每一个缓存数据增加相应的缓存标记，记录缓存的是否失效，如果缓存标记失效，则更新数据缓存。  3. 为 key 设置不同的缓存失效时间。  **缓存穿透**  缓存穿透是指用户查询数据，在数据库没有，自然在缓存中也不会有。这样就导致用户查询的时候，在缓存中找不到，每次都要去数据库再查询一遍，然后返回空（相当于进行了两次无用的查询）。这样请求就绕过缓存直接查数据库，这也是经常提的缓存命中率问题。有很多种方法可以有效地解决缓存穿透问题，最常见的则是采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的 bitmap 中，一个一定不存在的数据会被这个 bitmap 拦截掉，从而避免了对底层存储系统的查询压力。另外也有一个更为简单粗暴的方法，如果一个查询返回的数据为空（不管是数据不存在，还是系统故障），我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。通过这个直接设置的默认值存放到缓存，这样第二次到缓冲中获取就有值了，而不会继续访问数据库。  **缓存预热**  缓存预热就是系统上线后，将相关的缓存数据直接加载到缓存系统。这样就可以避免在用户请求的时候，先查询数据库，然后再将数据缓存的问题！用户直接查询事先被预热的缓存数据！  **缓存更新**  缓存更新除了缓存服务器自带的缓存失效策略之外（Redis 默认的有 6 中策略可供选择），我们还可以根据具体的业务需求进行自定义的缓存淘汰，常见的策略有两种：  （1）定时去清理过期的缓存；  （2）当有用户请求过来时，再判断这个请求所用到的缓存是否过期，过期的话就去底层系统得到新数据并更新缓存。  **缓存降级**  当访问量剧增、服务出现问题（如响应时间慢或不响应）或非核心服务影响到核心流程的性能时，仍然需要保证服务还是可用的，即使是有损服务。系统可以根据一些关键数据进行自动降级，也可以配置开关实现人工降级。降级的最终目的是保证核心服务可用，即使是有损的。而且有些服务是无法降级的（如加入购物车、结算）。 |
| **人资问题(至少30个)** |
| 问题1：你的离职原因是啥？  回答：因为项目已经上线了，而且我在这家公司已经工作了很长时间了，也想给自己换个环境，还有就是接触新的项目提升自己。  问题2：你之前的公司是在哪里？  回答：朝阳区朝外大街MEN大厦  问题3：那你上班是坐几号线？  回答：从广渠门坐特12外,经过3站, 到朝阳门南站下车  问题4：之前的项目周期是多久？  回答：18年7月到19年5月，十个月左右  问题5：项目的人员配备是怎样的？  回答：4个后台，2个前端，2个运维，1个测试  6.在五年的时间内，你的职业规划是什么？  我希望能在公司稳定发展，并且个人技术水平也要利用空闲时间进行提升，并且能够为公司创造更多的利益；    7.你朋友对你的评价怎样？  我的朋友觉得我是个值得信赖的人，并且我也乐于帮助他们解决一些生活或者工作上的困难，我们关系很融洽；    8.你还有什么问题要问的吗？  办理入职时要带什么资料？  什么时候入职合适？  如果入职了有没有人带一带，以便尽快熟悉公司业务。    9.如果面试录用了你，一段时间后发现你不适合这个职位，你怎么办？  通过自身努力学习、虚心求教在，尽快适应尽量不出现这种情况，如果实在不适合不给公司添加任何麻烦。      10.工作时，你认为领导要求的方式不是最好的，你应该怎么做？  原则上服从领导，私底下找机会请教、婉转的表达自己的想法看看领导是否会改变，如果没改变就按领导的走。    11.如果你的工作出现失误，给本公司造成经济损失，你认为该怎么办？  我本意是为公司努力工作如果造成损失首要问题是想办法去弥补或挽回损失，分清责任各负其责，如果是本人的责任自愿受罚，如果是团队成员的失误帮助其找到原因并解决。    12.如果你做的工作受到上级领导的表扬，但你的主管说是他做的，你怎样？  如果没有领导的正确指导和大力支持自己是不能做好的，是谁做好工作并不重要，只要对公司有利就行。    13.谈谈你对跳槽的看法？  正常的跳槽能促进人才合理流动，应该支持。频繁跳槽对单位和个人双方都不利，应该反对。    14.工作中你难以和同事、上司相处，你该怎么办？  会服从领导指挥配合同事工作，从自身找原因分析是不是自己的问题让领导不满意同事看不惯，是不是为人处事做的不好，如果是就努力去改正，如果不是就去找机会沟通，找出问题关键并解决。      15.假设你工作做的很好，得到领导肯定，但同事孤立你，你怎么看这个问题？  工作做的好是好事也是本分，以后也会更加努力。同事孤立需要检讨一下自己是不是没有正确认识成绩，取得成绩要共享荣誉，遇到问题要检讨自己。  16.你对我们公司了解多少？  关键点：面试前要提前对面试企业全面的了解一下。  介绍点：了解内容包括（企业文化、性质、经营范围、取得的成就等）。    17.请说出你选择这份工作的动机？  我之前了解过公司的文化，我很喜欢\*\*\*\*，我觉得公司文化的某些方面和我的观念一致，我觉得在这样的公司能够实现自己的价值，能够让自己的能力最大化的为公司创造价值；        20.最能概括你自己的三个词是什么？  适应力强、有责任心、做事善始善终等。    21.你的业余爱好是什么？  比较喜欢打篮球和登山。    22.作为被面试者给我打一下分？  亲和力强，责任意识强、为人和善等。      23.你怎么理解你应聘的职位？  应聘的职位和自身很适合，也有经验，可以和领导以及同事更完美的配合。    24.喜欢这份工作的哪一点？  有挑战性的工作，可以在为公司创造效益的时候更大的提升自己。      25.为什么要离职？  由于自身的原因，想找一个适合自己发挥的平台让自己能更好的发展。    26.说说你对行业、技术发展趋势的看法？  游戏开发行业是吸金度最高的行业，也是行业的领头羊技术含量较高，行业发展前景很好，在人们对物质条件满意的时候就会选择游戏，这就让我们游戏行业的用户量只增不减。    27.对工作的期望与目标是什么？  期望就是让自己得到更好的发展，目标是用自己的努力为公司创造更高的效益。    28.说说你的家庭？  家庭的生活经历构建了自身吃苦耐劳、认真负责、有始有终的良好素养。    29.你觉得你还欠缺什么？  社会经验不足、沟通能力需要提高    30.你欣赏哪种性格的人？  欣赏守信和负责的人，守信的人可以放心做朋友，负责的人可以一起共事。 |