|  |
| --- |
| 技术问题(至少50个) |
| 问题1:什么是应用程序?  应用程序是指可以执行的软件,每一个应用程序都会有一个进程.  问题2:什么是进程?  进程是指正在运行的程序,它是线程的集合。  问题3:什么是线程?  线程是正在独立运行的一条执行路径，一个独立的执行单元。  注：多线程并不是提高宽带速度而只是程序效率。  问题4:线程的几种状态?   1. 新建状态(new) 2. 就绪状态(Runnable等待cpu调度就绪状态)   c、运行状态(Running cpu开始执行run)  d、阻塞状态(Blocked)  e、死亡状态(Dead 调用stop方法)  问题5：什么是多线程？  为了提高程序的执行效率。  问题6：多线程的应用场景？  多线程下载、QQ、爬虫、前端开发ajax(异步上传)、分布式job(需要同一时刻执行多个任务调度)、  总结：使用多线程体现程序的效率。  问题7：创建线程的方式？  使用继承thread+重写run()方法体  使用实现runnable接口实现多线程(利用静态代理的设计模式)  使用匿名内部类方式  使用实现Callable接口（jdk1.5并发包）  使用线程池创建线程(我们在企业时使用这种方式)  注:run()方法:线程需要执行的任务。  怎样调用线程？将线程类实例化，调用Start()方法并不是调用run()方法。  使用开启多线程，代码不会从上往下执行。  例：使用继承方式创建线程好还是使用runnable接口好？  接口方式，首先开发都是以面向接口编程，同时接口使用多个继承。  问题8:什么是死锁?如何解决死锁?  过多的同步容易造成死锁  解决死锁的方式是生产者消费者模式利用信号灯法进行解决  信号灯法:生产者生产  消费者等待 生产者完成后通知消费  消费者 消费  生产者等待  消费者完成通知生产  问题9：死锁产生的原因？  是由于访问共享资源顺序不当造成的。简单的说，死锁就是指两个或两个以上的线程在执行过程中，因争  夺资源而造成的一种互相等待的现象，如果没有外力作用，他们都将无法继续执行下去。  问题10:死锁如果产生如何避免？  多线程产生死锁需要四个条件，分别是互斥性，保持和请求，不可剥夺性还有要形成闭环，这四个条件缺一不可，只要破坏了其中一个条件就可以破坏死锁，其中最简单的方法就是线程都是以同样的顺序加锁和释放锁  \*多线程产生死锁的四个必要条件：  互斥条件：一个资源每次只能被一个进程使用。  保持和请求条件：一个进程因请求资源而阻塞时，对已获得资源保持不放。  不可剥夺条件：进程已获得资源，在未使用完成前，不能被剥夺。  循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。  有序的资源分配法，以同样的顺序加锁和释放锁  问题11:解决线程安全问题的方案？  a、通过加synchronized同步锁  b、避免使用全局变量和静态变量  c、ThreadLocal类用来提供线程内部的局部变量  问题12守护线程是什么？  守护线程是与main函数相关，是运行在后台的一种特殊进程。有一个特征是与主线程一起销毁。在 Java 中垃圾回收线程就是特殊的守护线程。  应用场景：App下载东西时,App关闭后台还在下载。  问题13. 说一下 runnable 和 callable 有什么区别？  runnable 没有返回值，callable 可以拿到有返回值，callable 可以看作是 runnable 的补充。  问题14: sleep() 和 wait() 有什么区别？  类的不同：sleep() 来自 Thread，wait() 来自 Object。  释放锁：sleep() 不释放锁；wait() 释放锁。  用法不同：sleep() 时间到会自动恢复；wait() 可以使用 notify()/notifyAll()直接唤醒。  问题15:线程的 run() 和 start() 有什么区别？  start() 方法用于启动线程，run() 方法用于执行线程的运行时代码。run() 可以重复调用，而 start() 只能调用一次。  问题16:Object 对象的方法有哪些？你所了解的？  对[Java](http://www.chinawin.net/tag/java/).lang.Object类的一些重要方法进行重写  hashCode(),toString(),equals(),finalize(),clone(),wait(),notify()/notifyAll() 这八个方法  一 Equals()方法：  1.何时需要重写equals()  当一个类有自己特有的“逻辑相等”概念（不同于对象身份的概念）。  2.hashCode()方法：  当改写equals()的时候，总是要改写hashCode()  根据一个类的equals方法（改写后），两个截然不同的实例有可能在逻辑上是相等的，但是，根据Object.hashCode方法，它们仅仅是两个对象。因此，违反了“相等的对象必须具有相等的散列码”。  3.toString()方法  toString()将对象转换为字符串  问题17:默认的端口号  ftp 21 文件传输协议  http 80 超文本传输协议  oracle 默认端口号1521  mysql默认端口号 3306  Mongodb 默认端口号 27017  Tomcat 默认的端口号 8080  Redis 默认端口号 6379,默认16个数据库  Svn 默认端口号 443  Dubbo默认端口号:20880  Zookeeper:默认端口号2181  Kafka：9092  Rabbitmq: 默认端口号 程序连:5672,控制台:15672,集群:25672  问题18:SpringMVC运行原理  tomcat启动时，也是先加载web.xml，找到spring mvc的前端总控制器DispatcherServlet,并且通过DispatcherServlet来加载相关的配置文件信息。当浏览器发出一个请求之后，首先找到DispatcherServlet，通过这个控制器和代码中的requestMapping注解找到对应的controller中的方法，参数直接通过方法参数接收就可以，然后调用service、dao操作数据库，返回数据。如果页面跳转，controller的方法可以返回ModelAndView和string类型，再通过spring-mvc配置文件中的视图解析器找到对应的页面。  数据可以用request或ModelAndView返回到页面。如果是ajax请求，可以直接返回List或对象，加上responseBody注解，进行返回。  问题19:解释一下什么是 aop?  aop 是面向切面编程，通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术。简单来说就是统一处理某一“切面”（类）的问题的编程思想，比如统一处理日志、异常等。  问题20:解释一下什么是 ioc？  ioc：Inversionof Control（中文：控制反转）是 spring 的核心，对于 spring 框架来说，就是由 spring 来负责控制对象的生命周期和对象间的关系。  简单来说，控制指的是当前对象对内部成员的控制权；控制反转指的是，这种控制权不由当前对象管理了，由其他（类,第三方容器）来管理。  问题21. spring 有哪些主要模块？  spring core：框架的最基础部分，提供 ioc 和依赖注入特性。  spring context：构建于 core 封装包基础上的 context 封装包，提供了一种框架式的对象访问方法。  spring dao：Data Access Object 提供了JDBC的抽象层。  spring aop：提供了面向切面的编程实现，让你可以自定义拦截器、切点等。  spring Web：提供了针对 Web 开发的集成特性，例如文件上传，利用 servlet listeners 进行 ioc 容器初始化和针对 Web 的 ApplicationContext。  spring Web mvc：spring 中的 mvc 封装包提供了 Web 应用的 Model-View-Controller（MVC）的实现。  问题22. spring 常用的注入方式有哪些？  setter 属性注入  构造方法注入  注解方式注入  问题23:spring 事务实现方式有哪些？  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。  问题24. 说一下 spring mvc 运行流程？  spring mvc 先将请求发送给 DispatcherServlet。  DispatcherServlet 查询一个或多个 HandlerMapping，找到处理请求的 Controller。  DispatcherServlet 再把请求提交到对应的 Controller。  Controller 进行业务逻辑处理后，会返回一个ModelAndView。  Dispathcher 查询一个或多个 ViewResolver 视图解析器，找到 ModelAndView 对象指定的视图对象。  视图对象负责渲染返回给客户端。  问题25. spring mvc 有哪些组件？  前置控制器 DispatcherServlet。  映射控制器 HandlerMapping。  处理器 Controller。  模型和视图 ModelAndView。  视图解析器 ViewResolver。  问题26:什么是分布式事务  简单的说，就是一次大的操作由不同的小操作组成，这些小的操作分布在不同的服务器上，且属于不同的应用，分布式事务需要保证这些小操作要么全部成功，要么全部失败。本质上来说，分布式事务就是为了保证不同数据库的数据一致性。  问题27:分布式事务的产生的原因  1数据库分库分表  当数据库单表一年产生的数据超过1000W，那么就要考虑分库分表，具体分库分表的原理在此不做解释，以后有空详细说，简单的说就是原来的一个数据库变成了多个数据库。这时候，如果一个操作既访问01库，又访问02库，而且要保证数据的一致性，那么就要用到分布式事务。  2应用SOA化  所谓的SOA化，就是业务的服务化。比如原来单机支撑了整个网站，现在对整个网站进行拆解，分离出了诸多个模块。各个模块有自己对用的数据库存储信息，如果要同时对两三个模块进行操作，那么就会涉及到数据库的数据能否一致性，为了保证数据一致性，就需要用到分布式事务。  问题28:Java 特性?什么是多态?  Java的特性封装、继承、多态  封装隐藏了类的内部实现机制，可以在不影响使用的情况下改变类的内部结构，同时也保护了数据。对外界而已它的内部细节是隐藏的，暴露给外界的只是它的访问方法。  多态性是指允许不同子类型的对象对同一消息作出不同的响应。简单的说就是用同样的对象引用调  用同样的方法但是做了不同的事情。多态性分为编译时的多态性和运行时的多态性。  继承:子类继承父类重写父类里面的方法。  回答问题29:Vue的技术了解?  Vue.js框架是构建构建用户界面的渐进式框架,只关注视图自底而上的增量开发,目标是通过简单的API实现响应的数据可以绑定和组合视图组件。  为什么你们项目中会用到vue?  没有了HTML里面的div盒子的相互嵌套,没有了JQuery对DOM的复杂操作,像传统开发一样全部混合在HTML中，那么要对它们进行处理会十分的费劲，并且如果其中有几个结构之间存在藕断丝连的关系，那么会导致代码上出现更大的问题Vue 的核心库只关注视图层,有声明式，响应式的数据绑定，与组件化的开发。  回答问题30什么是渐进式框架？  看来，渐进式代表的含义是：主张最少。  每个框架都不可避免会有自己的一些特点，从而会对使用者有一定的要求，这些要求就是主张，主张有强有弱，它的强势程度会影响在业务开发中的使用方式。  而Vue它只是个轻量视图而已，只做了自己该做的事，没有做不该做的事，仅此而已。  自底而上和增量开发是什么意思？  增量开发就是老的代码不要去动，直接增加新的编码，使老代码保持封闭，这样便于管理。  自底而上就是像树的底部到顶部类似对于编程来说就是从编写基础到扩大规模到升级功能。  回答问题31:WebService和httpcllient 区别?  HttpClient对认证机制+、提供了全面的支持。用户认证过程需要一组用户名/密码进行认证用户身份。HttpClient附带多个AuthScheme(a fu si gei mu)的实现。 AuthScheme接口表示一个抽象的面向挑战-应答的认证机制。解析和处理目标服务器发送的挑战并且对受保护资源的请求做出应答。  WebService一般通过iis进行匿名调用。这种方式不安全。我们可以使用CXF的拦截器通过自定义的Interceptor，可以改变请求和响应的一些消息处理，其中最基本的原理还是一个动态代理。设置拦截器。来验证用户的账号和密码。  WebService是完全基于XML的，HttpClient是基于HTTP协议的；WebService传输的是对象，HttpClient传输的是JSON格式的数据  回答问题32:Rabbitmq消息队列?  当初我们做短信发送的时候用到了rabbitmq消息队列，我们当时有一个短信发送平台，平台采用监听消息队列方式进行短信的发送，我们在项目中通过配置一个RebbitConfig在上面加@Configuration注解，然后在里边new一个Queue对象创建一个消息队列，然后我们再发送短信的地方注入AmqpTemplate,最后通过调用AmqpTemplate.convertAndSend将消息发送到队列当中，convertAndSend里边有两个参数，第一个是消息队列名称，第二个是消息队列发送进去的内容。短信发送平台通过RabbitListener queues等于队列名称，监听队列，通过RabbitHandler注解获取到消息队列中的数据，然后通过httpClient调用短信发送接口，将短信发送出去并将发送的数据以及发送状态记录到数据库中。  回答问题33:jsonp跨域请求实原理?  先创建一个script 设置他的src 回调函数也在src中设置 然后在页面中返回json字符串放入到回调函数中，在对数据进行操作。这样一来用户的登录凭证token就可以在不同的模块中随意传输，就达到了只登陆一次就可以访问任意的功能模块。  回答问题34:为什么要使用熔断器？  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，这就是服务雪崩效应。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主键放大的过程。我们使用Hystrix来解决这个问题，Hystrix能使你的系统在出现依赖的服务失效的时候，通过隔离系统所依赖的服务，防止服务级联失败，同时提供失败回退机制，更优雅地应对失效，并使系统能更快的从异常中恢复。应用：Feign本身支持Hystrix，不需要额外引入依赖。  回答问题35:什么是JTA?  JTA是Java Transaction API简称 ，即Java事务管理器，主要管理本地事务。本地事务是只同一个数据库源中，多次数据库连接之间的事务，使得这多个数据表操作遵循ACID（原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability））JTA是基于XA架构上建模的，在JTA 中，事务管理器抽象为javax.transaction.TransactionManager接口，并通过底层事务服务（即JTS）实现。  回答问题36：什么是XA?  XA是由X/Open组织提出的两阶段提交协议，分布式事务的规范。XA规范主要定义了(全局)事务管理器和(局部)资源管理器之间的接口。XA接口是双向的系统接口，在事务管理器以及一个或多个资源管理器之间形成通信桥梁。XA之所以需要引入事务管理器是因为，在分布式系统中，从理论上讲两台机器理论上无法达到一致的状态，需要引入一个单点进行协调。事务管理器控制着全局事务，管理事务生命周期，并协调资源。资源管理器负责控制和管理实际资源（如[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "https://www.cnblogs.com/zhaoxinshanwei/p/_blank)或JMS队列）。  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：准备阶段和第二阶段：提交阶段。  回答问题37:什么是2pc,什么是3pc?  2pc即二阶段提交协议,二阶段提交协议主要分为来个阶段：准备阶段和提交阶段。(首先协调者（牧师）会询问两个参与者（二位新人）是否能执行事务提交操作（愿意结婚）。如果两个参与者能够执行事务的提交，先执行事务操作，然后返回YES，如果没有成功执行事务操作，就返回NO。)  三阶段提交协议（3PC）3PC最关键要解决的就是协调者和参与者同时挂掉的问题，3PC把2PC的准备阶段再次一分为二，这样三阶段提交就有CanCommit、PreCommit、DoCommit三个阶段。在第一阶段，只是询问所有参与者是否可可以执行事务操作，并不在本阶段执行事务操作。当协调者收到所有的参与者都返回YES时，在第二阶段才执行事务操作，然后在第三阶段在执行commit或者rollback。  回答问题38:谈谈分布式系统中CAP理论和BASE理论?  CAP理论认为在一个系统中对某个数据不存在一个算法同时满足一致性，可用性，分区容错性。往往需要牺牲某一项来满足其他两项。BASE理论即使无法做到强一致性,但每个应用都可以根据自身的业务特点，采用适当的方式来使系统达到最终一致性。  回答问题39:什么是柔性事务?  柔性事务是用来解决分布式事务的方案,所谓柔性事务，相比较与数据库事务中的ACID这种刚性事务来说，柔性事务保证的是“基本可用，最终一致。”这其实就是基于BASE理论，保证数据的最终一致性。  (1)柔性事务的基础?异步确保型、补偿型、最大努力通知型。  异步确保型:将一些同步阻塞的事务操作变为异步的操作,避免对数据库事务的争用,典型例子是热点账户异步记账,批量记账的处理。  最大努力通知型：一些交易的消息的通知  补偿型：TCC事务  例子:在一个长事务中,一个由两台服务器一起参与的 事务,服务器A发起事务,服务器B参与事务,B的事务需要人工参与,所以处理时间可能很长.如果按照ACID的原则.要保持事务的隔离性,一致性,服务器A发起的事务中使用到的事务资源将会被锁定,不允许其他应用访问到事务过程中的中间结果,直到整个事务被提交或者回滚,这就造成事务A中的资源被长时间锁定,系统的可用性将不可接受。  回答问题40：什么是Tcc事务?  TCC 将事务提交分为 Try - Confirm - Cancel 3个操作  Try：预留业务资源/数据效验 Confirm：确认执行业务操作Cancel：取消执行业务操作  TCC 事务应用场景  我们通过用户下单使用余额+红包支付来看一下TCC事务的具体应用。  假设用户下单操作来自3个系统下单系统、资金账户系统、红包账户系统，下单成功需要同时调用资金账户服务和红包服务完成支付  假设购买商品1000元，使用账户红包200元，余额800元，确认支付。  Try操作 tryX 下单系统创建待支付订单 tryY 冻结账户红包200元 tryZ 冻结资金账户800元  Confirm操作 confirmX 订单更新为支付成功 confirmY 扣减账户红包200元 confirmZ 扣减资金账户800元  Cancel操作 cancelX 订单处理异常，资金红包退回，订单支付失败 cancelY 冻结红包失败，账户余额退回，订单支付失败 cancelZ 冻结余额失败，账户红包退回，订单支付失败  回答问题41:Atomic包的简单介绍?  这个包提供了一系列原子类。这些类可以保证多线程环境下，当某个线程在执行atomic的方法时，不会被其他线程打断，而别的线程就像自旋锁一样，一直等到该方法执行完成，才由JVM从等待队列中选择一个线程执行。Atomic类在软件层面上是非阻塞的，它的原子性其实是在硬件层面上借助相关的指令来保证的。  什么是LCN?  LCN是解决分布式事务的一个框架,LCN并不生产事务，LCN只是本地事务的搬运工  1  Lcn原理:发起方调用接口,去xml创建事务分组,将分组的id传入,参与方收到id,做一个假的关闭,不提交事务,当发起方代码运作完成后通知协调者,然后进行转发,转发给参与方,看事务回滚还是提交.  问题42:什么是Nginx  回答:  Nginx是一个高级的轻量级的web服务器,由俄罗斯科学家开发的，具有如下优点:  1.占用内存少，并发量强，支持多种并发连接，效率高.  2.能够作为负载均衡服务器和(内部直接支持 Rails 和 PHP)代理服务器。Nginx用C编写开销和CPU占有小.  3.安装启动简单，配置简洁，bug少，一般几个月不需要重新启动且不会宕机，稳定性和安全性好.问题43:Nginx的作用  回答:  反向代理、负载均衡、配置主备tomcat、动静分离  问题44:Nginx 应用场景?  回答:  做HTTP服务器、反向代理服务器、静态资源服务器  问题45:什么是分布式架构?  分布式架构就是将传统结构按照模块进行拆分，不同的人负责不同的模块，不会产生代码冲突问题，方便开发。  问题46:什么是微服务架构?  微服务类似于SOA架构，但是比SOA架构粒度更细，更轻量  问题47:什么是SOA架构?  SOA架构就是将业务逻辑层提取出来，将相似的业务逻辑形成一个服务，提供外部访问接口，服务之间访问通过RPC调用实现。  问题48:Eureka如何实现高可用?  启动多台Eureka服务器，然后作为SpringCloud服务互相注册，客户端从Eureka集群获取信息时，按照注册的Eureka顺序对第一个Eureka进行访问。  问题49:谈谈雪崩效应  雪崩效应是在大型互联网项目中，当某个服务发生宕机时，调用这个服务的其他服务也会发生宕机，大型项目的微服务之间的调用是互通的，这样就会将服务的不可用逐步扩大到各个其他服务中，从而使整个项目的服务宕机崩溃.发生雪崩效应的原因有以下几点   1. 单个服务的代码存在bug. 2请求访问量激增导致服务发生崩溃 3.服务器的硬件故障也会导致部分服务不可用.   问题50:服务雪崩效应产生的原因?  因为Tomcat默认情况下只有一个线程池来维护客户端发送的所有的请求，这时候某一接口在某一时刻被大量访问就会占据tomcat线程池中的所有线程，其他请求处于等待状态，无法连接到服务接口.  51. 什么是 spring boot？  spring boot 是为 spring 服务的，是用来简化新 spring 应用的初始搭建以及开发过程的。  52. 为什么要用 spring boot？  配置简单  独立运行  自动装配  无代码生成和 xml 配置  提供应用监控  易上手  提升开发效率  53. spring boot 核心配置文件是什么？  spring boot 核心的两个配置文件：  bootstrap (. yml 或者 . properties)：boostrap 由父 ApplicationContext 加载的，比 applicaton 优先加载，且 boostrap 里面的属性不能被覆盖；  application (. yml 或者 . properties)：用于 spring boot 项目的自动化配置。  54. spring boot 配置文件有哪几种类型？它们有什么区别？  配置文件有 . properties 格式和 . yml 格式，它们主要的区别是书法风格不同。  . properties 配置如下：  spring. RabbitMQ. port=5672  . yml 配置如下：  spring:  RabbitMQ:  port: 5672  . yml 格式不支持 @PropertySource 注解导入。  55. spring boot 有哪些方式可以实现热部署？  使用 devtools 启动热部署，添加 devtools 库，在配置文件中把 spring. devtools. restart. enabled 设置为 true；  使用 Intellij Idea 编辑器，勾上自动编译或手动重新编译。  56. jpa 和 hibernate 有什么区别？  jpa 全称 Java Persistence API，是 Java 持久化接口规范，hibernate 属于 jpa 的具体实现。  57. 什么是 spring cloud？  spring cloud 是一系列框架的有序集合。它利用 spring boot 的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用 spring boot 的开发风格做到一键启动和部署。  58. spring cloud 断路器的作用是什么？  在分布式架构中，断路器模式的作用也是类似的，当某个服务单元发生故障（类似用电器发生短路）之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个错误响应，而不是长时间的等待。这样就不会使得线程因调用故障服务被长时间占用不释放，避免了故障在分布式系统中的蔓延。  59. spring cloud 的核心组件有哪些？  Eureka：服务注册于发现。  Feign：基于动态代理机制，根据注解和选择的机器，拼接请求 url 地址，发起请求。  Ribbon：实现负载均衡，从一个服务的多台机器中选择一台。  Hystrix：提供线程池，不同的服务走不同的线程池，实现了不同服务调用的隔离，避免了服务雪崩的问题。  Zuul：网关管理，由 Zuul 网关转发请求给对应的服务。  60. 为什么要使用 hibernate？  hibernate 是对 jdbc 的封装，大大简化了数据访问层的繁琐的重复性代码。  hibernate 是一个优秀的 ORM 实现，很多程度上简化了 DAO 层的编码功能。  可以很方便的进行数据库的移植工作。  提供了缓存机制，是程序执行更改的高效。  61. 什么是 ORM 框架？  ORM（Object Relation Mapping）对象关系映射，是把数据库中的关系数据映射成为程序中的对象。使用 ORM 的优点：提高了开发效率降低了开发成本、开发更简单更对象化、可移植更强。  62:SVN? svn主干以及分支的开发模式 解决版本冲突－使用SVN主干与分支功能  1  前言  大多数产品开发存在这样一个生命周期：编码、测试、发布，然后不断重复。通常是这样的开发步骤：  1)    开发人员开发完毕某一版本（如版本A）功能后，提交测试；  2)    测试人员对待发布版本A进行测试，同时开发人员继续开发新功能(如版本B)；  3)    测试人员提交bug，研发人员修复bug，同时继续开发新功能；  4)    重复第3步骤，直到待发布版本A测试通过测试后，发布第一版本  这样就会存在以下问题：  1)    如何从代码库中(A+B)分离出待发布版本A，进行测试和发布；  2)    如果单独存放待发布版本A，那么开发组必须同时维护此版本库A以及当前最新代码库(A+B)，操作冗余且容易出错。  在SVN中，通常采用主干（trunk）与分支（branches）的方法，解决以上问题。  2  相关概念和原理  在SVN中创建代码库时，通常会创建trunk、branches、tags三个子目录，当然，你也可以用其他名称来实现主干和分支的功能    trunk－主干，或称主线，顾名思义，是开发的主线。    branches－分支，是从主线上分出来，独立于主线的另一条线。可以创建多个分支。一个分支总是从主干一个备份开始的，从那里开始，发展自己独有的历史(如下图所示)。在版本控制的系统中，我们经常需要对开发周期中的单独生命线作单独的修改，这条单独的开发生命线就可以称为Branches，即分支。分支经常用于添加新的功能以及产品发布后的bug修复等，这样可以不影响主要的产品开发线以及避免编译错误等。当我们添加的新功能完成后可以将其合并到主干中。    tags－标记，主要用于项目开发中的里程碑，比如开发到一定阶段可以单独一个版本作为发布等，它往往代表一个可以固定的完整的版本。即主干和分支都是用来进行开发，而标记是用来进行阶段发布的。安全公司的配置库有专门的发布区，所以tags并不需要创建，在这里只是提供说明，不推荐使用。  branches以及tags在TortoiseSVN中创建方法是一致的，它们都是通过存储类似Linux中的lunch快捷方式一样，只是创建了指向某个版本的链接，而不会真正将此版本的内容复制到分支或者标记中，这样既可以节省空间，也可以很快速的创建，被称为“廉价的拷贝”。  为了便于创建分支和标记，通常习惯于将Repository版本库的结构布置为:/branches,/tags,/trunk。分别代表分支，标记以及主干。  还有一点值得注意的是，SVN不推荐在创建的tag基础上Revision，这种情况应用branches，因为tag一般保持不变不作任何修改。  3  代码的分支管理策略  关于代码管理的分支和发布策略，目前主要有两种：一种是主干作为新功能开发主线，分支用作发布。另一种是分支用作新功能开发，主干作为稳定版的发布。  3.1  分支用来发布  典型操作步骤如下：  1)  开发者提交所有的新特性到主干。 每日的修改提交到/trunk：新特性，bug修正和其他。  2)  这个主干被拷贝到“待发布”分支。 当小组认为软件已经做好发布的准备（如，版本1.0）然后/trunk会被拷贝到/branches/1.0。  3)  项目组继续并行工作，一个小组开始对分支进行严酷的测试，同时另一个小组在/trunk继续新的工作（如，准备2.0），如果一个bug在任何一个位置被发现，错误修正需要来回运送。然而这个过程有时候也会结束，例如分支已经为发布前的最终测试“停滞”了。  4)  分支已经作了标记并且发布，当测试结束，/branches/1.0作为引用快照已经拷贝到/tags/1.0.0，这个标记被打包发布给客户。  5)  分支多次维护。当继续在/trunk上为版本2.0工作，bug修正继续从/trunk运送到/branches/1.0，如果积累了足够的bug修正，管理部门决定发布1.0.1版本：拷贝/branches/1.0到/tags/1.0.1，标记被打包发布。  整个过程随着软件的成熟不断重复：当2.0完成，一个新的2.0分支被创建，测试、打标记和最终发布，经过许多年，版本库结束了许多版本发布，进入了“维护”模式，许多标记代表了最终的发布版本。  这种分支管理策略被广泛的应用于开源项目。比如freebsd的发布就是一个典型的例子。  freebsd的主干永远是current，也就是包括所有最新特性的不稳定版本。然后随着新特性的逐步稳定，达到一个发布的里程碑以后，从主干分出来一个stable分支。freebsd是每个大版本一个分支。也就是说4.x，5.x，6,x各一个分支。每个发布分支上只有bug修改和现有功能的完善，而不会再增加新特性。新特性会继续在主干上开发。当稳定分支上发生的修改积累到一定程度以后，就会有一次发布。发布的时候会在稳定分支上再分出来一个 release分支。以6.x为例，就会有6.0,6.1,6.2…等发布分支。  这种发布方法非常适用于产品线的发布管理。产品是要卖的，以前卖给客户的版本仍需要继续维护，而为了以后的市场，新功能也不断地在增加。这种管理方法对已发布产品的维护工作和下一代产品的开发工作进行了隔离。对于已经发布的产品，只有维护的补丁发布。而新发行的产品不仅包括了所有的bug修改，还包括了新功能。  这种方法具有如下缺点：首先，必须对主干上的新功能增加进行控制。只能增加下一个发布里面计划集成进去的新特性。而且，已经在主干上集成的新特性中的任何一个，如果达不到里程碑的要求，稳定分支就不能创建，这很有可能影响下一个发布的计划。开源项目可能这方面的压力小一些，但是商业产品开发如果碰到这种情况就危险了。还有一个缺点就是bug修改必须在各个分支之间合并。从分支和合并的一些实践经验上看，各个长期存在的分支之间必须要周期性的进行合并，否则很容易引发合并冲突。可是各个stable分支以及release分支之间恰好是不能进行合并而且还要长期存在的。因此，采用这种分支策略可能碰到的最大问题就是某个分支上的bug修改内容往其它分支merge的时候出现的冲突。而且一旦发现一个bug，调查这个bug影响哪些分支的工作会随着维护的发布分支的数量而增加。  在非产品开发的外包软件项目里面，这种发布方法的好处体现不出来，而缺点仍然存在。外包项目的特点是客户永远需要“最新”的代码，因此对已经发布的某个分支进行维护的情况很少出现（在测试的时候会出现）。而且发布的方法和产品的发布也不一样。产品的发布，只要把发布分支上的代码编译成安装盘就可以了，而外包的发布往往是把上一次发布和这一次发布之间发生变化的代码送给客户。如果每次发布都是一个分支的话，将会出现两个分支上的比较。强大的版本控制工具当然支持这种比较，但是很多版本工具不支持分支之间的比较，而只支持分支内的不同版本之间的比较。因此为了避免发布方法受工具的限制，就要避免出现分支间比较的情况。针对外包开发的特殊情况，只有采用另外一种分支管理策略。  3.2   主干用来发布  与第一种分支策略正好相反，主干上永远是稳定版本，可以随时发布。bug的修改和新功能的增加，全部在分支上进行。而且每个bug和新功能都有不同的开发分支，完全分离。而对主干上的每一次发布都做一个标记而不是分支。分支上的开发和测试完毕以后才合并到主干。  这种发布方法的好处是每次发布的内容调整起来比较容易。如果某个新功能或者bug在下一次发布之前无法完成，就不可能合并到主干，也就不会影响其他变更的发布。另外，每个分支的生命期比较短，唯一长期存在的就是主干，这样每次合并的风险很小。每次发布之前，只要比较主干上的最新版本和上一次发布的版本就能够知道这次发布的文件范围了。  这种发布模式也有缺点。如果某个开发分支因为功能比较复杂，或者应发布计划的要求而长期没有合并到主干上，很可能在最后合并的时候出现冲突。因此必须时刻注意分支离开主干的时间。如果有的分支确实因为特殊的需要必须长期存在，那就必须定期把主干的更新往这个分支上合并。为了减少这种合并发生的次数，并且限定合并的范围，要为每次发布预先建立一个发布分支，然后所有的开发分支根据自己的发布计划向各个发布分支合并。当下一次发布的分支上已经集成了所有的变更并且测试完毕以后，把这个发布分支内容合并到主干，发布主干，然后锁定或者删除这个分支。然后把主干上的所有更新合并到后面几个发布分支里面去。外包项目的发布周期一般都比较短，往往客户验收测试的周期就是发布周期。所以这种方法就够用了。如果发布周期很长，各个发布分支之间还要定期的从前向后合并。这种发布方法还有一个缺点就是测试。不像第一种分支策略，发布的分支就是测试的分支。这种发布模式的测试分支往往是各个发布分支，在正式发布之前才把下一个发布分支上的更新合并到主干，这就引入了合并出错的风险，而主干上的程序是没有经过测试的。幸好从这个发布模式上看，下一个发布分支的合并基础应该和主干上一次发布内容相同，所以引入合并错误的风险很低。还有一种建议就是不设置主干，下一个发布分支就是主干，直接发布下一个发布分支的变更内容，然后把变更合并到再下一个发布分支上去。以此类推。  3.3  注意事项  1)     做分支上做开发的时候，必须定期使分支与主干同步，避免开发完成后合并(merge)回主干时出现严重冲突(confict)；  2)     进行合并前，处理掉工作副本上的所有本地修改，方便合并失败时进行回滚(revert)；  3)     进行合并时，特别注意 新增/删除 操作，因为很多冲突都是这类操作引起的；  4)     完成一个分支的功能并合并回主干后，抛弃该分支，后续其它功能的开发使用新建的分支。当然，也有办法继续使用该分支；  5)     辅助文档是必需的。为了观察分支的创建和合并的过程，至少需要一份类似泳道图的文档标记每一次分支创建和合并的过程；  6)     开发分支往主干或者发布分支合并的次数应该尽可能少。一般来讲应该在单体测试结束合并到主干或者发布分支，然后进行结合测试。如果结合测试里发现bug不应该在原来的开发分支上继续修改，而应该创建新的分支进行修改；  7)     分支创建和合并的log必须规范。便于以后查找。基本的log信息应该包括从哪个分支的哪个版本创建分支；把哪个分支的从哪版本到哪个版本范围内的变更合并到了哪个分支的哪个版本，合并后的版本号。这些信息有一些是版本控制工具本身可以很方便查找到的，就可以省略  4  操作步骤  在代码库中创建trunk、branches、tags目录，分别为主干、分支和标记，这样的布局是为了更清晰的区别主线、分支和标记三者的位置。在主干上提交代码，到可发布的程度时，创建分支。  为便于比较结果，我们在主干中上传一个文件readme.txt（版本为659）：  4.1  创建分支（标记）  将主干trunk签出（checkout）到本地，在本地checkout的trunk目录上单击鼠标右键，在弹出菜单中选择“TortoiseSVN” →“Branch/tag…”  在下图弹出的窗口中，将“To URL” 指向branches目录并输入分支的具体目录名。默认的目标URL将会是你当前工作拷贝所处的源URL，必须给分支/标记编辑一个新路径。SVN不会自动递归创建目录，要自己先创建好父目录。比如想创建分支/branches/V1.0，那么V1.0可以不用自己创建，但是/branches要先创建好。这里是branches/V1.0，我们即将创建的分支便存放于此处，点击OK  上图中红色方框内Create copy revision in the repository下的选项：  u  HEAD revision in the repository：拷贝当前主干中的最新版本。不需要从你的工作副本中传输任何数据，这个分支的建立是非常快的。  u  Specific revision in repository：拷贝主干中的某个指定版本。假如你在上周发布了项目时忘记了做标记，这将非常有用。如果记不起来版本号，通过点击鼠标右键来显示版本日志，同时从这里选取版本号。和上次一样不需要从你的工作副本中传输任何数据，这个分支建立起来是非常快的。  u  Working copy：新的分支是一个完全等同于你的本地工作副本的一个拷贝。如果你更新了一些文件到你的工作副本的某个旧版本里，或者你在本地做出了修改，这些改变将准确无误地进入拷贝中。自然而然地这种综合的标记会包含正在从工作副本传输到版本库中的数据，如果这些数据还不存在的话。  选择完毕后单击【OK】按钮，则分支创建完毕。再次查看配置库，可以看到刚才创建的分支中包括主干中的文档“readme.txt”，版本为659，同主干一致。  标记的创建方法同分支一样，都是对主干的拷贝操作（实际是对某一版本的链接）。  4.2  合并分支  分支用来维护独立的开发支线，在一些阶段，你可能需要将分支上的修改合并到最新版本，或者将最新版本的修改合并到分支  为便于比较结果，我们修改分支中的readme文件（此时版本为664），同时添加一个文件：  如果想将分支合并到主干上，在本地checkout出的主干（trunk）目录上单击鼠标右键，在弹出菜单中选择“TortoisesSVN”→“Merge”  在弹出的“Merge”菜单中选择类别：  在“URL to merge form”输入框中选择分支的URL，在“Reverse range to merge”填入版本，可点击【show log】按钮选择需要合并的版本。需要注意的是Merge并非字面上所示的将两个分支归并到一起，而是diff-and-apply的意思，比较两个分支的差异并归并差异。输入完毕后单击【Next】：  选择合并选项后（如“Compare whitespaces”），单击【Merge】，完成合并操作。  如果在合并过程中发生冲突，SVN会进行提示：  进行合并后，在本地的trunk目录会显示以下文件：  冲突的文件图标中会有一个叹号，同时系统自动生成3个文件：  u  readme.txt为合并前主干中的版本  u  readme.txt.merge-left.r.664：为664版本，即创建分支时主干中的版本  u  readme.txt.merge-right.r665：为665版本，即合并前分支中的版本  可以直接打开文件进行手动修改，冲突的内容会以议<<<<<<<…………>>>>>>>标识  也可以选中该文件，右键→TortoiseSVN→Edit conflicts，TortoiseMerge窗口会显示冲突文件对比，可以在merged对话框中进行编辑：  修改完毕后，右键→TortoiseSVN→Resolved，此时系统自动生成的3个文件会自动删除，冲突文件的图标会变为未提交状态，右键→SVN commit，提交到配置库。  当有多个文件conflict时，需要逐个resolve。  如果合并后的内容不满意，可以通过撤销来取消这次的合并操作，前提是未对合并后的文件做提交操作。  总结如下：  如果是需要将分支的改动合并到主线上，需要在主线的工作副本下进行合并，合并的范围是需要从分支上上次合并的版本到当前分支上最新的版本，如果主线和分支都修改了相同的文件，合并后会出现冲突，然后解决冲突，提交，如果是第一次合并，则起始版本号是上次建立分支的版本号；  相反，如果是需要将主线的改动合并到分支上，需要在分支的工作副本下进行合并，合并的范围是需要从主线上上次合并的版本到当前主线上最新的版本，合并后会出现冲突（冲突的前提如上种情况），然后解决冲突，如果主线修改但是分支没有修改，则主线上合并的变更内容会增加到当前副本中，提交，如果是第一次合并，则起始版本号是上次建立分支的版本号  合并的工作是把主线或者分支上合并范围内的所有改动列出，并对比当前副本的内容，由合并者手工修改冲突。如果当前工作副本是主线的，则合并的范围是分支上的改动，如果工作副本是分支的，则合并范围是主线上的改动。  经常有人会说，树冲突是很难解决的一类冲突，其实一旦了解了其原理，要解决也不难。先回顾下对于树冲突的定义。      树冲突：当一名开发人员移动、重命名、删除一个文件或文件夹，而另一名开发人员也对它们进行了移动、重命名、删除或者仅仅是修改时就会发生树冲突。  出现冲突时，一般会提示冲突的信息是什么。过后我们可以使用svn st来查看当前状态。svn st的各种状态代表什么，请参考此博文svn st状态详解。  先介绍一下概念  Delete ： 其中目录结构变化，都认为是Delete  Edit： 是指修改文件  Local ： 是你本地修改  Incoming ：是别人修改，你要Update或Merge进来。  这样应该有4个组合，但是Edit对Edit的组合应该是File Conflict，这个容易解决，不在Tree Conflict 讨论范围，所以有3种组合。再需要区别Update和Merge，就有了6种情况。分别是  Local delete, incoming edit upon update  Local edit, incoming delete upon update  Local delete, incoming delete upon update  Local missing, incoming edit upon merge  Local edit, incoming delete upon merge  Local delete, incoming delete upon merge  分别对这几种情形解释如下：  1.Local delete, incoming edit upon update（本地删除，更新后传入修改）  产生原因：1.A修改文件Foo.c后提交到版本库中，B将Foo.c重命名为Bar.c或者删除了Foo.c或者直接将Foo.c的父目录Foo直接删除 2.B更新工作副本会提示该冲突，在working copy显示为Foo.c在本地删除，被标记为冲突。如果是重命名，则Bar.c被标记为新增，但是不包括A的修改。  解决：A与B要确认是否采用A的修改与是否重命名。如果采用A的修改，并且要重命名则修改后，标记冲突解决，svn resolved，最后提交；如果不采用A的修改，直接标记冲突解决提交即可。  2.Local edit, incoming delete upon update （本地编辑，更新后传入删除）  产生原因：1.A对Foo.c重命名为Bar.c并提交到版本库（或者A将Foo.c的上级目录Foo修改为Bar），B在他的工作副本中对Foo.c进行修改。2.B提交前更新，会提示如此错误。  解决：同样需要两个人进行协商后修改。  3.Local delete, incoming delete upon update （本地删除，更新后传入删除）  产生原因：1.A将Foo.c重命名为Bar.c后提交，B对Foo.c重命名为Bix.c。2.B更新本地工作副本是会提示该树冲突。  解决：通过日志查找文件被删除即重命名的原因，A与B协商后最终确认采用哪个名称。  4.Local missing, incoming edit upon merge （本地丢失，合并后传入修改）  产生原因：1.A在主干上修改Foo.c，B在分支上将Foo.c重命名为Bar.c。2.B合并A在主干上的修改。  解决：B先标记冲突解决，然后将Foo.c拷贝至本地，将A的修改合并至自己的文件中或者直接放弃A的修改，采用自己的修改。  5.Local edit, incoming delete upon merge （本地修改，合并后传入删除）  产生原因：1.A将Foo.c重命名为Bar.c(或者将Foo.c的父目录Foo改为Bar)，B在分支上修改Foo.c。2.B合并A的修改时提示该冲突。Bar.c被标记为增加，Foo.c被标记为冲突。  解决：同样根据日志查找到修改的源头，两人协商后解决。  6.Local delete, incoming delete upon merge （本地删除，合并后传入删除）  产生原因：1.A在主干上将Foo.c重命名为Bar.c，B在分支上将Foo.c重命名为Bix.c。2.B合并A的修改时会提示冲突。重命名后的文件被标记为新增，原来文件被标记为树冲突。  解决：通过日志查找到文件被改名的时刻，两人协商后解决。  63:什么是线程安全问题?  答:当多个线程共享同一个全局变量,做写的时候,可能会受到其他线程的干扰,导致数据有问题,这种现象叫做线程安全问题。做读的时候，不会产生线程安全问题。  多个线程共享同一个局部变量，做写的操作，不会产生线程安全问题。  64：怎样解决线程安全问题？  利用同步代码块保证线程之间同步，保证数据的原子性（数据不能受到其他线程的干扰）。多个线程想同步，必须用同一把锁。 |
| **人资问题(至少30个)** |
| 问题1：你上一家单位的离职原因?  回答：因为我在上一家单位我们的项目一期已经做得差不多了,我们组的老大说留下几位技术骨干就ok了。我想了一下与我的职业发展规划不符，希望获得一份更好地工作。对自己要求比较高然后就选择离职。  回答2：你对加班的看法？  回答：在我看来加班分为两种,一种是我没有完成领导交代给我的任务,自己自愿加班,完成自己的工作进度。从而迎接明天新的任务。在有另外一种是在我们这一行加班是不可避免的，如果公司在需要我在工作上加班时我会义不容辞的,因为这样可以更好地提升自己,利于我的职业成长。  回答3：简单做一下自我介绍？  面试官您好，我叫王龙，今年26岁，来自于山西临汾，今天来咱们公司是来应聘JAVA工程师职位的。我毕业后一直从事于JAVA开发工作到现在已经3年了。我在上一家公司做的是JAVA开发工程师，主要的工作职责是对项目模块的设计开发，做过社交类的交友项目和广告类的有赞广告平台项目。尤其是社交项目，我们利用当前最流行的技术进行开发,实现前后端分离。我在项目中主要负责用户模块，点赞模块，和交友模块。以上就是我简单的自我介绍。  回答4：说说你最大的缺点?  我觉得我有时候会过分在意别人的感受, 比方说, 不敢直接表达不同意见, 因为觉得会让对方丢面子,其实这样做很不利于快速有效地开展工作。我希望自己能够逐渐学会更加爽快, 对人对事更加直接。  回答5：说说你最大的优点?  对于工作有一份执着，如果没有完成交代的任务时，会不放弃的将它完成，然后总结工作当中出现问题的。以便下次能够更好、更快的完成。规划自己小的目标，  以便自己成长。  回答6：你对薪资的要求?  首先我对工资没有硬性要求，我看重的是找对工作的机会，只要能够很好地发展，我相信贵公司在处理我的问题上会友善合理，所以只要条件公平，我会坦然接受的。  回答7：如果通过这次面试我们录用了你，但工作一段时间却发现你根本不适合这个职位，你怎么办?  如果通过的我个人的努力无法胜任该工作的话，我会要求贵公司再给我几天时间，我会更加努力向领导和同事学习业务知识和处事经验。力争减少差距，尽快得到公司领导的认可。如果通过我的努力还是无法达到岗位的要求，我会向贵公司申请调换工作，如果还是没有合适的工作调动的情况下，我会选择主动离职。  回答8：谈谈你对跳槽的看法?  跳槽对于每一个人来讲，都有不同的想法和看法，要是频繁跳槽的话，对于公司的印象是我的工作没有稳定性，此人不够稳重。而对于我来讲呢，跳槽是因为上一家单位的工作做到后期，对我的发展空间不是很大，与个人的职业生涯规划不相符，所以选择跳槽，当然也属于个人原因。  回答9：你的职业规划是什么?  我的职业规划就是首先确定自己的小目标，把公司的交代的任务完成好之后，利用其它的一些时间去有关工作技术方面的书籍，然后去学习了解一些新的技术,确认自己的发展目标,坚持不懈的走下去。  回答10：工作中难以和同事、上司相处，你该怎么办？  克服自己服从领导的指挥，配合同事的工作。从自己身上找原因，看看哪里做的不合适,努力改正。努力与他们沟通找到共通点然后很好地相处,把工作做到最好.  回答11：你对我们公司了解多少？  在去公司面试前,我了解了一些有关贵公司的主营业务,和近几年的贵公司业务类型.主要经营的业务范围.  回答12：你能为我们公司带来什么呢？  如果贵公司录用我以后,我会让公司减少用人的成本,而在利益上获取更好的利益,当然在此期间我还会继续学习技术和其他方面的一些知识,为公司以后的发展努力.  回答 13:你欣赏哪种性格的人?  我个人比较欣赏有能力,而且幽默的人,这种性格的人给我带来的感觉非常随和,让同事或者朋友更容易接触,与这种人共事不仅心情愉悦,而且会让我们在工作上有很高的效率.  回答14:怎样对待自己的失败?  首先失败是不可避免的,失败是成功之母,最重要的是调整好自己的心态 ,寻找自己失败的原因,弥补失败中的不足,正确认识问题的重要性,改正自己的缺点,争取不要让自己在同一个地方摔倒两次,不断去完善自己.  回答问题15:最能概括你自己的三个词是什么?  有责任心和做事有始终,适应力强.  回答问题16:.假如家事与公事冲突，你如何解决？  首先看待事情的重要性,如果公事重要的话我会选择留下继续工作.如果家事发生一些状况的话我会向领导报备一下,完了将手头的工作进行一个简单的安排和交接,完了将一些可能出现的问题向领导说明清楚,然后对可能出现的问题进行关注,然后在处理自己的事情.  回答问题17:你通常如何对待别人的批评?  首先从别人眼里看自己,对自己是一种成长.别人就好比一面镜子,会告诉我们的好与坏.在别人眼中好的自己,继续做好自己不骄傲,如果因为我自己而影响到别人的话,我会努力听从别人的意见,努力改正自己的缺点.  回答问题18:.你工作经验欠缺，如何能胜任这项工作？  如果自身工作经验欠缺影响公司的话,我会抽空向别人请教一些经验上的问题,积极学习强化自身的工作能力.避免耽误工作项目的进度.  回答问题19:你希望与什么样的上级共事?  社会上有各种形形色色的人,没有好坏之分,能与不同的上级共事,我认为是一件非常荣幸的事情,只要大家都是为了很好地工作,即使工作当中有一点小的摩擦还是可以接受的,大家都是为了工作嘛.  回答问题20:如果你这次面试失败，你怎么打算？  即使是面试失败了我认为没有什么可怕,因为既然我能在这能你诸位说出我的经历与项目我认为我还是成功的,至少我努力了,我会继续总结今天的面试的不足,然后改变自己,加强自己的各方面能力.同时我也会祝愿贵公司在今后的发展中蒸蒸日上,同时也希望你们能尽快为自己的公司招揽更优秀的技术人才.  回答问题21:谈谈如何适应办公室工作的新环境  对于新的环境我会与周围的同事先熟悉起来,了解自己岗位的职责,同时还会根据领导指示和工作安排，制定工作计划，提前预备，并按计划完成。多请示并及时汇报，遇到不明白的要虚心请教。抓间隙时间，多学习，努力让自己能够尽快的融入这个团队当中.  回答问题21:你是怎样安排自己的周末时间的?  对于周末时间我会看一些技术方面的书籍,浏览一些博客和网页,如果还有时间的话和同事朋友一起出去吃个饭,交流一些技术.总之合理安排自己的时间,让自己的周末过得更有意义.  回答问题22:.喜欢这份工作的哪一点？  首先我是一个爱研究各种奇奇怪怪东西的人,所以我对这份工作的新技术比较会感兴趣一些,即使它比较难一些,但是我还是喜欢钻研下午,有一种不达目的不罢休的精神,估计这就是我喜欢这份工作的原因吧.  回答问题:23你还有什么问题要问吗?  咱们公司有新员工培训吗?对于这个岗位有什么样的发展空间吗?如果我通过面试我需要看关于哪些方面框架的书籍,能够尽快的融入这个集体当中?咱们公司现在所做项目的背景有哪些?  回答问题:24:说说你对行业、技术发展趋势的看法？  现在是互联网时代,新技术日新月异,每天都会有新的技术出现,而对于我们这个行业缺的就是技术人才,所以我们行业无论是技术还是技术性的人才都会有一个很好地发展.只要我们继续不放弃技术的学习,我们是不会被这个技术大圈所淘汰的.  回答问题25:除了本公司外，还应聘了哪些公司?  首先我是对这份工作是比较喜欢的,所以我面试的还是一些互联网公司  回答问题26:对工作的期望与目标是什么？  我的目标就是让自己的技术更上一层楼,将公司的项目做好,为公司解决技术上的问题,做出好的业绩让公司认可我,为了达到这个目标我会更加努力好好学习技术.  回答问题27:如果在工作中出现失误,给公司造成损失,你认为该怎么办?  我本是为公司努力工作的,如果造成经济损失首先我会选择弥补或者挽回经济损失,如果无能为力的话希望得到单位的支持.再就是分清楚职责,如果我的责任我会甘愿受罚,如果是别人的话我会尽可能的帮助他一起弥补,不幸灾乐祸.毕竟我们属于一个团体.  回答问题28 什么会让自己更有成就感?  为贵公司竭力效劳,尽我所能,完成项目.当我们做出项目完成需求时我认为在那时会有成就感.  回答问题29  眼下你生活中最重要的是什么?  对于我来说能够在这个领域找到自己喜欢的工作是最重要的,当然能在贵公司入职我认为也是最重要的.  回答问题30 为了做好你工作份外之事,你该怎样获得他人额的支持与帮助?  每个公司都在不断变化的发展的过程中,公司也希望员工也是如此,所以我会适应公司的变化,并且对变化做出积极地响应.  回答问题 31工作当中你学习到了什么?  在工作之内我学会了一些技术和经验的积累,工作之外我学会了与人交流沟通,如何与别人友好的相处,如何能与自己的同事交流,对于工作怎样合理安排才能提高工作效率.  ........ |