

联想面试题

-刘博文整理

1.接触过的 Java 算法有哪些?

答: 1.冒泡排序

通过循环的方式,比较相邻的元素,如果第一个比第二个大,就进行交换,直到没有需要再交换的数字,这样越小的元素就经过交换慢慢的到数组顶端。

```
public static void main(String[] args) {  
    //1.定义要排序的数组  
    int[] x = {94,36,78,9,13,47,33,144,3,19};  
    //2.算法逻辑  
    for (int i = 1; i < x.length; i++) {  
        for (int j = 0; j < x.length-i; j++) {  
            if (x[j] > x[j+1]) {  
                int temp;  
                temp = x[j];  
                x[j] = x[j+1];  
                x[j+1] = temp;  
            }  
        }  
    }  
}
```

联想面试题

-刘博文整理

2.快速排序

通过数列中的基准值进行排序,将一个数列分成2个子序列。

```
public static void main(String[] args) {
    //1.定义要排序的数组
    int[] x = {94,36,78,9,13,47,33,144,3,19,78,134,29,30,88,67,311};
    int[] quickSort = quickSort(x, 0, x.length-1);
    for (Integer integer : quickSort) {
        System.out.println(integer);
    }
}

public static int[] quickSort(int[] x,int start,int end){
    int i,j;
    i = start;
    j = end;
    if ((x == null) || (x.length == 0)) {
        return null;
    }
    while(i < j){
        while(i < j && x[i] <= x[j])
            j--;
        if (i < j) {
            int temp = x[i];
            x[i] = x[j];
            x[j] = temp;
        }
        while (i < j && x[i] < x[j])
            i++;
        if (i < j) {
            int temp = x[i];
            x[i] = x[j];
            x[j] = temp;
        }
    }
    if (i - start > 1){
        quickSort(x, 0, i-1);
    }
    if (end - j > 1) {
        quickSort(x, j+1, end);
    }
    return x;
}
```

联想面试题

-刘博文整理

3.选择排序

找到最小元素,然后放到序列末尾,以此类推,直到排序完毕。

```
public static void main(String[] args) {
    //1.定义要排序的数组
    int[] x =
{94,36,78,9,13,47,33,144,3,19,78,134,29,30,88,67,311,31,59,81,189,151};
    //2.排序逻辑
    for (int i = 0; i < x.length; i++) {
        int k = i;
        for (int j = k + 1; j < x.length; j++) {
            if (x[j] < x[k]) {
                k = j;
            }
        }
        if (i != k) {
            int temp = x[i];
            x[i] = x[k];
            x[k] = temp;
        }
    }
}
```

4.插入排序

队列中的第一个元素取出,构建新的队列认为已经排序,取出未排序的队列中的下一个元素,在已经排序的序列中从后向前扫描,如果已排序的元素大于新的元素,将该元素移动到下一个位置,直到已排序的元素小于或者等于新元素的位置,将新元素插入到该位置中。

```
public static void main(String[] args) {
    int[] x =
{94,36,78,9,13,47,33,144,3,19,78,134,29,30,88,67,311,31,59,81,189,151};
    //2.排序逻辑
    int i,j,target;
    int n = x.length;
    for (i = 1; i < n; i++) {
        j = i;
        target = x[i];
        while(j > 0 && target < x[j-1]){
            x[j] = x[j-1];
            j--;
        }
        x[j] = target;
    }
}
```

联想面试题

-刘博文整理

2.String & StringBuffer & StringBuilder 的区别?

答: 1.可变与否?

String 类使用字符数组保存字符串,有 final 修饰符,所以 String 对象是不可变的。

StringBuilder 和 StringBuffer 都继承 AbstractStringBuilder 类,在 AbstractStringBuilder 类中也是使用字符数组保存字符串的,但是是可变的。

2.是否多线程安全?

String 中的对象是不可变的,所以是线程安全的。

StringBuffer 对方法或者调用的方法加了同步锁,是线程安全的。

StringBuilder 并没有对方法加锁,所以是非线程安全的。

3.StringBuffer 和 StringBuilder 的共同点?

StringBuffer 和 StringBuilder 都是调用 AbstractStringBuilder 中的公共方法,但是 StringBuffer 会在方法上加 Synchronized 关键字进行同步。

所以如果程序不是多线程的,那么使用 StringBuilder 效率高于 StringBuffer。

3.关于 dubbo 的一些问题?

答: 1.dubbo 的服务开发流程?

dubbo 采用全 Spring 配置方式,只需要在 Spring 中加载 dubbo 即可。

2.dubbo 的暴露服务的流程?

```
<dubbo:application name="工程名"/>
```

```
    <dubbo:registry protocol="zookeeper"
```

```
        address="zookeeper 地址:端口" />
```

用 dubbo 协议在 20880 端口暴露服务

```
<dubbo:protocol name="dubbo" port="20880" />
```

声明要暴露的服务接口

```
<dubbo:reference interface="暴露的工程地址" id="类名[首字母小写]" timeout="配置连接时间" >
```

3.dubbo 引用服务的流程?

使用 dubbo 引用服务

```
<dubbo:application name="项目名称"/>
```

加载 zookeeper 注册中心

```
<dubbo registry protocol="zookeeper" address="X.X.X.X:2181"/>
```

使用 dubbo 引用服务

```
<dubbo:reference interface="引用的工程地址" id="类名[首字母小写]" timeout="配置连接时间" >
```

4.dubbo 是如何服务的?

(1)服务提供者启动,根据协议信息绑定到配置好的 Ip 和端口上,如果已经绑定则跳过。

(2)注册服务信息到注册中心。

(3)客户端启动,根据接口的信息和协议订阅注册中心的服务,注册中心将存活的服务地址通知到客户端。

(4)在客户端需要调用服务时,从内存中拿到上次通知的所有存活的服务地址,根据路由信息和负载均衡选择

联想面试题

-刘博文整理

最终的调用服务地址,发起调用。

5.dubbo 为什么使用 zookeeper 注册中心?

dubbo 是一个分布式服务框架,如果没有分布式需求,其实是不需要的,只有在分布式的时候才有对 dubbo 的需求,本质上就是一个服务调用的框架。如果一个消费者需要调用服务,我的所有服务器都提供相同的服务,但是如果让消费者去选择一个调用的话要么 F5 要么在消费者这边加代码逻辑进行判断用来达到负载均衡的效果,还有每次调用的时候都要查询当前有哪些服务提供者,是很耗开销的。

如果在消费者和服务者之间加一层服务路由,由服务路由提供功能和代理服务的地址,代理地址是稳定的,这样就算服务者提供者地址改变了,消费者直接调用代理服务器地址给服务路由,让路由自己去查询,减小开销,zookeeper 就扮演一个服务路由的角色。

4.关于 Redis 的一些问题?

1.Redis 为什么可以做缓存?

在一些项目中,大量的访问关系型数据库进行 CRUD 操作,会造成系统资源占用过大,Redis 的 Key-Value 的结构比关系型数据库要快,而且单线程的 Redis 并不需要做太大的控制,可以存放一些读写量大,不适合存储在关系型数据库中的数据。

2.Redis 如何使用?

因为 Redis 是使用 C 语言编写,所以必须在 C 语言的环境下运行,在 Linux 中使用 `yum install gcc-c++` 可以下载 C 语言环境。

3.Redis 应用场合?

取最新的 N 个数据的操作,实时取得 TopN 的操作,需要精确到毫秒的操作,取得某段时间内所有数据的操作,构建队列系统,缓存。

4.Redis 可以用来存储什么?

取最新的 N 个数据的操作,实时取得 TopN 的操作,需要精确到毫秒的操作,取得某段时间内所有数据的操作,构建队列系统,缓存。

5.Redis 的存储结构?

非关系型数据库,使用 key-value 方式存储。

6.Redis 集群的搭建方法?

Redis 最少需要三个节点,为了保证集群的高可用性,需要每一个节点有一个备份机,Redis 集群至少需要 6 台机器,可以使用 Ruby 脚本搭建。

```
yum install ruby
```

```
yum install rubygems
```

安装好过后需要去每台 Redis 服务器上修改 `redis.conf` 文件,需要把支持集群的配置文件更改为 Yes。

```
cluster-enabled Yes
```

然后使用 Ruby 脚本进行配置,搭建集群,把每台服务器的 Redis 地址和端口号进行配置。

7.Redis 的事务?

Redis 因为是单线程实现的,所以事务比较简单,给客户端发送 MULTI 命令后,服务器返回 Ok。然后再依次输入本次事务中所需要的操作,每次输入一个命令服务器不是马上返回 Ok,而是返回 QUEUED 命令,这代表服务器接收了本次命令并且暂时保存起来,当输入 EXEC 命令的时候,本次事务中的所有命令才会依次执行。

如果本次事务中有一个命令错误,那么其他的命令全部不执行,如果客户机在发送 EXEC 命令之前掉线了,那么服务器会清空队列,命令全部不执行,如果客户机在发送了 EXEC 命令之后掉线了,那么之前的事务全部会执行。

联想面试题

-刘博文整理

8.Redis 数据丢失怎么办?

- ①首先需要做的就是止损,从主存储器中导入数据,避免损失。
- ②查看 `expired_keys`,查看是否为过期数据。
- ③查看 `Maxmemory` 是否达到 **100%**,查看数据是否被强行淘汰。
- ④排除是否 `flushall`,删除了所有数据库,确认是否为程序或者是人为操作。
- ⑤查看 `comdstat_del`,是否为程序大批量删除。
- ⑥查看慢日志,查看开始丢失数据的时间后的所有操作,判断问题所在。

9.Redis 的数据类型?

Redis 包含五中数据类型,分别为:

- (1)**String** 字符串 **key-value** 类型
- (2)**Hash** 字典 可以将一些结构化的信息打包成 **HashMap**,如果需要修改就取出,进行反序列化,然后修改之后再存储回去。
- (3)**List** 列表,队列信息。
- (4)**Set** 集合[无序不重复] 判断交集,并集,差集等。
- (5)**Sorted Set** 集合[有序不重复] 通过添加一个权重参数 **Score**,使得元素按照 **Score** 进行有序排序。

5.关于 Cookie 和 Session 的区别?

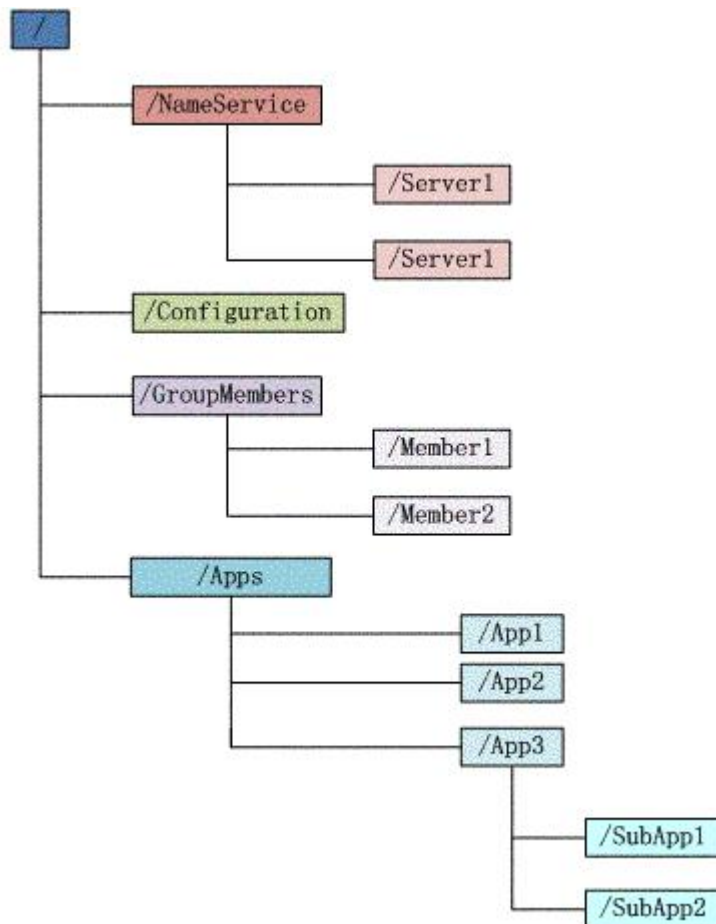
Cookie 是客户端保持状态的方法,**Session** 是服务器保持状态的方法。

Session 是基于 **Cookie** 进行信息处理的。

Cookie 是不安全的,别人可以通过分析本地的 **Cookie** 进行 **Cookie** 欺骗,考虑到安全问题应该使用 **Session**。

Session 会在一段时间内保存在服务器上,有时效性。

6.关于 ZooKeeper 数据结构的问题?



在 ZooKeeper 的数据结构中,每一个节点都称为一个 znode,每个 znode 由三部分组成。

Stat,此为状态信息,描述该 znode 的版本,权限等信息。

Data,与该 znode 关联的数据。

Children,该 znode 下的子节点。

Znode 可以被监控,包括这个目录节点中存储的数据被修改,子节点目录的变化等,一旦变化可以通知设置监控的客户端,这个是 ZooKeeper 的核心特征。

7.SpringMVC 的调用流程?

1.Http 请求到来,DispatcherServlet 负责请求分发,在分发之前需要借助 Spring 提供的 HandlerMapping 定位到具体的 Controller。

2.Spring Controller 将处理 DispatcherServlet 的请求,类似于 Struts 的 Action,Controller 可以处理 HttpServletRequest 和 HttpServletResponse 的请求。

3.接收到请求之后,通过后台逻辑的 Service 和 Dao,将 ModelAndView 对象返回给 DispatcherServlet 前端控制器。

联想面试题

-刘博文整理

8. 各种 MQ(消息中间件)的应用场景有什么?

1.MQ 消息中间件是分布系统的重要的组件,主要解决应用耦合,异步消息,流量削峰等问题,实现高性能,高可用,可伸缩和最终一致性架构。

2.应用场景有:

(异步处理)注册 -- >

当用户注册的时候,将注册信息写入数据库,需要发送注册邮件和注册短信。如果使用串行的方式,将数据写入数据库-->发送邮件-->发送短信,以上三个任务都完成过后,返回给客户端。但是如果使用并行的方式将数据写入数据库-->发送邮件

-->发送短信 以上三个任务完成之后,返回给客户端。可以缩短处理消息的时间。

(应用解耦)电商系统中,用户下单之后,订单系统需要通知库存系统,如果库存系统无法访问,那么 订单减库存则会失败,从而导致订单失败。

如果使用消息队列,那么在订单系统完成持久化处理之后,将消息写入消息队列,返回用户订单下单成功。

库存系统订阅下单的消息,获取下单信息,再根据信息进行库存操作。

即使下单时库存系统不能正常使用,也不会影响下单,因为下单之后,订单系统写入消息队列之后就不需要关系后续操作,实现了订单系统与库存系统的解耦。

(流量削峰)秒杀活动中,一般会因为流量过大,导致流量暴增,应用挂掉,所以为了解决这个问题,一般需要在应用前端加入消息队列。

用户请求被服务器接收之后,首先写入消息队列,加入队列长度超过最大值,则抛弃用户请求或者跳转到错误页面,然后秒杀业务根据队列中的消息请求再做后续的处理。

(日志处理)日志采集客户端,负责日志数据的采集,定时写入 Kafka 队列,Kafka 队列负责日志数据的接收,存储和转发,日志处理应用负责订阅并消费 Kafka 队列中的日志数据。

9.web.xml 的配置有哪些,为什么要这么配置,每个监听器的具体作用?

1.Web.xml 自带的 Schema 配置,其余的元素都放在<web-app>中。

2.上下文参数,声明应用范围的初始化参数。<context-param>

3.过滤器配置,将一个名字与一个实现 javax.servlet.Filter 接口的类相关类。<filter>

4.监听器配置<listener>

5.Servlet 配置<servlet>

6.会话超时配置<session-config>

7.MIME 类型配置<mime-mapping>

8.指定欢迎文件页面配置<welcome-file-list>

9.错误页面配置<error-page> -- > <error-code> 通过错误码进行配置
--><exception-type> 通过异常的类型进行配置

10.TLD 配置<taglib>

11.资源管理对象配置<resource-env-ref>

12.资源工厂配置<resource-ref>

13.安全限制配置<security-constraint>

14.登录验证配置<login-config>

15.Spring & Stauts 配置。

16.Socket 和 Netty 的具体流程有什么区别?

我需要想想。

17.nio 和 oio 分别是什么,有什么区别?(理解可能有些偏差)

nio:非阻塞式高伸缩性网络,

oio:阻塞式网络。

区别: oio 中,每个线程只能处理一个 channel(同步的,该线程与该 channel 绑定),线程发起 IO 请求,不管内核是否准备好 IO 操作,从发起请求起,线程一直阻塞,直到操作完成。

nio 中,每个线程可以处理多个 channel(异步)。

18.MySql 的数据引擎有哪几种,每一种的优缺点是什么?

1.Innodb 引擎,是一个事务性的存储引擎,有行级锁定和外键约束。

优:适合处理多重并发的更新请求,支持事务,容灾,支持外键约束,支持自动增加列属性。

缺:不支持 FULLTEXT 索引,不保存表的具体行数,对于 AUTO_INCREMENT 类型的字段,Innodb 中必须包含该字段的索引。删除表时,不会重新建立表,而是一行一行的删除。

2.Federated 引擎,是访问 MySql 服务器的一个代理,默认是禁用的。

3.Archive 引擎,归档引擎,仅仅支持插入和查询两种功能。

优:有很好的压缩机制,可以存储大量的独立的历史记录的数据,拥有很高的插入速度。

缺:虽然可以查询,但是查询速度较慢,而且不支持更新和删除操作。

4.CSV 引擎,可以将 CSV 文件作为 MySql 的表来使用的引擎。

优:可以将 CSV 文件作为 MySql 的表来使用。

缺:不支持索引,CSV 引擎表的所有字段必须是非空的。

5.Performance_Schema 引擎,主要是用来收集数据库服务器的性能参数,MySql 用户不能创建存储该类型的表。

6.BlackHole 引擎,黑洞引擎,会丢弃所有插入的数据,服务器会记录下 BlackHole 表的日志,所以可以用来复制数据到备份数据库。

7.Memory 引擎,Memory 引擎采取的逻辑介质是内存,响应速度是非常快的,但是当 MySql 守护进程崩溃的时候数据会丢失。

优:目标数据比较小,而且非常频繁的访问,在内存中存放数据,响应速度很快。

支持散列索引和 B 树索引。

缺:如果数据量太大会造成内存溢出,

要求存储的数据是长度不变的格式。

8.Mrg_Myisam 引擎,Merge 存储引擎,是一组 Myisam 的组合,意思就是它将 Myisam 引擎的多个表聚合起来,但是内部没有数据,真正的数据存储 Myisam 引擎的表中,可以直接进行查询删除更新等操作。

联想面试题

-刘博文整理

9. Myisam引擎, Myisam存储引擎独立于操作系统, 可以在Windows上使用也可以将比较简单的数据转移到Linux操作系统上去, 在创建表的时候, 会创建三个文件, .frm文件用于存储表的定义, .myd文件用于存储表的数据, .myi文件存储的是索引, 存储的是索引。

优: 查询速度很快, 如果数据库表中 Insert 和 Update 的操作比较多的话比较适用, 对表进行加锁。

缺: 不支持事务, 但是可以在 Service 层进行控制, 不支持外键。

19. MySQL 的主从方式如何配置?

1. 确保主数据库和从数据库一模一样,
2. 在主数据库上创建同步账号,
3. 配置主数据库的 my.ini (Linux 中是 my.cnf)
4. 配置从数据库的 my.ini (Linux 中是 my.cnf)
5. 验证是否成功,
6. 测试同步数据。

20. 简述几种 Java 的设计模式?

1. 单例模式: 使内存中保持一个对象, 通过静态方法向外界提供这个类的实例。
2. 工厂模式: 统一提供实例对象的引用。
3. 建造模式: 一个对象的组成可能是由其他很多的对象一起组成的, 封装这些复杂性, 就可以使用建造模式。
4. 门面模式: 外部与一个子系统的通信必须通过一个统一的面门 (Facade) 对象进行, 这就是门面模式。
5. 策略模式: 当几个类中有相似的方法, 将其中通用的部分提取出来, 从而使拓展更容易。类似于 Md5Utils & MailUtils。

21. Java 中接口和抽象类有什么区别?

1. 抽象类和接口都不能直接实例化, 如果要实例化, 抽象类变量必须指向实现所有抽象方法的子类对象, 接口变量必须指向实现所有接口方法的类对象,
2. 抽象类要被子类继承, 抽象类要被类实现,
3. 接口只做方法的声明, 抽象类中可以做方法的声明也可以做方法的实现,
4. 接口里定义的变量只能是公共的静态的常量, 抽象类中的普通变量,
5. 抽象类里的抽象方法必须全部被子类实现, 如果子类不能实现父类的抽象方法, 那么子类只能是抽象类, 同样如果实现接口的时候不能全部实现接口方法, 那么该类也只能为抽象类,
6. 抽象方法只能声明, 不能实现, 抽象类里可以没有抽象方法, 但如果一个类里有抽象方法, 那么这个类必须是抽象类, 抽象要被实现, 所以不能是静态的, 也不能是私有的。
7. 接口可以继承接口, 并可以多继承接口, 但类只能单继承,
8. 抽象是重构的结果, 接口是设计的结果。

22. 单点登录是如何实现(SSO)?

1. 通过专门的 Redis 服务器模拟 Session,
2. 因为网站按照不同的业务区分了子域名, 用户操作的话可能跨越多个实例, 用 Session 的话, 就需要解决 Session 共享的问题, 所以用 Redis 模拟 Session, 用户从浏览器中将登录信息传到 Web Server 中处理, Web Server 首先对用户的信息进行验证, 当验证通过的时候, 对用户的客户端信息进行散列, 形成一个

联想面试题

-刘博文整理

Token 令牌,这个 Token 即为 Redis 中的 Key,同时需要将获取的内容组成 Value,根据需要进行选择格式,比如说 Json 格式,然后设置一个 expire_time 为过期时间,保存进 Redis,将生成的 Token 保存在客户端的 Cookie 中。

3.当客户需要验证登陆内容的时,客户端将 Cookie 传到 Web Server 中,Web Server 获取到 Cookie 中 Token 的值,判断 Token 是否存在于 Redis 中,若存在将 Redis 中保存的值返回到 Web Server 中进行处理,判断此次获取是否合法。[

①客户端的信息是否和生成的 Token 相匹配(Ip,浏览器信息等),

②Value 保存的内容是否和当前操作匹配。

]如果合法,将通过验证将正确的信息返回给用户,并且延长 expire_time 或者重置 expire_time。

23.什么是单例模式,如何实现单例模式的线程安全?

单例模式的意思就是在内存中保持一个对象,并且通过静态方法向外界提供这个类的实例。

```
Public class Singleton{
    Private static Singleton instance;
    Private Singleton(){}
    Public static synchronize Singleton getInstance(){
        if(instance == null){
            instance = new Singleton();
        }
        Return instance;
    }
}
```

24.MVC 模式是什么意思,分别代表什么?

MVC 是一种程序架构,包含 M(Model) V(View) C(Controller)

视图与用户进行交互,通过事件改变控制器,通过控制器导致模型改变,模型改变导致参数改变,然后视图通过获取改变的参数。

MVC 不是 Java 的设计思想也不是 JavaWeb 的设计思想,是所有面向对象语言中应该遵守的规范,通过各部分分工协调工作,导致耦合降低,提高维护性和拓展性。

25.线程的生命周期?

1.创建状态:生成线程对象的时候,没有调用该对象的 start 方法,这时线程处于创建状态。

2.就绪状态:调用了对象的 start 方法的时候,该线程的处于就绪状态,但是还未设置成当前线程或者从等待中回来过后也会处于就绪状态。

3.运行状态:线程称为当前运行线程,执行 run 方法。

4.阻塞状态:线程正在运行的时候被暂停,通常是为了等待某个时间的发生或者某个资源的就绪,sleep,wait 方法都可以造成阻塞状态。

5.死亡状态:当一个线程调用完成过后或者调用 stop 方法后,该线程就会死亡,对于已经死亡的线程,无法调用其的 start 方法。

26.关于单例模式的一些问题?

1.懒汉式[线程不安全]

```
Public class Singleton{
    Private static Singleton instance;
    Private Singleton(){}
    Public static Singleton getInstance(){
        if(instance == null){
            instance = new Singleton();
        }
        Return instance
    }
}
```

2.懒汉式[线程安全]

```
Public class Singleton{
    Private static Singleton instance;
    Private Singleton(){}
    Public static synchronize Singleton getInstance(){
        if(instance == null){
            instance = new Singleton();
        }
        Return instance;
    }
}
```

3.饿汉式

```
Public class Singleton{
    Private static Singleton instance = new Singleton();
    Private Singleton(){}
    Public static Singleton getSingleton(){
        Return instance;
    }
}
```

27.main 线程为什么使用 static 修饰?

main 是 public static,只是语法规定,是迁就 Java 的语法。(此问题的争议性太大)就像.class 文件的魔数一样,大家猜了很多原因,最后作者一句我随便选的数字而已。

28. Servlet 和 jsp 的区别?

- 1.jsp 第一次运行的时候会编译成 Servlet,驻留在内存中以供调用。
- 2.jsp 是 Web 开发技术,Servlet 是服务端运用的小程序,我们访问一个 jsp 页面的时候,服务器会将这个 jsp 页面转变成 Servlet 小程序运行得到结果后,反馈给用户的浏览器。
- 3.Servlet 相当于一个控制层再去调用相应的 JavaBean 处理数据,最后把结果返回给 jsp。
- 4.Servlet 主要用于转向,将请求转向到相应的 jsp 页面。
- 5.jsp 更多的是进行页面的显示,Servlet 更多的是处理业务。
- 6.Servlet 可以实现 jsp 的所有功能。
- 7.开发页面是 jsp 可以直接进行编写。
- 8.jsp 潜入 Java 代码,Servlet 嵌入 HTML 代码。
- 9.在一个标准的 MVC 架构中,Servlet 作为 Controller 接受用户请求并转发给相应的 Action 处理,JSP 作为 View 主要用来产生动态页面,EJB 作为 Model 实现你的业务代码。

29. Servlet 的生命周期?

- 1.初始化阶段,调用 `init()` 方法。
- 2.响应客户请求阶段,调用 `service()` 方法。
- 3.终止阶段,调用 `destroy()` 方法。

30. 线程同步的方法?

- 1.同步方法,既有 `synchronized` 关键字修饰的方法,由于每一个 Java 对象都有一个内置锁,当用此关键字修饰方法时,内置锁会保护整个方法。
- 2.同步代码块,既有 `synchronized` 关键字修饰的语句块,被关键字修饰的语句会自动加上内置锁,从而实现同步。
- 3.使用特殊域变量(`volatile`)实现线程同步。
- 4.使用重入锁实现线程同步。
- 5.使用局部变量实现线程同步。

31. 什么是 static?

- 1.被 `static` 关键字修饰的方法或者变量不需要依赖对象来进行访问,只要类被加载了,就可以通过类名去进行访问,`static` 可以用来修饰类的成员方法,类的成员变量,另外可以编写 `static` 代码块来优化程序性能。
- 2.`static` 方法,`static` 方法一般被称为静态方法,由于静态方法不依赖任何对象就可以进行访问,因此对于静态方法来说,是没有 `this` 的,因为它不依附于任何对象,所以在静态方法中不能访问类的非静态成员变量和费静态成员方法。
- 3.`static` 变量,即为静态变量,静态变量被所有的对象所共享,在内存中只有一个副本,它当仅当在类初次加载时会被初始化。
- 4.`static` 代码块,形成静态代码块用来优化程序性能,`static` 代码块可以置于类的任何地方,类中可以有多个 `static` 代码块,在类初次被加载的时候会按照 `static` 块的顺序来执行每个代码块并且只会执行一次。

32.什么是 final?

1. **final** 是一个 Java 中保留的关键字,可以声明成员变量,方法,类和本地变量,一旦你将引用声明为 **final**,就不能改变这个引用了。
2. **final** 修饰的变量是只读的。
3. **final** 修饰的方法是不可以被子类的方法重写的,**final** 方法比非 **final** 方法要快,因为在编译的时候已经静态绑定了,不需要在运行时再次绑定。
4. **final** 修饰的类通常功能是完整的,他们不能被继承。

33.FastDFS 的原理?

1. **FastDFS** 的概念:**FastFSD** 是一款 C 语言编写的开源的分布式文件系统,充分考虑了冗余备份,负载均衡,线性扩容等机制,并且注重高可用,高性能,使用 **FastFSD** 可以搭建一套高性能的文件服务器集群提供文件上传文件下载的等服务。
2. **FastDFS** 的架构:**FastFSD** 的架构包括 **Tracker Server** 和 **Storage Server**. 客户端请求 **Tracker Server** 进行文件的上传和下载,通过 **Tracker Server** 调用 **Storage Server** 进行文件的上传和下载. **Tracker Server** 的作用是负载均衡和调度,**Storage Server** 的作用的存储。
Tracker 可以实现集群,集群中每个 **Tracker** 的每个节点地位相等。
Storage 可以分为多个组,每个组保存不同的文件,每个组内部可以有多个成员,组员内部保存的内容是一样的,组员的地位相等。
3. **FastFSD** 的上传流程:
Client 请求 **Tracker Server**-->**Tracker Server** 查询可用的 **Storage**-->把 **Storage** 的 **Ip** 和端口返回给 **Client**-->**Client** 上传文件-->**Storage** 生成 **File_id**-->将上传内容写入到磁盘-->返回给 **Client** 路径信息和文件名-->**Client** 存储文件信息。
Storage 定时向 **Tracker Server** 上传状态信息。
[返回的 **File_id** 包括组名/虚拟磁盘路径/数据两级目录/文件名]

34.MyBatis 中的 ResultType 和 resultMap 的区别?

1. **ResultType**:当使用 **ResultType** 做 **Sql** 语句返回结果类型处理时,对于 **Sql** 语句查询出的字段在相应的 **pojo** 中必须有和它相同的字段对应,而且 **ResultType** 中的内容就是 **pojo** 在本项目中的位置。适合单表查询。
2. **ResultMap**:当使用 **ResultMap** 做 **Sql** 语句返回结果类型处理时,通常需要在 **mapper.xml** 中定于 **ResultMap** 进行 **Pojo** 和相应表字段的对应。

35.Spring 中常用的注解有哪些?

- @Component**:标准一个普通的 **spring Bean** 类。
- @Controller**:标注一个控制器组件类。
- @Service**:标注一个业务逻辑组件类。
- @Repository**:标注一个 **DAO** 组件类。

36.Spring 使用注解开发还是 xml 开发?

使用 XML 配置的优缺点:

- 1.XML 配置方式进一步降低了耦合,使得应用更加容易拓展,即使对配置文件进一步修改也不需要工程进行修改和重新编译。
- 2.在处理大的业务量的时候,用 XML 配置应该更加好一些,因为 XML 更加清晰的表达了各个对象之间的关系,各个业务类之间的调用,同时 Spring 相关配置也能一目了然。
- 缺:3.配置文件读取和解析需要花费一定的时间,配置文件过多的时候难以管理,无法对配置的正确性进行校验。增加了测试难度。

使用注解配置的优缺点:

- 1.在 class 文件中,可以降低维护成本,注解的配置机制很明显简单。
- 2.不需要第三方的解析工具,利用 Java 反射技术就可以完成任务。
- 3.编辑器可以验证正确性。
- 4.提高开发效率。
- 缺:5.如果需要对注解进行修改,那么需要重新编译整个工程。
- 6.业务类之间的关系不如 XML 配置那样容易把握。
- 7.如果在代码的注解太多,直接影响代码质量,对于代码的简洁有一定影响。

37.查看日志的命令是什么?

```
tail -f xxx.log
```

38.Redis 和 Memcached 的区别,为什么使用 Redis 不使用 Memacached?

- 1.Redis 和 Memcached 都是将数据存放在内存中,都是内存数据库,不过 memcached 还可以缓存其他的东
西,例如 图片/视频等。
- 2.Redis 不仅仅支持简单的 Key-Value,还可以提供 List,Set,Hash 等数据结构的存储。
- 3.Redis 在物理内存用完时,可以将一些很久没用到的 Value 写入到磁盘中,重新释放内存。
- 4.过期策略不同,memcached 在 set 时候就要指定,Redis 可以通过 expire_time 指定。
- 5.分布式不同,memcached 可以一主多从,Redis 可以一主多从也可以一主一从。
- 6.容灾程度不同,memcached 挂掉之后数据不可以恢复,Redis 数据丢失之后可以通过 aof 恢复。
- 7.Redis 支持数据的备份,即 master-slave 模式的数据备份。
- 8.应用场景不一样,Redis 作为 NoSql 数据库使用外,还能用作消息队列,数据堆栈和数据缓存
等,Memcached 适合缓存 Sql 语句,数据集,用户临时性数据,延迟查询数据和 Session 等。

39.Linux 中解压命令的区别?

```
1.tar -z(配合解压 gz)x(解开一个包文件)v(显示详细信息)f(表示使用归档文件)
```

40.什么是索引,哪些数据可以有索引,索引的意义是什么?

1.什么是索引?

索引是对数据库表中的一列或者多列的值进行排序的一种结构。

2.什么数据可以有索引?

应该考虑在 Where 和 Group By 上创建索引。

联想面试题

-刘博文整理

3.索引的意义是什么?

- (1)通过创建唯一性索引,可以保证数据库表中每一行数据的唯一性。
- (2)可以大大加快数据的检索速度,这也是创建索引的最主要的原因。
- (3)可以加速表和表之间的连接,特别是在实现数据的参考完整性方面特别有意义。
- (4)在使用分组和排序子句进行数据检索时,同样可以显著减少查询中分组和排序的时间。
- (5)通过使用索引,可以在查询的过程中,使用优化隐藏器,提高系统的性能。

41.查看日志前 50 行的命令,搜索日志关键字前 50 行或后 50 行的命令?

1.查询前 50 行


```
head -n 50 xxx.log
```

2.搜索日志关键字前 50 行与后 50 行

```
cat -n xxx.log|grep "关键字"获取到行号,cat -n xxx.log|tail -n +92|head -n 50
```

42.Sql 语句题?

题目:给一张表,表里有编号,姓名,班级和成绩,写一条 sql 语句查出各个班级的前五



id	name	score	class
1	王一	351	一班
2	王二	213	一班
3	王三	421	一班
4	王四	216	一班
5	王五	634	一班
6	王六	535	一班
7	王七	142	一班
8	王九	843	二班
9	王八	546	二班
10	刘大	224	二班
11	刘二	224	二班
12	刘三	111	二班
13	刘四	222	二班
14	老王八	333	二班
15	表哥	444	二班
16	赵本三	555	二班

```
SELECT a.id,a.name,a.class,a.score
```

```
FROM student a LEFT JOIN student b ON a.class=b.class AND  
a.score<b.score
```

```
GROUP BY a.id,a.name,a.class,a.score
```

```
HAVING COUNT(b.id)<5
```

```
ORDER BY a.class,a.score DESC;
```


43. 有哪些常见的异常?

1. `ClassCastException` (类转换异常)
2. `ClassNotFoundException` (空指针异常)
3. `IndexOutOfBoundsException` (数组越界异常)
4. `BufferOverflowException` (I/O 操作异常)

44. 连接 jdbc 的几个步骤分别是什么?

1. 注册驱动,
2. 获取连接对象,
3. 创建对象执行语句,
4. 处理结果集,
5. 释放资源

45. List 和 Map 具体有哪些实现类?

1. List

- a) --> `ArrayList` 数组结构, 相当于动态数组, 容量可以动态增加, 提供了 CRUD 功能, 可以通过元素序号快速获取元素, 实现了序列化接口, 对于查询和赋值来说 `ArrayList` 比较快。
- b) --> `LinkedList` 链表结构, 双向链表, 可以作为堆栈, 队列, 对于新增和删除来说 `LinkedList` 比较快。

2. Map

- a) --> `HashMap` `HashMap` 基于最新的 `Map` 类, 默认是不线程同步的, 需要使用额外的同步机制, `map Collection.synchronizeMap(Map m)` 方法解决。
- b) --> `HashTable` `HashTable` 是基于 Java 中旧的 `Dictionary` 类, 是线程同步的, 在多线程应用中不需要专门的操作就可以安全的使用。
- c) --> `LinkedHashMap`
- d) --> `TreeMap`

46. 什么是二叉树?

二叉树是每个节点最多有两个子树的树结构, 分为空二叉树, 根节点二叉树, 只有左子树, 只有右子树, 完全二叉树. 第 i 层有 $2^{(i-1)}$ 个节点. 第 K 层 $(2^K) - 1$ 个结点. 用途分为二叉查找树和二叉堆. 红白树是自平衡二叉树, 可以获得较高的查找性能。

47. 如何实现每个表每天只能插入一条语句, 如果再插入的话就会报错?

MySQL 的触发器机制。

联想面试题

-刘博文整理

48.==和.equals 的区别?

基本的数据类型,八大原始数据类型如果需要比较的话需要使用==方法进行比较。

复合数据类型在使用==方法进行比较的时候,比较的是他们在内存中存放的地址值.所有的类都是继承Object 的基类的,基类中的 equals 方法是比较地址值的,但是在 String,Data,Integer 等一些类中对 equals 方法进行了覆盖重写,有自身的实现方法而不是比较地址中的值。

49.关于 Maven 的一些问题?

1.Maven 常用的命令有哪些?

-->clean 清除编译,
-->package 打包,
-->install 安装到本地,
-->deploy 发布,上传到仓库,
-->compile 编译,
-->test 测试运行。

2.如何打成 War 包?

在配置文件中配置<packaging>war</packaging>
运行插件或者命令 mvn -package 。

3.如何上传 Jar 包到服务器?

配置文件 pom.xml 中配置 repository 为服务器信息,
setting.xml 配置文件中配置 server 配置,
执行 mvn -deploy 命令。

50.MyBatis 中 Sql 语句中包含%如何处理?

在 sql 语句中用 concat 命令连接或者用<![CDATA[.....]]>处理。

51.SpringMVC 中默认的配置文件名是否可以修改?如何修改配置文件名?

可以修改,可以在 web.xml 中进行配置。

52.如何通过 Spring 上下文获取到一个 bean?

1.

```
ApplicationContext ac = new FileSystemXmlApplicationContext("applicationContent.xml");  
ac.getBean("beanId");
```

2.

```
ApplicationContext ac =  
WebApplicationContextUtils.getRequiredWebApplicationContext(ServletContext sc);
```

53.SpringBoot 框架的一些问题?

1.SpringBoot 项目和其他项目有什么不同?

纯 Java 的配置方式,简单并且方便,配合各种 Starter 的使用,基本上可以做到自动化配置,配置构件工具打成 Jar 包之后,Java -jar 进行部署还是比较方便的。

文档略少,搭建容易,出了问题解决相对于 XML 配置来说比较麻烦。

2.Tomcat 嵌入式是什么?

在之前的项目中,将一个 Web 项目 Copy 到 tomcat 中进行运行,而 Tomcat 嵌入式是将 Tomcat 嵌入到主程序中进行运行。

好处就是在进行更新的时候,可以先使用自定义的程序进行自动化更新,待更新完毕之后再启动 Tomcat 或者其他的 JavaEE 容器进行运行。

修改了项目的运行方式,之前是以 Tomcat 为核心,现在是由启动程序为核心。

3.什么是约定优于配置?

约定优于配置即为按约定编程,是一种软件设计范式,减少开发人员需要做决定的数量,获得简单的好处而且也不失灵活性,本质上就是说,如果模型中有一个叫做 xxx 的类,那么数据库中对应的表就会默认命名为 xxx,如果在偏离这一约定的时候,例如将表命名为 xxa,才需要写有关这个名字的配置。

4.SpringBoot 如何连接数据库?

(1)首先将 SpringBoot 的数据库连接 Jar 包和 MySQL 的相关 Jar 包依赖添加到 pom.xml 中,进行依赖。

(2)在 application.properties 配置文件中加入数据库相关配置。

(3)创建配置类进行配置,

```
@Configuration
@MapperScan("com.miqian.user.dao")
@PropertySource("classpath:db.properties")
public class DBConfig {

    @Resource
    private Environment env;

    @Bean(destroyMethod = "close")
    public DruidDataSource dataSource() {
        DruidDataSource dataSource = new DruidDataSource();

        dataSource.setDriverClassName(env.getRequiredProperty("db.driverClassName"));
        dataSource.setUrl(env.getRequiredProperty("db.url"));
        dataSource.setUsername(env.getRequiredProperty("db.username"));
        dataSource.setPassword(env.getRequiredProperty("db.password"));
    }
}
```

联想面试题

-刘博文整理

<pre>dataSource.setValidationQuery("select 1"); dataSource.setMaxActive(50); dataSource.setInitialSize(10); dataSource.setMinIdle(15); dataSource.setMaxWait(60000); return dataSource; }</pre>
<pre>@Bean public DataSourceTransactionManager transactionManager() { return new DataSourceTransactionManager(dataSource()); } @Bean public SqlSessionFactoryBean sqlSessionFactory() throws Exception { SqlSessionFactoryBean sessionFactory = new SqlSessionFactoryBean(); sessionFactory.setDataSource(dataSource()); sessionFactory.setTypeAliasesPackage("com.miqian.user.domain"); PageInterceptor pagePlugin = new PageInterceptor(); pagePlugin.setProperties(new Properties()); Interceptor[] plugins = {pagePlugin}; sessionFactory.setPlugins(plugins); configureSqlSessionFactory(sessionFactory.getObject().getConfiguration()); return sessionFactory; }</pre>
<pre>private void configureSqlSessionFactory(org.apache.ibatis.session.Configuration configuration) { configuration.setCacheEnabled(true); configuration.setLazyLoadingEnabled(true); configuration.setUseColumnLabel(true); configuration.setUseGeneratedKeys(false); configuration.setAutoMappingBehavior(AutoMappingBehavior.PARTIAL); configuration.setDefaultExecutorType(ExecutorType.SIMPLE); configuration.setDefaultStatementTimeout(25); configuration.setSafeRowBoundsEnabled(false); configuration.setMapUnderscoreToCamelCase(false); configuration.setLocalCacheScope(LocalCacheScope.SESSION); configuration.setJdbcTypeForNull(JdbcType.OTHER); configuration.setLazyLoadTriggerMethods(new HashSet<>(Arrays.asList("equals", "clone", "hashCode", "toString"))); }</pre>

联想面试题

-刘博文整理

54. 电商项目中购物车的逻辑问题, 订单的逻辑问题?

1. 购物车逻辑

2. 订单逻辑