## 常见面试题翻车合集

#### 1.去掉 main 方法的 static 修饰符，程序会怎样？

A：程序无法编译

B：程序正常编译，正常运行

C：程序正常编译，正常运行一下马上退出

D：程序正常编译，运行时报错

答：D

题目解析：运行时异常如下：

错误: main 方法不是类 xxx 中的 static, 请将 main 方法定义为:

public static void main(String[] args)

#### 2.以下程序运行的结果是？

public class TestClass {  
 public static void main(String[] args) {  
 System.out.println(getLength());  
 }  
 int getLength() {  
 private String s = "xyz";  
 int result = s.length();  
 return result;  
 }  
}

A：3

B：2

C：4

D：程序无法编译

答：D

题目解析：成员变量 s 不能使用任何修饰符（private/protected/public）修饰，否则编译会报错。

#### 3.以下程序有几处错误？

abstract class myAbstractClass  
 private abstract String method(){};  
}

A：1

B：2

C：3

D：4

答：C

题目解析：类少一个“{”类开始标签、抽象方法不能包含方法体、抽象方法访问修饰符不能为 private，因此总共有 3 处错误。

#### 4.以下程序执行的结果是？

class A {  
 public static int x;  
 static {  
 x = B.y + 1;  
 }  
}  
public class B {  
 public static int y = A.x + 1;  
 public static void main(String[] args) {  
 System.out.println(String.format("x=%d,y=%d", A.x, B.y));  
 }  
}

A：程序无法编译

B：程序正常编译，运行报错

C：x=1,y=2

D：x=0,y=1

答：C

#### 5.switch 语法可以配合 return 一起使用吗？return 和 break 在 switch 使用上有何不同？

答：switch 可以配合 return 一起使用。return 和 break 的区别在于 switch 结束之后的代码，比如以下代码：

String getColor(String color) {  
 switch (color) {  
 case "red":  
 return "红";  
 case "blue":  
 return "蓝";  
 }  
 return "未知";  
}  
  
String getColor(String color) {  
 String result = "未知";  
 switch (color) {  
 case "red":  
 result = "红";  
 break;  
 case "blue":  
 result = "蓝";  
 }  
 return result;  
}

对于以上这种 switch 之后没有特殊业务处理的程序来说，return 和 break 的效果是等效的。然而，对于以下这种代码：

String getColor(String color) {  
 switch (color) {  
 case "red":  
 return "红";  
 case "blue":  
 return "蓝";  
 }  
 return "未知";  
}  
  
String getColor(String color) {  
 String result = "未知";  
 switch (color) {  
 case "red":  
 result = "红";  
 break;  
 case "blue":  
 result = "蓝";  
 }  
 if (result.equals("未知")) {  
 result = "透明";  
 } else {  
 result += "色";  
 }  
 return result;  
}

如果 switch 之后还有特殊的业务处理，那么 return 和 break 就有很大的区别了。

#### 6.一个栈的入栈顺序是 A、B、C、D、E 则出栈不可能的顺序是？

A：E D C B A

B：D E C B A

C：D C E A B

D：A B C D E

答：C

题目解析：栈是后进先出的，因此：

* A 选项：入栈顺序 A B C D E 出栈顺序就是 E D C B A 是正确的；
* B 选项：A B C D 先入栈，D 先出栈，这个时候 E 在入栈，E 在出栈，顺序 D E C B A 也是正确的；
* C 选项：D 先出栈，说明 A B C 一定已入栈，因为题目说了入栈的顺序是 A B C D E，所以出栈的顺序一定是 C B A，而 D C E A B 的顺序 A 在 B 前面是永远不可能发生的，所以选择是 C；
* D 选项 A B C D E 依次先入栈、出栈，顺序就是 A B C D E。

#### 7.可以在 finally 块中使用 return吗？

答：不可以，finally 块中的 return 返回后方法结束执行，不会再执行 try 块中的 return 语句。

#### 8.FileInputStream 可以实现什么功能？

A：文件夹目录获取

B：文件写入

C：文件读取

D：文件夹目录写入

答：C

题目解析：FileInputStream 是文件读取，FileOutputStream 才是用来写入文件的，FileInputStream 和 FileOutputStream 很容易搞混。

#### 9.以下程序打印的结果是什么？

Thread t1 = new Thread(){  
 @Override  
 public void run() {  
 System.out.println("I'm T1.");  
 }  
};  
t1.setPriority(3);  
t1.start();  
Thread t2 = new Thread(){  
 @Override  
 public void run() {  
 System.out.println("I'm T2.");  
 }  
};  
t2.setPriority(0);  
t2.start();

答：程序报错 java.lang.IllegalArgumentException，setPriority(n) 方法用于设置程序的优先级，优先级的取值为 1-10，当设置为 0 时，程序会报错。

#### 10.如何设置守护线程？

答：设置 Thead 类的 setDaemon(true) 方法设置当前的线程为守护线程。

守护线程的使用示例如下：

Thread daemonThread = new Thread(){  
 @Override  
 public void run() {  
 super.run();  
 }  
};  
// 设置为守护线程  
daemonThread.setDaemon(true);  
daemonThread.start();

#### 11.以下说法中关于线程通信的说法错误的是？

A：可以调用 wait()、notify()、notifyAll() 三个方法实现线程通信

B：wait() 必须在 synchronized 方法或者代码块中使用

C：wait() 有多个重载的方法，可以指定等待的时间

D：wait()、notify()、notifyAll() 是 Object 类提供的方法，子类可以重写

答：D

题目解析：wait()、notify()、notifyAll() 都是被 final 修饰的方法，不能再子类中重写。选项 B，使用 wait() 方法时，必须先持有当前对象的锁，否则会抛出异常 java.lang.IllegalMonitorStateException。

#### 12.ReentrantLock 默认创建的是公平锁还是非公平锁？

答：默认创建的是非公平锁，看以下源码可以得知：

/\*\*  
 \* Creates an instance of {@code ReentrantLock}.  
 \* This is equivalent to using {@code ReentrantLock(false)}.  
 \*/  
public ReentrantLock() {  
 sync = new NonfairSync();  
}

Nonfair 为非公平的意思，ReentrantLock() 等同于代码 ReentrantLock(false)。

#### 13.ReentrantLock 如何在一段时间内无阻塞尝试访问锁？

答：使用 tryLock(long timeout, TimeUnit unit) 方法，就可以在一段时间内无堵塞的访问锁。

#### 14.枚举比较使用 equals 还是 ==？

答：枚举比较调用 equals 和 == 的结果是一样，查看 Enum 的源码可知 equals 其实是直接调用了 ==，源码如下：

public final boolean equals(Object other) {  
 return this==other;  
}

#### 15.在 Spring 中使用 @Value 赋值静态变量为什么 null？怎么解决？

答：因为在 Springframework 框架中，当类加载器加载静态变量时，Spring 上下文尚未加载，因此类加载器不会在 bean 中正确注入静态类，导致了结果为 null。可使用 Setter() 方法给静态变量赋值，代码如下：

@Component  
public class ConfigValue {  
 private static String accessKey;  
 public String getAccessKey() {  
 return accessKey;  
 }  
 @Value("${accessKey}")  
 public void setAccessKey(String accessKey) {  
 ConfigValue.accessKey = accessKey;  
 }  
}  
/\*  
 \* 调用赋值变量  
 \*/  
@Component  
public class TestClass {  
 @Autowired  
 private ConfigValue configValue;  
 public void method() {  
 // 读取配置文件  
 configValue.getAccessKey();  
 }  
}

#### 16.如何自己实现一个定时任务？

答：启动一个后台线程，循环执行任务。代码示例如下：

Thread getTocketThread = new Thread(new Runnable() {  
 @Override  
 public void run() {  
 while (true) {  
 try {  
 // 执行业务方法  
 TimeUnit.HOURS.sleep(2); // 每两小时执行一次  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
});  
if (!getTocketThread.isAlive()) {  
 System.out.println("启动线程");  
 getTocketThread.start();  
}

#### 17.如何定义一个不定长度的数组？

答：在 Java 中使用数组必须要指定长度，如果长度不固定可使用 ArrayList、LinkedList 等容器接收完数据，再使用 toArray() 方法转换成固定数组。

#### 18.如何优雅的格式化百分比小数？

答：使用数字格式化类 DecimalFormat 来处理，具体实现代码如下：

double num = 0.37500;  
DecimalFormat df = new DecimalFormat("0.0%");  
System.out.println(df.format(num)); // 执行结果：37.5%

#### 19.什么是跨域问题？为什么会产生跨域问题？

答：跨域问题指的是不同站点直接，使用 ajax 无法相互调用的问题。跨域问题是浏览器的行为，是为了保证用户信息的安全，防止恶意网站窃取数据，所做的限制，如果没有跨域限制就会导致信息被随意篡改和提交，会导致不可预估的安全问题，所以也会造成不同站点间“正常”请求的跨域问题。

#### 20.跨域的解决方案有哪些？

答：常见跨域问题的解决方案如下：

* jsonp（只支持 get 请求）；
* nginx 请求转发，把不同站点应用配置到同一个域名下；
* 服务器端设置运行跨域访问，如果使用的是 Spring 框架可通过 @CrossOrigin 注解的方式声明某个类或方法运行跨域访问，或者配置全局跨域配置，请参考以下代码：
* // 单个跨域配置 @CrossOrigin(origins = “http://xxx”, maxAge = 3600) @RestController public class testController{ }
* // 全局配置 @Configuration public class CorsConfiguration { @Bean public WebMvcConfigurer corsConfigurer() { return new WebMvcConfigurerAdapter() { @Override public void addCorsMappings(CorsRegistry registry) { registry.addMapping("/api/\*\*“).allowedOrigins(”https://xxx"); } }; } }

#### 21.什么原因会导致 Nginx 转发时丢失部分 header 信息？该如何解决？

答：部分 header 信息丢失的原因是，丢失的 header 的 key 值中有下划线，因为 Nginx 转发时，默认会忽略带下划线的 header 信息。

解决方案有两个，一是去掉 key 值中的下划线，二是在 Nginx 的配置文件 http 中添加“underscores *in* headers on;” 不忽略有下划线的 header 信息。

#### 22.如何设计一个高效的系统？

答：要设计一个高效的系统，通常要包含以下几个方面。

##### （1）优化代码

代码优化分为两种情况：

* 代码问题导致系统资源消耗过多的问题，比如，某段代码导致内存溢出，往往是将 JVM 中的内存用完了，这个时候系统的内存资源消耗殆尽了，同时也会引发 JVM 频繁地发生垃圾回收，导致 CPU 100% 以上居高不下，这个时候又消耗了系统的 CPU 资源。这种情况下需要使用相应的排查工具 VisualVM 或 JConsole，找到对应的问题代码再进行优化；
* 还有一种是非问题代码，这种代码不容易发现，比如，LinkedList 集合如果使用 for 循环遍历，则它的效率是很低的，因为 LinkedList 是链表实现的，如果使用 for 循环获取元素，在每次循环获取元素时，都会去遍历一次 List，这样会降低读的效率，这个时候应该改用 Iterator （迭代器）迭代循环该集合。

##### （2）设计优化

有很多问题可以通过我们的设计优化来提高程序的执行性能，比如，使用单例模式来减少频繁地创建和销毁对象所带来的性能消耗，从而提高了程序的执行性能。

##### （3）参数调优

JVM 和 Web 容器的参数调优，对系统的执行性能也是也很大帮助的。比如，我们的业务中会创建大量的大对象，我们可以通过设置，将这些大对象直接放进老年代，这样可以减少年轻代频繁发生小的垃圾回收（Minor GC），减少 CPU 占用时间，从而提升了程序的执行性能。Web 容器的线程池设置以及 Linux 操作系统的内核参数设置，也对程序的运行性能有着很大的影响，我们根据自己的业务场景优化这两项内容。

##### （4）使用缓存

缓存的使用分为前端和后端：

* 前端可使用浏览器缓存或者 CDN，CDN 的全称是 Content Delivery Network，即内容分发网络。CDN 是构建在现有网络基础之上的智能虚拟网络，依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，使用户就近获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率。CDN 的关键技术主要有内容存储和分发技术；
* 后端缓存，使用第三方缓存 Redis 或 Memcache 来缓存查询结果，以提高查询的响应速度。

##### （5）优化数据库

数据库是最宝贵的资源，通常也是影响程序响应速度的罪魁祸首，它的优化至关重要，通常分为以下六个方面：

* 合理使用数据库引擎
* 合理设置事务隔离级别，合理使用事务
* 正确使用 SQL 语句和查询索引
* 合理分库分表
* 使用数据库中间件实现数据库读写分离
* 设置数据库主从读写分离

##### （6）屏蔽无效和恶意访问

前端禁止重复提交：用户提交之后按钮置灰，禁止重复提交；

用户限流，在某一时间段内只允许用户提交一次请求，比如，采取 IP 限流。

##### （7）搭建分布式环境，使用负载分发

可以把程序部署到多台服务器上，通过负载均衡工具，比如 Nginx，将并发请求分配到不同的服务器，从而提高了系统处理并发的能力。

## 更多资源下载交流请加微信：Morstrong,加入永久会员,网盘更新更快捷！

# 本资源由微信公众号：光明顶一号，提供支持