# JAVA研发工程师笔试题

姓名： 性别： 电话：

1. **基础部分**
2. **cookie和session的区别？列举几个常用的Set-Cookie属性。**

解答：主要区别是Session是在服务端保存的一个数据结构，用来跟踪用户的状态，这个数据可以保存在集群、数据库、文件中；  
Cookie是客户端保存用户信息的一种机制，用来记录用户的一些信息，也是实现Session的一种方式

**Set-Cookie属性：**

**name**字段为一个cookie的名称。

**value**字段为一个cookie的值。

**domain**字段为可以访问此cookie的域名。

非顶级域名，如二级域名或者三级域名，设置的cookie的domain只能为顶级域名或者二级域名或者三级域名本身，不能设置其他二级域名的cookie，否则cookie无法生成。

顶级域名只能设置domain为顶级域名，不能设置为二级域名或者三级域名，否则cookie无法生成。

二级域名能读取设置了domain为顶级域名或者自身的cookie，不能读取其他二级域名domain的cookie。所以要想cookie在多个二级域名中共享，需要设置domain为顶级域名，这样就可以在所有二级域名里面或者到这个cookie的值了。  
顶级域名只能获取到domain设置为顶级域名的cookie，其他domain设置为二级域名的无法获取。

**path**字段为可以访问此cookie的页面路径。 比如domain是abc.com,path是/test，那么只有/test路径下的页面可以读取此cookie。

**expires/Max-Age** 字段为此cookie超时时间。若设置其值为一个时间，那么当到达此时间后，此cookie失效。不设置的话默认值是Session，意思是cookie会和session一起失效。当浏览器关闭(不是浏览器标签页，而是整个浏览器) 后，此cookie失效。

**Size**字段 此cookie大小。

**http**字段  cookie的httponly属性。若此属性为true，则只有在http请求头中会带有此cookie的信息，而不能通过document.cookie来访问此cookie。

**secure** 字段 设置是否只能通过https来传递此条cookie

1. **Java集合类框架的基本接口有哪些？HashSet和HashMap的区别**

解答：HashMap是Hashtable的轻量级实现（非线程安全的实现），他们都实现了Map接口，主要区别

在于HashMap允许空（null）键值（key）,由于非线程安全，效率上高于Hashtable。HashMap允许将null作为一个entry的key或者value，而Hashtable不允许。HashMap把Hashtable的contains方法去掉了，改成containsvalue和containsKey。因为contains方法容易让人引起误解。Hashtable继承自Dictionary类，而HashMap是Java1.2引进的Map interface的一个实现。最大的不同是，Hastable的方法是synchronize的，而HashMap不是，在多个线程访问Hashtable时，不需要自己为它的方法实现同步，而HashMap 就必须为之提供同步。

1. **Java异常处理中，try {}里有一个return语句，那么紧跟在这个try后的finally {}里的code会不会被执行，什么时候被执行，在return前还是后?**

解答：会执行，在return前执行。

1. **在java语言中int 和 Integer 有什么区别**

解答：int是基本数据类型，Integer是int的包装类，属于引用类型

1. **下面哪个流类属于面向字符的输入流（）**

A 、BufferedWriter B、FileInputStream     C、ObjectInputStream   D、InputStreamReader

 答案：D

 解析：Java的IO操作中有面向字节(Byte)和面向字符(Character)两种方式。  
面向字节的操作为以8位为单位对二进制的数据进行操作，对数据不进行转换，这些类都是InputStream和OutputStream的子类。  
面向字符的操作为以字符为单位对数据进行操作，在读的时候将二进制数据转为字符，在写的时候将字符转为二进制数据，这些类都是Reader和Writer的子类。

1. **一般创建线程有几种方式？有什么不同？**

继承Thread类、实现Runnable接口、使用Executor框架来创建线程池；

实现Runnable接口：已经继承了别的对象的情况下，需要多继承，可以采用这种方式。

使用Executor框架：线程池非常高效，很容易实现和使用。

1. **跳出for循环的方式？如何跳出多层for循环？**

break、continue；

在最外层的for上，加标识符，可以跳出多层for循环

1. **谈谈你对java堆和栈的认识，他们两者的主要区别？**

堆--用new建立，垃圾自动回收负责回收

a、堆是一个"运行时"数据区，类实例化的对象就是从堆上去分配空间的；

b、在堆上分配空间是通过"new"等指令建立的；

c、Java针对堆的操作和C++的区别就是，Java不需要在空间不用的时候来显式的释放；

d、Java的堆是由Java的垃圾回收机制来负责处理的，堆是动态分配内存大小，垃圾收集器可以自动回收不再使用的内存空间。

e、但缺点是，因为在运行时动态分配内存，所以内存的存取速度较慢。

a、栈中主要存放一些基本类型的变量（int, short, long, byte, float, double, boolean, char）和对象句柄；

b、栈的存取速度比堆要快；

c、栈数据可以共享；

d、栈的数据大小与生存期必须是确定的，缺乏灵活性。

1. **列举三种springboot中常用的注解，并说明一下其作用**

@RequestMapping

注解提供路由信息。它告诉Spring任何来自"/"路径的HTTP请求都应该被映射到对应方法上

@ComponentScan

表示将该类自动发现扫描组件。如果扫描到有@Component、@Controller、@Service等这些注解的类，并注册为Bean，可以自动收集所有的Spring组件，包括@Configuration类

@value

通过@value注解来读取application.properties里面的配置

@ResponseBody

表示该方法的返回结果直接写入HTTP response body中，一般在异步获取数据时使用，在使用@RequestMapping后，返回值通常解析为跳转路径，加上@Responsebody后返回结果不会被解析为跳转路径，而是直接写入HTTP response body中。

@Component

泛指组件，当组件不好归类时，可以使用这个注解进行标注

@RequestParam

用在方法的参数前面

@PathVariable

路径变量注解，其参数与大括号里的名字一样要保持一致

@Bean

意思是产生一个bean,并交给spring管理。

@AutoWired

自动导入依赖的bean。byType方式。把配置好的Bean拿来用，完成属性、方法的组装

1. **SpringMVC的工作流程?**

1. 用户发送请求至前端控制器DispatcherServlet

2. DispatcherServlet收到请求调用HandlerMapping处理器映射器。

3. 处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。

4. DispatcherServlet通过HandlerAdapter处理器适配器调用处理器

5. 执行处理器(Controller，也叫后端控制器)。

6. Controller执行完成返回ModelAndView

7. HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet

8. DispatcherServlet将ModelAndView传给ViewReslover视图解析器

9. ViewReslover解析后返回具体View

10. DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）。

11. DispatcherServlet响应用户

1. **数据库部分**
2. **请说明SQL操作中delete from table 和 truncate table的区别？**

解答：两者都可以用来删除表中所有的记录。区别在于：truncate是DDL操作，它移动HWK，使HWK值为0，不需要 rollback segment .而Delete是DML操作需要rollback segment 且花费较长时间.

1. **多表查询时，优化查询sql的方法都有哪些？平时工作中，你是如何确定追踪、优化一个sql需要优化的？**

解答：优化sql

A、从业务逻辑方面来考虑，是否业务逻辑对应的表建的有问题

B、对where条件顺序来优化，oracle和mysql不同

C、对外键类似的属性建立索引、联合索引

D、不使用NULL作为索引列、不用or、不用NOT等等细节问题

E、数据量大的时候，考虑分区、分表（综合业务来弄）

F、数据量特别大的时候，考虑redis等缓存来弄

平时工作中，可以利用tomcat、weblogic的sql日志追踪功能来定位哪些sql执行的慢；可以利用c3p0、druid的监控平台来定位查询有问题的sql；

定位好的sql，查看其执行计划，mysql（explain、desc）、oracle（explain plan for），根据其索引情况具体来优化sql

1. **mysql数据库和oracle数据库，在查询sql时，是如何实现分页的？**

mysql利用limit特性

oracle利用rownum和临时表

1. **mysql数据库和oracle数据库，在同时修改多表数据时，是如何实现的？比如要求修改teacherId为1的老师所管理学生的分数（score），统一修改为80分。**

**具体的表结构如下：**

**CREATE TABLE `teacher` (**

**`teacherId` int(11) NOT NULL,**

**`teacherName` varchar(255) DEFAULT NULL,**

**`studentId` int(11) DEFAULT NULL,**

**PRIMARY KEY (`teacherId`)**

**) ENGINE=InnoDB DEFAULT CHARSET=utf8;**

**SET FOREIGN\_KEY\_CHECKS=1;**

**CREATE TABLE `student` (**

**`studentId` int(11) NOT NULL,**

**`studentName` varchar(255) DEFAULT NULL,**

**`score` varchar(255) DEFAULT NULL,**

**PRIMARY KEY (`studentId`)**

**) ENGINE=InnoDB DEFAULT CHARSET=utf8;**

mysql：update student,teacher set student.score='80' where student.studentId=teacher.studentId and teacher.teacherId=1;

oracle：update student a set score='80' where exists (select 1 from teacher b where a.studentId=b.studentId and b.teacherId =1);

1. **JAVA程序题**
2. **编程题：简要用两种方式实现，将两个类型相同的list合并。**

**解答：**

1、利用List的addAll方法

2、利用hashSet的key唯一特性来弄

3、利用for循环来弄

1. **下面程序的运行结果是什么？为什么？**

String str1 = "hello";

String str2 = "he" + new String("llo");

System.err.println(str1 == str2);

答案：false

解析：因为str2中的llo是新申请的内存块，而==判断的是对象的地址而非值，所以不一样。如果是String str2 = str1，那么就是true了。

1. **下面程序的运行结果是什么?**

class PcitcA {

public PcitcA() {

System.out.println("PcitcA");

}

{ System.out.println("I'm A class"); }

static { System.out.println("static A"); }

}

public class PcitcB extends PcitcA {

public PcitcB() {

System.out.println("PcitcB");

}

{ System.out.println("I'm B class"); }

static { System.out.println("static B"); }

public static void main(String[] args) {

System.out.println("start-------");

　　　　 new PcitcB();

System.out.println("end---------");

　　 }

}

结果是：

static A

static B

start-------

I'm A class

PcitcA

I'm B class

PcitcB

end---------

1. **求出两个数的最大公约数和最小公倍数**

// 最大公约数

public static int get\_gcd(int n1, int n2) {

int gcd = 0;

if (n1 < n2) {// 交换n1、n2的值

n1 = n1 + n2;

n2 = n1 - n2;

n1 = n1 - n2;

}

if (n1 % n2 == 0) {

gcd = n2;

}

while (n1 % n2 > 0) {

n1 = n1 % n2;

if (n1 < n2) {

n1 = n1 + n2;

n2 = n1 - n2;

n1 = n1 - n2;

}

if (n1 % n2 == 0) {

gcd = n2;

}

}

return gcd;

}

// 最小公倍数

public static int get\_lcm(int n1, int n2) {

return n1 \* n2 / get\_gcd(n1, n2);

}

1. **请用代码实现单态模式，并简要说一下懒汉式和饿汉式的区别。**

//懒汉式单例模式

public class MySingleton {

//设立静态变量

private static MySingleton mySingleton = null;

private MySingleton(){

//私有化构造函数

System.out.println("-->懒汉式单例模式开始调用构造函数");

}

//开放一个公有方法，判断是否已经存在实例，有返回，没有新建一个在返回

public static MySingleton getInstance(){

System.out.println("-->懒汉式单例模式开始调用公有方法返回实例");

if(mySingleton == null){

System.out.println("-->懒汉式构造函数的实例当前并没有被创建");

mySingleton = new MySingleton();

}else{

System.out.println("-->懒汉式构造函数的实例已经被创建");

}

System.out.println("-->方法调用结束，返回单例");

return mySingleton;

}

}

//饿汉式单例模式

public class MySingleton2 {

//设立静态变量，直接创建实例

private static MySingleton2 mySingleton = new MySingleton2();

private MySingleton2(){

//私有化构造函数

System.out.println("-->饿汉式单例模式开始调用构造函数");

}

//开放一个公有方法，判断是否已经存在实例，有返回，没有新建一个在返回

public static MySingleton2 getInstance(){

System.out.println("-->饿汉式单例模式开始调用公有方法返回实例");

return mySingleton;

}

}

1、饿汉式是[线程安全](https://www.baidu.com/s?wd=%E7%BA%BF%E7%A8%8B%E5%AE%89%E5%85%A8&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLuHNhm1fzuHTdn16LnHDL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW0srH6sP1mYrjD1nW0sPHb3r0)的,在类创建的同时就已经创建好一个静态的对象供系统使用,以后不在改变。懒汉式如果在创建实例对象时不加上synchronized则会导致对对象的访问不是[线程安全](https://www.baidu.com/s?wd=%E7%BA%BF%E7%A8%8B%E5%AE%89%E5%85%A8&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLuHNhm1fzuHTdn16LnHDL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW0srH6sP1mYrjD1nW0sPHb3r0)的。  
2、从实现方式来讲他们最大的区别就是懒汉式是延时加载,他是在需要的时候才创建对象,而饿汉式在虚拟机启动的时候就会创建，饿汉式无需关注多线程问题