# 3年JAVA技能总结-基础篇

1. 编译和反编译

JAVA编译：java文件转class文件

JAVA反编译：将class字节码文件转成java文件

JAVA常用的反编译工具：

javap：jdk自带的一个工具，可以对代码反编译，也可以查看java编译器生成的字节码

jad：<http://www.javadecompilers.com/jad>下载地址

<https://blog.csdn.net/xuwb123xuwb/article/details/80528511>使用地址

javac SwitchMain.java

jad -sjava SwitchMain.class

JAD已经很久不更新了，在对Java7生成的字节码进行反编译时，偶尔会出现不支持的问题，在对Java 8的lambda表达式反 编译时就彻底失败。

**jd-gui:可以直接拖拽jar编译**

1. 反射

//方式一：Class.forName("")

//获得字节码文件

Class c1 = Class.forName("fanshe.User");

//创建对象

c1.newInstance();

//方式二：类名.class

Class c2 = User.class;

c2.newInstance();

//方式三

Class c3 = new User().getClass();

c3.newInstance();

1. IntegerCache [-128,127]

String className = "java.lang.Integer$IntegerCache"; //静态内部类：$

try {

Class<?> clazz = Class.forName(className);

Field low = clazz.getDeclaredField("low");

low.setAccessible(true);

Field high = clazz.getDeclaredField("high");

high.setAccessible(true);

System.out.println("high=" + high.get(clazz) + ",low=" + low.get(clazz));

Field cache = clazz.getDeclaredField("cache");

cache.setAccessible(true);

Integer[] ca = (Integer[])cache.get(clazz);

System.out.println(Arrays.toString(ca));

} catch (ClassNotFoundException e) {

e.printStackTrace();

} catch (NoSuchFieldException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

}

1. String和StringBuilder、StringBuffer的区别

String:字符串数值不可变；

StringBuffer：字符串可修改，可以动态构造字符数据。StringBuffer类是可以通过Append()来修改值。线程安全。

StringBuilder：线程不安全。

三者在执行速度方面的比较：StringBuilder > StringBuffer > String

对于三者使用的总结：

1.如果要操作少量的数据用 = String

2.单线程操作字符串缓冲区下操作大量数据 = StringBuilder

3.多线程操作字符串缓冲区下操作大量数据 = StringBuffer

1. 三个与取整有关的方法

Math.ceil():表示向上取整；Math.ceil(11.3)=12;Math.ceil(-11.3)=-12。

Math.floor():表示向下取整；Math.floor(11.6)=12;Math.floor(-11.6)=-12。

Math.round():表示四舍五入；Math.round(11.5)=12;Math.round(-11.5)=-11;

Marh.round(11.3)=11;Math.round(-11.3)=--11;

1. ”==”和equals方法究竟有什么区别？

==：表示两个变量的值是否相等，比较两个基本数据类型的数据或者引用变量，用==。

equals:用于比较两个独立对象的内容是否相同。字符串的比较也用equals。

1. java中volatile和synchronized有什么区别?
2. volatile本质是在告诉jvm当前变量在寄存器（工作内存）中的值是不确定的，需要从主存中读取；synchronized则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞住。
3. volatile仅能使用在变量级别；synchronized则可以使用在变量、方法、和类级别的。
4. volatile仅能实现变量的修改可见性，并不能保证原子性；而synchronized则可以保证变量的修改可见性和原子性。
5. volatile不会造成线程的阻塞；synchronized可能会造成线程的阻塞。
6. volatile标记的变量不会被编译器优化；synchronized标记的变量可以被编译器优化。
7. Hash算法解决冲突的四种方法：
8. String类的hashCode()方法
9. Object类中的方法以及每个方法的作用

public class Object {

public final native void notify();

public final native void notifyAll();

public final native void wait(long timeout) throws InterruptedException;

//当前线程调用“锁对象”的wait()方法，当前线程释放对象锁，进入等待队列WAITING

//调用obj的wait()方法前，必须获得obj锁，也就是必须写在synchronized(obj) 代码段内

public final void wait() throws InterruptedException {

wait(0);

}

}

1. forward和redirect

转发过程：客户端首先发送一个请求到服务器，服务器匹配Servlet，并指定执行。当这个Servlet执行完后，它要调用getRequestDispacther()方法，把请求转发给指定的Servlet\_list.jsp，整个流程都是在服务端完成的，而且是在同一个请求里面完成的，因此Servlet和jsp共享同一个request，在Servlet里面放的所有东西，在student\_list.jsp中都能取出来。因此，student\_list.jsp能把结果getAttribute()出来，getAttribute()出来后执行完把结果返回给客户端，整个过程是一个请求，一个响应。

重定向过程：客户端发送一个请求到服务器端，服务器匹配Servlet，这都和请求转发一样。Servlet处理完之后调用了sendRedirect()这个方法，这个方法是response方法。所以，当这个Servlet处理完后，看到response.sendRedirect()方法，立即向客户端返回个响应，响应行告诉客户端你必须再重新发送一个请求，去访问student\_list.jsp，紧接着客户端收到这个请求后，立刻发出一个新的请求，去请求student\_list.jsp,在这两个请求互不干扰、相互独立，在前面request里面setAttribute()的任何东西，在后面的request里面都获得不了。因此，在sendRedirect()里面是两个请求，两个响应。

Forward是在服务器端的跳转，就是客户端一个请求给服务器，服务器直接将请求相关参数的信息原封不动的传递到该服务器的其他jsp或Servlet去处理。而sendRedirect()是客户端的跳转，服务器会返回客户端一个响应报头和新的URL地址，原来的参数信息如果服务器没有特殊处理就不存在了，浏览器会访问新的URL所指向的Servlet或jsp，这可能不是原来服务器上的webService了。

总结：

       1、转发是在服务器端完成的，重定向是在客户端发生的；

       2、转发的速度快，重定向速度慢；

       3、转发是同一次请求，重定向是两次请求；

       4、转发地址栏没有变化，重定向地址栏有变化；

       5、转发必须是在同一台服务器下完成，重定向可以在不同的服务器下完成。

1. 递归和尾递归

递归

private static long sum(int num){

if(num == 1)

return 1;

return num + sum(num - 1);

}

尾递归

private static long sum(int num, long total) {

if(num == 0)

return total;

return sum(num - 1, total + sum);

}

1. 内存溢出和内存泄露区别
2. CPU相关

单核CPU:单核处理器，只有一个核心处理器

多核CPU:双核处理器，

超线程HT:Hyper-Threading

总核数 = 物理CPU个数 X 每颗物理CPU的核数

总逻辑CPU数 = 物理CPU个数 X 每颗物理CPU的核数 X 超线程数

查看物理CPU的个数：cmd+systeminfo

查看物理CPU数、CPU核心数、线程数：wmic+cpu get

1. SQL注入

通过把SQL命令插入到Web表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令

1. Javaweb防止表单重复提交的几种方式：

前端控制

后台控制：

1. 4种对象引用级别

从JDK1.2版本开始，把对象的引用分为四种级别，从而使程序能更加灵活的控制对象的生命周期。细分的准则是体现在被GC回收的优先级上，这四种级别由高到低依次为：

强引用StrongReference：强引用表示一个对象处在【有用，必须】的状态，是使用最普遍的引用。如果一个对象具有强引用，那么垃圾回收器绝不会回收它。就算在内存空间不足的情况下，Java虚拟机宁可抛出OutOfMemoryError错误，使程序异常终止，也不会通过回收具有强引用的对象来解决内存不足的问题。

Student student = new Student(); // 这就是强引用

软引用SoftReference

弱引用WeakReference

虚引用PhantomReference

1. 对象拷贝

深拷贝：在对类中引用数据类型（类的实例对象）进行拷贝的时候，创建了一个新的对象，并且复制其内的成员变量

浅拷贝：只对类中基本数据类型进行了拷贝，而对引用数据类型只是进行了引用的传递，而没有真实的创建一个新的对象

浅拷贝和深拷贝是相对的，如果一个对象内部只有基本数据类型，那用 clone() 方法获取到的就是这个对象的深拷贝，而如果其内部还有引用数据类型，那用 clone() 方法就是一次浅拷贝的操作

1. 装箱和拆箱

拆箱：将包装类转为基本数据类型的过程--->...Value

装箱：将基本数据类型转为包装类的过程--->valueOf

eg:拆箱的过程就是通过Integer 实体调用intValue()方法；

装箱的过程就是调用了 Integer.valueOf(int i) 方法，帮你直接new了一个Integer对象

那么哪些地方会进行自动拆装箱？

1.添加到集合中时，进行自动装箱

2.涉及到运算的时候，“加，减，乘， 除” 以及 “比较 equals,compareTo”，进行自动拆箱

1. JAVA代理模式

动态代理：在运行时，创建目标类，可以调用和扩展目标类的方法。

Java 中实现动态的方式：JDK 中的动态代理 和 Java类库 CGLib

悲观锁和乐观锁

中间件：在大型网站中，帮助各子模块间实现互相访问，消息共享或统一访问等功能的软件产品  
 远程服务框架中间件：主要解决各子模块之间互相访问的问题。 如分布式服务框架Dubbo，RPC框架等  
 消息队列中间件：主要解决各子模之间消息共享的问题。是一种EDA架构（event driven architecture）：通过在低耦合的模块之间传输事件消息来完成模块之间的合作。大型网站中，消息队列采用发布/订阅模式。消息发布者发布消息到消息队列，消息消费者只要对该类消息感兴趣，就可以订阅该消息，对其他的业务没有任何的影响。  
 数据访问中间件：用于简化应用层对数据层的访问  
 Dubbo  
 Kafka  
 Zookeeper  
 RocketMQ  
 RabbitMQ

分布式理论：CAP、BASE、2PC、3PC、TCC

数据库：  
 关系型数据库sql：  
 Oracle：  
 Mysql：  
 索引：如何实现  
  
 sql语句：  
 union  
 union all  
 left join  
 inner join  
 outer join  
 存储过程：  
 常用函数：  
 非惯性数据库Nosql：  
 Redis：  
 Mongodb：  
 HBase：  
  
大数据框架  
 Hadoop：  
 Spark：  
 Storm：

框架：  
 Spring:  
 IOC  
 AOP  
 SpringMVC  
 Hibernate  
 Mybatis

搜索引擎  
 Solr:  
 ElasticSearch:

热更新：  
  
苹果内购IAP：  
  
切支付：  
  
CLH队列  
  
hellogithub.com

Linux常用命令  
 批量杀进程https://www.cnblogs.com/abclife/p/7380397.html  
 使用awk批量杀进程的命令：  
 ps -ef | grep firefox | grep -v grep | awk '{print "kill -9 "$2}'|sh  
 使用cut批量杀进程的命令：  
 ps -ef | grep firefox | grep -v grep | cut -c 9-15 | xargs kill -9

1. hashCode和identityHashCode区别

Object.hashCode()：Object类的native方法，所有的类都可以进行对hashCode方法重写。这时hash值是重写后的hashCode

System.identityHashCode(object)标识哈希码：返回给定对象的哈希码，该代码与默认的方法 hashCode() 返回的代码一样，无论给定对象的类是否重写 hashCode()。null 引用的哈希码为 0

区别：前者是obj的hashcode()方法计算hash值，后者是根据Object类hashCode()方法来计算hash值，如果两者不同则说明hashcode方法被重写了

1. transient关键字
2. Class.forName和ClassLoader的区别
3. 原码、反码、补码、位移

<https://www.runoob.com/java/java-basic-datatypes.html>

原码：正数符号位置0，负数符号位置1

反码：正数本身，负数除符号位外，按位取反

补码：正数本身，负数等于反码+1

位移

>> 右移：正数符号位补0，负数符号位补1（符号位为最高位，0-正数,1-负数）,右移不是循环移动

<< 左移：低位始终补0，左移是循环移动

>>> 无符号右移：符号位（最高位）始终补0（不论正负），右移不是循环移动

对于byte,short,int都是转成int后，再进行位移操作

1. JAVA排序
2. Java虚拟机  
    (1) Java虚拟机的内存布局  
    (2) GC算法及几种垃圾收集器  
    (3) 类加载机制，也就是双亲委派模型  
    (4) Java内存模型  
    (5) happens-before规则  
    (6) volatile关键字使用规则  
    Java内存泄露的问题调查定位：jmap，jstack的使用等等
3. JSONP跨域问题

<https://segmentfault.com/a/1190000011145364>

Access to XMLHttpRequest at 'https://www.layui.com/demo/table/user/?page=5&limit=10' from origin 'file://' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

测试：<meta name="referrer" content="no-referrer" />

JSONP跨域：跨域问题是由于浏览器为了防止CSRF攻击，避免恶意攻击而带来的风险而采取的同源策略限制。当一个页面中使用XMLHTTPRequest对象发送HTTP请求时（XHR请求），必须保证当前页面和请求的对象是同源的，即协议、域名和端口号要完全一致，否则浏览器就会阻止此跨域请求返回的数据。

原理；浏览器只对XHR(XMLHttpRequest)请求有同源请求限制，而对script标签src属性、link标签ref属性和img标签src属性没有这这种限制，利用这个“漏洞”就可以很好的解决跨域请求。JSONP就是利用了script标签无同源限制的特点来实现的，当向第三方站点请求时，我们可以将此请求放在<script>标签的src属性里，这就如同我们请求一个普通的JS脚本，可以自由的向不同的站点请求。

<script type=”text/javascript”>

function x(result) { //回调

console.log(result);

}

var sc = document.createElement(“script”);

sc.src = “http://localhost:8081/DreamWeb/ctrl/proposal/jsonp?callback=x”;

document.body.append(sc);

</script>

$.ajax({

url : "http://localhost:8081/DreamWeb/ctrl/proposal/jsonp",

data : {

ana : 1,

uuid : ''

},

dataType: 'jsonp',

jsonp : 'callback',

success :function (result) {

console.log(result);

},

error : function(result) {

console.log(result);

}

});

script标签设置src属性为请求的地址，并将回调函数作为参数

服务端构建JS脚本，返回给客户端的数据:

return request.getParameter("callback")+ "(" + "[{x:1},{x:2}]" + ")";

客户端在回调函数中解析服务器生成的数据

1. 时间复杂度和空间复杂度

<https://www.jianshu.com/p/f4cca5ce055a>

时间复杂度O(n):如果一个算法的执行次数是 T(n)，那么只保留最高次项，同时忽略最高项的系数后得到函数 f(n)，此时算法的时间复杂度就是 O(f(n))。常见的算法的时间 复杂度之间的关系为：

O(1)<O(logn)<O(n)<O(nlog n)<O(n^2)<O(2^n)<O(n!)<O(n^n)

1. 调用栈的数组

通过new Exception().getStackTrace();可以获得方法的调用栈的数组，那么就可以在持久化的入口处获得传入不正确值的方法调用栈了

private static void findMethodCall() {  
 StackTraceElement[] stackTrace = new RuntimeException().getStackTrace();  
 for (StackTraceElement stackTraceElement : stackTrace) {  
 System.out.println("类名=" + stackTraceElement.getClassName() + "方法名=" + stackTraceElement.getMethodName());  
 }  
}

1. java.awt.headless 模式

Headless模式是系统的一种配置模式。在启用该模式下，系统缺少了显示设备、键盘或鼠标。

使用方式

1.System.setProperty("java.awt.headless", Boolean.toString(true));

2.java -Djava.awt.headless=true

3.通过反射设置java.awt.GraphicsEnvironment中这个属性的值为true

public static synchronized Toolkit getDefaultToolkit() {

if (toolkit == null) {

java.security.AccessController.doPrivileged(

new java.security.PrivilegedAction<Void>() {

public Void run() {

Class<?> cls = null;

String nm = System.getProperty("awt.toolkit");

try {

cls = Class.forName(nm);

} catch (ClassNotFoundException e) {

ClassLoader cl = ClassLoader.getSystemClassLoader();

if (cl != null) {

try {

cls = cl.loadClass(nm);

} catch (final ClassNotFoundException ignored) {

throw new AWTError("Toolkit not found: " + nm);

}

}

}

try {

if (cls != null) {

toolkit = (Toolkit)cls.newInstance();

if (GraphicsEnvironment.isHeadless()) { //是否启用了HeadLess模式

toolkit = new HeadlessToolkit(toolkit);

}

}

} catch (final InstantiationException ignored) {

throw new AWTError("Could not instantiate Toolkit: " + nm);

} catch (final IllegalAccessException ignored) {

throw new AWTError("Could not access Toolkit: " + nm);

}

return null;

}

});

loadAssistiveTechnologies();

}

return toolkit;

}

1. org.springframework.http.HttpStatus状态码
2. 囚徒困境

|  |  |  |
| --- | --- | --- |
| A╲B | 坦白 | 抵赖 |
| 坦白 | 8,8 | 0,10 |
| 抵赖 | 10,0 | 1，1 |

谁养鱼

1. 海盗分金

5个海盗抢得100枚金币，他们按抽签的顺序依次提方案：首先由1号提出分配方案，然后5人表决，投票要超过半数同意方案才被通过，否则他将被扔入大海喂鲨鱼，依此类推。

演绎推理编辑

假设

5个海盗抢到了100枚金币，每一颗都一样的大小和价值。

他们决定这么分：

1. 抽签决定自己的号码（1，2，3，4，5）
2. 首先，由1号提出分配方案，然后大家5人进行表决，当半数以上的人同意时（包括半数），按照他的提案进行分配，否则将被扔入大海喂鲨鱼。
3. 如果1号死后，再由2号提出分配方案，然后大家4人进行表决，当且仅当半超过半数的人同意时，按照他的提案进行分配，否则将被扔入大海喂鲨鱼。

4. 依次类推......

条件：每个海盗都是很聪明的人，都能很理智的判断得失，从而做出选择。

问题：第一个海盗提出怎样的分配方案才能够使自己的收益最大化?

（如果在规则中加上下面一条会更加完善：海盗在自己的收益最大化的前提下乐意看到其他海盗被扔入大海喂鲨鱼。不加也说的过去，因为其他海盗被扔入大海喂鲨鱼符合每个海盗的最大化利益。）

x + y = 202

x + 2 = y;

100 0

99 0 1

99 0 1 0

98 0 1 0 1

1. apply、call、bind

//<https://www.jianshu.com/p/80ea0d1c04f8>

Function.apply(obj, [params1, params2, ...]); //第二个参数是数组

Function.call(obj, params1, params2, ...);

Function.bind(obj, params1, params2, ...)();

参数obj：改变函数体内的this的指向，obj将代替Function类中的this对象，这个时候obj具备了Function的属性和方法，换句话说就是obj继承了Function的属性和方法。

bind会生成一个新的函数，需要手动调用

数组追加

var x1 = [];

var x2 = [];

Array.prototype.push.apply(x1, x2); //x1中追加入x2

var numbers = [1,2,-1];

Math.max.call(Math, numbers);

Max.min.apply(Math, 1, 2, -1);

Max.min.apply(null, 1, 2, -1)

number 本身没有 max 方法，但是 Math 有，我们就可以借助 call 或者 apply 使用其方法

var x = document.getElementById(“\*”); //返回值是NodeList对象，属于伪数组，不能直接调用Array下的push或pop等方法，可以通过以下方法转换：

var nodes = Array.prototype.slice.call(x); //将x转换成为标准数组的方法

prototype

//判断是否是数组

if(typeof Array.isArray === "function" && Array.isArray(arr)) {

//判断当前浏览器是否支持indexOf

if (Array.prototype.indexOf)

if(typeof arr.indeOf === "function")

1. JS数组对象相关方法

prototype

toString将数组转成字符串，逗号分隔

filter创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素

array.filter(function(currentValue,index,arr), thisValue);

注意： filter() 不会对空数组进行检测。

注意： filter() 不会改变原始数组

参数二thisValue: 可选。对象作为该执行回调时使用，传递给函数，用作 "this" 的值。如果省略了 thisValue ，"this" 的值为 "undefined"

find返回通过测试（函数内判断）的数组的第一个元素的值

array.find(function(currentValue, index, arr),thisValue);

注意: find() 对于空数组，函数是不会执行的。

注意: find() 并没有改变数组的原始值

参数二thisValue: 可选。对象作为该执行回调时使用，传递给函数，用作 "this" 的值。如果省略了 thisValue ，"this" 的值为 "undefined"

findIndex返回传入一个测试条件（函数）符合条件的数组第一个元素位置

同上

forEach

array.forEach(function(currentValue, index, arr), thisValue);

from用于通过拥有 length 属性的对象或可迭代的对象来返回一个新数组

Array.from(object, mapFunction, thisValue);

|  |  |
| --- | --- |
| 参数 | 描述 |
| object | 必需，要转换为数组的对象。 |
| mapFunction | 可选，数组中每个元素要调用的函数。 |
| thisValue | 可选，映射函数(mapFunction)中的 this 对象。 |

eg:var newArr = Array.from([1, 2, 3], x => x \* 2);

//判断是否是数组 if(arr instanceof Array) if(Array.isArray(arr))

//验证是否是数组(前提是toString没有被重写过)

function isArray(o) {

return Object.prototype.toString.call(o) == “[object Array]”;

}

concat用于连接两个或多个数组

arr1.concat(arr2); //concat并不会改变arr1本身，它会返回一个新的数组

Array.prototype.push.apply(arr1,arr2);

includes用来判断一个数组是否包含一个指定的值，如果是返回 true，否则false

arr.includes(searchElement, fromIndex);

indexOf

lastIndexOf

array.indexOf(item, fromIndex);

isArray判断是否是数组

Array.isArray(arr);

join用于把数组中的所有元素转换一个字符串

var str = array.join(separator); //默认逗号作为分隔符，返回一个字符串

keys用于从数组创建一个包含数组键的可迭代对象

var keys = arr.keys(); //

eg: var fruits = ["Banana", "Orange", "Apple", "Mango"];

var x = fruits.keys();

document.getElementById("demo1").innerHTML = x.next().value; 0

document.getElementById("demo2").innerHTML = x.next().value; 1

document.getElementById("demo3").innerHTML = x.next().value; 2

map返回一个新数组，数组中的元素为原始数组元素调用函数处理后的值

array.map(function(currentValue,index,arr), thisValue);

注意： map() 不会对空数组进行检测。

注意： map() 不会改变原始数组

pop用于删除数组的最后一个元素并返回删除的元素

var ele = array.pop();

shift移除数组第一个元素并返回删除的元素

var ele = array.shift();

push向数组的末尾添加一个或多个元素，并返回新的长度

var newLen = array.push(item1, item2, ..., itemX);

unshift 在数组起始位置添加元素，并返回新的长度

var newLen = array.unshift(item1,item2, ..., itemX);

reduce接收一个函数作为累加器，数组中的每个值（从左到右）开始缩减，最终计算为一个值

var sum = array.reduce(function(total, currentValue, currentIndex, arr), initialValue);

注意: reduce() 对于空数组是不会执行回调函数的

reduceRight从右向左

reverse颠倒数组中元素的顺序，直接修改原数组

array.reverse();

slice可从已有的数组中返回选定的元素，可提取字符串的某个部分，并以新的字符串返回被提取的部分

var newArr = array.slice(start, end); //并不会改变原数组

|  |  |
| --- | --- |
| **参数** | **描述** |
| start | 可选。规定从何处开始选取。如果是负数，那么它规定从数组尾部开始算起的位置。也就是说，-1 指最后一个元素，-2 指倒数第二个元素，以此类推。 |
| end | 可选。规定从何处结束选取。该参数是数组片断结束处的数组下标。如果没有指定该参数，那么切分的数组包含从 start 到数组结束的所有元素。如果这个参数是负数，那么它规定的是从数组尾部开始算起的元素。 |

将类数组转成数组

Array.prototype.slice.call(argements)

splice添加或删除数组中的元素，会改变原有数组

array = array.splice(index, howmany, ele1, ele2, ...);

|  |  |
| --- | --- |
| **参数** | **描述** |
| index | 必需。规定从何处添加/删除元素。 该参数是开始插入和（或）删除的数组元素的下标，必须是数字。 |
| howmany | 可选。规定应该删除多少元素。必须是数字，但可以是 "0"。 如果未规定此参数，则删除从 index 开始到原数组结尾的所有元素。 |
| item1, ..., itemX | 可选。要添加到数组的新元素 |

some用于检测数组中是否存在某元素满足指定条件（函数提供）

array.some(function(currentValue,index,arr),thisValue); //返回true或false

some() 方法用于检测数组中的元素是否满足指定条件（函数提供）。

some() 方法会依次执行数组的每个元素：

如果有一个元素满足条件，则表达式返回true , 剩余的元素不会再执行检测。

如果没有满足条件的元素，则返回false。

注意： some() 不会对空数组进行检测。

注意： some() 不会改变原始数组。

sort用于对数组的元素进行排序，会改变原始数组

array.sort(sortfunction)

1. 常用的设计模式

(1) 你的项目中用到了哪些设计模式，如何使用。  
(2) 知道常用设计模式的优缺点。  
(3) 能画出常用设计模式的UML图。  
1.单例模式：  
 饿汉式：典型的空间换时间，当类装载的时候就会创建类实例，不管你用不用，先创建出来，然后每次调用的时候，就不需要判断了，节省了运行时间  
 懒汉式：  
 双重校验锁懒汉式：  
 静态内部类：由JVM来保证线程安全  
 枚举：  
2.模板方法模式：抽取公共实现，剥离单独实现  
3.建造者模式：将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示  
4.观察者模式：java.util.Observer和java.util.Observable  
5.代理模式：由于某些原因需要给某对象提供一个代理以控制对该对象的访问。这时，访问对象不适合或者不能直接引用目标对象，代理对象作为访问对象和目标对象之间的中介  
 主要优点有：  
 代理模式在客户端与目标对象之间起到一个中介作用和保护目标对象的作用；  
 代理对象可以扩展目标对象的功能；  
 代理模式能将客户端与目标对象分离，在一定程度上降低了系统的耦合度；  
 主要缺点是：  
 在客户端和目标对象之间增加一个代理对象，会造成请求处理速度变慢；  
 增加了系统的复杂度  
  
 静态代理：编译期间手动生成代理对象  
 动态代理：  
 JDK动态代理：在程序运行时JVM才为被代理对象生成代理对象  
 限制：被代理的对象必须实现一个或多个接口，若想代理没有实现接口的类，就需要使用Cglib实现  
 Cglib动态代理（子类代理）：在内存中构建一个子类对象从而实现对目标对象功能的扩展  
6.适配器模式：将一个类的接口转换成客户希望的另外一个接口，使得原本由于接口不兼容而不能一起工作的那些类能一起工作。  
 适配器模式分为类结构型模式和对象结构型模式两种，前者类之间的耦合度比后者高，且要求程序员了解现有组件库中的相关组件的内部结构，所以应用相对较少些  
 该模式的主要优点如下。  
 客户端通过适配器可以透明地调用目标接口。  
 复用了现存的类，程序员不需要修改原有代码而重用现有的适配者类。  
 将目标类和适配者类解耦，解决了目标类和适配者类接口不一致的问题。  
 其缺点是：对类适配器来说，更换适配器的实现过程比较复杂  
  
 适配器模式（Adapter）包含以下主要角色。  
 目标（Target）接口：当前系统业务所期待的接口，它可以是抽象类或接口。  
 适配者（Adaptee）类：它是被访问和适配的现存组件库中的组件接口。  
 适配器（Adapter）类：它是一个转换器，通过继承或引用适配者的对象，把适配者接口转换成目标接口，让客户按目标接口的格式访问适配者。  
7.原型模式：原型模式属于对象的创建模式。通过给出一个原型对象来指明所有创建的对象的类型，然后用复制这个原型对象的办法创建出更多同类型的对象。这就是选型模式的用意  
 有两种表现形式：（1）简单形式、（2）登记形式，这两种表现形式仅仅是原型模式的不同实现  
  
 原型模式分三个角色，抽象原型类，具体原型类，客户类。  
 抽象原型类（prototype）:它是声明克隆方法的接口，是所有具体原型类的公共父类，它可以是接口，抽象类甚至是一个具体的实现类。  
 具体原型类（concretePrototype）：它实现了抽象原型类中声明的克隆方法，在克隆方法中返回一个自己的克隆对象。  
 客户类（Client）：在客户类中，使用原型对象只需要通过工厂方式创建或者直接NEW（实例化一个）原型对象，然后通过原型对象的克隆方法就能获得多个相同的对象。由于客户端是针对抽象原型对象编程的所以还可以可以很方便的换成不同类型的原型对象  
  
 浅克隆：如果原型对象的成员变量是值类型（八大基本类型，byte,short,int,long,char,double,float,boolean）.那么就直接复制，如果是复杂的类型（枚举，String,对象）就只复制对应的内存地址  
 在换个说法，就是复杂类型的成员变量（String,枚举，啥的）用的是一个。修改了克隆对象的原型对象也会变。他们是共用的。而值类型不是共用的  
 深克隆：全部复制，然后各自独立。你修改克隆对象对于原型对象没有丝毫影响  
8.桥接模式：将抽象与实现分离，使它们可以独立变化。它是用组合关系代替继承关系来实现，从而降低了抽象和实现这两个可变维度的耦合度  
9.装饰者模式：指在不改变现有对象结构的情况下，动态地给该对象增加一些职责（即增加其额外功能）的模式，它属于对象结构型模式  
 装饰（Decorator）模式的主要优点有：  
 采用装饰模式扩展对象的功能比采用继承方式更加灵活。  
 可以设计出多个不同的具体装饰类，创造出多个不同行为的组合。  
 其主要缺点是：  
 装饰模式增加了许多子类，如果过度使用会使程序变得很复杂  
 装饰模式主要包含以下角色。  
 抽象构件（Component）角色：定义一个抽象接口以规范准备接收附加责任的对象。  
 具体构件（Concrete Component）角色：实现抽象构件，通过装饰角色为其添加一些职责。  
 抽象装饰（Decorator）角色：继承抽象构件，并包含具体构件的实例，可以通过其子类扩展具体构件的功能。  
 具体装饰（ConcreteDecorator）角色：实现抽象装饰的相关方法，并给具体构件对象添加附加的责任  
10.命令模式：将一个请求封装为一个对象，使发出请求的责任和执行请求的责任分割开。这样两者之间通过命令对象进行沟通，这样方便将命令对象进行储存、传递、调用、增加与管理  
 主要优点如下。  
 降低系统的耦合度。命令模式能将调用操作的对象与实现该操作的对象解耦。  
 增加或删除命令非常方便。采用命令模式增加与删除命令不会影响其他类，它满足“开闭原则”，对扩展比较灵活。  
 可以实现宏命令。命令模式可以与组合模式结合，将多个命令装配成一个组合命令，即宏命令。  
 方便实现 Undo 和 Redo 操作。命令模式可以与后面介绍的备忘录模式结合，实现命令的撤销与恢复。  
 其缺点是：可能产生大量具体命令类。因为计对每一个具体操作都需要设计一个具体命令类，这将增加系统的复杂性。  
11.责任链模式：

META-INF/MANIFEST.MF文件

String version = SpringBootVersion.*getVersion*();

String version2 = SpringBootVersion.**class**.getPackage().getImplementationVersion()

1. 四大国产数据库

南大通用 武汉达梦 人大金仓 神舟通用

1. 全文检索

全文检索Full-Text Search：先建立索引，再对索引进行搜索的过程。

反向索引Inverse Index：由于从字符串到文件的映射是文件到字符串映射的反向过程

影响一个词(Term)在一篇文档中的重要性主要有两个因素：

Term Frequency (tf)：即此Term在此文档中出现了多少次。tf 越大说明越重要。

Document Frequency (df)：即有多少文档包含次Term。df 越大说明越不重要

工具

必须至少熟练使用一种IDE的开发工具，例如Eclipse、

Netbeans、JBuilder、Jdeveloper、IDEA、JCreator或者Workshop，包括进行工程管理、常用选项的设置、插件的安装配置以及进行调试。

命令

必须熟悉JDK带的一些常用命令及其常用选项，命令至少需要熟悉：appletviewer、HtmlConverter、jar、JAVA、javac、javadoc、javap、javaw、native2ascii、serialver，如果这些命令你没有全部使用过，那么你对JAVA实际上还很不了解，更谈不上精通JAVA技术了。

博学

掌握J2EE 、Oracle 、WebLogic、Jboss、Spring、Struts、Hibernate 等流行技术，掌握软件架构设计思想、搜索引擎优化、缓存系统设计、网站负载均衡、系统性能调优等实用技术。

规范

编写的代码必须符合流行的编码规范，例如类名首字母大写，成员和方法名首字母小写，方法名的第一个单词一般是动词，包名全部小写等，这样程序的可读性才比较好。

测试

必须熟悉使用junit编写测试用例完成代码的自动测试。

语法

必须比较熟悉，在写代码的时候IDE的编辑器对某一行报错应该

能够根据报错信息知道是什么样的语法错误并且知道任何修正。

思想

必须掌握OOP的主要要求，这样使用JAVA开发的系统才能是真正的JAVA系统。

排错

应该可以根据异常信息比较快速的定位问题的原因和大致位置。

API

JAVA的核心API是非常庞大的，但是有一些内容软件培训网认为是必须熟悉的，否则不可能熟练的运用JAVA，包括：

◆java.io包下的60%以上的类的使用，理解IO体系的基于管道模型的设计思路以及常用IO类的特性和使用场合。

◆java.util包下的80%以上的类的灵活运用，特别是集合类体系、规则表达式、zip、以及时间、随机数、属性、资源和Timer。

◆XML处理，熟悉SAX、DOM以及JDOM的优缺点并且能够使用其中的一种完成XML的解析及内容处理。

◆java.math包下的100%的内容。

◆java.net包下的60%以上的内容，对各个类的功能比较熟悉。

◆java.text包下的60%以上的内容，特别是各种格式化类。

◆熟练运用JDBC. 8）、java.security包下40%以上的内容，如果对于安全没有接触的话根本就不可能掌握JAVA。

◆java.lang包下的80%以上的类的功能的灵活运用。

◆Swing的基本内容，和AWT的要求类似。

◆AWT的基本内容，包括各种组件事件、监听器、布局管理器、常用组件、打印