java的优势

1：跨平台（平台指的是操作系统）

跨平台：指的是我们用java开发出来的应用程序 是不受操作系统限制的

底层的功臣：JVM = java虚拟机 = 秘书 + 保镖 + 翻译

秘书 类加载器 ClassLoader

保镖 字节码校验器 ByteCode Verifier

翻译 解释执行器 Interpreter

2：安全 健壮

健壮：程序在遇到突发情况的时候，是否还可以正常执行

3：免费 开源

4：简单

语法简单：去除了c++的一些东西，如：指针 运算符重载 手动的垃圾回收

理想简单：面向对象的思想 = OO思想

面向过程的思想：需要程序员以计算机的角度去思考问题

面向对象的思想：需要程序员拿着代码去模拟现实生活

类：一组类型相同事物高度抽象之后的集合概念。创建对象的模版

对象：类的一个具体的实例

引用：对象的名字，一个对象同时可以有多个名字，但是一个对象如果一个名字都没有的话会被gc回收掉 -》 零引用内存回收

属性：对象有什么

方法：对象会什么

5：动态更新

对于Sun公司官方提供的核心方法只保留一个指向的关系，刚若干年后核心类库升级了，我们的程序还是可以正常使用，这是挤兑c/c++的

============================================================================================

开发环境的搭建

1：安装jdk http://www.oracle.com

SDK = software development kits = 软件开发工具包(目录)

JDK = java + SDK = java软件开发工具包

JRE = java runtime environment = java运行环境：

jre = jvm + api(核心类库);

jvm = 类加载器 + 字节码校验器 + 解释执行器

jdk = jre + bin(常用的工具命令)

jdk -> jre -> jvm

面试题：jre和jdk之间的区别？

jdk表示java软件开发工具包，如果电脑上面安装jdk，既可以编写程序，又可以运行程序

jre表示java运行环境，如果电脑上面安装jre，只能运行java程序

2：设置环境变量(环境变量可以不设置，为了简化开发流程 提高开发效率)

PATH:操作系统 让操作系统更加快捷的找到一个文件/命令

PATH=C:\Program Files\Java\jdk1.6.0\_45\bin

CLASSPATH:类加载器 让类加载器明确去哪里加载.class文件，通常不需要设置 默认值：. .:当前目录

CLASSPATH相当于中国的低保制度

JAVA\_HOME:给其他的程序使用的 WEB

面试题：

1：在设置环境变量的时候最少需要设置几个环境变量？

0

2：在设置环境变量的时候通常需要设置几个环境变量？

1 -》 PATH

java中的软件包结构：\*\*\*

包结构 = 操作系统中目录（文件夹）结构目录（包）的作用？

1：按照文件的功能 性质进行分类 方便查找

2：给出不同的命名空间从而让重名的文件可以同时存在

打包语句：package

\*:打包语句必须出现在文件的第一行

============================================================================================

常用的工具命令：bin

javac 编译指令

java 运行指令

javadoc 生成帮助文档[说明书]

javadoc -d 说明书 XXX.java

jar 生成可执行的压缩归档文件

jar cvf XXX.jar YYY.class

java中的数据类型：

java是一个强类型的编程语言，所有的变量在第一次出现的时候，必须有数据类型

面试题：

java中的数据类型分为哪些？

2种/无数种

基本数据类型 简单数据类型[程序里面自带的数据类型]

java中的基本数据类型分为哪些？4类8种

布尔类型：boolean

\*:只能使用true/false进行赋值

\*:不能使用数字进行赋值 （c++）

字符类型：char 底层基于Unicode编码实现的

统一的都是2个字节 = 16位

赋值方式：

char x1 = 'a';//指定字面值的赋值方式

char x2 = '中';//指定字面值的赋值方式

char x3 = 99;//通过ASCII编码进行赋值

char x4 = '\u3d2d';//通过Unicode编码进行赋值

char x5 = '\t';//转义字符

\t \r \n \b

制表符 返回行首 换行 退格键

整数类型：符号位(1) + 数值位(n-1)

取值范围：-2(n-1)到2(n-1)-1

n-1:符号位占用一个位

-1：0占用一个位

byte short int long

n: 8 16 32 64

\* L/l

\*：整数类型默认类型是int类型

\*：long类型的结尾需要加上L/l，尽量写L

\*：当数值超出int类型最大边界的时候注意使用long类型

整数类型的赋值方式：

int num1 = 24;//十进制进行赋值 \*\*\*\*\*

int num2 = 024;//八进制进行赋值

int num3 = 0x24;//十六进制进行赋值

int num4 = 0b1011;//二进制进行赋值 since jdk7.0

十进制和二进制之间的转换（非重点）

byte:-128 127

short:-32768 32767

int:-2147483648 2147483647

long:-XXXX XXXX

浮点类型：符号位 + 幂值位 + 数值位

float double

32 64

\*

F/f D/d

float:符号位1 + 幂值位8 + 数值位23

double:符号位1 + 幂值位11 + 数值位52

面试题：

3.2E3F E F分别表示什么？

E:表示科学计数法 3.2\*10(3)

F:表示float类型

小结：

基本数据类型

布尔类型：boolean

字符类型：char \*：不区分中英文

char x = 'a';

char y = '中';

整数类型：byte short int(\*) long

int x = 45;

浮点类型：float double(\*)

引用数据类型

===========================================================================================

基本数据类型之间的转换：

面试题：

double类型后面可以加上D可以不加，什么时候加上D比较好？

double x = 45;//double = int ->double

double y = 45D;//double = double

小的数据类型可以直接给大的数据类型赋值，但大的数据类型需要强转为小的数据类型

byte和char之间永远需要强转

short和char之间永远需要强转

引用数据类型 自定义数据类型[程序员自己写的数据类型]

java中的运算符：+ - \* / %

[+ 当+遇到String类型 不再表示算数加法 而且表示追加连接]

[\* 当心数值超出int类型最大边界 注意使用long类型]

365L \* 24 \* 60 \* 60 \* 1000

[/ 整数除以整数 结果一定是整数]

5 / 0.0 = Infinity -> 无穷大

0.0 / 0.0 = Nan = not a number = 不是一个数

[% 取模运算符 求余数 最终结果的符号位取决于参与运算的第一个数的符号]

+= -= \*= /= %=....

[他们能够保证运算符的左边的数据类型不发生改变 -》 隐式的强转]

++ --[自增自减运算符]

[a:取值 b:运算 x++:ab ++x:ba]

> < >= <= ! = != ==

> < >= <=:返回类型都是boolean类型 -》 true/false

System.out.println(45 > 53);//true false 报错

!:非 !boolean

!true = false !false = true

!=:判断左右两边的值是否不相等

不相等：true 相等：false

System.out.println(45 != 45);//false

=和==之间的区别？

=：赋值运算符 将右侧的值交给左侧的变量进行保管

int x = 45;

==:判断左右两边的值是否相等

System.out.println(45 == 55);//false

&& || & |[逻辑运算符]

面试题：

&&和&之间的区别？

相同点：&&和&都是逻辑的与 相当于汉文中并且的意思

需要前后两个条件都成立才可以

不同点：&&叫做逻辑的短路与 有短路特性

短路特性：如果通过前面的条件 可以得出最终结果的话，后面的条件直接舍弃 不再判断

例子：

如果晚上我有空的话 而且你们愿意 我请大家唱歌

if(我有空 && 你们愿意){

我请大家唱歌

}

boolean && boolean

true 判断

false 短路

||和｜之间的区别？

相同点：||和｜都是逻辑的或 相当于汉文中或者的意思

需要前后一个条件成立就可以

不同点:||叫做逻辑的短路或 有短路特性

短路特性：如果通过前面的条件 可以得出最终结果的话，后面的条件直接舍弃 不再判断

例子：

QQ注册的时候 如果用户名长度>6位 或者密码长度<12位 显示注册失败

if(用户名长度>6 || 密码长度<12){

显示注册失败

}

boolean || boolean

true 短路

false 判断

& ｜ ^ ~[按位（二进制位）运算符]

& ｜ 什么情况下表示逻辑运算符？什么情况下表示按位运算符？

需要看运算符两边的数据类型：

boolean -> 逻辑运算符

数字 -》 按位运算符

基本用法（非重点）

&：按位的与 上下都为1的时候才写1

System.out.println(9 & 5);//1

9:8 + 1 = 0000....1001

5:4 + 1 = 0000....0101 &

0000....0001 :

|: 按位的或 上下只要有1就写1

System.out.println(9 | 5);//13

9:8 + 1 = 0000....1001

5:4 + 1 = 0000....0101 |

0000....1101 : 8 + 4 + 1 = 13

^: 按位的异或 上下不一样的时候才写1

System.out.println(9 ^ 5);//12

9:8 + 1 = 0000....1001

5:4 + 1 = 0000....0101 ^

0000....1100 : 8 + 4 = 12

~ : 按位的取反 0-》1 1-》0

int x = 9;

~x;

9:8 + 1 = 000000...1001

111111...0110 :

111111...1111 : -1 - 1 - 8 = -10

重点：

&:在某些情况下可以取代%

某些情况：

当我们拿着一个正数 % 2的n次方数的时候

结果完全等效于我们拿着这个正数 & 2的n次方数-1的结果

x % 2(n) -> x & 2(n)-1

%2 %4 %8 %16 .... %2(n)

27 1 3 3 11

&1 &3 &7 &15 ..... & 2(n)-1

27 1 3 3 11

27 & 1

27:16 + 8 + 2 + 1 = 0000...11011

7 : = 0000...00111 &

0000...00011 : 3

^ ： 可以不借助第三块空间的方式交换两个变量的值

int x = 100;

int y = 67;

//采用第三块空间的方式：

int z = x;

x = y;

y = z;

//不采用第三块空间的方式：

x = x ^ y;

y = x ^ y;

x = x ^ y;[char byte short int long]

>> << >>>[位(二进制位)移运算符]

<<相当于乘以 >>相当于除以

移动多少位相当于乘以/除以2的几次方数

\*2 \*4 \*8 \*16 .... \*2(n)

<<1 <<2 <<3 <<4 .... << n

/2 /4 /8 /16 .... /2(n)

>>1 >>2 >>3 >>4.... >>n

面试题：

1：在计算机里面如何高效的求出2的3次方数？

2 \* 2 \* 2

2 << 1 << 1 -> 2 << 2

2:在计算机里面 如何高效的求出2的31次方数-1的结果？

(2 << 30) - 1

3:>>和>>>之间的区别？

>>:叫做有符号右移 异动之后空缺的高位看原本的符号位

int x = 8;

x >> 2;

8 : 0000...1000

000000...10

>>>:叫做无符号右移 移动之后空缺的高位直接补0

01010000 >>> 2

00010100

11101010

00111010

? : [三目运算符 三元运算符]

1 ? 2 : 3

第一部分永远放boolean类型的判断，如果第一部分位true执行第二部分，如果第一部分条件位false 执行第三部分

===========================================================================================

java中的变量：

成员变量 实例变量 属性：定义在类体里面方法体外面

局部变量 自动变量：定义在方法体里面的变量

\*：java中没有全局变量的概念

成员变量和局部变量之间的区别？

1：定义的位置不同

成员变量：方法体外面

局部变量：方法体里面

2：作用范围不同

成员变量：对象有什么，依赖于对象存在，只要依赖的对象没有被回收，统统可以访问

局部变量：从定义的那一行开始，一直到所在的方法体结束之前可以访问，一旦所在的方法体执行结束之后，局部变量会立即消亡

3：默认值不同

成员变量：即使不赋值也有默认值

局部变量：没有默认值 要求在使用之前必须先赋值

\*：成员变量和局部变量可以重名的，当成员变量和局部变量重名的时候，如果直接访问该变量，默认是局部变量，如果想要访问成员变量，需要在变量的前面加上this

流程控制：

分支：

if else:

语法格式：

if(boolean条件){

执行语句1;

}else if(boolean条件){

执行语句2；

}else if(boolean条件){

执行语句3；

}else{

执行语句4;

}

\*:学会使用if else的排他特性简化代码

不要在下面的条件里面写出上面条件不成立的等价判断

switch case:

语法格式：

switch(参数){

case XXX : 执行语句;

case YYY : 执行语句;

case ZZZ : 执行语句;

default : 执行语句;

}

\*:学会使用break共享代码

将执行相同操作的代码放在一起进行共享

面试题：

switch case的参数可以传哪些数据类型？

jdk1.0 char byte short int

jdk5.0 enum->枚举

jdk7.0 String

循环：

for:

语法格式：

for(1;2;3){

4;

}

1:初始化循环变量

2:循环执行的条件

3:循环之后的变化

4:循环执行的代码

执行步骤：1243243243...2

while:

语法格式：

1;

while(2){

4;

3;

}

do while:

1;

do{

4;

3;

}while(2);

面试题：

while和do while之间的区别？

while先判断 符合条件在执行

do while先执行 然后在判断 能保证程序至少执行1次 -》 先斩后奏

循环控制：

1：循环嵌套：一个循环定义在另一个循环里面

===========================================================================================

java中的数组：一组类型相同存储空间连续的数据:容器

1:如何创建一个数组对象

数据类型[]

int[] data1 = new int[3];

int[] data2 = new int[]{45,66,18,29,55};

int[] data3 = {45,66,19,28};

2:如何得到数组里面的某一个元素：

数组对象[下标]

下标从0开始 最大下标数组长度 - 1

System.out.println(data2[4]);

3:如何得到数组的大小

数组对象.length

System.out.println(data3.length);

4:如何遍历数组对象

A:for + 下标

for(int x = 0;x < data.length;x++){

//x -> 下标

System.out.println(data[x]);

}

B:foreach forin since jdk5.0

for(int x : data){

//x -> 元素

System.out.println(x);

}

===========================================================================================

数组的复制：

1：错误的复制方式

只是内存里面的对象多了一个名字而已

2：笨重的复制方式

需要程序员挨个的手动进行复制

3：“克隆”的方法，可以不符合正常的应用场景

4：System.arraycopy(1,2,3,4,5);

1：要复制的老鼠组对象

2：老鼠组的起始下标位置

3：要复制到的目标数组

4：新数组的起始下标位置

5：要复制的长度

\*：当我们用System.arraycopy方法的时候，起始一三参数可以是同一个数组对象，从而实现的是一个数组的前后移动

数组的排序：

int[] data = new int[]{45,22,81,73,10};

手动排序 冒泡排序 \*\*\*\*\*

for(int x = 0;x < data.length - 1;x++){

for(int y = 0;y < data.length -1 - x;y++){

//得到前后两个元素

//y y+1 -> 下标

//data[y] data[y + 1]

if(data[y] > data[y + 1]){

//交换

/\*

data[y] = data[y] ^ data[y + 1];

data[y + 1] = data[y] ^ data[y + 1];

data[y] = data[y] ^ data[y + 1];

\*/

int z = data[y];

data[y] = data[y + 1];

data[y + 1] = z;

}

}

}

===========================================================================================

面向对象开发的三大特点：封装 继承 多态

封装：用private将不想被外界访问到的属性或者方法隐藏起来

private：私有的 只有本类可以访问

public：公共的 谁都可以访问

封装的好处？

类的开发者：将数据得到了保护，从而更加的安全

类的使用者：将精力放在核心业务逻辑上

封装对属性和方法都有要求：

对属性的要求：

所有的都需要用private修饰，每个属性配套提供两个public修饰的方法，一个是用来给属性进行赋值的方法 -》 setter

public void setXXX(参数){

this.XXX = XXX;

}

一个是用来得到属性值的方法 -》 getter

public XXX getXXX(){

return XXX;

}

对方法的封装：

判断哪些是核心方法

核心方法 -》 public

给核心方法提供帮助的方法 -》 private

继承 : 用extends实现两个类(子类 extends 父类)之间is a(是一个 是一种)的关系

\*：继承是最简单的代码共享方式之一

\*：java中的类只允许单根继承

class A{}

class B{}

class C extends A,B{}//错误

面试题：

父类的所有属性都会被子类继承得到吗？

父类的私有属性在子类的内存空间里面存在

但是不能继续使用 所以不能算是正常继承得到

多态：一个事物总有不同的名字去称呼他

一个对象总有不同的类型去定义他

多态的第一个作用：创建对象（非重点）

多态：父类类型 = new 子类对象

Animal xx = new Cat();

Person xx = new Student();

\*:当我们使用多态的方式创建完一个对象之后这个对象只能调用父类有的属性、方法

\*:在编译的时候编译器只看等号的左边进行语法检查

\*:在运行的时候 一个对象属于什么类型决定他能调用哪些属性、方法

多态的第二个作用：\*\*\*\*\*\*

放在参数里面用于解除代码之间的耦合度

方法定义（父类类型）

方法调用（子类对象）

wc(Person x)

wc(stu);

方法重载 方法过载 Overload

面试题：

方法重载的条件？

1：需要发生在同一个类体里面

2：方法名字一模一样

3：方法的参数不同

参数类型不同

参数的个数不同

参数的顺序不同

\*：方法重载对返回类型和修饰符没有要求

\*：方法重载作用？

同时满足用户的不同需求

System.out.println();//println()

System.out.println(34);//println(int)

System.out.println('a');//println(char)

System.out.println("etoak");//println(String)

System.out.println(45.6);//println(double)

System.out.println(false);//println(boolean)

方法覆盖 方法重写 Override

面试题：

什么情况下满足方法覆盖？

需要发生在有继承关系的两个类中 而且是在子类里面发生的

（子类在继承得到父类的某些方法之后 觉得父类的实现不好

于是在子类里面重新实现一下）

达尔文 -》 《进化论》

public void test()throws Exception{}

| |

脑袋 躯干 尾巴

脑袋：访问权限修饰符 >= 父类的访问权限

Sun公司认为子类重新实现的方法应该更加优秀

更加优秀的方法应该给更多的人使用 这样才利于java的发展

躯干：返回类型 + 方法签名 一模一样

尾巴：异常处理部分 <= 父类的异常 （范围）

异常：程序在执行的时候 可能会出现的隐患

Sun公司认为子类重新实现的方法应该更加优秀，更加优秀的方法不应该有更多的隐患

面试题：

父类抛出一个异常 子类能不能抛出更多的异常？

可以 但是需要保证子类所有的异常范围<父类的异常范围

从jdk5.0开始 方法覆盖的时候 可以在子类要覆盖的

方法上面加上@Override -》 注解

表示下面的这个方法一定要覆盖父类的某个元素

异常：Exception

小：NullPointerException:N

ClassNotFondException:C

ArrayIndexOutOfException:A

父类：

protected void test()throws Exception{}

子类：

protected/public void test()throws N,C,A{}

===========================================================================================

构造方法：

\*：java中没有构造函数的概念

构造方法是创建对象的时候调用的方法-》收尾的工作

1：构造方法作用？

构造方法的出现其实为了方便程序员做开发的

可以在创建对象同时直接对属性进行赋值

\*：java中只要是个类就一定有构造方法

即使我们没有写 一同也会提供一个默认的构造方法 -》 无参空体

如果我们想一边创建对象一边对属性进行赋值的话，需要自己提供构造方法，一旦我们写出自己的构造方法之后，默认的构造方法将不再提供 -》 低保制度 ：CLASSPATH

2：构造方法的特点：

A:没有返回类型 连void都不能写 修饰符 + 方法签名

B:构造方法的名字需要和类名一模一样

3: 构造方法的首行：

默认是super():

super():表示要执行本构造方法之前 先去执行父类的构造方法，默认找父类的无参构造方法，如果父类没有无参构造方法：

A:提供父类的无参构造方法

B:在super里面传参数，指定他找父类的哪一个构造方法

构造方法的首行还可以出现this():

this():表示要执行本构造方法之前 先去执行本类的其他的构造方法，具体执行本类的哪一个构造方法 看参数类型

\*：构造方法首行只能出现super()/this()中的一个

面试题：

构造方法能不能方法重载？

可以重载

构造方法能不能方法覆盖？

不能

方法覆盖的前提是需要先继承得到

但是构造方法没办法被子类继承得到 所以不能进行覆盖

===========================================================================================

参数传递：

java只有值传递

java中的基本数据类型传值 引用数据类型传地址

String类里面两个常见的面试题：

1：new和不new之间的区别？

String str1 = "etoak";

String str2 = new String("etoak");

不new的方式涉及到常量池查找机制

2: String、StringBuffer、StringBuilder都是表示java中的字符串，他们三者之间的区别？

A:String和StringBuffer/StringBuilder之间的区别？

String str = new String("etoak");//[]:5

StringBuffer buffer = new StringBuffer("etoak");//[]:5 + 16

StringBuffer/StringBuilder在创建对象的时候会多预留16块缓冲区（缓冲区用来对字符串不断追加连接的）

B:StringBuffer和StringBuilder之间的区别？

StringBuffer同一时间允许一个线程进行访问，效率较低，但是不会出现并发错误，相当于把子肉

StringBuilder同一时间允许多个线程一起进行访问，效率较高，但是可能会出现并发错误相当于KFC

String类里面常用的20个方法：

和长度有关的方法：

返回类型 方法签名 作用

int length() 得到字符串的字符个数

和数组有关的方法：

返回类型 方法签名 作用

byte[] getBytes() 将字符串转换成字节数组

char[] toCharArray() 将字符串转换成字符数组

String[] split(String) 将字符串按照指定的内容劈开

和判断有关的方法：

返回类型 方法签名 作用

boolean equals(String) 判断两个字符串的内容是否一模一样

boolean equalsIgnoreCase(String) 忽略大小写的比较两个字符串的内容是否一模一样

boolean contains(String)判断字符串里面是否包含指定的内容

boolean startsWith(String)判断字符串是否以指定的内容开头

boolean endsWith(String) 判断字符串知否以指定的内容结尾

和改变内容有关的方法：

和改变内容有关的方法都不会直接操作原本的字符串

只是将符合条件的字符串返回给我们

所以需要注意接收

返回类型 方法签名 作用

String toUpperCase() 将字符串转化成大写的

String toLowerCase() 将字符串全部转换成小写的

String replace(String,String)将字符串里面出现的某个内容全部替换成指定的内容

String replaceAll(String,String)将字符串里面出现的某个内容全部替换成指定的内容 [支持正则表达式]

String replaceFirst(String,String)将字符串里面第一次出现的某个内容替换成指定的内容

String trim() 去除字符串的前后空格

String substring(int x,int y)从下标x截取到下标y-1对应的元素

String substring(int x)从下标x截取到字符串的最后

和位置有关的方法：

返回类型 方法签名 作用

char charAt(int) 找到指定下标对应的字符

int indexOf(String) 找到执行内容第一次出现的下标

int lastIndexOf(String)找到指定内容最后一次出现的下标

===========================================================================================

面向对象的高阶特征：

访问权限修饰符：

private: 私有的 只有本类可以访问

default: 默认的 只有本包内可以访问

protected:受保护的 只有本内可以访问 包外有继承关系的子类中

public: 公共的 谁都可以使用的

各自能修饰的东西：

类 成员（属性/方法）

private: F T

default: T T

protected: F T

public: T T

面试题：

A类里面定义一个public修饰的方法

问在B类里面能不能访问到？

不一定

如果B类想要访问到A类里面的方法

必须在B类里面能够先访问到A类

然后在访问A类里面的方法

static : 静态的 修饰符 相当于汉文中的形容词

属性：静态属性

特点：整个类型共享一份的属性，不是每个对象都有一个的属性，需要拿着类名去调用

普通属性：表示一个对象有一份的属性，需要拿着对象去调用

System.out(static)

面试题：

java中变量的概念大 还是属性概念大？

变量

static为什么不能修饰局部变量？

static修饰的变量要求类一加载的时候，内存里面就要创建，但是局部变量当方法调用的时候，内存里面才会开辟空间，类的加载永远在前面，方法调用在后面，这两个时间点赶不上一起，所以static不能修饰局部变量

方法：静态方法

特点：拿着类名去调用 静态方法里面只能直接的访问静态的成员

Math.random()

System.currentTimeMillis()

System.arraycopy()

面试题：

1：普通方法调用起来简单还是静态方法？

静态方法

2：问什么不把一个类里面所有的方法都定义成静态的呢？

静态方法虽然调用起来简单，但是静态方法里面访问的东西比较有局限，静态方法里面只能直接的访问静态的成员，如果想要在静态方法里面访问非静态的成员，需要创建对象，拿着对象去调用

普通方法里面既可以直接的访问静态的成员，又可以直接的访问非静态的成员

{}

代码块：初始化一个普通属性的

当我们想初始化一个静态属性的时候

需要用静态代码块 -》 static{}

普通代码块：给普通属性进行赋值 普通属性是每个对象都有一份的属性

创建对象的时候执行 创建几个对象执行几次

静态代码块：给静态属性进行赋值，静态属性是整个类型共享一份的属性，当类第一次被加载的时候执行，而且只执行一次

模式：java中本没有模式的概念，用的程序员多了于是有了模式 -------周树人

单例模式：控制一个类型只能创建一个对象（有且只有一个对象）

1：私有化构造方法 防止类体之外别人随意的创建对象

2：创建一个私有的 静态的 属于本类类型的对象

3：提供一个公共的 静态的 返回本类对象的方法

TreeSet（唯一） -> 二叉树 -> 比较器（单例模式）

final: 最终的 修饰符 相当于汉文中的形容词

类：最终类 "骡子类"

特点：不会有子类 但是可以有父类

面试题：

String类和Math类在开发的时候 Sun公司为什么都有final进行修饰？

String类表示字符串类，在开发的时候是很基础很核心的一个类，越是基础，越是核心的类所有的程序员月应该保持一致

Math类表示数学类，里面定义的都是定理 公理，不能按照程序员自己的意愿进行修改

方法：最终方法

特点：不能被覆盖 可以被继承

Sun公司不在乎有多少人通过继承得到最终方法，但是需要保证的是这个方法是最后一个版本，不能在修改

变量（属性+局部变量）：最终变量 常量

特点：一旦赋值 就不能在修改值

final:基本数据类型 -》 值

final:引用数据类型 -》 地址

final double π = 3.14;

abstract : 抽象的 修饰符 相当于汉文中形容词

类：抽象类 这个类型不形象 不具体

特点：不允许创建对象

面试题：

1：抽象类是类吗？

是

2: 抽象类有构造方法吗？

有

java中只要是个类就一定有构造方法

3：抽象类是类，有构造方法，但是不能创建对象，抽象类里面的构造方法用来干什么？

给子类的构造方法首行的super()去调用的

方法：抽象方法，表示这个类型一定会这个方法，但是现在给不出具体的实现，待留给子类去实现

特点：不能方法体 直接()结束;

\*：一个类里面只要出现了抽象方法，那么这个类一定要变成抽象类

\*：抽象类里面既可以定义抽象方法，有可以定义普通方法

===========================================================================================

接口：interface 相当于工业生产中的规范

java中的第二大类型：(编译之后 生成对应的.class文件)

class interface enum @interface

类 接口 枚举 注解

如何定义接口：

interface XXX{

//属性

//接口里面定义的属性默认加上三个修饰符：

//public static final

int x = 45;

String name = "xxx";

//方法：

//接口里面定义的方法默认加上两个修饰符：

//public abstract

void test();

int show();

}

面试题：

1：两两之间的关系：

类和类之间的关系：extends

类和接口之间的关系：implements(实现 遵循)

接口和接口之间的关系：extends

java中的类允许单根继承

java中的接口允许多重继承

java中的类可以在继承一个类的同时实现多个接口

2: 当方法覆盖的时候 需要加@Override

类和类之间的方法覆盖 ： jdk5.0开始

类和接口之间的方法覆盖 ： jdk6.0开始

3：接口不能创建对象

4：接口和抽象类之间的区别？\*\*\*\*\*

A:分别是java中的两个类型：

接口：interface

抽象类：class

B:里面定义的属性不同

接口：里面定义的所有属性默认都是静态的最终变量：public static final

抽象类：里面定义的所有属性默认就是普通属性

C:里面定义的方法不同

接口：里面只能定义抽象方法

抽象类：里面既可以定义抽象方法 又可以定义普通方法

Object:大

clone():"克隆"一个对象的方法

1:先覆盖这个类的clone方法

public XXX clone()throws CloneNotSupportedException{

Object obj = super.clone();

return (XXX)obj;

}

2:当前类需要实现一个接口

class XXX implements Cloneable{

}

finalize():对象的"遗言"方法 -> gc

finalize方法是当对象要被gc回收的时候，会主动调用的一个方法

toString():制定一个对象打印显示的内容，当我们要打印一个引用数据类型的对象的，如果在没有覆盖toString的时候 打印：类型@XXX

如果打印的内容想要看的懂 -》 toString()

public String toString(){

return XXX;

}

equals():制定一个类型的比较规则

当我们向要将内存里面不同的两个对象视为相等对象，需要覆盖equals，如果equals方法没有覆盖，比较两个对象的地址

hashCode(): 制定一个对象的散列特征码 -> HashSet HashMap

散列:将一大组数据分散为不同的小组

注意：当此方法(equals)，被重写时，通常有必要重写hashCode方法，以维护 hashCode 方法的常规协定，该协定声明相等对象必须具有相等的哈希码。

===========================================================================================

集合：装多个元素：容器 可以装不同类型的元素 + 个数没有限制

数组：装多个元素：容器 必须装类型相同的元素 + 个数限制

JCF:java Collections FrameWork

java集合框架

Collection Map

[单值类型的集合] [键值对类型的集合]

List Set SortedMap

[有序] [无序]

[不唯一] [唯一]

SortedSet

[有序]

[唯一]

面试题：

List Set Map是不是属于同一个等级？

List Set属于Collection的子接口

Collection和Map属于同一等级

List:

ArrayList:\*\*\*\*\*

特点：有序 不唯一

底层数据结构：数组实现的

ArrayList0:包装类

第一个作用：

由于ArrayList底层基于Object[]实现的，所以所有的引用数据类型都可以存放在集合里面，但是基本数据类型不能存放，为了保证集合里面，也可以装基本数据类型 -》 包装类

int x = 45;

//基本数据类下-》 包装类

//jdk5.0之前

Integer y = Integer.valueOf(x);

//jdk5.0开始

Integer y = x;//自动打包

Double x = new Double(45.5);

//包装类 -》 基本数据类下

//jdk5.0之前

double y = x.doubleValue();

//jdk5.0开始

double y = x;//自动解包

Integer会自动缓存：-128到127之间所有的数字

第二个作用：

包装类里面提供了一个方法(parseXXX)

将String类下转换成对应的基本数据类型

String x = "45";

int y = Integer.parseInt(x);

ArrayList1:集合的基本用法及和特点

1：如何创建ArrayList对象

jdk5.0之前 默认往集合里面添加的都是Object类型

ArrayList list = new ArrayList();

jdk5.0开始 可以泛型

泛型：规定集合里面只能装某种数据类型

ArrayList<泛型> list = new ArrayList<泛型>();

jdk7.0开始 后面的泛型会自动推断

ArrayList<泛型> list = new ArrayList<>();

2:如何往集合里面元素：

A:一次添加元素：

list.add(元素);

B:一次添加多个元素：

Collections.addAll(集合,元素1,元素2,元素3....);

面试题：

Collection和Collections之间的区别？

Collection是单值类型同一的父接口：interface

Collections是集合的工具类：class

3:如何得到某一个元素：

集合对象.get(下标);

4:如何得到集合的元素个数：

集合对象.size()

5:如何判断集合里面是否存在某一个元素：

集合对象.contains(元素);

6:如何遍历集合对象

A:for + 下标

for(int x = 0;x < list.size();x++){

//x -> 下标

//list.get(x) -> 元素

}

B:foreach

for(泛型 x : list){

//x -》 元素

}

C:迭代器

for(Iterator<泛型> car= list.iterator();car.hasNext();){

//car.next() -> 元素

}

===========================================================================================

ArrayList1:基本用法

1：创建对象

jdk5.0

ArrayList<泛型> list = new ArrayList<泛型>();

jdk7.0

ArrayList<泛型> list = new ArrayList<>();

2:如何往集合里面添加元素

ArrayList -> add(元素)

list.add(元素);

Collections -> addAll()

Collections.addAll(集合,元素1,元素2....)

class Collections{

public static void addAll(ArrayList<Object> list,Object ... data){

for(Object obj : data){

list.add(obj);

}

}

}

3:如何得到集合的元素个数

list.size();

4:如何得到某一个元素：

list.get(下标)

5：如何判断集合里面是否出现某一个元素：

list.contains(Object obj)

6:遍历

A:for + 下标

for(int x = 0;x < list.size();x++){list.get(x) -> 元素}

B:foreach

for(泛型 x : list){x -> 元素}

C:迭代器

for(Iterator<泛型> car = list.iterator();car.hasNext();){car.next() -> 元素}

ArrayList2:如何从集合里面删除某一个元素：

list.remove(int 下标);

下标指向谁删除谁

清空集合对象：list.clear();

ArrayList3:如何从集合里面删除某一个元素：

list.remove(Object 元素);

指定元素进行删除

\*：一个remove的方法只能删掉一个元素

ArrayList4:

ArrayList集合里面有一个remove(Object obj)的方法

底层需要尊重equals的比较机制，参数obj只是一个参照物而已，底层会拿着参照物和集合

里面的每一个元素作equals比较，只要集合里面有一个元素和obj视为相等对象，那么将集合对应的元素删除

ArrayList5:

要被删除的对象会主动的调用他自己类的equals方法

和集合里面的每一个元素作比较

ArrayList6:\*\*\*\*\*

当我们用迭代器遍历集合的过程中，不允许对集合的整体，进行添加/删除操作，否则都会触发CME异常CME:ConcurrentModificationException(并发修改异常)如果需求一定要求我们一边遍历一边删除，那么只能使用

迭代器的删除方法：car.remove();

\*:foreach底层基于迭代器实现的 foreach -> CME

ArrayList7:

ArrayList list = new ArrayList();//[]:10

ArrayList list = new ArrayList(50);//[]:50

当我们创建一个数组类型的对象的时候 必须要明确底层空间大小

ArrayList底层基于Object[]实现的 所以当我们在创建

ArrayList对象的时候 其实也可以明确底层开辟多大空间

在ArrayList构造方法里面串参数

参数：底层数组对象开辟多大空间

当我们调用ArrayList类里面的无参构造方法的时候 底层

默认开辟10块连续的存储空间 不代表只能装10个元素

集合里面可以装无数个元素：

集合会自动扩容：

jdk6.0及之前 x \* 3 / 2 + 1 10-》16->25....

jdk7.0及之后 x + (x >> 1) 10->15->22->33....

java开发的时候尽量表面扩容：

1：创建出来一个更大的数组对象

2：将老数组里面的元素复制进新数组里面

3：改变引用指向

4：回收老数组对象

5：继续添加元素

扩容：list.ensureCapacity(int 容量);

缩容：list.trimToSize();

ArrayList8：模拟实现ArrayList底层方法

===========================================================================================

List：

ArrayList \*\*\*\*\*

Vector:语法和ArrayList一模一样

面试题：\*\*\*\*\*

ArrayList和Vector之间的区别？

1:同步线程不同

ArrayList同一时间允许多个线程进行访问

效率较高 但是可能会出现并发错误

Vector同一时间允许一个线程进行访问

效率较低 但是不会出现并发错误

从jdk5.0开始 集合的工具类Collections里面提供一个方法（synchronizedList），可以将线程不安全的ArrayList变成线程安全的集合对象，于是Vector渐渐被淘汰

2:扩容机制不同

ArrayList:分版本

jdk6.0及之前 x \* 3 / 2 + 1

jdk7.0及之后 x + (x >> 1)

Vector:分构造方法

Vector(10) -> 2倍扩容 10 -》 20 -》 40...

Vector(10,3) ->定长扩容10 -》 13 -》 16

3:出现的版本不同

Vector:since jdk1.0

ArrayList:since jdk1.2

LinkedList:语法和ArrayList一模一样

面试题：

ArrayList和LinkedList之间的区别？

ArrayList和LinkedList底层数据结构不同

数据结构不同 导致优劣势不同

ArrayList:底层基于数组实现的

优点：随机访问 遍历查找

缺点：添加、删除元素

创建新的数组对象

将老数组赋值过去

改变引用指向

继续添加元素

LinkedList:底层基于链表实现

优点：添加、删除元素 （改变引用指向）

缺点：随机访问 遍历查找

\*：LinkedList里面尽量避免使用get(int 下标)

Stack:用数组模拟栈结构