转载请注明链接：http://blog.csdn.net/feather_wch/article/details/78733436

> 猎羽版Java百题大全，目标是收录Java的22个分类中所有知识点。希望对希望掌握Java的人有一定帮助。因为都是本人一点点积累总结而来，难免会有错误和疏漏，希望能不吝惜赐教，非常感谢！

Java百题大全

版本：2018/8/2-1(15:41)

0-最基本知识点-15题

1、java特点

> 是一次编译到处运行

2、JDK组成

> 1. jre（java运行环境）

2. 类库（3600多种，常用160多）
3. 工具（javac.exe，java.exe）

3、java代码编译到运行流程：

1. 编写源文件(.java文件)
2. javac.exe生成字节码文件(.class文件)
3. 解释器(java.exe)将其加载到java虚拟机中运行(jvm)

4、下面基本类型是否使用正确

```
int a = 1.2; ×
int a = (int)1.2; √

float f = 3.4; ×  解析：3.4字面为double
float f = 3.4f; √

int a = 3;
int b = a + 3.4; × 解析：a自动转为高精度
```

5、System.exit(0)是什么？非0代表什么？

1. System.exit(): 终止当前运行的java虚拟机
2. 0: 正常退出
3. 非0: 异常中止

6、封装体现在哪里？

1. 将具体实现保护在类内部
2. 访问控制修饰符
3. 包的功能

7、包的功能

1. 同名类区分开
2. 方便管理
3. 控制访问范围-使用方法：
   package com.xiaoqiang;
   import com.xiaoqiang;

8、继承要点

1. 子类最多继承一个父类
2. jdk所有类为object子类
3. 方法重载：子类不能缩小父类方法的访问范围

9、final 作用：功能（常量）

1. 方法不希望被overide
2. 变量不希望被改变
3. final class不能被继承

10、this

1. this用于指明具体的object
2. this只能用在非静态方法中。

11、什么是引用拷贝

1. 引用拷贝：对象之间直接等于。

12、什么是对象拷贝

1. 对象拷贝：使用 `clone` 进行拷贝。
2. 浅拷贝 和 深拷贝 都是 对象拷贝

13、浅复制(Shallow Copy)定义和实现方法

1. 定义： 浅拷贝 只会复制 主对象 ， 不会复制 主对象 里面的对象。
2. 成员变量 是 基本数据类型 ， 浅拷贝 具有两份不同的数据。对一个对象的成员变量进行修改，不会影响到另一个对象的成员变量。
3. 成员变量 是 引用数据类型 ， 浅拷贝 只是将 内存地址 复制一份给新对象，对其中一个对象的数据进行修改，会影响到另一个对象。
4. 实现方法：重写 clone 方法实现 浅拷贝 ： return super.clone()即可

```
public Student clone(){
    try {
        return (Student) super.clone();
    } catch (CloneNotSupportedException e) {
        e.printStackTrace();
    }
    return null;
}
```

巧记(浅烤定时) -> 浅浅的沙滩中烤贝壳吃，用定时器烤出最完美的味道。
1. 只复制主(制服之主) -> 【浅】水中，一个制服之主在玩cosplay
2. 基本,不影响(饥饿的笨蛋，布音响) -> 【烤】贝壳的时候，一个饥饿的笨蛋，想烤布音响吃。
3. 引用,内存地址,影响(饮用水，内存低质，音响) -> 大家约【定】，饮用水，来充当内存低质量的音响，来省钱。
4. 重写,clone,super,异常(重金编写，克隆羊多莉，超级，异常) -> 古【时】，就有人花费重金编写了一本书，里面介绍在克隆羊多莉，肚子里面有一个超级，异常的肿块，不就就会

14、深拷贝(deep Copy)定义和实现方法

1. 定义：会拷贝 对象 和 对象 内部所有的属性。(会为 引用数据类型 的对象开辟内存空间，而不是像 浅拷贝 一样只传递 内存地址)
2. 实现方法:
    1. 重写 clone 方法：依次将所有属性进行clone和set。
    2. 通过 对象序列化 实现深拷贝。

```
// 第一种：clone
        public Object clone(){
                Student s = null;
                try {
                        // 1. 浅拷贝
                        s = (Student)super.clone();
                } catch (CloneNotSupportedException e) {
                        e.printStackTrace();
                }
                // 2. 深拷贝
                s.setTeacher((Teacher)getTeacher().clone());//这里设置一下教师的克隆对象即可
                return o;
        }
```

巧记(深烤定时) -> 深深的沙滩中烤贝壳吃，用定时器烤出最完美的味道。
1. 对象 内部 属性(老婆，内部，树的形状) -> 【深】水中，老婆的内部变成了树的形状，触手
2. 引用 开辟 空间(饮用水，打开屁股，空中飞剑) -> 【烤】贝壳的时候，手拿着饮用水在喝，打开屁股放了个屁，空中飞来一把飞剑。
3. 重写 clone 属性 clone set(重金编写，克隆羊多莉，树的形状，cs) -> 约【定】，重金编写一个小说，clone羊和树形状的妖怪，在打CS比赛。
4. 序列化(井然有序的列兵，一人拿着一朵大红花) -> 【时】间到了，井然有序的列表，人手一个大红花，准备出征香港。

15、浅复制和深复制的区别

1. 速度： 深复制 相比于 浅复制 速度较慢，并且开销较大。
2. 基本数据类型 ： 两者的该成员变量都是两份不同数据，互不影响。
3. 引用数据类型 ：
    1. 浅拷贝的是同一份内存空间，互相影响。
    1. 深拷贝，是两份不同的数据，互不影响。
4. 实现方式： 深拷贝 在重写clone()方法的情况下，就是 浅拷贝 + 额外的引用对象的clone

巧记：浅深区别 ->
1. 速度-快-慢-开销大(飞机,...) -> 【浅】浅的水面上，飞机飞过，先快，再慢，导致油的开销很大。
2. 基本-均不影响(饥饿的笨蛋，军用的布音响) -> 【深深】的水面以下，饥饿的笨蛋，在找军用的布音响
3. 引用-相同-不同(饮用水壶，相同，...) -> 海底的【区】域中，只找到一堆饮用水壶，这些饮用水壶都是相同的，和要找的军用布音响完全不相同。
4. 实现-浅clone-属性clone(实线，...) -> 离【别】了海域，用一根实线，把浅滩上的clone羊给牵走，结果遇到了树形状的妖怪，把clone羊给抢走了。

1-对象导论-4题

1、OOP是什么?

> Object-oriented programming(OOP)面向对象编程。

2、 OOP五大基础特性

> - 万物都是对象---Everything is an object.
> - 程序由一堆对象组成，对象通过交互完成功能。
> - 创造的新对象可以由其他对象组成，这样将复杂内容隐藏在简单对象背后，就可以将复杂内容构建进程序中。
> - 每个对象都是一个class(类)的实例(instance)
> - 一个类的所有对象都可以接收同样的消息——例如circle对象也是shape对象，因此circle对象一定能接受shape的消息。这种可替换性也是OOP非常有用的概念

巧记：OOP基特-望远镜，看见旗子，插在基地里面，特别清楚。
1．万物（晚五） --- 晚上五点
2．程序，对象，交互（电脑，对象，焦糊） --- 电脑，老婆在写代码，有一股焦糊的味道。
3．新对象，复杂，简单，构建（新对象，父亲皇杂，煎鸡蛋，狗贱）--- 新对象，父亲是皇杂，在煎鸡蛋，和狗一样贱
4．类，实例（雷，视力）--- 一道雷电下来，视力都看不见了
5．类，对象，同样消息，替换性（雷，对象，童扬小溪，客人踢唤醒） -- 又一道雷电下来，对象跳了起来，看见童扬在小溪里面游泳，客人踢了我一脚，把我唤醒了，我在做梦呀。

3、 抽象是什么?

> 1. 汇编语言就是底层机器的抽象
> 2. C语言是汇编语言的抽象
> 3. C语言的抽象还是需要我们关注计算机的结构。C语言就是典型的面向过程编程。
> 4. OOP使得程序员不局限于具体的实现。

4、什么是对象?

> 一个对象拥有:
>
> 1. state---也就是拥有内部数据
> 2. behavior---方法
> 3. identity(身份)---每个对象拥有唯一的内存地址(用于区别于其他任何对象)。

2-一切都是对象-11题

1、java和C++的联系

> 1. java和C++都是混合语言
> 2. java是更纯的OOP语言
> 3. C++混合程度高，因为C++是向下兼容C语言的，使得一些C++的方面更加复杂。

2、内存有哪几部分？具体作用?

内存	位置	速度	存放对象	其他
register寄存器	处理器中	最快的内存		无法控制。数量非常有限。
stack	RAM中	仅次于寄存器	基本数据类型， 对象的引用	通过栈指针来分配和释放内存
heap	RAM中	仅次栈	所有对象	比stack消耗更多的功夫在资源调度和回收上。 不需要程序员控制资源释放。
常量内存(constant storage)			常量	有些常量在嵌入式系统中也会被选择性放到ROM中。
Non-Ram Storage（非RAM内存）			存放程序之外的数据， 比如数据库	

3、对象通过引用操作

4、 原始类型是什么？有什么用?

> 1. java提供了原始类型数据,不通过引用进行操作，而是直接拥有该数据
> 2. 原始类型数据都被存储在 `stack` 中。因此原始类型是非常高效的。

5、 原始类型有哪些？所占大小是多少?

> 8(原始数据类型) or 9(原始类型) 种

原始类型	字节	位
boolean	1	8bits
byte	1	8bits
char	2	16bits
short	2	16bits
int	4	32bits
long	8	64bits
float	4	32bits
double	8	64bits
void		

6、Java中所有的数都是有符号的

7、 Java中需要高精度数的时候，用什么对象?

> 1. BigInteger和BigDecimal
> 2. 可以和int、float结合使用，只是需要使用方法而不是操作符。

8、class由什么组成?

> 成员变量和成员方法

9、Java中如何处理数组越界

> Java会进行范围判定，越界后会抛出异常。

10、static的作用

> 1. 表明特定的成员变量和方法不依赖于具体的对象。也会被称为类数据和类方法(class data\class method)
> 2. static代码块, 是静态代码块
>   1. 类中可以有多个static代码块，在类加载的时候，会按照顺序依次执行一次。
>   2. static代码块适合只需要一次的初始化操作。
>   3. static代码块，父类先执行，子类后执行。父类的成员变量初始化和构造器，早于子类的成员变量初始化和构造器
> 3. 类数据和类方法，可以通过object或Class调用。
> 4. 静态类方法不能访问非静态方法和变量。

11、tags是什么

> 提供作者、版本、是否废弃等信息
> 如@author等等

3-操作符-4题

1、!,~作用

> !非，~按位取反

2、子面值是什么？内容?

```
0x2f 十六进制，float
0x2F float

0177 八进制
200L long int
200l long in
200  int
0.1F float
0.1f float
0.1D double
0.1d double

1.39 e-43f = 1.39 x 10-43 //科学计数法
```

3、位运算符

```
& 位与
| 位或
^ 位异或
~ 位取反
```

4、位移操作符

```
<< 补0
>> 补符号位0 or 1
>>> 补0
```

4-控制执行流程-2题

1、switch可以用于哪些数据类型

> 仅有五种：byte、short、int、char、enum

2、java中一共有哪些控制执行的方法

```
if-else
while
do-while
for
return
break
switch
continue
goto：只是保留字，还没有使用。
```

5-初始化与清理

6-访问权限-1题

1、public、private、protected、默认的区别

修饰符	同类	同包	子类	不同包
public	√	√	√	√
protected	√	√	√	
默认	√	√		
private	√			

7-复用类

8-多态-3题

1、什么是多态?

> 一个引用（类型）在不同情况的多种状态

2、抽象类?

> 形如：abstract class Animal(){};
>
> 1. 不一定有抽象方法

2. 不能被实例化
3. 子类必须实现抽象方法
4. 可以有变量
5. 抽象方法不能被实现

3、接口与抽象类区别

> 1. 接口是对继承的一种补充

2. 去掉接口不会影响其余代码?
3. 接口能继承接口，不能继承类

9-接口-7题

1、接口的作用

> 1. 接口中变量为 static,final int a = 0。常用作全局变量。例如：Usb.a

2. 接口可以继承接口
3. 体现了多态，以及高内聚低耦合的特点

2、接口和抽象类的区别

> 1. interface所有方法不能实现。是更为抽象的抽象类。

2. 类能implements 多个接口，但只能继承一个父类

3、接口是否会继承?

> 子类会继承父类关于接口的实现

4、 抽象类abstract

> - 一个class包含abstract方法，该class也必须为abstract类

- 不能实例化abstract类
- 一个具体类继承抽象类，必须实现抽象类的所有抽象方法
- 抽象类可以继承抽象类，并且可以不实现其中的抽象方法
- 抽象类可以不包含任何抽象方法。

5、 Interface

> - 在两个类之间建立一种协议“protocol”

- 能实现多重继承
- 接口可以有成员变量，但是会隐含的加上static和final

- 接口的方法methods必须为public

6、接口作用：

> 1. 解耦

2. 多重继承
3. 可以用于声明一组常量：接口中成员变量自动加上static和final。(这些数据会存储在静态存储区域)

7、类继承超类并且实现了另一个接口

> class继承class和实现interface，之前所有同名的方法，必须要可以通过参数区分，如果仅仅返回值不同，是无法区分会报错的。因此，不同Interface最好method name不同

8、抽象的定义和作用以及实现方法

> 1. 定义 ： 将一类事物的 共同特征 总结出来并且构造出一类事物的过程(数据抽象 和 行为抽象 ， 不会去关注 属性和行为 内部具体的细节)
> 2. 作用 ： 1-隐藏 具体实现 方式 2-提供 扩展的能力
> 3. 实现方式 ： 抽象类和接口

9、抽象类的特点

> 1. 抽象类 是 类对象的抽象集合
> 2. 抽象类 用关键字 `abstract` 修饰，不能用 `final` 修饰 抽象类 。
> 3. 抽象方法 不能用 `private`、`static`、`final`或`native` 修饰。
> 4. 包含 抽象方法 的类一定要是 抽象类
> 5. 抽象类 不一定要有 抽象方法
> 6. 抽象类 不能被实例化

10、接口的特点

> 1. 接口 是 类行为方法的抽象集合
> 2. 接口 用 `interface` 修饰
> 3. 接口 中所有方法都是 抽象且公有的
> 4. 接口 中的属性都是 全局变量 =public static final(不用写)
> 5. 接口 只能继承 接口
> 6. 接口 不能实例化-继承接口的 子类 可以实例化

11、接口和抽象类的区别

> 1. 抽象的层次: 1-抽象类抽象整个事务包括属性和方法 2-接口是对行为的抽象
> 2. 抽象类 单继承 ； 接口 多重继承
> 3. 抽象类 方法和属性 可以抽象或者不抽象；接口中的必须抽象
> 4. 抽象类的实现类可以改变 其成员变量 ， 接口的实现类不可以改变 属性=常量

12、子类继承父类的访问修饰符只能比父类的更大或者一致

10-内部类-4题

1、内部类和组合(Composition)的区别

2、内部类的作用

> - 代码隐藏机制：将class放置于其他class之间。
> - 内部类知道如何与周遭的class(surrounding class = 包含内部类的外部类)交流-【可以自由使用外部类的所有变量和方法】
> - 通过内部类写的代码是简洁清爽的

3、 内部类的一个特殊实例

> 如果你想在任何外部类的非静态方法以外的方法中获得内部类的对象，都需要调用 外部类名 + . 点+ 内部类名。 而且需要在外部类中额外创建一个返回该内部类的方法。

```
public class Test {

        class C{//内部类
        }
        public C getNewC() {//返回内部类的方法
                return new C();
        }

        public static void main(String[] args) {
                Test test = new Test();
                Test.C c = test.getNewC(); //在外部类的静态方法中获得对象
        }
}
```

4、内部类会拥有外部类的链接link(可能会导致内存泄露！)

> 一个内部类的对象会拥有其外部类的链接/引用/link，这样能用来访问该外部类的对象的成员。

5、内部类和静态内部类的区别

> - 使用方法不同

```
static class Outer {
        class Inner {}
        static class StaticInner {}
}

Outer outer = new Outer();
Outer.Inner inner = outer.new Inner();
Outer.StaticInner inner0 = new Outer.StaticInner();
```

> - Oracle说法：一个被称为静态嵌套类，一个被称为内部类。
> 嵌套就是指两者没太大关系，静态内部类可以完全独立存在，但是就是借用外部类的壳来隐藏自己。

> - 静态内部类起到了一种注释的效果。A.B表明B类与A相关。且降低包的深度，方便类的使用

11-集合-16题

1、集合有哪些？(9)

> 1. ArrayList
> 2. LinkedList
> 3. SynchronizedList
> 4. Vector
> 5. Stack
> 6. Hashtable
> 7. HashMap
> 8. ConcurrentHashMap
> 9. HashSet

2、集合的包

> java.util.*

3、集合的区别

	线程安全性	初始容量	扩容机制	其他
ArrayList	×-异步	0+10(第一次add时)	1.5 * n	
LinkedList	×-异步	不存在	不存在	
SynchronizedList	√-同步	和原始List一致	和原始List一致	
Vector	√-同步	10	2 * n	
Stack	√-同步	10	2 * n	

	线程安全性	初始容量	扩容机制	其他
Hashtable	√-同步	11	2 * n + 1	
HashMap	×-异步	16	2 * n	
ConcurrentHashMap	√-同步	16	2 * n	

> 1. 线程安全的有几种集合：5种
> 2. 非线程安全的有几种集合：3种

ArrayList

4、ArrayList的特点(7)

> 1. `ArrayList` 基于 数组实现(底层是`Object`数组) ,访问元素效率高，插入删除元素的效率低。
> 2. 实现 `List`接口 ： 意味着 有序、可以重复、可以有`null`元素
> 3. 实现 `RandomAccess`接口 ： 意味着 `ArrayList` 支持 快速随机访问 ， `Collections` 对于实现了 `RandomAccess`接口 的List会使用 索引二分查找算法 进行查找。
> 4. 实现 `Cloneable`接口 ： 标识着可以被辅复制。 `ArrayList` 的 `clone()`复制 其实是 浅复制
> 5. 实现 `Serializable`接口 ： 标识着集合可以被序列化。
> 6. 绝大多数情况下应该指定 `ArrayList` 的 初始容量 ： 避免进行 耗时的扩容操作 ， 在能 预知 的情况下尽量使用刚刚好的 列表 ， 从而不浪费任何资源。
> 7. 非线程安全

5、ArrayList的扩容机制

> 1. `ArrayList` 的默认容量为 `10` (本质默认初始化为 空数组 ， 第一次 `add` 时会扩容到10)
> 2. 扩容 以 `1.5`倍 进行扩容。
> 3. 扩容 ： 会创建一个更大的数组，然后将旧数组的数据都复制过去，该过程是 低效率 的

6、ArrayList源码总结出的效率技巧

> `List` 提供了 `indexOf`和`lastIndexOf` 查询 指定元素的索引(`index`) ， 这些方法是从头和从尾部一个个查询的。显然可以直接使用 `Collections`的`binarySearch` 通过二分查找，效率会更高。

Vector

7、Vector的特点

> 1. 基于 数组实现
> 2. 支持 快速随机访问
> 3. 线程安全：在不需要 同步时 ， 性能比 `ArrayList`低 ， 需要同步时使用 `SynchronizedList` 更好。
> 4. 扩容机制 ： 每次扩容都是原来容量大小的 `2`倍
> 5. 适合大量数据的时候。

Stack

8、Stack的特点

> 1. `Stack` 继承自 `Vector` ， 扩展了 栈 的相关方法。
> 2. 具有 `pop`、`push`、`peek` 等方法(`LinkedList` 也有这些方法)。

LinkedList

9、LinkedList的特点

> 1. `LinkedList` 基于 链表实现 ， 插入和删除效率高，访问元素效率低(这是链表的特点)
> 2. `LinkedList` 可以作为 队列、栈 来使用。
> 3. 队列 （ 先进先出 ） ： offerFirst、pollLast
> 4. 栈 （ 后进先出 ） ： pop、push
> 5. 非线程安全

SynchronizedList

10、SynchronizedList的特点

> 1. 线程安全
> 2. 通过 `Collections.synchronizedList(new ArrayList<>())`-参数传入任何`List` 就会返回 `SynchronizedList` ， 使得该 `List` 是线程安全的。

Hashtable和HashMap

11、Hashtable和HashMap的区别

> 1. 作者： `HashMap` 的作者比 `Hashtable` 的作者多了一个人： `Doug Lea` 写了 `util.concurrent` 包并且著有 并发编程圣经：`Concurrent Programming in Java`
> 2. 诞生时间： `HashMap` 产生于 `JDK1.2` 相比于 `Hashtable` 更晚。
> 3. 弃用状况： `Hashtable` 基本上已经被弃用：1- `Hashtable` 是 线程安全 ， 效率比较低。2- `Hashtable` 没有遵循 驼峰命名法
> 4. 父类： `HashMap` 继承自 `AbstractMap` , `Hashtable` 继承自 `Dictionary`
> 5. 接口数量： `Hashtable` 比 `HashMap` 多剔红了 两个接口 ： elements和contains
> 6. key和value是否为null： `Hashtable` 不允许 `key`和`value` 为 `NULL` ， `HashMap` 支持： `key=null`的键只能有一个 ， `get()`返回为`null`，可能是`value`为`null`，也可能是没有该`key`，需要通过`containKey`来判断是否具有某个`key`
> 7. 线程安全性- `Hashtable` 是 线程安全(每个方法都加入`Synchronized`) ， `HashMap` 是 非线程安全 的。 `HashMap` 效率比 `Hashtable` 高很多，而且需要自己进行同步处理。如果需要 线程安全 可以使用 `ConcurrentHashMap` ， 也比 `Hashtable` 效率高很多倍。
> 8. 遍历方式- `Hashtable` 使用老旧的 `Enumeration` 的方式， `HashMap` 使用 `Iterator`迭代器
> 9. 初始容量和扩容方式： `Hashtable`初始为`11` ， 扩容是 `2 * n + 1` ， `HashMap`初始为`16` ， 扩容是 `2 * n`
> 10. 计算 `hash`值 的方式不同： `HashMap` 比 `Hashtable` 的计算效率更高。（涉及到位运算，以及通过额外计算打散数据来减少hash冲突的问题）
> 相同1： 两者都实现了：Cloneable(可复制)、Serializable(可序列化)

HashSet和HashMap

12、HashSet和HashMap的区别与相同

HashMap	HashSet
实现了Map接口	实现Set接口
存储键值对	仅存储对象
调用put（）向map中添加元素	调用add（）方法向Set中添加元素
HashMap使用键（Key） 计算Hashcode	HashSet使用成员对象来计算hashcode值，对于两个对象来说hashcode可能相同 (因此使用前要确保重写hashCode（）方法和equals（）方法)
HashMap较快， 因为它是使用唯一的键获取对象	HashSet较慢
非线程安全	非线程安全

13、要求线程安全用什么?

> Vector、hashtable、SynchronizedList、ConcurrentHashMap

14、没有线程安全要求用什么?

> ArrayList、LinkedList、HashMap

15、有键值对?

> HashMap、HashTable

16、数据量很大且要求线程安全

> Vector

synchronizedMap

ConcurrentHashMap

CopyOnWriteArrayList

12-异常-15题

1、Throwable和三大异常子类

> Throwable是所有异常和错误的超类。包含三个异常子类。
>
> 1. Error
> 2. Exception
> 3. RuntimeException:如NullPointerException等异常都自动由java抛出

2、4种finally不会被执行的情况

> 1.system.exit()
> 2.线程异常
> 3.finally异常
> 4.CPU关闭

3、为什么需要异常处理?

> 工程师最理想状态是在编译时就能发现所有错误，但是不是所有的错误都可以被找到。这些错误需要在运行时被处理，这就需要异常处理机制。
> 异常作用：在合适层级修复问题，以完成预期功能，并提高系统的健壮性、安全性。

4、 异常处理的优点

> 1.提高系统的稳健性。
> 2.减少处理错误的代码的复杂度。
> 3.不需要再在方法调用中去处理，而是直接在exception handler去处理。

5、 为什么要throw exception

> 当前异常的环境无法处理该异常，就需要将问题提交给更高层去处理。
> 抛出异常也可以看做一种return机制，当抛出异常时，其所处的方法就会退出。

6、抛出异常时的流程

> 1. exception对象会被创建，如同其他java对象被new创建一样，创建在堆heap中。
> 2. 当前异常的路径会停止，并且异常对象的引用会被当前context抛出。
> 3. 此时，异常处理机制(exception-handling mechanism)会接管，并且开始寻找合适的地方去继续执行程序。
> 4. 合适的地方-exception handler，异常处理，会进行问题修复。

7、 try-catch和switch区别

> try-catch由上至下匹配异常类型，一旦匹配到就会进入相应的catch进行处理，处理完成后不会继续向下匹配。而switch必须通过break来停止流程。

8、 自定义异常

> `class SimpleException extends Exception{}`,一般都不需要自己实现内容，直接继承Exception即可。一个异常最重要的部分就是类名。

9、 catch中rethrow异常

```
catch(exception e){
  throw e;
}
```

这里再次抛出的异常e，本身类型是其初始的类型，而不会变成Exception类型。这种特性被称为-exception chaining,异常链

10、finally作用

> 1. 执行无论异常与否都必须执行的代码。

> 2. 可以用于文件、网络连接等内容的清理工作。
> 3. 在涉及到break和continue的部分，finally也一定会执行(在java中都可以消除goto的需求)

11、 finally和return

> 1. 在finally之前的任何return都不会影响finally块的执行。

12、 finally缺陷:会丢失异常

> 1.finally中抛出异常可能会导致丢失异常

```
try {
    try {
            //抛出异常a
    } finally {
            //抛出异常b
    }
} catch (Exception e) {
     //获得异常b，而异常a丢失
}
```

> 2.finally中return

```
try {
      throw new RuntimeException();
} finally {
      //在finally中return，会导致本该抛出的异常却丢失了
   return;
}
```

13、exception 限制

> 子类重写的方法，抛出的异常不能比superclass该方法抛出的异常范围大。
> 子类重写的方法，抛出的异常要在superclass该方法抛出的异常范围之内。
> 子类构造器中，可以抛出任何范围的异常，而不会受到限制。

14、Constructors(构造器)中产生异常，如何关闭需要清理的资源，如文件?

> 绝对不能在finally中清理资源。
>
> 1. 这样会导致无论异常与否资源都会被清理
> 2. 而且前面的异常可能会导致需要被清理的资源本质上也没创建。

15、不知道如何处理某些异常：将其转换为系统处理的异常

> throw new RuntimeException(e); //不知道如何处理，直接转换为Runtime异常抛出。

13-String-10题

1、String比较

> astr.equals(bstr) 比较对象是否相等
> == 仅仅是比较首地址

2、String的特点(4)

> 1. `String` 是 不可变的 ： 修改`String`不会在原有的内存地址修改，而是重新指向一个新对象
> 2. `String` 用 `final`修饰 ， 不可以 继承 ： String本质是 `final`的`char[]`数组 ， 所以 `char[]` 数组的内存地址不会被修改，而且 `String` 没有对外暴露修改 `char[]`数组 的方法。
> 3. `String` 的 不可变性 ： 可以保证 线程安全 以及 字符串常量池 的实现。
> 4. 频繁的 增删操作 不建议使用 `String`

3、StringBuffer的特点(1)

> 1. 线程安全 ： 适用于 多线程

4、StringBuilder的特点(1)

2. 非线程安全 ： 适用于 单线程

5、String的内部原理

1. String 内部是一个 final修饰 char[]数组 ： 名称为 value
2. String的构造 本质上就是对 value数组 赋初值的过程。
3. String 的其他API本质都是对 value数组 操作的过程。如： substring 是通过 数组的复制 完成的，最终会 返回 新建的 String ， 不会影响到原有的数组。

6、String API的分类（12）

1. 构造方法
2. 字符串长度：length
3. 字符串某一位置：charAt
4. 提取子串：substring
5. 字符串比较：compareTo、compareToIgnore、equals、equalsIgnoreCase
6. 字符串的连接：concat
7. 字符串的查找：indexOf、lastIndexOf
8. 字符串的替换：replace
9. 前后空格的移除：trim
10. 起始字符串/终止字符串是否与给定字符串相同：startsWith、endWith
11. 是否包含字符串：contains
12. 基本类型转换为字符串类型：valueOf

7、AbstractStringBuilder是什么（2）

1. StringBuider 和 StringBuffer 都是继承自 AbstractStringBuilder
2. 内部是一个char[]数组 ， 没有final 修饰符。

8、AbstractStringBuilder的API分类（7）

1. 扩容：ensureCapacity、newCapacity、hugeCapacity--- 扩容方式：以前大小 * 2 + 2
2. 追加：append（容量不够就扩容，容量够就追加到 value数组 的最后）
3. 插入字符串：insert
4. 删除字符串：delete（删除[start, end]之间的字符，通过复制的方式实现）
5. 提取子串：substring(new一个新String)
6. 字符串的替换：replace
7. 字符串某位置的字符：charAt

9、StringBuffer的特点（3）

1. 继承自 AbstractStringBuilder
2. 构造 默认是创建 容量为16 的 AbstractStringBuilder
3. 线程安全： StringBuffer 的所有方法，都是直接使用父类的方法，并都是用 synchronized 进行加锁保护。

10、StringBuffer的特点（4）

1. 继承自 AbstractStringBuilder
2. 非线程安全 ： StringBuilder 的所有方法，都是直接使用父类的方法，但是没有使用 synchronized 进行加锁保护。

14-类型信息

15-泛型-13题

1、多元组tuple是什么？作用?

将一组对象包装到一个对象中。用于一些methods需要返回多个对象的时候。

2、在同时有需要两个、三个等更多返回对象的时候怎么处理?

实现TwoTuple，然后Three、Four层层继承(类似于装饰者模型)

3、泛型的典型应用：stack、List等数据结构

4、泛型接口的定义和作用

> public interface Generator...
> 泛型接口常被用于各种类的生产器中。

5、泛型方法定义

> 泛型方法需要将泛型参数列表，放置于返回值前(`public <T> void genericMethod(T a)`)

6、泛型方法和泛型类的区别

> 1. 泛型类需要在实例化时指定参数的类型
> 2. 泛型方法不需要指明，编译器会自动识别。

7、泛型方法可以和可变参数完美配合

8、泛型的边界限制

> `class Colored<T extends MyColor>` ,T必须为MyColor的子类

9、泛型边界限制同时有类和接口

> `<T extends MyClass & MyInterface>` 要先类后接口

10、泛型限制在泛型方法中的使用

> `public <T extends Number> T showKeyName(Generic<T> container){...}` 需要在泛型列表中进行限制，而不能在参数中限制。

11、泛型通配符

> ?表示是所有类型的超类

12、静态方法与泛型

> 如果静态方法要使用泛型，则必须将静态方法变成泛型方法。

13、泛型数组：Java终不能创建一个确切的泛型类型的数组

```
List<String>[] ls = new ArrayList<String>[10];  × 不可以
List<?>[] ls = new ArrayList<?>[10];   √ 可以
List<String>[] ls = new ArrayList[10]; √ 可以
```

16-数组-2题

1、基本数据类型数组

```
int a[] = {1,2,3};
int a[] = new int[3];
int []a = new int[3];
```

2、对象数组定义

> 对象数组仅仅存放引用，定义数组后还需要给每个元素new一个对象。

17-容器

18-I/O

19-枚举

20-注解-7题

1、注解作用

> 为源代码文件提供元数据

2、注解原则

> 注解不能干扰程序的运行，无论增加或者删除注解，代码都能运行。

3、注解的分类：

> 标准注解(没有元素的注解)
> 元注解(负责注解其他注解)

4、元注解的作用：

> 编写文档、代码分析、编译检查

5、提取注解

> java.lang.reflect.Field有getDeclaredFields()、field.isAnnotationPresent、field.getAnnotation..等方法能够提取注解(参考下面实例)

6、标准注解有哪些？作用

> @Override, 表明一个重载自父类的方法，避免出现方法名错误或者参数写错。
> @Deprecated, 当该元素被使用的时候产生警告。
> @SuppressWarnings， 镇压/关闭掉不合适的编译器警告。

7、元注解有哪些？作用

> @Target 表明该注解用在哪里
> @Retention 注解信息持续多久(会被编译器丢弃，还是运行时还存在，等等)
> @Documented 在javadocs中包含该注解
> @Inherited 允许子类基础父类的注解

21-并发

22-反射-5题

1、Java中识别对象和类信息的两种方法?

> 传统RTTI(它假定我们在编译时已经知道了所有的类型信息)；反射机制(它允许我们在运行时发现和使用类的信息)

2、RTTI(RunTime Type Information)是什么?

> 允许你在程序运行的时候发现和使用类型信息。会在所有类第一次使用的时候，将class对象(保存在.class文件)动态加载到JVM。都必须在编译时已知。

3、 反射机制实现方法

> class类和reflect类库对反射进行了支持，通过里面的Field、Method和Constructor等类，能够确定对象信息，而在编译时不需要知道类的任何事情。但是反射的.class文件在运行的时候也必须是已知的，.class文件可以在本地也可以通过网络获得。

4、 反射和RTTI的本质区别

> RTTI：编译器在编译时打开和检查.class文件
> 反射：运行时打开和检查.class文件

5、JVM加载对象的方法

> 类的信息会存在方法区，类加载器会通过方法区的类信息，在堆Heap上创建一个类对象(Class对象)，这个类对象是唯一的，后续的New等操作都是通过这个唯一的类对象作为模板去进行new等操作。

扩展资料

1. Android程序员-Java面试题