Java复习重点

Java复习 面向对象 多线程 IO 集合 泛型 反射 网络编程 设计模式

基础

自动递增/递减

- **前缀式**
  前缀式：先执行运算，再生成值
- **后缀式**
  后缀式：先生成值，再执行运算

String、StringBuffer和StringBulider

- **String**：提供值不可改变的字符串
- **StringBuffer**:提供值可改变的字符串，线程是安全的，效率低下
- **StringBulider**:提供值可改变的字符串，线程不安全，效率高.在不考虑线程安全的情况下优先使用它

Java 中基本类型和字符串之间的转换

其中，基本类型转换为字符串有三种方法：

1. 使用包装类的 toString() 方法
2. 使用String类的 valueOf() 方法
3. 用一个空字符串加上基本类型，得到的就是基本类型数据对应的字符串

```
//将基本数据类型转换为字符串
int c=10;
String str1=Integer.toString(c);
String str2=String.valueof(c);
String str3=c+"";
```

再来看，将字符串转换成基本类型有两种方法：

1. 调用包装类的 parseXxx 静态方法
2. 调用包装类的 valueOf() 方法转换为基本类型的包装类，会自动拆箱

```
String str="8";
int a=Integer.parseInt(str);
int b=Integer.valueof(str);

注：其他基本类型与字符串的相互转化这里不再一一列出，方法都类似
```

Java中常用类

- **String类**
  String 类提供了许多用来处理字符串的方法，例如，获取字符串长度、对字符串进行截取、将字符串转换为大写或小写、字符串分割等，下面我们就来领略它的强大之处吧

方法	说明
int length()	返回当前字符串的长度
int indexOf(int ch)	查找ch字符在该字符串中第一次出现的位置
int indexOf(String str)	查找str子字符串在该字符串中第一次出现的位置
int lastIndexOf(int ch)	查找ch字符在该字符串中最后一次出现的位置
int lastIndexOf(String str)	查找str子字符串在该字符串中最后一次出现的位置
String substring(int beginIndex)	获取从beginIndex位置开始到结束的子字符串
String substring(int beginIndex, int endIndex)	获取从beginIndex位置开始到endIndex位置的子字符串
String trim()	返回去除了前后空格的字符串
boolean equals(Object obj)	将该字符串与指定对象比较，返回true或false
String toLowerCase()	将字符串转换为小写
String toUpperCase()	将字符串转换为大写
char charAt(int index)	获取字符串中指定位置的字符
String[] split(String regex,int limit)	将字符串分割为子字符串，返回字符串数组
byte[] getBytes()	将该字符串转换为byte数组

```
public class HelloWorld {
    public static void main(String[] args) {
        // Java文件名
        String fileName = "HelloWorld.jav";
        // 邮箱
        String email = "laurenyang@imooc.com";

        // 判断.java文件名是否正确：合法的文件名应该以.java结尾
        /*
        参考步骤：
        1、获取文件名中最后一次出现"."号的位置
        2、根据"."号的位置，获取文件的后缀
        3、判断"."号位置及文件后缀名
        */
        //获取文件名中最后一次出现"."号的位置
        int index = fileName.indexof(".");

        // 获取文件的后缀
        String prefix =fileName.substring(index+1);
        // 获取文件的后缀方式二
        String [] arr=fileName.split(".");//按特定符号将字符串分割
        String prefix2=arr[1];


        // 判断必须包含"."号，且不能出现在首位，同时后缀名为"java"
        if
(fileName.indexOf(".")!=-1&&fileName.indexOf(".")!=0&&prefix.equal
s("java")) {
            System.out.println("Java文件名正确");
        } else {
            System.out.println("Java文件名无效");
        }

        // 判断邮箱格式是否正确：合法的邮箱名中至少要包含"@"，并且"@"是
在"."之前
         /*
        参考步骤：
        1、获取文件名中"@"符号的位置
        2、获取邮箱中"."号的位置
        3、判断必须包含"@"符号，且"@"必须在"."之前
        */
        // 获取邮箱中"@"符号的位置
        int index2 = email.indexOf("@");

        // 获取邮箱中"."号的位置
        int index3 = email.indexOf('.');

        // 判断必须包含"@"符号，且"@"必须在"."之前
```

```
        if (index2 != -1 && index3 > index2) {
            System.out.println("邮箱格式正确");
        } else {
            System.out.println("邮箱格式无效");
        }
    }
}
```

```
public class HelloWorld {
    public static void main(String[] args) {
        // 定义一个字符串
        String s =
"aljlkdsflkjsadjfklhasdkjlflkajdflwoiudsafhaasdasd";

        // 出现次数
        int num = 0;


        // 循环遍历每个字符，判断是否是字符 a ，如果是，累加次数
        for (int i=0;i<s.length();i++)
        {
            // 获取每个字符，判断是否是字符a
            if (s.charAt(i)=='a') {
                // 累加统计次数
                num++;
            }
        }
        System.out.println("字符a出现的次数: " + num);

        // 将字符串转换为大写
        String str1=str.toUpperCase();

        //将字符串转化为字节数组byte[]
        byte [] b=s.getBytes();
        for(int i=0;i<b.length;i++){
            System.out.println(b[i]);
        }

        //==和equals的使用
        String s2=new
String("aljlkdsflkjsadjfklhasdkjlflkajdflwoiudsafhaasdasd");
        bolean value=s.equals(s1);//比较值是否相等
        bolean address=(s==s1);//比较地址是否相等，新new出来的地址肯定不
相等
    }
}
```

- **StringBuilder 类的常用方法**

  StringBuilder 类提供了很多方法来操作字符串：

方法	说明
StringBuilder append(参数)	追加内容到当前StringBuilder对象的末尾
StringBuilder insert(位置,参数)	将内容插入到StringBuilder对象的指定位置
String toString()	将StringBuilder对象转换为String对象
int length()	获取字符串的长度

```
public class HelloWorld {
    public static void main(String[] args) {
        // 创建一个空的StringBuilder对象
        StringBuilder str=new StringBuilder();

        // 追加字符串
        str.append("jaewkjldfxmopzdm");

        // 从后往前每隔三位插入逗号
        for(int i=str.length()-3;i>0;i-=3)

            str.insert(i,',');
        }

        // 将StringBuilder对象转换为String对象并输出
        System.out.print(str.toString());
    }
}
```

- **Date和SimpleDateFormat**

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class HelloWorld {

    public static void main(String[] args) throws ParseException {

        // 使用format()方法将日期转换为指定格式的文本
        SimpleDateFormat sdf1 = new SimpleDateFormat("yyyy年MM月dd
日 HH时mm分ss秒");
        SimpleDateFormat sdf2 = new SimpleDateFormat("yyyy/MM/dd
HH:mm");
        SimpleDateFormat sdf3 = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");

        // 创建Date对象，表示当前时间

        Date now=new Date();
        // 调用format()方法，将日期转换为字符串并输出
        System.out.println(sdf1.format(now));
        System.out.println(sdf2.format(now));
        System.out.println(sdf3.format(now));

        // 使用parse()方法将文本转换为日期
        String d = "2014-6-1 21:05:36";
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");

         // 调用parse()方法，将字符串转换为日期
        Date date =sdf.parse(d);

        System.out.println(date);
    }
}
```

- **Calendar**

1.Date 类最主要的作用就是获得当前时间，同时这个类里面也具有设置时间以及一些其他的功能，但是由于本身设计的问题，这些方法却遭到众多批评，不建议使用，更推荐使用 Calendar 类进行时间和日期的处理。

2.java.util.Calendar 类是一个抽象类，可以通过调用 getInstance() 静态方法获取一个 Calendar 对象，此对象已由当前日期时间初始化，即默认代表当前时间，如 Calendar c = Calendar.getInstance();

```
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

public class HelloWorld {

    public static void main(String[] args) {
        // 创建Canlendar对象
        Calendar c = Calendar.getInstance();

        // 使用Calendar获取年月日时分秒
        int year=c.get(Calendar.YEAR);
        int month=c.get(Calender.MONTH)+1;
        int day=c.get(Calendar.DAY_OF_MONTH);
        int hour=c.get(Calendar.HOUR_OF_DAY);
        int minute=c.get(Calendar.MIUNTE);
        int second=c.get(Calendar.SECOND);

        // 将Calendar对象转换为Date对象
        Date date = c.getTime();
        System.out.println("当前时间："+date);

        // 还可以Calendar的时间值，以毫秒为单位
        Long time=c.getTimeInMillis();
        System.out.println("当前毫秒数："+time);

        // 创建SimpleDateFormat对象，指定目标格式
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");

        // 将日期转换为指定格式的字符串
        String now = sdf.format(date);
        System.out.println("当前时间：" + now);
    }
}
```

- **Math类**
  Math 类位于 java.lang 包中，包含用于执行基本数学运算的方法， Math 类的所有方法都是静态方法，所以使用该类中的方法时，可以直接使用类名.方法名，如：Math.round();

返回值	方法名	解释
long	round()	返回四舍五入后的整数
double	floor()	返回小于参数的最大整数
double	ceil()	返回大于参数的最小整数
double	random()	返回[0,1)之间的随机数浮点数

```
public class HelloWorld {

    public static void main(String[] args) {

        // 定义一个整型数组，长度为10
        int[] nums = int[10];

        //通过循环给数组赋值
        for (int i = 0; i < nums.length; i++) {
            //随机数产生方法一
            // 产生10以内的随机数
            //其中Math.random()方法是一个可以产生[0.0,1.0]区间内的一个双精度浮点数的方法
            //产生一个1-50之间的随机数：int x=1+(int)(Math.random()*50)
            int x = (int)(Math.random()*10);

            nums[i] = x;// 为元素赋值
        }

        // 使用foreach循环输出数组中的元素
        for (int random:nums) {
            System.out.print(num + " ");
        }


        //  四舍五入
        double a=12.81;
        long c=Math.round(a);//结果为13
        double d=Math.floor(a);//结果为12.0  向下取整
        double e=Math.ceil(a);//结果为13.0  向上取整
    }

    //随机数产生方法二
    //通过java.util包中的Random类的nextInt方法来得到1-10的int随机数
    Random ra =new Random();
    for (int i=0;i<30;i++)
    {System.out.println(ra.nextInt(10)+1);}
    //Random可以产生随机整数、随机float、随机double，随机long，带种子和不带种子

}
```

抽象类与接口

1.抽象类由关键在abstract修饰

2.应用场景：

- 某个父类只是知道子类应该包含怎样的方法，但无法准确的知道这些子类如何实现这些方法。
- 从多个具有相同特征的类中抽象出一个抽象类，有这个抽象类为模板，从而避免了子类设计的随意性

3.作用：限制规定子类必须实现某些方法，但不关注实现细节
4.使用规则：

- abstract定义抽象类
- abstract定义抽象方法，只声明，不需要实现
- 包含抽象方法的类是抽象类
- 抽象类中可以包含普通方法，也可以没有抽象方法（抽象方法没有方法体）
- 抽象类不能直接创建，可以定义引用变量

```
public abstract class Phone{
    public abstract void call();
    public abstract void send();
}
```

```
public class Nokin extends Phone{

    public abstract void call(){
        syso("a");
    }
    public abstract void send(){
        syso("b");
    }
}
```

```
public class Android extends Phone{
    public abstract void call(){
        syso("c");
    }
    public abstract void send(){
        syso("d");
    }

}
```

```
public class Test{
    public void main(){
    //通过父类的引用创建子类对象
    Phone nokin=new Nokin();
    nokin.call();
    Phone android=new Phone();
    android.send();
    }
}
```

1接口定义了某一些类所要遵守的`规范`，接口不关心这些类的内部数据，也不关心内部的实现细节，它只规定这些类里必须提供某些方法。
2.定义时关键字为interface

```
[修饰符] interface 接口名 [extends 父接口1，父接口2..]{
    零个或多个常量定义
    零个货多个抽象方法
}
```

3.使用接口
一个类可以实现一个或多个接口，实现关键字为implements,通过实现多个接口是对java中类的单继承做一个补充

```
[修饰符] class 类名 extends 父类implemnets接口1，接口2...{

}
```

4.示例

```
public interface IplayGame{
public void playGame();
}
```

```
public class Android extends implements IplayGame{

    public void playGame(){
        syso("android play Game");
    }
}
```

```
public class Psp implements IplayGame{

    public void playGame(){
        syso("psp play Game");
    }
}
```

```
public class Test{

    public void main(){
        //接口的引用指向实现了接口的对象
        IplayGame ip1=new Android();
        ip1.playGame();
        IplayGame ip2=new Psp();
        ip2.playGame();

        //使用匿名内部类的方式实现接口
        new IplayGame(){
            public void playGame(){
                syso("使用匿名内部类的方式实现接口");}.playGame();
        }
    }
}
```

面向对象

类与对象

类：确定对象将会拥有的特征。类是对象的类型，具有相同属性和方法的一组对象集合，是模板，是抽象概念
对象：客观存在的事物，是具体的实体
面向对象：人关注事物信息

封装

- **概念**：将类的信息隐藏在类的内部，不允许外部程序直接访问，而是通过该类提供的方法来实现对隐藏信息的操作和访问，隐藏该隐藏的，暴露该暴露的

- **好处**
  1.只能通过规定的方法来访问数据
  2.提高安全性
  3.隐藏类的实现细节，方便修改和实现

- **实现**
  1.修该可见属性Private
  2.创建get/set方法
  3.在getter/settter方法中加入属性的控制语句

- **this和super**
  1.this表示当前类或当前对象
  2.this关键字代表当前对象。this.属性：操作当前对象的属性；this.方法：调用当前对象的方法
  3.封装对象的属性的时候
  4.super表示当前类或对象的父类.访问父类属性：super.age;访问父类方法：super.eat();
  5.如果显示的调用构造方法，super()一般要将此句放在构造函数第一句
  6.如果子类的构造方法没有显示的调用父类的构造方法，则系统会默认调用父类的无参构造方法
  7. `子类的构造的过程当中必须调用父类的构造方法`

- **Java中内部类**
  1.成员内部类：外部类中嵌套内部类
  2.局部内部类：定义在方法中的类
  3.静态内部类：由static修饰的成员内部类
  4.匿名内部类：通常出现在点击事件、多线程。btn.setOnclickListen(new OnClikListen(){});

继承

- **概念**
  继承是类与类的一种关系，是一种“is a”的的关系，java是单继承
- **好处**
  1.子类拥有父类的所有属性和方法（不能是private修饰）
  2.复用父类所写的代码
- **重写和重载**
  1.重写(父子类)
  如果子类对父类的方法不满意，是可以重写父类继承的个方法的，调用方法是优先调用子类的方法。语法规则：a.返回值类型，b.方法名，c.参数类型及个数.都要与父类继承的方法相同
  方法的重写要遵循“ `两同两小一大` ”规则：

> 方法名相同
> 形参列表相同
> 子类的方法返回值类型应小于或等于父类的方法返回值类型
> 子类方法声明抛出的异常类应小于或等于父类方法声明抛出的异常类
> 子类方法的访问权限应大于或等于父类方法的访问权限。

2.重载
如果 同一个类 中包含了两个以上方法的方法名相同，但形参列表不同，则被称为方法重载。
方法重载其实就是上述要素的“两同一不同”。

```
确定一个方法的三要素：
调用者-->同一个类
方法名-->方法名相同
形参列表-->形参个数和类型不同
```

- **继承的初始化顺序**
  1.初始化父类再初始化子类
  2.先执行初始化对象中的属性，再执行构造方法中的的初始化

- **Java中的final和static**
  1.final关键字
  使用final关键字做标识有“最终的”含义
  final可以修饰类、方法、属性和变量
  final–>修饰类，该类不允许被继承
  final–>修饰方法，该方法不允许被重写
  final–>修饰属性，不会自动进行隐式的初始化，需人为在构造方法中进行初始化
  final–>修饰变量，变量在声明时只能赋一次值，也即变为 常量

2.static关键字
Java 中被 static 修饰的成员称为静态成员或类成员。它属于整个类所有，而不是某个对象所有，即被类的所有对象所共享。静态成员可以使用类名直接访问。

```
1、 静态方法中可以直接调用同类中的静态成员，但不能直接调用非静态成员。
2、如果希望在静态方法中调用非静态变量，可以通过创建类的对象，然后通过对象来访问非静态变量。
3、在普通成员方法中，则可以直接访问同类的非静态变量和静态变量
4、静态方法中不能直接调用非静态方法，需要通过对象来访问非静态方法。
5、静态初始化块只在类加载时执行，且`只会执行一次`，同时静态初始化块只能给静态变量赋值，不能初始化普通的成员变量。
6、执行顺序
静态变量->静态代码块->变量->初始化块->构造器
```

多态
对象具有多种形态，其主要体现在1.引用多态，2.方法多态
1.引用多态

- 父类的引用可以指向本类的对象
- 父类的引用可以指向子类的对象

```
public class Animal{
    public void eat(){
    syso("Animal");
    }
    public void drink(){
    syso("Animal_drink");
    }
}
```

```
public class Dog extends Animal{
    public void eat(){
    syso("Dog");
    }
}
```

```
public class Test{
    public static void main(String[] args)
    //父类的引用可以指向本类的对象
    Animal obj1=new Animal();
    //父类的引用可以指向子类的对象
    Animal obj2=new Dog();

    obj1.eat();//调用父类方法
    obi2.eat();//调用子类重写的方法

    obj2.drink();//调用子类继承父类后的方法

}
```

2.方法多态

- 创建本类对象时，调用方法为本类方法
- 创建子类对象时，调用的方法为子类重写的方法或者继承的方法
- 如果子类中有独有的方法，就不允许有父类引用的子类对象来调用此方法。而只能通过该类引用的该类对象来调用此方法
  3.引用类型转换
- 向上类型转换（隐式/自动类型转化），是小类型到大类型的转换–>无风险（杯子里的水倒到茶壶里）
- 向下类型转换（强制类型转化），是大类型转化小类型–>有风险（茶壶里的水倒到杯子，有可能会溢出）
- instanceof运算符，来解决引用对象的类型，避免类型转换的安全性问题

```
Dog dog=new DOg();
Animal animal=dog;//自动类型提升
Dog dog2=(Dog)animal;//向下转，强转

//通过instanceof运算符来避免类型转化的安全性问题
if(animal instanceof Cat){
    Cat cat=(Cat)animal;
}

注意：java中A instanceof B中instanceof用来判断内存中实际对象A是不是B类型
```

文件IO

集合\泛型

反射

Java多线程

Java网络编程

序列化与反序列化

设计模式（常见13种）