

Day23

Review month1

JCF => Java Collections Framework => Java集合框架

```
1      JCF
2
3      Collection      Map
4
5      List      Set      SortedMap
6
7      SortedSet
```

List: 有序 不唯一

Set: "唯一"

SortedSet: 有序 "唯一"

```
1  ArrayList  LinkedList  Vector  Stack
2  HashSet
3  TreeSet
4  HashMap   Hashtable   ConcurrentHashMap
```

所有单值类型集合统一的操作有哪些?

add(obj) remove(obj) contains(obj) size()

clear() iterator() forEach() stream()

addAll() removeAll() retainAll() ---> 返回差集

```
1  小明:   手机   键盘   篮球
2  小红:   手机   键盘   口红
3
4  进货:   小明.addAll(小红)
```

List集合有序吗? 如何按照程序员需要的顺序进行排序呢?

有序 默认添加顺序..

```
1  1> Collections.sort(list,比较器);
```

Map集合 如何按照其值对象进行排序?

Map集合没法按照值对象进行排序

哪怕有序Map集合 也是**只能按照主键对象**排序

```
1  如果需求决定 一定要按照值对象排序
2  可以创建一个新的Map集合 让键做值 值做键
```

内部类：成员内部类 局部内部类 匿名内部类

定义在一个类类体当中的类 被称作内部类

What

为什么要使用内部类：

Why

内部类是Java当中共享数据最简单的方式之一
还能体现类和类之间的专属关系~

- 1 *： 内部类同样是类 编译之后会形成.class
- 2 其命名规则为 外部类名字\$内部类名字
- 3 注意 同一篇代码 不能出现等价定义

内部类的分类：能够共享到外部类的哪些成员

成员内部类：

外部类的所有[静态+非静态]成员[属性+方法]

如何理解其与外部类的关系：

蛔虫和牛的关系

如何创建其对象：

```
Outer.Inner in = new Outer().new Inner();
```

静态内部类：

外部类的静态成员[属性+方法]

如何理解其与外部类的关系：

寄居蟹和蛤蜊壳的关系

房客和房东的关系

如何创建其对象：

```
Outer.Inner in = new Outer.Inner(); --->并不影响外部类，所以不创建
```

表达类与类的专属关系

- 1 *： 静态内部类能够共享的是外部类的静态成员
- 2 但是静态成员本身就能直接访问
- 3 还用内部类去共享吗？
- 4 显然不用，所以，内部类的存在 并不是为了共享数据
- 5 而是为了表达类和类的专属关系

局部内部类：

如果定义在外部类的静态方法中：

只能共享外部类静态成员

如果定义在外部类的非静态方法中：

能够共享外部类所有成员

另外 还有其所在的方法体当中的局部变量也能共享

只是JDK8.0之前必须用final修饰

8.0开始可以不加final 但默认就是final 传参，并放到构造方法中

- 1 如何理解其与外部类的关系：
- 2 老师和学生的关系（特定范围内的关系）
- 3 如何创建其对象：
- 4 `Inner in = new Inner();`
- 5 *: 有位置限定： 定义结束之后 所在方法结束之前

匿名内部类：

- #: 如果生个孩子就是为了拿去卖钱的
就不用费尽心思给他起名字
- #: 某些场景下 我们自己的名字根本不重要
反而父母或者长辈的名字才重要

```
1 new 接口() {  
2     完成抽象方法的具体实现  
3 };  
4  
5 或者  
6  
7 new 父类(给父类构造方法传参){  
8     完成抽象方法的具体实现  
9 };  
10  
11 *: 匿名内部类能够共享外部类的哪些成员  
12 取决于定义它的位置  
13 可能等价于上述三种的一种
```

*: 匿名内部类能够共享外部类的哪些成员

取决于定义它的位置

可能等价于上述三种的一种

```
1 //匿名内部类的语法创建一个医生对象  
2 Person bq = new Person("扁鹊"){  
3     @Override  
4     public void eat(){  
5         System.out.println("医生吃火腿");  
6     }  
7 };  
8  
9 bq.eat();
```

Java当中如何完成数据共享

- 1.使用静态变量完成共享 Okay
- 2.使用参数传递完成共享 Okay
- 3.使用内部类完成共享 Okay

一个凄美的爱情故事

一个小男孩 和 一个小女孩

他们约会去看电影 买完电影票之后 只够买一杯大可乐

于是找服务员要了两根吸管 共享同一杯可乐...

```
1 | 广寒宫 -> 电影院 Cinema
2 | 貂蝉 -> 可乐 cola
3 | 吕布 -> 男孩 Boy
4 | 董卓 -> 女孩 Girl
```

Day24

review

内部类：定义在一个类 **类体**当中的类 被称作内部类

内部类的**作用**：

- 1.用来**共享数据**
- 2.用来表达**类和类的专属关系**

内部类的分类：

成员内部类：共享外部类**所有**[静态+非静态]**成员**[属性+方法]

静态内部类：共享外部类**所有静态成员**

局部内部类：

定义在**静态方法**中：只能共享静态成员

定义在**非静态方法**中：能够共享**所有成员**

由于局部内部类定义在**方法体**当中

所以还能共享其所在方法体当中的局部变量

只是**JDK8.0之前**必须加**final**修饰

8.0开始可以不加final但**默认就是最终变量**

- ```
1 | 匿名内部类：取决于定义它的位置 可能等价于上述三种的某一种
```

**如何创建内部类对象：**

1成员内部类： Outer.Inner in = new Outer().new Inner();

2静态内部类： Outer.Inner in = new Outer.Inner();

3局部内部类： Inner in = new Inner();

\*:有位置限定 定义完成之后 所在方法结束之前

```
4匿名内部类： new 父类/接口(){
 完成抽象方法的具体实现;
}
```

**共享数据的三种方式：**

- 1.使用**静态变量**
- 2.使用**参数传递**(不要只会构造方法 还得会setter)
- 3.使用**内部类**

## Exception (异常)

异常 Exception => 例外

Exception : 程序运行过程当中出现的**例外情况**而已

Java当中所有"问题"的体系结构

```
1 | Throwable
2 | [可以向外抛出的]
```

```
1 | Error Exception
2 | [错误] [异常]
3 |
4 | RuntimeException
5 | [运行时异常]
```

### \*: Error 和 Exception 的区别?

**Error** 是错误 通常是指由于**硬件环境**或者**系统原因**导致的相对较严重的问题  
OutOfMemoryError..

**Exception** 是异常 **程序运行过程当中**出现的例外情况

### \*: 运行时异常 和 非运行时异常的区别?

- 1、**非运行时异常** 在编译的时候 就要求程序员必须给出处理方案  
否则编译无法通过 它们都直接**继承Exception**
- 2、运行时异常 在编译的时候不要求给出处理方案 编译直接通过  
问题会在**运行时直接体现出来** 它们**继承RuntimeException**

### \*: 常见的运行时异常 11种

#### 1运算符:

**ArithmeticException** = 算术异常

```
System.out.println(5 / 0);
```

#### 2数组:

**NegativeArraySizeException** = 负数的数组大小异常

```
int[] data1 = new int[-5];
```

**ArrayIndexOutOfBoundsException** = 索引值超出边界异常

```
int[] data2 = new int[]{11,22,33};
System.out.println(data2[3]);
```

#### 3字符串:

**NullPointerException** = 空指针异常

```
String s1 = null;
System.out.println(s1.length());
```

**StringIndexOutOfBoundsException** = 字符串索引值超出边界异常

```
String s2 = "ETOAK";
System.out.println(s2.charAt(5));
```

**NumberFormatException** = 数字格式异常

```
String s3 = "123a";
int price = Integer.parseInt(s3);
System.out.println(price + 5);
```

#### 1. 强制类型转换:

**ClassCastException** = 类型造型异常

```
Object stu = new Student();
Cacti cc = (Cacti)stu;
```

```
1 // 强制类型转换 ()
2 // instanceof
3 // ==
```

#### 4. 集合:

**IllegalArgumentException** = 非法参数异常

```
List list1 = new ArrayList<>(-7);
```

**IndexOutOfBoundsException** = 索引值超出边界异常

```
List list2 = new ArrayList<>();
Collections.addAll(list2, 11, 22, 33);
System.out.println(list2.get(3));
```

**IllegalStateException** = 非法状态异常

```
1 List<Integer> list3 = new ArrayList<>();
2 Collections.addAll(list3, 11, 22, 33, 44, 55);
3 Iterator<Integer> car = list3.iterator();
4 car.remove();
```

**ConcurrentModificationException** = 并发修改异常

## 为什么要处理异常?:

- A. 如果是非运行时异常 不做处理 编译都无法通过
- B. 一旦程序(线程)运行过程当中出现未作处理的异常  
虚拟机将直接中断程序(该线程)的执行

## 如何处理异常?:

#### 1. 抛还上级 throws

throws 出现在方法签名的最后  
用于表达 本方法当中出现指定种类的异常  
本方法当中不做处理 抛还给调用的上级进行处理

```

1 足以解决A 但无力解决B
2
3 2.自行处理 try catch finally
4 try{
5 可能出现异常的语句;
6 通常只写一句可能导致异常的语句;
7 除非需求决定前者出现异常 后者跳过
8 }catch(要捕获的异常类型 异常代号){
9 对捕获到的异常进行处理
10
11 0.隐瞒不报
12 1.简要的审:
13 异常代号.getMessage()
14 2.详细的审:
15 异常代号.printStackTrace()
16 }finally{
17 无论是否出现异常 最终都要执行的操作
18 通常是释放和关闭资源的操作
19 }
20
21 *: 一个try 后面可以跟上多个不同的catch
22 但是要求 类型必须前小后大 后者并列
23 总之不能前大后小 否则后者没有意义
24
25 *: 在JDK7.0之前 如果捕获到两种不同异常
26 要做的处理是相同的 也必须写多个catch分支
27 但是从JDK7.0开始 支持多重catch
28 一个catch捕获多种类型 中间用 |
29
30 *: 我们不要在finally当中写return语句
31 否则try 和 catch当中的所有return
 都失去意义了

```

如何在程序本没有异常出现的时候 主动制造异常出现的场景:

```

1 throw 出现在方法体当中 用于在没有异常的时候
2 主动制造异常出现的场景

```

## \*: throw和throws的区别

**throws** 出现在方法签名的最后

用于表达 本方法当中出现指定种类的异常

本方法当中不做处理 抛还给调用的上级进行处理

**throw** 出现在方法体当中 用于在没有异常的时候

主动制造异常出现的场景

## 如何自定义异常:

自己开发一个类 选择继承Exception / RuntimeException

非运行时异常 / 运行时异常

然后 在其构造方法的首行使用super() 指定异常的描述信息

## 五个常见异常：

### Plus01:

当类体当中的静态变量是调用有异常声明的方法完成赋值的时候  
我们不能在类体当中直接try catch 更不能在类的签名上throws  
此时如果想要编译通过 **必须借助静态初始化块 static{ }**  
在静态初始化块当中使用try catch进行处理

\*: 如果是个非静态变量 则可以使用 非静态初始化块 或者 构造方法 { }

```
1 public class TestExceptionPlus1{
2 public static void main(String[] args){
3
4 }
5 }
6 class A{
7
8 static int i;
9 static{
10 try{
11 i = get();
12 }catch(Exception e){
13 e.printStackTrace();
14 }
15 }
16 public static int get()throws Exception{
17 int x = (int)(Math.random()*5);//0-4
18 if(x == 2 || x == 4){
19 throw new Exception("生成的数字不吉利");
20 }
21 return x;
22 }
23 }
```

### Plus02:

在完成方法覆盖的时候 如果父类当中的方法没有任何throws声明

子类在覆盖该方法的时候 能不能声明抛出异常呢?

可以 但是只能向外抛出运行时异常 - 这样的行为没有意义

**因为在Java当中每个方法都默认抛出所有的运行时异常**

相当于 每个方法最后都有我们看不见的throws RuntimeException

```
1 public class TestExceptionPlus2{
2 public static void main(String[] args)throws RuntimeException{
3 C cc = new C();
4 cc.test();
5 }
6 }
7 class A{
8 public void test(){
9 System.out.println("这是父类的test()方法");
10 }
11 }
```



```

12 class C extends A{
13 @Override
14 public void test()throws RuntimeException{
15 System.out.println("这是子类的test()方法");
16 }
17 }

```

### Plus03:

当我们的代码当中出现连续的多行语句都有异常声明的时候  
 我们需要无论前者执行是否顺利 后者都要尝试去执行  
 则必须借助try catch finally的finally当中 嵌套使用try catch  
 我们把这种语法戏称为 **连环try...** [学IO流的时候要用~]

```

1 public class TestExceptionPlus3{
2 public static void main(String[] args){
3 SLT no1 = new SLT();
4 SLT no2 = new SLT();
5 SLT no3 = new SLT();
6
7 try{
8 no1.close();
9 return;
10 }catch(Exception e){
11 e.printStackTrace();
12 }finally{
13 try{
14 no2.close();
15 }catch(Exception e){
16 e.printStackTrace();
17 }finally{
18 try{
19 no3.close();
20 }catch(Exception e){
21 e.printStackTrace();
22 }
23 }
24 }
25 }
26 }
27 class SLT{
28 public void close()throws Exception{
29 int x = (int)(Math.random()*2);//0 or 1
30 if(x == 1){
31 throw new Exception("拧坏了关不上的异常");
32 }
33 System.out.println("正常的关闭了水龙头");
34 }
35 }

```

#### Plus04:

为了进行异常处理而添加的try catch语法结构是有{}的  
这对儿大括号同样能够控制变量的作用范围  
所以如果我们的某些变量 在下文程序当中还要继续使用  
就不能在try{}当中进行定义  
应当在try{}前面完成定义 并且以默认值赋值

**try{}当中只做重新赋值 不做变量定义**

```
1 public class TestExceptionPlus4{
2 public static void main(String[] args){
3 int num = 0;
4 try{
5 num = get();
6
7 }catch(Exception e){
8 e.printStackTrace();
9 }
10 System.out.println(num);
11
12 }
13
14 public static int get()throws Exception{
15 int x = (int)(Math.random()*5);//0-4
16 if(x == 2 || x == 4){
17 throw new Exception("生成的数字不吉利");
18 }
19 return x;
20 }
21 }
```

#### Plus05:

在某些场景下 **学会使用异常处理的机制代替传统的分支判断**  
**会有奇效!!!** 【某些场景 发生在下周... 和100有关】

```
1 public class TestExceptionPlus5{
2 public static void main(String[] args){
3
4 }
5 public static boolean check(String str){
6 //如果str当中全是数字 则返回true 反之false
7 try{
8 Integer.parseInt(str);
9 return true;
10 }catch(Exception e){
11 return false;
12 }
13
14 /*
15 for(int i = 0;i<str.length();i++){
16 char c = str.charAt(i);
17 if(c < '0' || c > '9'){
18 return false;
19 }
20 }
21 }
22 }
```

```

21 return true;
22 */
23 }
24 }

```

# day25

## Review

异常：程序运行过程当中出现的例外情况而已

Java当中所有问题的体系结构

Throwable

[可以向外抛出的]

```

1 Error Exception
2
3 RuntimeException

```

### #: **Error和Exception的区别?**

Error通常是指由于硬件环境或者系统原因导致的  
相对较严重的问题

Exception就是运行过程当中出现的例外情况而已

### #: **非运行时异常和运行时异常的区别?**

非运行时异常在编译的时候就必须要求给出处理方案  
否则编译无法通过

运行时异常在编译的时候不要求给出处理方案  
编译可以直接通过 问题在运行时直接体现出来

### #: **常见的运行时异常:**

#### 1 运算符

ArithmeticException = 算术异常

#### 2 数组

NegativeArraySizeException = 负数数组大小异常

ArrayIndexOutOfBoundsException = 数组索引值超出边界异常

#### 3 字符串

NullPointerException = 空指针异常

StringIndexOutOfBoundsException = 字符串索引值超出边界异常

NumberFormatException = 数字格式异常

#### 1 类型转换

ClassCastException = 类型造型异常

#### 4 集合

IllegalArgumentException = 非法参数异常 **illegalArgumentException**

IllegalStateException = 非法状态异常

IndexOutOfBoundsException = 索引值超出边界异常

ConcurrentModificationException = 并发修改异常

## #: 为什么要处理异常

- A. 非运行时异常不做处理编译无法通过
- B. 一旦线程执行过程当中出现未做处理的异常  
虚拟机将直接中断该线程的执行

## #: 如何处理异常

1. **throws 抛还上级** 足以解决A 但无力解决B
2. **try catch finally 自行处理**

```
1 try{
2 可能出现异常的语句;
3 }catch(要捕获的异常类型 给异常起个名字){
4 0. 隐瞒不报
5 1. 简要的审 e.getMessage()
6 2. 详细的审 e.printStackTrace();
7 }finally{
8 无论是否出现异常 最终都要执行的操作
9 通常是释放和关闭资源的操作
10 }
11
12 *: 一个try 后面可以有多个catch 但是
13 决不允许前者包含后者
14
15 *: JDK7.0开始支持多重catch
16 如果捕获到不同类型的异常想要做相同处理
17 则catch(类型1 | 类型2 | 类型3 e)
18
19 *: finally当中不该出现return语句
```

## 如何在没有异常的情况下 主动制造异常出现

### **throw**

用在方法体当中 用于主动制造异常出现的场景  
[没事找事型]

```
1 throws
2 用在方法签名的最后 用于表达本方法中出现指定种类的异常
3 本方法当中不做处理 抛还给调用的上级进行处理
4 [有事甩锅型]
```

## 如何自定义异常

自己开发一个类 选择**继承Exception / RuntimeException**  
在其构造方法的首行使用**super()**传参指定异常的描述信息

## 线程 Thread

**程序**: 保存在物理介质中的代码片段

**进程**: 一旦程序运行起来 就会编程操作系统其中的一个进程  
进行当中的程序

**线程**: 进程当中更加微观的概念  
**程序当中一条独立的执行线索**

为什么要使用多线程：

我们不否认在某些场景下使用多线程确实可以提高效率

但是使用多线程的根本目的不是提高效率

而是让程序同一时间能够做多件事情

从而可以服务多个用户 处理多个请求

线程的一生：

|   |         |          |         |      |
|---|---------|----------|---------|------|
| 1 | 新生      | 就绪       | 运行      | 消亡   |
| 2 | NewBorn | Runnable | Running | Dead |
| 3 |         |          |         |      |
| 4 |         | 阻塞       |         |      |
| 5 |         | Blocking |         |      |

如何实现线程：

1. extends Thread

2. implements Runnable

3. 金针菇..

|   |                |                     |  |
|---|----------------|---------------------|--|
| 1 | 请用两种不同方式实现两个线程 |                     |  |
| 2 | 要求第一个线程打印1-26  | extends Thread      |  |
| 3 | 要求第二个线程打印a-z   | implements Runnable |  |

如何控制线程：

0. setPriority(int) : 设置线程的优先级别

可选范围1-10 默认优先级为5

优先级高 代表抢到时间片的概率高而已

1. static sleep(long) : 让当前线程休眠指定的毫秒数 / 在主方法里调用xx.sleep(), 主线程进入睡眠/

\r 独立回车字符

2. static yield() : 让当前线程放弃时间片 直接返回就绪

3. join() : 让当前线程邀请调用方法的那个线程优先执行

在被邀请的线程执行结束之前

当前线程一直阻塞 不再继续执行

\*: 线程类当中所有的静态方法 不要关注谁调用方法

而要关注 调用出现在谁的线程体当中

出现在谁的线程体当中 就是操作对应的那个线程

别看谁点 看写在哪

\*: 线程类当中所有涉及到主动进入阻塞状态的方法

都必须进行异常处理

因为它们都有异常声明

它们都有**throws InterruptedException**

InterruptedException是个**非运行时异常**

BigOne

一个向往美好的和尚 与 诸多动物一起旅游的故事

1 |

- 《西游记》

故事当中涉及到如下多个线程：

有一个师傅线程 打印999次 "Only You"

有一个猴哥线程 打印888次 "俺老孙来也"

有一个八戒线程 打印777次 "分行李吧 回高老庄"

有一个悟净线程 打印666次 "大师兄 不好了 师傅被妖怪抓走了"

```
1 setPriority()
2 *: 师傅天天逼逼叨叨特别烦人 请让他优先级最低
3 *: 猴哥想给师弟们更多的表现机会 前三次拿到时间片
4 猴哥决定主动放弃 yield()
5 *: 八戒很懒 干活之前需要先休眠300毫秒 sleep()
6 *: 沙僧很依赖猴哥 猴子不死 他就不上 join()
```

**线程类其它常用方法：** **static activeCount()** / **setDaemon(true)** / **interrupt()** / **static currentThread**

**setName() + getName()** : 设置和得到线程的名字~

**static activeCount()** : 得到程序当中所有活跃线程的总数

活跃线程：就绪 + 运行 + 阻塞

**setDaemon(true)** : 设置线程成为**守护线程**

守护线程：守护线程是为其它线程提供服务的

当程序当中只剩下守护线程的时候

守护线程会自行消亡

\*: **守护线程应当无限循环 以防止其过早消亡**

\*: 设置成为守护线程 **必须早于线程的start**

否则会触发IllegalThreadStateException

\*: 守护线程**应该具有较低的优先级别**

**interrupt()** : 中断 打断线程的阻塞状态

**static currentThread()** : 得到正在运行状态的**线程对象**

1.它可以出现在主方法当中 ☆**用于获得主线程对象**

2.它还可以出现在run()调用的其它方法中

用于得到**当前线程是谁**~

X.它绝对不该直接出现在run()当中  
它得到的将完全等价于this

- 1 1. 线程的基础 如何实现线程 如何启动线程
- 2 2. 如何给线程设置名字 如何得到线程的线程
- 3 3. 单例模式 - 醉汉式
- 4 4. run()调用的其它方法中 如何得到当前线程
- 5 5. 多线程高并发的场景下 该如何选择键值对集合 ConcurrentHashMap
- 6 6. Map集合的基础操作 put() get() containsKey()
- 7 7. 分析数据的线程如何邀请生产数据的线程优先执行 join()
- 8 8. 连环try
- 9 9. CountdownLatch

```
import java.util.concurrent.*;
```

## CountDownLatch

/\*\*

**CountDownLatch** 是什么？核心方法有什么？

直译叫做**倒计时门闩** 是JUC包(**并发包**)当中的常用工具之一  
其作者是Doug Lea **用于主动制造阻塞**

```
1 核心方法：
2 构造方法： new CountdownLatch(门闩个数);
3
4 await() : 阻塞直到门闩都被打开 [分析数据的线程]
5 countDown() : 打开一个门闩 [每个生产数据的线程]
```

\*/

```
class X{
 static CountdownLatch latch = new CountdownLatch(3);
}
```

**X.latch.countDown();**//拔掉一个门闩

```
try{
 X.latch.await();//等待开门
}catch(Exception e){
 e.printStackTrace();
}
```

# day26

## review

**线程**：程序当中一条独立的执行线索...

而多线程编程 就是让程序当中拥有多条独立的执行线索  
从而可以同一时间做多件事 同一时间接待多个连接  
处理多个请求...

为什么要使用多线程：

我们不否认某些场景使用多线程确实可以提高效率..  
能够让程序同一时间做多件事情...

线程的五大状态：

新生 就绪 运行 消亡

```
1 | 阻塞(sleep() / join() / await())
```

如何创建线程：

1. extends Thread
2. implements Runnable

```
1 | 3. 明天见implements Callable<T>{}
2 |
3 | public T call()throws Exception{}
```

### 如何控制线程

- 0.**setPriority(int)**：设置线程的优先级别
- 1.**static sleep(long)**：让当前线程休眠指定的毫秒数
- 2.**static yield()**：让当前线程放弃时间片**直接返回就绪**
- 3.**join()**：让当前线程邀请调用方法的线程优先执行

线程类其它常用方法：

- setName() + getName()：设置和得到线程的名字
- static activeCount()：得到程序当中所有活跃线程总数
- setDaemon(true)**：设置线程成为守护线程
  - 1.守护线程必须无限循环 防止其过早消亡
  - 2.设置成为守护线程必须早于自身的start()
  - 3.守护线程应该具有较低的优先级
- interrupt()：中断线程的阻塞状态
- static currentThread()：得到当前正在运行的线程对象
  - 1.可以用于在主方法当中得到主线程
  - 2.可以在run()调用的其它方法中得到当前线程是谁
  - X.不该直接出现在run() 得到的必然就是this

=====



## synchronized

已知: **Vector**类的**add()** 和 **remove()** 都是**synchronized**修饰的

我们有一个Vector对象 名叫v

有两个线程对象 名叫t1 t2

- 1 当**t1线程**调用**v对象**的**add()** 方法已经开始执行了
- 2 但是还没执行结束呢 此时时间片耗尽
- 3 而**t2线程**抢到了时间片

问:

t2能不能调用v对象的add()? **false**

t2能不能调用v对象的remove()? **false**

已知: **Hashtable**类的**put()** 和 **remove()**都是**synchronized**修饰的

我们有两个Hashtable对象 名叫**h1** 和 **h2**

有两个线程对象 名叫t1 和 t2

- 1 当**t1线程**调用**h1对象**的**put()** 方法已经开始执行了
- 2 但是还没执行结束呢 此时时间片耗尽
- 3 而**t2线程**抢到了时间片

问:

t2能不能调用h1对象的add()? **false**

t2能不能调用h1对象的remove()? **false**

t2能不能调用h2对象的add()? **true**

t2能不能调用h2对象的remove()? **true**

**综上所述** 即便**synchronized**加在方法上 也是对对象进行加锁

Java世界当中 **只有每个对象才有互斥锁标记**

才能对其进行加锁 **Java**当中根本没有对方法加锁

---

## 锁池 等待池

**锁池**和**等待池**都是Java当中每个对象都有一份的空间

它们都是用于存放**线程任务**的

**锁池** 存放的是那些想要拿到**对象的锁标记**

但是还没能顺利拿到的线程

**等待池** 存放的是那些原本已经**拿到了**对象锁标记

怕与其它线程永久阻塞而**主动释放锁标记的线程**

它们的**区别**主要在三个方法

1.进入的时候**是否需要释放资源**

锁池不需要

等待池必须先要释放资源

2.离开的时候**是否需要调用方法**

锁池不需要

等待池必须要**notify()** / **notifyAll()**

3.离开之后**去往什么状态**

离开**锁池**返回**就绪**

离开**等待池**进入**锁池**

## 多线程共享数据的时候 可能导致**并发错误!**

**根本原因**: 多个线程共享操作同一份数据...

**直接原因**: 线程体当中连续多个操作 未必能够连续执行

很可能操作只完成了一部分

然后**时间片**突然耗尽

此时另一个线程抢到了时间片

**直接访问操作了**操作不完整的数据

**导火线**: **【时间片突然耗尽】**

多个线程共享操作同一份数据的时候 线程体当中连续的多行操作

未必能够连续执行 很可能操作只完成了一部分

**时间片**突然耗尽 此时另一个线程抢到时间片

直接访问了操作并不完整的数据

在语法上没有任何错误 **但数据却全是错的(逻辑错误)**

1 | 编译不报错 运行没异常 它就是数据都不对

## #: **并发修改异常** 和 **并发错误** 是什么关系?

**并发修改异常** 是为了**减少**和**避免**并发错误出现

由官方程序员主动校验 主动抛出的运行时异常

它是为了避免程序进一步执行就该出现**并发错误**了

**并发错误** **编译不报错 运行没异常 杀程序员于无形**

## 如何解决**并发错误**:

**加锁**

\*: 多个线程共享操作的那个对象 被称作 **临界资源**

### 1st.**互斥锁**:

synchronized 修饰符 同步的...

**修饰代码块:**

```
synchronized(临界资源){
 需要连续执行的操作1;
 需要连续执行的操作2;
 ;
}
```

**修饰方法:**

```
public synchronized void add(Object obj){
 需要连续执行的操作1;
 需要连续执行的操作2;
}
```

```
....;
}
```

\*: **Vector Hashtable StringBuffer** 之所以线程安全  
是因为它们底层大量的方法都使用了**synchronized**

\*: 单例模式懒汉式 需要synchronized修饰那个getter~

\*: synchronized隔代丢失...

## 2nd.可重入锁 JDK5.0新特性 Doug Lea

java.util.concurrent.locks.ReentrantLock  
Java包的工具包的并发包的锁包 可重入锁

**lock()**      **unlock()**  
加锁          释放锁

```
1 ReentrantLock构造方法可以传参指定
2 公平锁 / 非公平锁
3
4 new ReentrantLock(true)
```

## 死锁 what how

### 什么是死锁:

多个线程 相互持有对方想要申请的资源 不释放的情况下  
又去申请对方已经持有到的资源  
从而双双进入对方已经持有的资源的锁池当中  
产生了永久的阻塞  
- 死锁现象 DeadLock

1. 中美科学家联合国饿死事件
2. 泉城路奔宝事件
3. AABB事件

### 如何解决死锁:

一块空间: 等待池

三个方法: **wait() / notify() / notifyAll()**

```
1 wait() : 让当前线程放弃对象的锁标记
2 并且进入调用方法的那个对象的等待池当中
3
4 notify() : 让当前线程从调用方法的那个对象的
5 等待池中随机的唤醒一个线程
6
7 notifyAll() : 让当前线程从调用方法的那个对象的
8 唤醒所有等待的线程
```

\*: 注意 这三个方法不是线程类的方法 而是Object类的方法

因为Java当中每个对象都有等待池 都可能要操作等待池  
所以这三个方法被定义在Object当中了

\*: 必须拿到对象的锁标记 才能操作对象的等待池

这三个方法都必须在已经持有锁标记的前提下才能使用

xxx.wait() 必然出现在synchronized(xxx){}

否则不但操作失败 还会触发运行时异常

IllegalMonitorStateException

Monitor = 监视器 = 锁标记 = 互斥锁 = 锁旗标 = 互斥锁标记

两个线程交替执行~

顺丰陆运 不可能一个大卡车只安排一个司机师傅..

|   |             |          |
|---|-------------|----------|
| 1 | 王师傅         | 黎师傅      |
| 2 | 1. 开4个小时    |          |
| 3 | 2. 王师傅主动休息  |          |
| 4 |             | 3. 开4个小时 |
| 5 |             | 4. 唤醒王师傅 |
| 6 |             | 5. 主动休息  |
| 7 | 6. 让黎师傅做好准备 |          |

## day27

### Review

并发错误:

多个线程共享操作同一份数据

线程体当中连续的多行语句未必能够连续执行

很可能操作只完成了一部分 时间片突然耗尽

而此时另一个线程直接访问了操作不完整的数据

在语法上 这没有任何错误 但是逻辑上 数据全是错的..

如何解决并发错误:

加锁

1.使用synchronized修饰符操作对象的互斥锁

a> 修饰代码块

```
synchronized(临界资源){
 需要连续执行的操作1;
 需要连续执行的操作2;
}
```

```

1 b> 修饰方法
2 public synchronized void add(Object obj){
3 需要连续执行的操作1;
4 需要连续执行的操作2;
5 }
6
7 *: 方法的synchronized修饰符不会被子类继承得到
8 这个synchronized隔代丢失

```

```

1 2. 直接使用可重入锁
2 java.util.concurrent.locks.ReentrantLock;
3
4 lock() unlock();
5 加锁 解锁
6
7 公平锁 & 非公平锁
8 new ReentrantLock(true);

```

### 什么是死锁

多个线程相互持有对方想要申请的资源  
 不释放的情况下 又去申请对方已经持有的资源  
 从而双双进入对方已经持有的资源的锁池当中  
 产生了永久的阻塞  
 - DeadLock

### 如何解决死锁:

一块空间: 对象的等待池  
 三个方法: wait() / notify() / notifyAll()

```

1 *: 这三个方法是Object类的方法
2 *: 这三个方法必须在已经持有对方锁标记的前提下才能使用
3 所以它们必然出现在synchronized 的{}当中
4 否则不但操作失败 还会触发运行时异常
5 IllegalMonitorStateException

```

### 线程交替执行

1. 执行逻辑
2. 主动wait
  3. 执行逻辑
  4. 唤醒notify()
  5. 主动wait()
6. 唤醒notify()

### \*: 锁池和等待池的区别?

进入是否需要释放资源  
 离开是否需要调用方法  
 离开之后去往什么状态

## shutdown() shutdownNow()

首先 它们都能禁止新任务再次提交

其次 它们都不能结束那些正在执行当中的线程任务

- 1 它们的区别在于 那些已经提交上去 还没开始执行的线程
- 2
- 3 **shutdown()** : 能够让进行中的和排队中的都执行完
- 4 **shutdownNow()** : 会直接退回排队中的线程任务

## 线程池

since JDK5.0 Doug Lea  
java.util.concurrent.\*;

- 1 **线程池** 是一种标准的资源池模式
- 2
- 3 **资源池**是指在用户出现之前提前预留活跃资源
- 4 从而在用户出现的第一时间 直接满足用户对资源的需求
- 5 并且将资源的**创建**和**销毁** 都委托给资源池完成
- 6 从而优化用户体验

- 1 假如一个线程的完整执行时间为T
- 2 则  $T = t1 + t2 + t3$
- 3 **t1**: 在操作系统当中映射一个线程所消耗的时间
- 4 **t2**: 线程核心逻辑执行的时间 **run()**
- 5 **t3**: 在操作系统当中销毁一个线程所消耗的时间
- 6
- 7 假如**run()**当中的代码非常简短
- 8 则**t2**所占T的比例就会很小
- 9 此时我们会觉得付出和回报不成比例 喧宾夺主

### \*: 什么是资源池 为什么要使用资源池?

**资源池**会在用户出现之前提前预留活跃资源  
从而在用户出现的第一时间直接满足用户对资源的需求  
并且将资源的创建和销毁都委托给**资源池**完成  
从而优化用户体验...

### \*: 实现线程有哪些方式?

1. **extends Thread**  
public void run(){}
2. **implements Runnable**  
public void run(){}
3. **implements Callable**  
public T call()throws Exception{}

## \*: 创建线程的第三种方式 优势是什么?

- 1.能够有return返回数据
- 2.能够向外抛出异常

## \*: shutdown() 和 shutdownNow()的区别?

## \*: 核心类库当中官方提供的常用的线程池种类有哪些?

- 1.newFixedThreadPool(): 修复后可重用的
- 2.newCachedThreadPool(): 缓存机制的 60S = 1M
- 3.newSingleThreadExecutor(): 单一实例的执行器

## \*: 如果自己创建线程池执行器 必须五个参数分别是什么?

- 1.核心线程的数量
- 2.最大线程的数量
- 3.KeepAliveTime => 保持活着的时间
- 4.TimeUnit => 时间单位
- 5.一个队列 用于存放超过最大需要排队的线程任务...

\*: 什么是并发错误

\*: 如何解决并发错误

\*: 什么是死锁

\*: 如何解决死锁

\*: 线程的七大状态

\*: 控制线程的方法有哪些?

\*: InterruptedException

先启动线程 后设置其成为守护线程 不但会失败 还会触发异常

\*: IllegalMonitorStateException

没有拿到对象锁标记就直接去操作对象的等待池

: 懒汉式单例

private 2 + static 2 + public + synchronized

\*: 锁池和等待池的区别?

\*: throw和throws的区别?

\*: Exception和Error的区别

\*: 运行时异常和非运行时异常的区别

\*: 常见的运行时异常

\*: throws try catch finally

\*: catch(类型1 | 类型2 e) 多重catch 7.0

\*: 自定义异常

\*: 什么是内部类 内部类的作用是什么

\*: 内部类的分类 每种内部类如何创建对象

/\*

JDK5.0的时候Callable接口的出现 弥补了原本Runnable接口的两大不足

1.run()被定义为void方法 线程执行结束没法返回数据

2.run()没有任何throws声明 逼迫程序员必须try catch

\*/

```
1 //Executors.newSingleThreadExecutor();
2 //单一实例的
3 //Executors.newCachedThreadPool();
4 //缓存机制的
5 ExecutorService es = Executors.newFixedThreadPool(2);
6 //修复后可重用的(固定大小)
7 ThreadOne t1 = new ThreadOne();
8 es.submit(t1); //将线程任务提交给执行器服务
```

## day28

### review (考试题)

#### 01.内部类简述以及内部类的作用?

定义在一个类类体当中的类 被称作内部类

内部类可以用于共享数据

也可以用于表达类和类的专属关系

#### \*: Java当中常见的共享数据的方式有哪些?

a> 使用静态变量 [交替打印 static Object obj]

b> 使用参数传递 [并发错误 一个打印 一个修改]

c> 使用内部类 [死锁 泉城路奔宝]

#### 02.内部类有几种?每种内部类能够共享外部类的什么数据?

4种

成员内部类: 能够共享外部类的所有[静态+非静态]成员[属性+方法]

静态内部类: 能够共享外部类的静态成员[属性+方法]

局部内部类:

如果定义在静态方法中 只能共享外部类静态成员

如果定义在非静态方法中 能够共享外部类所有成员

另外它出现在外部类的方法体当中

所以还能共享所在的外部类方法中的局部变量

只是JDK8.0之前必须加final 8.0开始可以不加

匿名内部类: 根据定义的位置 可能等价于上述三种任意一种

\*: 成员内部类如何创建对象?

Outer.Inner in = new Outer().new Inner();



\*: 请使用匿名内部类语法创建一个**比较器对象**

```
Comparator cmp = new Comparator(){
 @Override
 public int compare(Integer i1,Integer i2){
 return i2.compareTo(i1);
 }
};
```

```
1 Set<Integer> set = new TreeSet<>(new Comparator<Integer>(){
2 @Override
3 public int compare(Integer i1,Integer i2){
4 return i2.compareTo(i1);
5 }
6 });
```

### 03.Error和Exception的区别?什么是Exception?

Error通常是指由于硬件环境或者系统原因导致的

程序员通过编码无法解决的 相对较严重的问题

Exception就是指程序运行过程当中出现的**例外情况**而已

### 04.运行时异常和非运行时异常的区别?

运行时异常在编译的时候**不要求给出处理方案**

编译能够直接通过 问题会在运行的时候直接体现出来

它们都直接继承**RuntimeException**

非运行时异常在编译的时候**必须给出处理方案**

否则编译无法通过 它们直接继承**Exception**

```
1 *: 无论运行时异常还是非运行时异常一定都在运行的时候出现
2 编译的时候找你要处理方案 不是异常出现
```

### 05.常见的运行时异常 至少8种

过

### 06.如何处理异常?

a> 抛还上级 throws

throws 出现在方法签名的最后

用于表达本方法中出现指定种类的异常

本方法中不做处理 抛还给调用的上级进行处理

b> 自行处理 try catch finally

try{

可能出现异常的语句;

}catch(要捕获的异常类型 异常代号){

对捕获的异常进行处理

**0.隐瞒不报**

**1.简要的审 getMessage()**

**2.详细的审 printStackTrace();**

**3.谎报 throw new ???Exception();**

```

1 }finally{
2 无论是否出现异常最终都要执行的操作
3 通常是释放和关闭资源的操作；
4 }

```

\*: JDK7.0之前和之后 如果对多种捕获到的异常要做相同的处理  
各自应该怎么写

```

1 JDK7.0之前 JDK7.0开始
2 try{ try{
3
4 }catch(Exception1 e){ }catch(Exception1 | Exception2 e){
5
6 }catch(Exception2 e){ }
7 *: 多重catch
8 }

```

\*: finally当中的语句一定会执行吗？如何不让他执行呢？

`System.exit(0); //结束虚拟机`

## 07.如何自定义异常并指定异常表述，请写出代码？

```

class EtoakException extends RuntimeException{
 public EtoakException(){
 super("异常的描述信息");//message
 }
}

```

\*: throw和throws的区别？

throw 用在方法体当中 用于在本没有异常的情况下  
主动制造异常出现

【没事找事型】

throws 出现在方法签名的最后

用于表达本方法中出现指定种类的异常

本方法中不做处理 抛还给调用的上级进行处理

【有事甩锅性】

\*: 请写出几个你常见的非运行时异常

IOException

CloneNotSupportedException

AWTException (Robot)

```

1 InterruptedException (sleep() join() wait() get())

```

## 08.创建线程的方式有哪些？请写出代码

```
class A extends Thread{
 @Override
 public void run(){
 ...;
 }
}
class B implements Runnable{
 @Override
 public void run(){
 ...;
 }
}
class C implements Callable{
 @Override
 public T call()throws Exception{
 ...;
 }
}
```

\*: Callable接口啥时候出现的 谁写的?

JDK5.0的时候 Doug Lea 在JUC包当中提供的实现

\*: JUC包当中都有哪些内容?

a> 多线程高并发的场景下 更加好用的集合

ConcurrentHashMap & CopyOnWriteArrayList & CopyOnWriteArraySet

b> 多线程开发中常用的各种工具

倒计时门🚪 CountdownLatch

c> 第三种实现线程的方式 以及线程池的实现

ExecutorService

d> locks子包当中提供了新的加锁机制 可重入锁

ReentrantLock

## 9.线程是什么 多线程的适用场景有哪些？

线程是程序当中一条独立的执行线索

而多线程编程就是让程序当中拥有多条不同的执行线索

- 1 当程序当中需要同一时间做多件事 同一时间处理多个请求
- 2 都必须使用多线程 （某些场景下多线程也可以提高效率）

## 10.线程状态有哪些？

|   |      |    |     |    |
|---|------|----|-----|----|
| 1 | 新生   | 就绪 | 运行  | 消亡 |
| 2 |      |    |     |    |
| 3 |      | 阻塞 |     |    |
| 4 |      |    |     |    |
| 5 | 普通阻塞 | 锁池 | 等待池 |    |

## 11.控制线程的方法有哪些？

- 0.setPriority(int) : 设置线程优先级 可选范围1-10 默认5
- 1.static sleep(long) : 让当前线程休眠指定的毫秒数
- 2.static yield() : 让当前线程放弃时间片返回就绪
- 3.join() : 当前线程邀请调用方法的线程优先执行

## 12.并发修改异常和并发错误有什么区别？

并发错误编译不报错 运行没异常 语法完全正确 只是逻辑上数据全是错的  
并发修改异常是为了避免程序运行出现并发错误  
而由官方程序员主动校验 主动抛出的运行时异常  
以防止程序进一步执行就该出现并发错误了

## 13.什么场景下会导致并发错误 怎么处理并解决并发错误？

多个线程同时操作同一份数据 线程体当中连续的多行语句  
未必能够连续执行 很可能操作只完成了一部分  
时间片突然耗尽 此时另一个线程抢到时时间片  
直接拿走了操作不完整的数据  
语法上没有任何错误 数据在逻辑上都是错的

- 1 加锁
- 2 1. 语法级别的加锁 synchronized 操作互斥锁
- 3 2. 面向对象的加锁 ReentrantLock 可重入锁

## 14.synchronized特性有哪些 synchronized可以修饰什么

隔代丢失

synchronized修饰的方法可以被子类继承但是其synchronized特性就不见了

- 1 synchronized 可以修饰代码块 可以修饰方法

## 15.造成死锁的原因？该如何解决？

多个线程相互持有对方想要申请的资源 不释放的情况下  
又去申请对方已经持有的资源  
从而双双进入对方资源的锁池当中 产生永久的阻塞

- 1 使用对象的等待池和操作等待池的方法：
- 2 wait() notify() notifyAll()

## 16.锁池和等待池的区别？

过

## 17.shutdown()和shutdownNow()的区别？

过

## 18.线程池是什么 为什么要使用线程池

**线程池**是一种标准的资源池模式 **资源池**会在用户出现之前  
提前预留活跃资源 从而在用户出现的第一时间  
直接提供给用户资源  
并且将资源的创建和销毁都委托给资源池完成 从而优化用户体验

1 | 有效的提高效率 优化用户体验

## 19.创建线程池的三种方法

`newFixedThreadPool(int)` : (固定大小)修复后可重用的  
`newCachedThreadPool()` : 缓存机制的  
`newSingleThreadExecutor()` : 单一实例的

## 20.如果要自己创建**线程池执行器** 需要指定几个参数 各自代表什么含义

- 5个
- 1.线程池中**核心线程的数量**
- 2.线程池中**最大线程数量**
- 3.`KeepAliveTime` **保持活着的时间** 数词
- 4.`TimeUnit` **时间单位** 量词
- 5.**一个队列** 用于存放排队的线程任务

# day29

## java.io.File 文件

构造方法:

`new File(String 路径)`  
`new File(String 父目录,String 文件名)`  
`new File(File 父目录对象,String 文件名)`

```
1 *: 路径 = 相对路径 or 绝对路径
2 相对路径: 从程序认定的主目录出发定位我们要找的文件
3 绝对路径: 从盘符或者根目录出发定位我们要找的文件
4
5 if(str.charAt(1) == ':' || str.startsWith("/")){
6 绝对路径
7 }else{
8 相对路径
9 }
10
11 *: File.separator 路径分界符
12
13 挖掘机技术哪家强? 中国山东找蓝翔
14 蓝翔里面有小明.exe 请用兼容性最好的方式
15 定位文件小明.exe
```

```

16
17 File f1 = new File("中国\\山东\\蓝翔\\小明.exe");
18 // 50
19
20 File f2 = new File("中国/山东/蓝翔/小明.exe");
21 // 60
22
23 StringBuffer buff = new StringBuffer();
24 buff.append("中国").append(File.separator);
25 buff.append("山东").append(File.separator);
26 buff.append("蓝翔").append(File.separator);
27 buff.append("小明.exe");
28 File f3 = new File(buff.toString());
29 // 80
30 File sd = new File("中国","山东");
31 File bf = new File(sd,"蓝翔");
32 File xm = new File(bf,"小明.exe");
33 //100

```

## 特等优先级方法：3个 `listRoot()` `list()` `listFiles()`

`static listRoots()` : 列出当前计算机的所有根目录

`String[] list()` : 列出一个目录当中所有的文件名字

`File[] listFiles()` : 列出一个目录当中所有的文件对象

## 一等优先级方法：12个

4: 只读操作的方法

```

1 exists() : 判断File对象代表的目录或者文件是否已经存在
2 *: File对象既可以代表已经存在的文件
3 也可以代表尚不存在的文件
4
5 isFile() : 判断File对象代表的是不是一个文件
6 isDirectory() : 判断File对象代表的是不是一个目录
7 *: File对象既可以代表一个文件
8 又可以代表一个目录
9
10 length() : 得到文件的字节个数 long类型
11 length length() size() length()
12 数组 字符串 集合 文件
13 int int int long
14 *: 该方法不要对目录调用 得到的是非预期结果
15
16 3 写操作的方法
17
18 ===== 注意 如下三个方法高危 谨慎测试 =====
19
20 delete() : 删除File对象代表的文件或者目录...
21 *: 注意 该方法不会经过回收站..就没了
22 *: 注意 如果要删除的是个目录 则必须保证目录是空的
23 否则删除失败
24
25 mkdirs() : make directories = 创建多层目录结构
26 *: File类还有一个方法叫mkdir()

```

```

27 这个方法只能建一层不存在的目录结构
28 renameTo() : 重命名文件或者目录
29
30 a.renameTo(c); a源文件 c目标文件
31 *: a必须exists() c必须!exists()
32 *: 如果a和c给出不同的目录结构 可以实现移动剪切

```

```

1 3 得到信息
2 getName() : 得到文件或者目录的名字
3 getParent() : 得到文件或者目录的父目录
4 getAbsolutePath() : 得到文件或者目录的绝对路径
5
6 2 修改时间
7 setLastModified() : 设置文件最后一次修改时间
8 lastModified() : 得到文件最后一次修改时间

```

\*: 如何解析时间戳:

1. java.util.Date

```

 getYear() getMonth()+1 getDate()
 getHours() getMinutes() getSeconds()

```

2. java.util.Calendar

```

 getInstance() setTimeInMillis()
 get(x) 1 2+1 5 11 12 13 7-1

```

3. java.text.SimpleDateFormat

**format()** : 从时间戳到字符串

**parse() + getTime()** : 从字符串到时间戳

## 总结:

## File类的方法

构造方法

```

 new File(String 路径)
 new File(String 父目录,String 文件名)
 new File(File 父目录对象,String 文件名)

```

## 一等优先级方法 12

```

4 exists() isFile() isDirectory() length()
3 delete() mkdirs() renameTo()
3 getName() getParent() getAbsolutePath()
2 setLastModified() lastModified()

```

## 时间戳解析 解析时间的三种方式和n多方法~

```
1 1st. java.util.Date
2 getYear()+1900 getMonth()+1 getDate()
3 getHours() getMinutes() getSeconds()
4 getDay()
5
6 2nd. java.util.Calendar
7 getInstance()
8 setTimeInMillis()
9 get(x) 1 2+1 5 11 12 13 7-1
10
11 3rd. java.text.SimpleDateFormat
12 format() : long -> String
13 parse() + getTime() : String -> long
```

## day30

### review

### java.io.File

#### 构造方法:

```
new File(String 路径)
new File(String 父目录,String 文件名)
new File(File 父目录对象,String 文件名)
```

```
1 *: 相对路径 or 绝对路径
2 *: File.separator 路径分界符..
```

#### 特等优先级方法:

```
static listRoots(): 得到一台计算机的所有根目录
String[] list(): 列出一个目录当中所有的文件名字
File[] listFiles(): 列出一个目录当中所有的文件对象
```

#### 一等优先级方法:

```
4: exists() isFile() isDirectory() length()
3: delete() mkdirs() renameTo()
3: getName() getParent() getAbsolutePath()
2: setLastModified() lastModified()
```

#### 解析时间戳:

```
1st. java.util.Date
 getYear()+1900 getMonth()+1 getDate()
 getHours() getMinutes() getSeconds()
```



```

1 2nd. java.util.Calendar
2 getInstance()
3 setTimeInMillis()
4 get(x) 1 2+1 5 11 12 13 7-1
5
6 3rd. java.text.SimpleDateFormat
7 format() : long -> String
8 parse() + getTime() : String->long

```

## IO流

IO流~ I = Input = 输入 O = Output = 输出

1 流 = 数据从源点传输到汇点的"管道"

流的分类:

按照方向分: 输入流 输出流 \*:参照物 = 当前程序

按照单位分: 字节流 字符流

按照功能分: 节点流 过滤流(包装流、处理流)

一起上路:

**InputStream** 所有字节输入流统一的父类 抽象类

```

int read()
int read(byte[] data)
int read(byte[] data,int off,int len)

```

**OutputStream** 所有字节输出流统一的父类 抽象类

```

write(int data)
write(byte[] data)
write(byte[] data,int off,int len)

```

**FileInputStream** 输入流 字节流 节点流

**FileOutputStream** 输出流 字节流 节点流

\*: 它们都是节点流 构造方法允许传入File对象/String文件路径

\*: 它们都是节点流 但是不能连接目录 只能连接文件...

\*: FileInputStream 最常用的read(byte[])

\*: FileOutputStream 最常用的write(byte[],int,int)

\*: FileInputStream 以-1作为读取结束的标志

\*: FileOutputStream 是节点输出流

节点输出流创建对象的时候 如果连接的文件不存在  
也会在建流的那一刻 自动创建出来

所以File类当中有个方法叫createNewFile() 咱没讲

但是 如果连连接的目录结构都不存在

不但不会创建 还会出现异常

所以File类当中有个方法叫mkdirs() 一等优先级

\*: FileOutputStream是节点输出流 它有极强的杀伤性

节点输出流创建对象的时候 即便连接的文件已经存在

也会在创建流的那一刻被新的空白文件**直接替换**  
如果我们的需求是在原有内容之后追加连接新内容而不要替换  
可以**构造方法传参 指定追加模式开启**  
**new FileOutputStream("abc.txt",true);**  
\*: 必须学会标准**try-catch** 和 **TWR**两种处理异常

## day31

### review

**流**: 数据从**源点**传输到**汇点**的**"管道"**

三种分类:

按照**方向**: 输入流 输出流  
按照**单位**: 字节流 字符流  
按照**功能**: 节点流 过滤流

**InputStream** 所有字节输入流统一的父类 抽象类

```
int read()
int read(byte[] data) *
int read(byte[] data,int off,int len)
```

**OutputStream** 所有字节输出流统一的父类 抽象类

```
write(int data)
write(byte[] data)
write(byte[] data,int off,int len) *
```

**FileInputStream** 输入流 字节流 节点流

**FileOutputStream** 输出流 字节流 节点流

- \*: 它们都是节点流构造方法允许传入File对象/String路径
- \*: 它们都是**节点流**但是**只能连接文件不能连接目录** [**FileNotFoundException .. 拒绝访问**]
- \*: **FileInputStream** 最常用的**read(byte[])**
- \*: **FileOutputStream** 最常用的却是**write(byte[],int,int)**
- \*: **FileInputStream** 以-1作为读取结束的标识
- \*: **FileOutputStream** 是节点输出流
- 节点输出流**创建对象的时候 如果连接的**文件**不存在  
也会自动创建出来 不用程序员自己建  
但是如果**连接的目录结构都不存在** 将直接异常(**FileNotFoundException ...系统找不到指定路径**)
- \*: 节点输出流创建对象的时候 如果连接的文件已经存在  
也会在创建流的那一刻被新的空白文件**直接替换**  
如果不想替换 而要**追加新内容** 构造方法必须传参  
**new FileOutputStream("a.txt",true);**
- \*: **TWR** 带有资源控制的try catch语法 (自动关闭)

## [课程内容]

### DataOutputStream DataInputStream

InputStream 所有字节输入流统一的父类 抽象类

OutputStream 所有字节输出流统一的父类 抽象类

FileInputStream 输入流 字节流 节点流

FileOutputStream 输出流 字节流 节点流

BufferedInputStream 输入流 字节流 过滤流

BufferedOutputStream 输出流 字节流 过滤流

\*: 作为过滤流的它们是为了给原本的流添加缓冲空间

从而提高每次读写吞吐量 进而提高效率的

\*: 它们都是过滤流 不能直接连接文件 只能连接其它的流

\*: 它们的构造方法第二个参数 都可以指定缓冲空间大小

默认8192字节 8k

\*: BufferedInputStream 最常用的是read()

\*: BufferedOutputStream 最常用的是write(int data)

\*: BufferedInputStream 同样以-1作为读取结束的标识

\*: BufferedOutputStream 是带缓冲区的输出流

使用带缓冲区的输出流 务必注意及时清空缓冲

以防止数据滞留缓冲空间导致丢失...

缓冲区清空:

1.满了自动清空无需操作

2.关闭流的操作会触发清空缓冲

3.主动调用方法清空: flush();

DataInputStream 输入流 字节流 过滤流

DataOutputStream 输出流 字节流 过滤流

\*: 作为过滤流的它们 给原本的流添加读写基本数据类型的功能

\*: 作为过滤流的它们 不能直接连接文件 只能连接其它的流

```
1 *: boolean char byte short int long float double
```

```
2 *: DataInputStream 核心方法 readXxxx() - 有返回值
```

```
3 *: DataOutputStream 核心方法 writeXxxx() - 有参数
```

```
4 *: DataInputStream 不能再以-1作为读取结束的标识了
```

```
5 而是一旦到达文件结尾还继续尝试读取
```

```
6 将直接触发EOFException -> End Of File
```

### ObjectInputStream ObjectOutputStream

ObjectInputStream 输入流 字节流 过滤流

ObjectOutputStream 输出流 字节流 过滤流

\*: 作为过滤流的它们 给原本的流添加读写对象的功能

\*: 作为过滤流的它们 不能直接连接文件 只能连接其它的流

\*: ObjectInputStream 核心方法 readObject() - 有返回值

\*: ObjectOutputStream 核心方法 writeObject() - 有参数

\*: ObjectInputStream 同样不以-1作为读取结束的标志

\*: **想要持久化 必须先要序列化**

想要保存到磁盘上的对象的类型

必须要实现**Serializable接口**

如果这个类型当中有其它引用类型的属性

就连这些属性的类型也必须要实现序列化接口

如果某些属性无关紧要 不需要参与持久化保存

可以使用修饰符**transient**修饰

transient 短暂的 转瞬即逝的 不参与持久化的

\*: 如果想要持久化的是一个集合对象

则必须保证集合当中的元素的类型

必须实现序列化接口

如果要持久化的是一个使用了比较器的TreeSet或者TreeMap

就连这个比较器的类 也必须要实现序列化接口

(因为这个比较器是TreeSet或者TreeMap的一个属性)

=====

## 字节流内容汇总

字节流要点汇总:

InputStream 所有字节输入流统一的父类 **抽象类**

OutputStream 所有字节输出流统一的父类 **抽象类**

FileInputStream 节点流 用于连接文件

FileOutputStream 节点流 用于连接文件

\*:不能连接目录 只能连接文件

\*:FileOutputStream **有极强的杀伤性**

new对象的时候 会覆盖文件 除非指定追加模式

BufferedInputStream 给原本的流**添加缓冲空间** 从而提高读取效率  
read()

BufferedOutputStream 给原本的流添加缓冲空间 从而提高写出效率  
write(int data)

\*: 带缓冲区的输出流 一定注意及时清空缓冲 否则数据丢失

DataInputStream 给原本的流添加**读取基本数据类型**的功能  
readXxxx()

DataOutputStream 给原本的流添加**写出基本数据类型**的功能  
writeXxxx();

ObjectInputStream 给原本的流添加**读取对象**的功能  
readObject();

ObjectOutputStream 给原本的流添加**写出对象**的功能  
writeObject();

## \*: 复制文件的时候 文件变大了

没有使用三个参数的write(byte[],int,int)

会将上次读取的剩余数据 再次写出到目标文件

## \*: 复制文件的时候 文件变小了

使用了带缓冲区的输出流 没有及时清空缓冲

数据滞留在缓冲空间当中

## \*: NotSerializableException

要持久化到磁盘当中的对象的类型 没有实现序列化接口

## \*: InvalidClassException

不成立的类型异常

使用ObjectOutputStream写出对象之后

使用ObjectInputStream读取对象之前

对这个类进行了修改 从而导致

序列化的唯一序列号 不一致了

```
1 Buffered Data Object
2 加缓冲提高效率 读写基本数据类型 读写对象
```

学好IO流必须要会的8大需求:

```
1 1.使用FileInputStream+FileOutputStream
2 配合一个数组 完成文件复制
3
4 2.使用BufferedInputStream + BufferedOutputStream
5 一次一个字节的 完成文件复制
6
7 3.将内存当中一个基本数据类型的变量 DataOutputStream
8 例如 double PI = 3.14159265;
9 保存到磁盘文件当中
10
11 4.从文件当中读取那个基本数据类型的double回来
12 DataInputStream readDouble()
13
14 5.将内存当中一个引用类型的对象 ObjectOutputStream
15 Student stu = new Student()保存到磁盘文件中
16 *: 必须序列化...
17
18 6.从文件当中读取一个引用类型的对象 ObjectInputStream
19
20 7.
21
22 8.
```

=====

# day32

## review

InputStream      所有字节输入流统一的父类 抽象类  
OutputStream    所有字节输出流统一的父类 抽象类  
FileInputStream    字节流 输入流 节点流  
FileOutputStream   字节流 输出流 节点流  
BufferedInputStream   字节流 输入流 过滤流(添加缓冲提高读取效率) **read()**  
BufferedOutputStream   字节流 输出流 过滤流(添加缓冲提高写出效率) **write(int data)**  
DataInputStream    字节流 输入流 过滤流(提供读取基本数据类型功能) **readXxxx()**  
DataOutputStream   字节流 输出流 过滤流(提供写出基本数据类型功能) **writeXxxx()**  
ObjectInputStream   字节流 输入流 过滤流(提供读取对象功能) **readObject()**  
ObjectOutputStream   字节流 输出流 过滤流(提供写出对象功能) **writeObject()**

=====

## 字符流最核心的两个需求

### 7.以一行为单位**写出**文本文件 **PrintWriter**

核心代码:

```
PrintWriter pw = new PrintWriter("春晓.txt");
pw.println("春眠不觉晓");
pw.close();
```

```
1 TWR:
2 try(PrintWriter pw = new PrintWriter("春晓.txt")){
3 pw.println("春眠不觉晓");
4 }catch(Exception e){
5 e.printStackTrace();
6 }
```

### 8.以一行为单位**读取**文本文件 **BufferedReader**

核心代码:

```
FileReader fr = new FileReader("focus.txt");
BufferedReader br = new BufferedReader(fr);
String str;
while((str = br.readLine())!=null){
 System.out.println(str);
}
br.close();
```

```
1 TWR:
2 try(BufferedReader br = new BufferedReader(new FileReader("focus.txt"))){
3 String str;
4 while((str = br.readLine())!=null){
5 System.out.println(str);
6 }
7 }catch(Exception e){
8 e.printStackTrace();
9 }
```

## focus

**Reader** 所有**字符输入流**统一的父类 抽象类

```
int read()
int read(char[] data)
int read(char[] data,int off,int len)
```

**Writer** 所有**字符输出流**统一的父类 抽象类

```
write(int data)
write(char[] data)
write(char[] data,int off,int len)
```

**FileReader** 字符流 输入流 节点流

**FileWriter** 字符流 输出流 节点流

- \*: 它们都是节点流 构造方法可以传入**File对象/String路径**
- \*: 它们都是节点流 但是**只能连接文件 不能连接目录**  
否则直接出现异常 FileNotFoundException (拒绝访问)
- \*: FileReader 最常用的read(char[] data)
- \*: FileWriter 最常用的却是write(char[],int,int)
- \*: FileReader 以-1作为读取结束的标识
- \*: FileWriter 是**节点输出流**  
节点输出流创建对象的时候 即便连接的文件不存在  
也会在创建流的那一刻 自动创建出来 不需要自己建  
File类有个createNewFile() 咱们没讲  
但是如果其连接的目录结构都不存在 将直接异常  
File类的mkdirs() **一等优先级**
- \*: **FileWriter** 是节点输出流 有极强的**杀伤性**  
如果创建对象的时候 连接的文件已经存在  
也会在创建流的那一刻 被新的空白文件直接替换  
如果不想替换 想要追加连接新内容  
可以构造方法传参 **指定追加模式开启**  
new FileWriter("a.txt",true)

**BufferedReader** 字符流 输入流 过滤流

**BufferedWriter** 字符流 输出流 过滤流

- \*: 作为**过滤流**的它们 给原本的流添加一个变长的缓冲空间  
从而实现以行(hang)为单位的读写
- \*: 作为过滤流的它们 **不能直接连接文件 只能连接其它的流**
- \*: BufferedReader String **readLine()**
- \*: BufferedWriter write(String) + newLine()
- \*: BufferedReader 以**null**作为读取结束的标识

**PrintWriter** 比 **BufferedWriter** 强大很多~

- 1.它既可以当做**节点流** 又可以当做**过滤流**  
构造方法允许传入 File对象/String文件路径/流
- 2.它既可以连接字节流 又可以连接字符流  
构造方法允许传入 OutputStream / Writer
- 3.当做**节点流**使用的时候 **构造方法第二个参数可以指定字符编码**  
**PrintWriter pw = new PrintWriter("a.txt","utf-8");**
- 4.当做过滤流使用的时候 **构造方法第二个参数可以指定自动清空缓冲**

```

 PrintWriter pw = new PrintWriter(fw,true);
5.println() = write() + newLine()
6.我们对他的孪生兄弟特别熟悉 PrintStream
 System.out out属性就是PrintStream类型的

```

=====

## 复习 ()

1 用FileInputStream + FileOutputStream

配合一个**大数组**完成文件复制

7 以**行**为单位写出文本文件(指定编码)

```

PrintWriter pw = new PrintWriter("a.txt","utf-8");
pw.println("内容");
pw.close(); //

```

```

1 *: PrintWriter pw =
2 new PrintWriter(new FileWriter("a.txt",true),true);

```

8 以一行为单位读取文本文件 (万一要求指定编码就不能直接上字符流)

```

FileInputStream fis = new FileInputStream("a.txt");
InputStreamReader r = new InputStreamReader(fis,"utf-8");
BufferedReader br = new BufferedReader(r);
String str;
while((str = br.readLine())!=null){
 System.out.println(str);
}
br.close();

```

2 用**BufferedInputStream + BufferedOutputStream**

一次一个字节的完成文件复制

3  
4  
5  
6

\*: File类方法

```

4 exists() isFile() isDirectory() length()
3 delete() mkdirs() renameTo()
3 getName() getParent() getAbsolutePath()
2 lastModified() setLastModified()

```

```

1 *: 特等优先级的 lastRoots() list() listFiles()
2 *: 写操作的 delete() mkdirs() renameTo() setLastModified()
3 *: 特别长的 getAbsolutePath()
4
5 | 获得时间 lastModified() setLastModified()
6 v

```



\*: 时间戳解析

```
java.util.Date
 getYear()+1900 getMonth()+1 getDate()
 getHours() getMinutes() getSeconds()
```

```
1 java.util.Calendar
2 getInstance()
3 setTimeInMillis()
4 get(x) 1 2+1 5 11 12 13 7-1
5
6 java.text.SimpleDateFormat
7 有long 到 String : format()
8 有String 到 long : parse() + getTime()
```

\*: 递归遍历 核心代码

\*: 流

FileInputStream + FileOutputStream

```
1 EOFException -> DataInputStream ObjectInputStream
2 InvalidClassException ->
3
4 *: TWR!
5 *: 文件变大了 变小了
```

=====

## day33

### URL

URL => 统一资源定位符 => 网址

1. 直接下载文件 -> 字节流 原封不动的直接读取和写出2进制数据

```
URL url = new URL("");
URLConnection uc = url.openConnection();
InputStream is = uc.getInputStream();
FileOutputStream fos = new FileOutputStream("../");
...
```

2. 读取互联网上的数据 -> 字符流

## day34

### 01. 请用兼容性最好的方式创建File对象代表文件

中国/山东/济南/易途/小明.exe

```
1 *: 路径分界符 和 File类构造方法
2
```

```

3 a> String str = "中国"+File.separator+"山东"+File.separator+"济
 南"+File.separator+"易途"+File.separator+"小明.exe";
4 File file = new File(str);
5 *: 涉及到大量字符串连续追加 效率较差
6
7 b> StringBuffer buff = new StringBuffer("中国");
8 buff.append(File.separator).append("山东");
9 buff.append(File.separator).append("济南");
10 buff.append(File.separator).append("易途");
11 buff.append(File.separator).append("小明.exe");
12 File file = new File(buff.toString());
13
14 c> File sd = new File("中国","山东");
15 File jn = new File(sd,"济南");
16 File etoak = new File(jn,"易途");
17 File xm = new File(etoak,"小明.exe");

```

## 02.获取E盘下(e:/etoak/img/et2301/Jay.png)的文件最后修改时间

并格式化为(年-月-日 时:分:秒)

```

1 File类方法得到最后修改时间 格式化显示时间
2
3 File file = new File("e:/etoak/img/et2301/Jay.png");
4 long time = file.lastModified();
5 SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
6 String ok = sdf.format(time);

```

## 03.递归遍历D:\所有.jpg图片

略

## 04.请写出字节过滤流都有哪些 各自的作用是什么?

BufferedInputStream 添加缓冲空间提高读取效率 [小卖部备货]

BufferedOutputStream 添加缓冲空间提高写出效率 [你的肚子]

DataInputStream 添加读取基本数据类型功能

DataOutputStream 添加写出基本数据类型功能

ObjectInputStream 添加读取对象功能

ObjectOutputStream 添加写出对象功能

## 05.Java当中如何解析日期和时间 请列出三种不同方式 (要求有具体方法)

```

a> java.util.Date
 getYear()+1900 getMonth()+1 getDate()
 getHours() getMinutes() getSeconds()
 getTime() getDay()

```

```

1 b> java.util.Calendar
2 getInstance() setTimeInMillis()
3 get(x) 1 2+1 5 11 12 13 7-1
4
5 c> java.text.SimpleDateFormat
6 format()
7 parse()+getTime()

```

## 06. 请将文件et2301.txt的最后一次修改时间修改为三天前?

```

1 三天前 = 原本修改时间的三天前
2 a> File file = new File("et2301.txt");
3 long time = file.lastModified();
4 time -= 3L*24*60*60*1000;
5 file.setLastModified(time);
6
7 三天前 = 现在的三天前
8 b> File file = new File("et2301.txt");
9 long time = System.currentTimeMillis();
10 time -= 3L*24*60*60*1000;
11 file.setLastModified(time);

```

## 07. 将字符串 String str = "2023-02-24 16:29"转换成时间戳(毫秒数)

```

String str = "2023-02-24 16:29";
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm");
long time = sdf.parse(str).getTime();

```

## 08. 使用TWR语法完成文件复制 源文件(six.mp4)目标文件(666.mp4)

```

try(FileInputStream fis = new FileInputStream("six.mp4");FileOutputStream fos = new
FileOutputStream("666.mp4")){
 byte[] data = new byte[65536];
 int len;
 while((len = fis.read(data))!=-1){
 fos.write(data,0,len);
 }
}catch(Exception e){
 e.printStackTrace();
}

```

## 09. EOFException是什么异常 怎么触发这个异常?

End Of File 意外达到文件结尾的异常

```

1 使用DataInputStream 或者 ObjectInputStream
2 读取数据的时候 已经到达文件结尾还继续尝试读取
3 将直接触发该异常

```

## 10. ObjectOutputStream的核心方法是什么?它如何表达读取结束?

writeObject()

这是输出流 不能读取数据

## 11. InvalidClassException在什么情况下会出现?

使用ObjectOutputStream写出对象之后

使用ObjectInputStream读取对象之前

对这个类的属性和方法做了改动

从而导致序列化的唯一版本号发生了变化 不一致了...

## 12. BufferedInputStream构造方法第二个参数是什么含义?

指定缓冲空间的大小

- |   |    |        |                  |
|---|----|--------|------------------|
| 1 | 缓冲 | Buffer | [教室里面多留空座 叫缓冲]   |
| 2 | 缓存 | Cache  | [知道你要用这些对象 提前造好] |

## 13. 在复制文件的时候 如果文件变小了 通常是为什么?

使用了带缓冲区的输出流 并没有及时清空缓冲

所以导致程序执行结束时 数据滞留在缓冲区了

## 14. 缓冲区在什么情况下会清空

a> 满了自动清空 无需操作

b> 关闭流的操作触发清空缓冲 (close() / TWR)

c> 主动调用flush();

## 15. 现在有utf-8编码的文本文件focus.txt 请逐行读取它的内容

```
FileInputStream fis = new FileInputStream("focus.txt");
InputStreamReader r = new InputStreamReader(fis,"utf-8");
BufferedReader br = new BufferedReader(r);
String str;
while((str = br.readLine())!=null){
 System.out.println(str);
}
br.close();
```

## 16. 请将《咏鹅》写出到ye.txt当中 要求ye.txt使用utf-8编码

```
PrintWriter pw = new PrintWriter("ye.txt","utf-8");
pw.println("《咏鹅》");
pw.close();
```

## 17. PrintWriter pw = new PrintWriter(new FileWriter("a.txt",true),true);

两个true的含义

```
1 | 第一个true是传给节点流FileWriter的
2 | 代表追加模式打开目标文件 appendMode
3 |
4 | 第二个true是传给PrintWriter的
5 | 代表自动清空缓冲区(见到换行) autoFlush
```

## 18.File类对磁盘有写操作的方法有哪些?

delete() mkdirs() renameTo()  
setLastModified()

```
1 | mkdir() createNewFile()
```

## 19.请写出两种最常见的字符流 并写出它们的核心方法

BufferedReader readLine()  
PrintWriter println()

## 20.文件下载

过...  
URL url = new URL("...");  
URLConnection uc = url.openConnection();  
InputStream is = uc.getInputStream();

```
1 | FileOutputStream fos = new FileOutputStream("etoak2301.png");
2 |
```

# day35

## Socket (java.net.Socket)

套接字 Socket => 一根"电话线" (当中包含两股线 一股输入 一股输出)

CS架构

C = Client = 客户端 S = Server = 服务器  
例如: QQ

BS架构 (J2EE)

```
1 | 服务器端: 客户端:
```

=====

## GUI

GUI => G = 图形 U = 用户 I = 接口

图形用户接口 => 用户图形界面

```
1 java.awt.*; 重量级组件 Button
2 javax.swing.*; 轻量级组件 JButton
3
4 java.awt.event.*; 事件监听模型
```

## 开发用户图形界面常见的6个步骤：

### 1.选择容器和组件

**组件：**Component 界面上能够看到的任何事物都被称之为一个组件

**容器：**Container 当组件当中能够添加其它组件的时候 这个组件就是容器

```
1 常见的容器： JFrame 窗体 JPanel 面板
2 常见的组件：
3 JButton 按钮
4 JTextField 单行文本框
5 JPasswordField 单行密码框
6 JTextArea 多行文本域
7 JLabel 标签
8 JMenuBar 菜单条
9 JMenu 菜单
10 JMenuItem 菜单项
```

### 2.初始化容器和组件

#### 3.选择布局管理器

a> **BorderLayout** 边框布局 它是JFrame的默认布局

把容器的可视范围分割为东西南北中五个区域

每个区域只允许添加一个组件

不尊重组件的原始大小

直接拉伸占满整块区域

当东西南北四个区域当中有未使用的区域的时候

将直接被中央及其小弟占领

b> **FlowLayout** 流水布局 它是JPanel的默认布局

它按照从左到右的顺序依次摆放添加的组件

尊重组件的原始大小 不会拉伸 不会缩放

如果一行摆放不开 则自动换行并且依然居中对齐

c> **GridLayout** 网格布局 它不是任何容器的默认布局

按照指定的行数列数将可视范围分割成网格

每个单元格 只能添加一个组件

不尊重组件原始大小 直接拉伸占满整个单元格

按照从左到右 自上而下的顺序 依次添加...

### 4.将组件添加进容器

容器.add(组件);

### 5.添加事件监听器

### 6.设置窗体属性

a> 设置窗体大小

b> 设置是否可见

c> 设置默认关闭操作

```
=====
1 import java.awt.*;
2 import javax.swing.*;
3 public class LoginFrame{
4 JFrame frame;
5 JPanel top,center,bottom;
6
7 JLabel lab1,lab2,pic;
8 JTextField username,password;
9 JButton register,submit,reset;
10
11 JButton[] bts;
12 public LoginFrame(){
13 frame = new JFrame("易途盛世大酒店 只能点餐系统 v0.8");
14 top = new JPanel();
15 center = new JPanel(new BorderLayout());
16 bottom = new JPanel(new GridLayout(4,9));
17
18 lab1 = new JLabel("用户名:");
19 lab2 = new JLabel("密码:");
20 pic = new JLabel(new ImageIcon("jay.png"));
21 username = new JTextField(12);
22 password = new JPasswordField(12);
23 register = new JButton("注册");
24 submit = new JButton("登录");
25 reset = new JButton("重置");
26
27 bts = new JButton[36];
28 for(int i = 0;i<bts.length;i++){
29 bts[i] = new JButton(i<10 ? i+" " : (char)(i+87)+"");
30 bottom.add(bts[i]);
31 }
32
33 center.add(pic);
34
35
36 top.add(lab1);
37 top.add(username);
38 top.add(lab2);
39 top.add(password);
40 top.add(register);
41 top.add(submit);
42 top.add(reset);
43
44 frame.add(top,"North");
45 frame.add(center);
46 frame.add(bottom,"South");
47
48 frame.setSize(800,640);
49 frame.setVisible(true);
50 frame.setDefaultCloseOperation(3);
51
52 }
53 public static void main(String[] args){
```

```
54 new JFrame();
55 }
56 }
```

## day36

### 01.软键盘的监听器

注意 重点在于 它需要监听36个按钮  
所以 在被点击之后 我们需要先得到 是谁被按了..

\*: 注意 **成员内部类** 为了方便共享外部类的组件和容器

核心代码:

```
String cmd = ae.getActionCommand();//得到动作指令...
String old = password.getText();//得到原本的文字
password.setText(old + cmd);
```

### 02.注册 登录

得到用户名框框里面的东西 得到密码框框里面的东西  
校验它们是否合法 不合法就骂人  
合法就先这样吧  
准备连服务器...

核心代码:

```
String name = username.getText();
String pwd = password.getText();
```

### 03.重置

把用户名和密码框框里面的内容都清空

核心代码:

```
password.setText("");
username.setText("");
```

### 04.菜单项的监听

换图片

```
ccc.removeAll();//移除原本的所有组件
ccc.add(new JLabel(new ImageIcon("img/"+cmd+".jpg")));//放入新图片
ccc.updateUI();//强制刷新
```

换标签的字



```
int price = nameAndPrice.get(cmd);
title.setText(cmd + " : " + price + " 元");
记录当前菜是谁
current = cmd;
```

## 05.我要点这道菜

```
list.add(current);
int price = nameAndPrice.get(current);
sum += price;
total.setText("当前总共消费: "+sum+" 元");
```

## 06.提交菜单

这个又需要连接后台服务器了 暂时不做

## 07.重置菜单

```
list.removeAll();
sum = 0;
total.setText("当前总共消费: 0 元");
```

=====

=====

## Frame登录代码

```
1 submit = new JButton("登录");
2 submit.addActionListener((ae) -> {
3 //1.采集用户数据
4 String name = username.getText();
5 String pwd = password.getText();
6 //2.校验数据是否合法
7 if(name.trim().length() == 0 || pwd.trim().length() == 0){
8 JOptionPane.showMessageDialog(frame,"请先输入用户名和密码之后重试");
9 return;
10 }
11 //System.out.println("数据准备就绪 即将连接服务器");
12
13 Request req = new Request();
14 req.setAskNo(1002);//登录
15 req.setParameter("username",name);
16 req.setParameter("password",pwd);
17
18 Response res = ClientNetTools.sendAndRead(req);
19 int result = res.getResult();
20 //6.根据结果提示用户
21 if(result == 0){
22 JOptionPane.showMessageDialog(frame,"登录成功");
23 frame.setVisible(false);//隐藏登录窗体
24 try{
25 new OrderFrame(name);//弹出点菜窗体
26 }catch(Exception e){
27 e.printStackTrace();
```

```
28 }
29 }else if(result == 1){
30 JOptionPane.showMessageDialog(frame,"用户名错误 请重新登录");
31 }else if(result == 2){
32 JOptionPane.showMessageDialog(frame,"密码错误 请重新登录");
33 }
34 });
```