专题一、数字和字符串

1.1 装箱和拆箱

封装类：所有的基本类型，都有对应的类类型，比如int对应的类是Integer ，这种类就叫做封装类

Number类：数字封装类有，Byte,Short,Integer,Long,Float,Double ，这些类都是抽象类Number的子类

基本类型转封装类、封装类转基本类型（见程序Test）

自动装箱：不需要调用构造方法，通过=符号自动把 基本类型 转换为 类类型 就叫装箱

自动拆箱：不需要调用Integer的intValue方法，通过=就自动转换成int类型，就叫拆箱

int的最大值可以通过其对应的封装类Integer.MAX\_VALUE获取

1.2 字符串转换

数字转字符串：方法一：使用String类的静态方法valueOf 方法二：先把基本类型装箱为对象，然后调用对象的toString

字符串转数字：调用Integer的静态方法parseInt

1.3数学方法：java.lang.Math提供了一些常用的数学运算方法，并且都是以静态方法的形式存在

四舍五入round、开方、随机数、次方、自然常数、Π

1.4格式化输出（见程序Test）

如果不使用格式化输出，就需要进行字符串连接，如果变量比较多，拼接就会显得繁琐

使用格式化输出，就可以简洁明了

%s 表示字符串

%d 表示数字

%n 表示换行

printf和format能够达到一模一样的效果，如何通过eclipse查看java源代码 可以看到，在printf中直接调用了format

换行符：换行符就是另起一行 --- 'n' 换行（newline）

回车符就是回到一行的开头 --- 'r' 回车（return）

在eclipse里敲一个回车，实际上是回车换行符

Java是跨平台的编程语言，同样的代码，可以在不同的平台使用，比如Windows,Linux,Mac

然而在不同的操作系统，换行符是不一样的

（1）在DOS和Windows中，每行结尾是 “rn”；

（2）Linux系统里，每行结尾只有 “n”；

（3）Mac系统里，每行结尾是只有 r。

为了使得同一个java程序的换行符在所有的操作系统中都有一样的表现，使用%n，就可以做到平台无关的换行（都可以用）

总长度、左对齐、补0、千分位分隔符、小数点位数、本地化表达（见程序Test）

1.5字符

Character常见方法：isLetter(), isDigit(), isWhitespace(), isUpperCase(), isLowerCase(), toUpperCase(), toLowerCase(), toString()

常见转义：（见程序Test）''用来输出字符''

1.6字符串

String是final的

immutable 是指不可改变的，比如创建了一个字符串对象String garen =盖伦;

不可改变的具体含义是指：不能增加长度；不能减少长度；不能插入字符；不能删除字符；不能修改字符

一旦创建好这个字符串，里面的内容 永远 不能改变

String 的表现就像是一个常量

1.7对字符串的操作

charAt, toCharArray, subString, split, trim, toLowerCase, toUpperCase, indexOf, lashIndexOf, contains, replaceAll, replaceFirst

1.8比较字符串

判断字符串内容相同：不能用==，用equals，equalsIgnoreCase

startWith，endsWith

1.9 StringBuffer

是可变长的字符串

方法：append delete insert reverse length capacity

比较String 和 StringBuffer的性能（见程序Test）

测试MyStringBuffer性能（见程序Test）

1047 20201228

专题二 日期和日历

2.1（java.util.Date）

创建Data对象：Date()和Date(long mils)

getTime() 得到一个long型的整数，这个整数代表 从1970.1.1 080000000 开始 每经历一毫秒，增加1

System.currentTimeMillis()：当前日期的毫秒数

new Date().getTime() 和 System.currentTimeMillis() 是一样的不过由于机器性能的原因，可能会相差几十毫秒，毕竟每执行一行代码，都是需要时间的

2.2 SimpleDateFormat 日期格式化类

日期转字符串（见程序Test）：format方法，y代表年, M代表月份，d代表，H代表24小时制的时，h代表12小时制的时，m代表分，s代表秒，S代表毫秒

字符串转日期（见程序Test）：parse方法，需要异常处理

2.3日历类Calendar（采用单例模式获取日历对象Calendar.getInstance();）

Calendar与Date进行转换：getTime方法；setTime方法

翻日历：add方法，在原日期上增加年月日；set方法，直接设置年月日（见程序Test）

专题三 异常处理

3.1 异常定义： 导致程序的正常流程被中断的事件，叫做异常

3.2

多异常处理（见程序Test）：1、分别进行catch 2、放在一个catch里统一捕捉 ：从 JDK7开始支持，好处是捕捉的代码更紧凑，不足之处是，一旦发生异常，不能确定到底是哪种异常，需要通过instanceof 进行判断具体的异常类型

finally：无论是否出现异常，finally中的代码都会被执行

throws： 处理办法也是两种，要么是try catch处理掉，要么也是抛出去。

3.3 异常分类

异常分类： 可查异常，运行时异常和错误3种 ，其中，运行时异常和错误又叫非可查异常

可查异常： CheckedException

可查异常即必须进行处理的异常，要么try catch住,要么往外抛，谁调用，谁处理，比如 FileNotFoundException，如果不处理，编译器，就不让你通过

运行时异常RuntimeException指： 不是必须进行try catch的异常

常见运行时异常 除数不能为0异常ArithmeticException ；下标越界异常ArrayIndexOutOfBoundsException ；空指针异常NullPointerException

在编写代码的时候，依然可以使用try catch throws进行处理，与可查异常不同之处在于，即便不进行try catch，也不会有编译错误

Java之所以会设计运行时异常的原因之一，是因为下标越界，空指针这些运行时异常太过于普遍，如果都需要进行捕捉，代码的可读性就会变得很糟糕。

错误Error，指的是系统级别的异常，通常是内存用光了，在默认设置下，一般java程序启动的时候，最大可以使用16m的内存

如例不停的给StringBuffer追加字符，很快就把内存使用光了。抛出OutOfMemoryError，与运行时异常一样，错误也是不要求强制捕捉的

有：AWTError、VirtualMachineError(StackOverflowErrorOutOfMemoryError)

3.4 Throwable类

Exception和Error都继承了该类，所以在捕捉的时候，也可以使用Throwable进行捕捉

3.5自定义异常类（见程序Test）

1019 2021129

专题四 IO

4.1 文件和文件夹对象File 使用相对路径和绝对路径创建

4.1.1 方法：exists isDirectory isFile length lastModified（见Test）setLastModified（见Test） renameTo()（见Test）

list()和listFile() getParent()和getParentFile() mkdir和mkdirs createNewFile()和getParentFile().mkdir（见文件Test） listRoots delete和deleteOnExit

4.2 流就是一系列数据

输入流： InputStream，Reader(unicode编码双字节字符)

FileInputStream FilterInputStream SequenceInputStream ByteArrayInputStream StringBufferInputStream PipedInputStream

CharArrayReader BufferedReader(LineNumberReader) FilterReader InputStreamReader(FileReader) StringReader

FilterInputStream BufferedInputStream PushbackInputSteam LineNumberInputStream DateInputStream(ObjectInputStream)

输出流：OutputStream，Writer

FileOutputSteam FilterOutputSteam 无 ByteArrayOutputStream 无 PipedOutputStream

CharArrayWriter BufferedWriter(无） FilterWriter OutputStreamWriter(FileWriter) StringWriter PrintWriter

FilterOutputStream BufferedOutputStream 无 无 DateOutputStream(ObjectOutputStream) PrintStream

4.3 字节流的读取和写入、含中文文件（见程序Test)

4.4 关闭流的方法：

在try中关闭：如果文件不存在，或者读取的时候出现问题而抛出异常，那么就不会执行这一行关闭流的代码，存在巨大的资源占用隐患。 不推荐使用；

在finally关闭：标准的严谨的关闭流的方式；

使用try()的方式；

4.5 字符流的读取的写入、含中文文件（见程序Test)

2332 202124

4.6 常见编码 ISO-8859-1 ASCII 数字和西欧字母；GBK GB2312 BIG5 中文；UNICODE (统一码，万国码)

2008 202125

4.7 缓存流读写（见程序Test）

4.8 数据流 要用DataInputStream 读取一个文件，这个文件必须是由DataOutputStream 写出的，否则会出现EOFException；

因为DataOutputStream 在写出的时候会做一些特殊标记，只有DataInputStream 才能成功的读取。

使用数据流的writeUTF()和readUTF() 可以进行数据的格式化顺序读写

4.9 对象流 一个对象以流的形式进行传输，叫做序列化。 该对象所对应的类，必须是实现Serializable接口

字节流的正确读写中文（见4.3）、字符流的正确读取中文（见test程序）

默认编码：Charset.defaultCharset()

4.10 System.out 是常用的在控制台输出数据的

System.in 可以从控制台输入数据：使用System.in.read虽然可以读取数据，但是很不方便，使用Scanner就可以逐行读取了

4.11 解决EOFException

在存入对象的时候使用集合的方式来存储对象,这样在获取时就只需要获取集合,然后遍历集合即可，这种是将本来的多个对象转化为一个集合对象,再从集合对象中去获取数据；

在存入对象前或者后面加一个标志,可以在第一位置加一个int类型的值来记录存入的对象个数,也可以在最后加入一个null来做结束标识；

使用异常处理方式

2137 202126

专题五 集合框架 （java集合分为Collection和Map两种体系）

5.1 ArrayList 实现了List接口

5.1.1 数组的局限性：容量限定；ArrayList容器的容量capacity会随着对象的增加，自动增长

5.1.2 常用方法

add contains get indexof remove set(用于替换指定位置的元素) size addAll clear

数组和ArrayList的转换，toArray(见test)如果要转换为一个Hero数组，那么需要传递一个Hero数组类型的对象给toArray()，这样toArray方法才知道，你希望转换为哪种类型的数组，否则只能转换为Object数组

asList接受的参数是T .... t，泛型不能是基本类型，不能用基本类型的数组转list，会识别成数组的list。用基本类型的对象类。

返回的对象是Arrays的内部类，对list的操作仍然反映在原数组上，因此这个list是定长的，不支持add、remove操作；

httpsblog.csdn.netzty1317313805articledetails82771034

5.1.3 遍历

用for循环；用增强型for（无下标，无法初始化）；用iterator（见程序Test）

5.2 其他集合

5.2.1 LinkedList

实现了List接口外，LinkedList还实现了双向链表结构Deque：addLast, addFirst, getLast, getFirst, removeLast, removeFirst

实现了Queue接口：offer, poll, peek

5.2.2 Stack push, pop

5.2.3 二叉树：三个遍历，层序遍历

1649 202137

5.2.4 HashMapkey, value

健不能重复，值可以重复，键重复会覆盖

比较查找内容的性能（ArrayList vs HashMap)(见程序Test）

get和containsKey有区别

5.2.5 HashSetT

Set中的元素，不能重复,既不是按照插入顺序，也不是按照hashcode的顺序， 不同jvm版本顺序不同

It makes no guarantees as to the iteration order of the set;

in particular, it does not guarantee that the order will remain constant over time.

HashSet自身并没有独立的实现，而是在里面封装了一个Map，HashSet作为Map的key，而所有value是一个名为PRESENT的static的Object对象

解决大量重复数据中有几种数据的问题（见Test)

5.2.6 Collection

是一个接口，子接口有List，Deque，Queue，Set

5.2.7 Collections是一个类，容器的工具类,就如同Arrays是数组的工具类（见Test）

reverse

shuffle 混淆List中数据的顺序

sort 对List中的数据进行排序

swap 交换两个数据的位置

rotate 把List中的数据，向右滚动指定单位的长度

synchronizedList 把非线程安全的List转换为线程安全的List（见Test专题八8.4）

5.3.1 比较ArrayList和HashSet

ArrayList 有顺序，HashSet 无顺序

List中的数据可以重复，Set中的数据不能够重复

重复判断标准是

首先看hashcode是否相同

如果hashcode不同，则认为是不同数据

如果hashcode相同，再比较equals，如果equals相同，则是相同数据，否则是不同数据

5.3.2 ArrayList和LinkedList比较

ArrayList 插入，删除数据慢，LinkedList， 插入，删除数据快（见Test）

ArrayList是顺序结构，所以定位很快，指哪找哪。LinkedList 是链表结构，定位慢，必须得一个一个的数过去

5.3.3 HashMap和HashTable比较

继承：HashTable继承自Dirctionary，HashMap继承自AbstractMap，二者均实现了Map接口；

线程安全性：HashTable的方法是同步的，即线程安全的。HaspMap的方法不是同步的，线程安全的。在多线程并发的情况下，可直接使用HashTable；要使用HashMap，就需要对HashMap的同步处理（见Test，未完成）

键值：HashTable中不允许有null键和null值，HashMap中允许出现一个null键，可以存在一个或者多个键的值都为null。

程序中，对于HashMap，如果使用get(参数为 键)方法时，返回结果为null，可能是该键不存在，也可能是该键对应的值为null，这就出现了结果的二义性。

因此，在HashMap中，使用get()方法来查询键对应的值，应该先使用containskey()方法判断是否存在该键。（见程序Test）

遍历：这两个在遍历方式的实现不同。HashTable和HashMap两者都实现了Iterator。但是，由于历史原因，HashTable还使用了Enumeration。

哈希值：HashTable是直接使用对象的hashCode。HashMap是重新计算hash值。

扩容：HashTable和HashMap的底层实现的数组和初始大小和扩容方式。HashTable初始大小为11，并且每次扩容都为：2old+1。HashMap的默认大小为16，并且一 定是2的指数，每次扩容都为old2。

5.3.4 集合中的几种set（比较顺序不同，见Test）

HashSet：哈希表结构，底层由HashMap实现，无特定顺序，不同版本虚拟机的顺序不同

LinkedListSet：按照插入顺序

TreeSet：也是基于Map来实现，具体实现TreeMap，其底层结构为红黑树（特殊的二叉查找树）

具有排序功能，分为自然排序(123456)和自定义排序两类，默认是自然排序

5.4.1 集合框架hashcode原理

HashMap查找比List查找快，原因：每个非基本类型对象都要对应的hashcode，查找时就不用一个一个比较相同，先找到对应的hashcode，再找值

（见Test）自定义字符串的hashcode：

如下是Java API提供的String的hashcode生成办法；

hash=s[0]∗31^(n−1) +s[1]∗31 ^(n−2)+...+s[n−1]=(...(s[0]∗31+s[1])∗31+s[2])∗31+...)+s[n−1]

s[0] 表示第一位字符

n表示字符串的长度

(见Test)自定义HashMap

（见Test）比较HashMap和ArrayList的性能

5.4.2 比较器（见Test）

5.4.3 聚合操作 （未完成）

JDK8之后，引入了对集合的聚合操作，可以非常容易的遍历，筛选，比较集合中的元素。

046 202144

2202 202148

专题六 泛型

httpswww.cnblogs.comminikobep11547220.html

httpswww.cnblogs.comEasonJimp7967096.html

6.1 集合中的泛型

6.1.1 不使用泛型带来的问题：造成繁琐的运行时类型转换异常

6.1.2 使用泛型的优点：能够在编译时而不是运行时检测出的类型转换异常

6.2 支持泛型的类

6.3 通配符（通配符使声明支持泛型的类的变量可以接受不一定唯一种的泛型，可以免去由有继承关系的不同泛型类型引起的多个支持泛型的类变量的声明，用在方法参数中可以减少不必要的重载）

6.3.1 extends （上界通配符）（见Test）

ArrayList heroList extends Hero 表示这是一个Hero泛型或者其子类泛型

6.3.2 super （下界通配符）（超类限定）

ArrayList heroList super Hero 表示这是一个Hero泛型或者其父类泛型

6.3.3 ？（非受限通配符）

等价于 extends Object

6.3.4 比较？和T的不同

T 是一个 确定的 类型，通常用于泛型类和泛型方法的定义，？是一个 不确定 的类型，通常用于泛型方法的调用代码和形参，不能用于声明类和泛型方法。

区别1：通过 T 来 确保 泛型参数的一致性

通过 T 来 确保 泛型参数的一致性

public T extends Number void test(ListT dest, ListT src)

通配符是 不确定的，所以这个方法不能保证两个 List 具有相同的元素类型

public void test(List extends Number dest, List extends Number src)

区别2：类型参数可以多重限定而通配符不行

使用 & 符号设定多重边界（Multi Bounds)（见Test）

区别3：通配符可以使用超类限定而类型参数不行

类型参数 T 只具有 一种 类型限定方式：

T extends A

但是通配符 可以进行 两种限定：

extends A

super A

6.4 泛型方法(见Test）

方法可以定义泛型类型

[publicprivateprotected] [static] 泛型(可受限) 返回类型 方法名(参数）

6.5 泛型类型转换

子类泛型对象不用转父类泛型声明，反之更不能：ArrayListHero a = new ArrayListADHero();错误的

6.6 ClassT 和Class的区别

最常见的是在反射场景下的使用（见Test）

831 2021410

专题七 Lambda

7.1 匿名类方式（见Test）

7.2 Lambda方式（见Test）：其实就是匿名方法

java会还原成匿名类方式，引入Lambda表达式，会使得代码更加紧凑，而不是各种接口和匿名类到处飞。

缺点：可读性差；不便于调试；版本支持jdk8

7.3 Lambda方法引用

7.3.1 引用静态方法（见Test）

7.3.2 引用对象方法（见Test）

7.3.3 引用容器中的对象方法， 引用Lambda的参数中的对象方法（见Test）

7.3.4 引用构造器

7.4 聚合操作 httpswww.runoob.comjavajava8-streams.html httpswww.ktanx.comblogp433

和以前的Collection操作不同， Stream操作还有两个基础的特征：

Pipelining 中间操作都会返回流对象本身。 这样多个操作可以串联成一个管道， 如同流式风格（fluent style）。 这样做可以对操作进行优化， 比如延迟执行(laziness)和短路( short-circuiting)。

内部迭代： 以前对集合遍历都是通过Iterator或者For-Each的方式, 显式的在集合外部进行迭代， 这叫做外部迭代。 Stream提供了内部迭代的方式， 通过访问者模式(Visitor)实现。

7.4.1 Stream和管道的概念

Stream 和Collection结构化的数据不一样，Stream是一系列的元素，来自数据源的元素队列。

管道指的是一系列的聚合操作，分3个部分：

7.4.2 管道源：流的来源。 可以是集合，数组，IO channel， 产生器generator 等

集合：xxx.stream() 并行流：xxx.parallelStream() 数组：Stream.of(xxx)或者Arrays.stream(xxx)

7.4.3 中间操作intermediate operation：

每个中间操作，又会返回一个Stream，比如.filter()又返回一个Stream, 中间操作是“懒”操作，并不会真正进行遍历。

主要分两类

对元素进行筛选：filter 匹配 distinct 去除重复(根据equals判断) sorted 自然排序 sorted(ComparatorT) 指定排序 limit 保留 skip 忽略

转换为其他形式的流mapToDouble 转换为double的流 map 转换为任意类型的流

7.4.4 结束操作terminal operation：

当这个操作执行后，流就被使用“光”了，无法再被操作。所以这必定是流的最后一个操作。

结束操作不会返回Stream，但是会返回int、float、String、 Collection或者像forEach，什么都不返回, 结束操作才进行真正的遍历行为，在遍历的时候，才会去进行中间操作的相关判断

结束操作才真正进行遍历行为，前面的中间操作也在这个时候，才真正的执行。

forEach() 遍历每个元素 toArray() 转换为数组 min(ComparatorT) 取最小的元素 max(ComparatorT) 取最大的元素 count() 总数 findFirst() 第一个元素

7.5 比较串行流，并行流的聚合操作和集合的查找性能（见程序Test）

总结：测试可知，在数据已经预先处理后直接读取结果是最快的，如MyHashMap；使用普通迭代比stream快一点，相差不大；而使用并行流的聚合操作是串行流的一半时间。

1059 2021411

专题八 多线程

8.1 概念和基本方式

多线程就是同一时间执行多个任务，进程是一个程序的启动运行，线程是进程内部同时进行的任务

8.1.1 继承线程类Thread

自定义多线程类继承Thread，重写run方法，使用时实例化该类，执行start方法

8.1.2 实现Runnable接口

自定义多线程类实现Runnable接口，重写run方法，使用时，创建该类对象，并由该对象实例化一个Thread，再执行start方法

8.1.3 匿名类（见Test）

使用匿名类，继承Thread,重写run方法，直接在run方法中写业务代码

匿名类的一个好处是可以很方便的访问外部的局部变量。

前提是外部的局部变量需要被声明为final。(JDK7以后就不需要了)，但这个变量不能在外面进行赋值，否则无法识别成final变量

8.1.4 使用多线程进行查找（见Test）

等待所有线程结束的方法 httpsblog.csdn.netliweisnakearticledetails12966761

8.2 常见的线程方法

sleep当前所在的线程暂停，因为当前线程sleep的时候，有可能被停止，会抛出InterruptedException

join加入到当前线程中。主线程：所有进程，至少会有一个线程即主线程，即main方法开始执行。必须先start才能join（见Test)

setPriority：线程优先级，当线程处于竞争关系的时候，优先级高的线程会有更大的几率获得CPU资源。（见Test）在资源十分充足时无法看到效果

yield当前线程，临时暂停，使得其他线程可以有更多的机会占用CPU资源，由运行态到就绪态，停止一下后再由就绪态到运行态。（暂时不知到效果）

setDaemon守护线程的概念是： 当一个进程里，所有的线程都是守护线程的时候，结束当前进程。守护线程通常会被用来做日志，性能统计等工作。(见Test）

守护线程最典型的应用就是 GC (垃圾回收器)

(1) start()之前设置，否则会IllegalThreadStateException。不能把正在运行的常规线程设置为守护线程。

(2) 在Daemon线程中产生的新线程也是Daemon的。

(3) 不要认为所有的应用都可以分配给Daemon来进行服务，比如读写操作或者计算逻辑。 因为你不可能知道在所有的User完成之前，Daemon是否已经完成了预期的服务任务。

1742 2021412

8.3 多线程 同步问题concurrency

多个线程同时修改一个数据的时候，可能导致的问题

8.3.1 同步问题演示（见Test）

8.3.2 分析原因：在一个线程真正进行访问修改某对象中的成员数据的同时，另一线程也在做类似操作，最后的结果是最后修改的值而不是两个线程的综合修改的值，这个值在业务上叫脏数据。

8.3.3 解決思路：在增加线程访问hp期间，其他线程不可以访问hp

8.3.4 解決辦法

用法一：

synchronized (someObject){

此处的代码只有占有了someObject后才可以执行

}

synchronized關鍵字的意義：表示【當前】綫程（关键字所在的线程）獨占該對象(見Test），并不是执行代码里面启动的线程。

释放同步对象的方式： synchronized 块自然结束，或者有异常抛出

用法二：

synchronized (this) {

}

表示当前对象为同步对象

在实例方法前，直接加上synchronized ，其所对应的同步对象，就是this（见Test）

用法三：

在类方法前加上synchronized， 同步对象是Class实例

httpswww.jb51.netarticle74566.htm （主要是第四大点）

难点：httpswww.cnblogs.comonlywujunarticles3530984.html

8.4 线程安全类

如果一个类，其方法都是有synchronized修饰的，那么该类就叫做线程安全的类

同一时间，只有一个线程能够进入 这种类的一个实例 的去修改数据，进而保证了这个实例中的数据的安全(不会同时被多线程修改而变成脏数据)

对线程安全类和非线程安全类进行多线程操作时，后者回快（发生大量同步时以及有延时才能看出来，几乎没区别，而且后者产生的是脏数据）（见Test）

8.4.1 HashMap和HashTable 区别见（5.3.3 ）

8.4.2 （见Test）StringBuffer和StringBuilder的区别，StringBuffer的方法都是有synchronized修饰的，StringBuffer就叫做线程安全的类，而StringBuilder就不是线程安全的类

8.4.3 Vector和ArrayList一模一样的~他们的区别也在于，Vector是线程安全的类，而ArrayList是非线程安全的。

8.4.4 借助Collections.synchronizedList，可以把ArrayList转换为线程安全的List。与此类似的，还有HashSet,LinkedList,HashMap等等非线程安全的类，都通过工具类Collections转换为线程安全的（见Test）

线程安全化后不可再对之前的对象进行操作，否则一样不安全，因为两者底层还是同一个集合

8.5 死锁演示（见Test）

8.6 线程之间的交互

线程之间有交互通知的需求。

例如：加血，减血两个线程处理同一个英雄。减血的线程发现血量=1，就停止减血，直到加血的线程为英雄加了血，才可以继续减血。

8.6.1 不好的解决方式：减血线程中使用while循环判断是否是1，是就不停循环等加血，不是就跳出循环

会大量占用CPU,拖慢性能

8.6.2 使用wait和notify进行线程交互（見Test）

并不是Thread线程上的方法，它们是Object上的方法。

调用wait是有前提条件的，一定是在synchronized块里，否则就会出错。

this.wait()表示 让占有this的线程等待，并临时释放占有

this.notify() 表示通知那些等待在this的线程，可以苏醒过来了。

notifyAll() 的意思是，通知所有的等待在这个同步对象上的线程，可以苏醒过来了，有机会重新占用当前对象了。

8.6.3 生产者消费者问题（見Test）如何實現生產少時加快生產，生產多少減慢生產？？？？

1. 使用栈来存放数据

1.1 把栈改造为支持线程安全

1.2 把栈的边界操作进行处理，当栈里的数据是0的时候，访问pull的线程就会等待。 当栈里的数据是20的时候，访问push的线程就会等待

2. 提供一个生产者（Producer）线程类，生产随机大写字符压入到堆栈

3. 提供一个消费者（Consumer）线程类，从堆栈中弹出字符并打印到控制台

4. 提供一个测试类，使两个生产者和三个消费者线程同时运行

8.6.4 終止綫程的三種方法 httpsblog.csdn.netanhuidelingerarticledetails11746365

8.6.5 Lock对象使用await和signal

8.7 綫程池

每一个线程的启动和结束都是比较消耗时间和占用资源的。如果在系统中用到了很多的线程，大量的启动和结束动作会导致系统的性能变卡，响应变慢。

为了解决这个问题，引入线程池这种设计思想。

8.7.1 綫程池的設計思路（见TEst）

线程池的思路和生产者消费者模型是很接近的。

1. 准备一个任务容器

2. 一次性启动10个 消费者线程

3. 刚开始任务容器是空的，所以线程都wait在上面。

4. 直到一个外部线程往这个任务容器中扔了一个“任务”，就会有一个消费者线程被唤醒notify

5. 这个消费者线程取出“任务”，并且执行这个任务，执行完毕后，继续等待下一次任务的到来。

6. 如果短时间内，有较多的任务加入，那么就会有多个线程被唤醒，去执行这些任务。

在整个过程中，都不需要创建新的线程，而是循环使用这些已经存在的线程

8.7.2 自定義綫程池（见Test）

8.7.3 测试自定义线程池（见Test）

创造一个情景，每个任务执行的时间都是1秒，刚开始是间隔1秒钟向线程池中添加任务，然后间隔时间越来越短，执行任务的线程还没有来得及结束，新的任务又来了。

就会观察到线程池里的其他线程被唤醒来执行这些任务

8.7.4 java本身的线程池

线程池类ThreadPoolExecutor在包java.util.concurrent下

ThreadPoolExecutor threadPool= new ThreadPoolExecutor(10, 15, 60, TimeUnit.SECONDS, new LinkedBlockingQueueRunnable());

第一个参数10 表示这个线程池初始化了10个线程在里面工作

第二个参数15 表示如果10个线程不够用了，就会自动增加到最多15个线程

第三个参数60 结合第四个参数TimeUnit.SECONDS，表示经过60秒，多出来的线程还没有接到活儿，就会回收，最后保持池子里就10个

第四个参数TimeUnit.SECONDS 如上

第五个参数 new LinkedBlockingQueue() 用来放任务的集合

execute方法用于添加新的任务

使用线程池工具类Executors

newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。

newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。

newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。

newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

8.7.5测试使用线程池来查找的性能（见Test）

8.8 Lock对象

与 synchronized (someObject) 类似的，lock()方法，表示当前线程占用lock对象，一旦占用，其他线程就不能占用了。

与 synchronized 不同的是，一旦synchronized 块结束，就会自动释放对someObject的占用。 lock却必须调用unlock方法进行手动释放，为了保证释放的执行，往往会把unlock() 放在finally中进行。

8.8.1 使用Lock对象实现同步效果（见Test）

在try中使用，finally释放（各个方法的一般形式httpswww.cnblogs.commyseriesp10784076.html）

获取锁 ，获取成功则占有锁，否则等待

void lock()

（见Test）

如果当前线程未被中断，则获取锁，可以响应中断

void lockInterruptibly()

lockInterruptibly()方法比较特殊，当通过这个方法去获取锁时，如果线程 正在等待获取锁，则这个线程能够 响应中断，即中断线程的等待状态。

例如，当两个线程同时通过lock.lockInterruptibly()想获取某个锁时，假若此时线程A获取到了锁，而线程B只有在等待，那么对线程B调用threadB.interrupt()方法能够中断线程B的等待过程。

lockInterruptibly()一般的使用形式如下：

public void method() throws InterruptedException {

lock.lockInterruptibly();

try {

.....

}

finally {

lock.unlock();

}

}

注意：当一个线程获取了锁之后，是不会被interrupt()方法中断的。

因为interrupt()方法只能中断阻塞过程中的线程而不能中断正在运行过程中的线程。

因此，当通过lockInterruptibly()方法获取某个锁时，如果不能获取到，那么只有进行等待的情况下，才可以响应中断的。

与 synchronized 相比，当一个线程处于等待某个锁的状态，是无法被中断的，只有一直等待下去。

返回绑定到此 Lock 实例的新 Condition 实例 ，线程间通信与协作

Condition newCondition()

仅在调用时锁为空闲状态才获取该锁，获取成功返回true并占用（需要解锁），否则返回false不会等待，可以响应中断

boolean tryLock()

如果锁在给定的等待时间内空闲，并且当前线程未被中断，则获取锁

boolean tryLock(long time, TimeUnit unit)

释放锁 最好时finally

void unlock()

8.8.2 Lock的线程交互（见Test）

使用synchronized方式进行线程交互，用到的是同步对象的wait,notify和notifyAll方法

Lock也提供了类似的解决办法，首先通过lock对象得到一个Condition对象，然后分别调用这个Condition对象的：await, signal,signalAll 方法

8.8.3 Lock和synchronized的对比

1. Lock是一个接口，而synchronized是Java中的关键字，synchronized是内置的语言实现，Lock是代码层面的实现。

2. Lock可以选择性的获取锁，如果一段时间获取不到，可以放弃。synchronized不行，会一根筋一直获取下去。 借助Lock的这个特性，就能够规避死锁，synchronized必须通过谨慎和良好的设计，才能减少死锁的发生。

3. synchronized在发生异常和同步块结束的时候，会自动释放锁。而Lock必须手动释放， 所以如果忘记了释放锁，一样会造成死锁。

4.在资源竞争不是很激烈的情况下，Synchronized的性能要优于ReetrantLock，但是在资源竞争很激烈的情况下，Synchronized的性能会下降几十倍，但是ReetrantLock的性能能维持常态。

8.8.4 借助tryLock的有限等待时间，解决死锁问题

8.8.5 Lock的实现类ReentrantLock

可重入锁。ReentrantLock是唯一实现了Lock接口的类，并且ReentrantLock提供了更多的方法。

构造方法（不带参数 和带参数 true： 公平锁； false 非公平锁）

8.8.6 ReadWriteLock

该接口只有两个方法：Lock readLock()用于读取操作的锁 Lock writeLock()返回用于写入操作的锁

维护了一对相关的锁，一个用于只读操作，另一个用于写入操作。只要没有 writer，读取锁可以由多个 reader 线程同时保持，而写入锁是独占的。

实现读时可并发，写时互斥 等操作

8.8.7 锁的基本概念

可重入锁：已申请锁的线程不需要重新申请锁，synchronized和ReentrantLock都是可重入锁

可中断锁：可以响应中断的锁，synchronized是不可中断的锁，ReentrantLock是可中断锁

公平锁： 尽量 以请求锁的顺序来获取锁，同是有多个线程在等待一个锁，当这个锁被释放时，等待时间最久的线程（最先请求的线程）会获得该锁。synchronized是非公平锁，ReentrantLock和ReentrantReadWriteLock默认是公平锁，可设置

8.9 原子访问

原子性操作概念：所谓的原子性操作即不可中断的操作，比如赋值操作。本身是线程安全的

i++是由3个原子性操作组成，不是线程安全的

原子类，jdk6，包java.util.concurrent.atomic，对应基本类型，自增incrementAndGet自减等方法都是原子性的

出现线程安全性问题的条件1）多线程环境2）有共享资源3）对共享资源的非原子性操作

8.9 线程状态总结

8.9.1 中断（各种情况）

Unless the current thread is interrupting itself, which is always permitted, the checkAccess method of this thread is invoked, which may cause a SecurityException to be thrown.

If this thread is blocked in an invocation of the wait(), wait(long), or wait(long, int) methods of the Object class, or of the join(), join(long), join(long, int), sleep(long), or sleep(long, int), methods of this class, then its interrupt status will be cleared and it will receive an InterruptedException.

If this thread is blocked in an IO operation upon an InterruptibleChannel then the channel will be closed, the thread's interrupt status will be set, and the thread will receive a ClosedByInterruptException.

If this thread is blocked in a Selector then the thread's interrupt status will be set and it will return immediately from the selection operation, possibly with a non-zero value, just as if the selector's wakeup method were invoked.

If none of the previous conditions hold then this thread's interrupt status will be set.

Interrupting a thread that is not alive need not have any effect.

8.9.2 等待操作比较

httpsgourderwa.blog.csdn.netarticledetails103701793

8.9.3 状态

httpsblog.csdn.netqq\_22771739articledetails82529874

httpswww.runoob.comjavathread-status.html（后面有测试程序）

2322 2021421

专题九 JDBC MySQL数据库连接 httpsblog.csdn.netqq\_22172133articledetails81266048

9.1 JDBC (Java DataBase Connection) 是通过JAVA访问数据库

9.1.1 httpsblog.csdn.netqq\_22172133articledetails81266048

加载数据库驱动（mysql8不需要手动加载）：通过Class.forName();

建立数据库连接 DriverManager.getConnection(url,user,pass);

创建Statement

执行Statement语句

关闭连接：数据库的连接是有限资源，相关操作结束后，养成关闭数据库的好习惯，先关闭Statement，后关闭Connection

可以使用try-with-resource的方式自动关闭连接，因为Connection和Statement都实现了AutoCloseable接口

打开时：Connection - PreparedStatement - ResultSet

关闭时：ResultSet- PreparedStatement - Connection

9.2 增删查改CRUD

增加 create：create database xxx; create table xxx ( column\_name column\_type, ... ); insert into xxx values();

删除 delete drop database xxx; drop table xxx; delete from xxx where xx = xx;

查询 retrieve：select xxx from xxx; show databases;

更新 update：update xxx set xxx = xxx where xxx = xxx; alter

9.2.1 Statement常用的方法

executeQuery（用于向数据库发送查询语句）

executeUpdate（用于向数据库发送增删改语句）

execute（用于发送任意语句）

addBatch(把多条sql语句放到一个批处理中）

executeBatch（向数据库发送一批sql语句执行）

9.3 执行查询语句

9.3.1 获取结果 httpsblog.csdn.netqq\_22172133articledetails81266048

Jdbc程序中的ResultSet用于代表Sql语句的执行结果。

Resultset封装执行结果时，采用的类似于表格的方式，

ResultSet 对象维护了一个指向表格数据行的游标，初始的时候，游标在第一行之前,next最后会在最后一行之后，即这两个地方为null，

调用ResultSet.next() 方法，可以使游标指向具体的数据行，进行调用方法获取该行的数据。

获取行：

ResultSet提供了对结果集进行滚动的方法：

next()：移动到下一行（见Test)

Previous()：移动到前一行

absolute(int row)：移动到指定行

beforeFirst()：移动resultSet的最前面。

afterLast() ：移动到resultSet的最后面。

获取值：

获取数据的get方法：

获取任意类型的数据：getObject(int index) getObject(string columnName)

获取指定类型的数据，例如：getString(int index) getString(String columnName)

getInt getLong ...

在取第二列的数据的时候，用的是get(2) ，而不是get(1). 这个是整个Java自带的api里唯二的地方，使用基1的，即2就代表第二个。另一个地方是在PreparedStatement这里

9.3.2 SQL判断账号密码是否正确（见Test）

正确方式是根据账号和密码到表中去找数据，如果有数据，就表明密码正确了，如果没数据，就表明密码错误。

不恰当的方式 是把uers表的数据全部查到内存中，挨个进行比较。

9.3.3 获取总数（见Test）

select count() from hero

9.3.4 设计分页查询（见Test）

select from hero limit 0,5

从0开始查5个

9.4 预编译PreparedStatement(见Test）

和 Statement一样，PreparedStatement也是用来执行sql语句的

与创建Statement不同的是，需要根据sql语句创建PreparedStatement

除此之外，还能够通过设置参数，指定相应的值，而不是Statement那样使用字符串拼接

注： 这是JAVA里唯二的基1的地方，另一个是查询语句中的ResultSet也是基1的。

9.4.1 优点1：参数设置

Statement 需要进行字符串拼接，可读性和维护性比较差

String sql = insert into hero values(null,+'提莫'+,+313.0f+,+50+);

PreparedStatement 使用参数设置，可读性好，不易犯错

String sql = insert into hero values(null,,,);

9.4.2 优点2：性能表现（见Test)

PreparedStatement有预编译机制，性能比Statement更快

PreparedStatement 执行多次，只需要1次把SQL语句传输到数据库端

数据库对带的SQL进行预编译

每次执行，只需要传输参数到数据库端

1. 网络传输量比Statement更小

2. 数据库不需要再进行编译，响应更快

9.4.3 优点3：防止SQL注入式攻击（见Test）

例如OR 1=1在数据后面一起提交

9.5 execute 和executeUpdate

相同点：都可以执行增加，删除，修改

不同点： execute可以执行查询语句然后通过getResultSet，把结果集取出来

executeUpdate不能执行查询语句

execute返回boolean类型，true表示执行的是查询语句，false表示执行的是insert,delete,update等等

executeUpdate返回的是int，表示有多少条数据受到了影响

9.6 特殊操作

9.6.1自增的设置和获取，删除元素要修改自增复位（见Test）

httpsblog.csdn.netpupilxiaomingarticledetails54694765

httpsblog.csdn.netqq\_40378034articledetails90736544

无论是execute还是executeUpdate都不会返回这个自增长id是多少。需要通过Statement的getGeneratedKeys获取该id

PreparedStatement需要Statement.RETURN\_GENERATED\_KEYS参数，以确保会返回自增长ID

9.6.2 获取表的元数据（见Test）

元数据概念：

和数据库服务器相关的数据，比如数据库版本，有哪些表，表有哪些字段，字段类型是什么等等。

9.7 事务

在Mysql中，只有当表的类型是INNODB的时候，才支持事务，所以需要把表的类型设置为INNODB,否则无法观察到事务.

修改表的类型为INNODB的SQL：alter table hero ENGINE = innodb;

查看表的类型的SQLshow table status from how2java;

前提：当前的MYSQL服务器本身要支持INNODB,如果不支持，请开启MYSQL INNODB

不使用事务的情况：几个操作需要捆绑一起执行时，若某个操作出现异常，则其他操作执行成功。

在事务中的多个操作，要么都成功，要么都失败

通过 c.setAutoCommit(false);关闭自动提交

使用 c.commit();进行手动提交

设置回滚点

9.7.1 batch 批量执行PreparedStatement中只有占位符参数不同的SQL

9.8 ORM ORM=Object Relationship Database Mapping

对象和关系数据库的映射(见Test）

9.9 DAO DAO=DataAccess Object

数据访问对象,数据库相关的操作都封装在这个类（见Test）

9.10 数据库连接池

9.10.1 传统方式连接

有多个线程都需要连接数据库执行SQL语句，那每个线程都会创建一个连接，在使用完毕后，关闭连接。

创建和关闭连接的过程是消耗时间的。过多线程并发，会造成系统卡顿。

一个数据库同时支持的连接总数是有限的。多线程并发量很大，那么数据库连接的总数就会被消耗光，后续线程发起的数据库连接就会失败。

9.10.2 使用数据库连接池

连接池在使用之前，就会 创建好 一定数量 的连接。

如果有任何线程需要使用连接，就从连接池里面 借用 ，而不是自己重新创建.

使用完毕后，又把这个连接 归还 给连接池 供下一次或者其他线程使用。

发生多线程并发情况，连接池里的连接被 借用光 了，那么其他线程就会 临时等待 ，直到有连接被归还回来，再继续使用。

整个过程，这些连接 都不会被关闭 ，而是 不断的被循环使用 ，从而节约了启动和关闭连接的时间。

9.10.3 自定义ConnectionPool（见Test）

a、构造方法约定了一个有多少个连接

b、自动关闭所有连接（实现AutoCloseable)或者手动关闭的方法

c、getConnection， 判断是否为空，如果是空的就wait等待，否则就借用一条连接出去

d、returnConnection， 在使用完毕后，归还这个连接到连接池，并且在归还完毕后，调用notifyAll，通知那些等待的线程，有新的连接可以借用了。

9.10.4 常用数据库连接池API

HikariCP ：（见Test）

httpsblog.csdn.nethuhumissarticledetails100038563utm\_medium=distribute.pc\_aggpage\_search\_result.none-task-blog-2~aggregatepage~first\_rank\_v2~rank\_aggregation-1-100038563.pc\_agg\_rank\_aggregation&utm\_term=hikaricp%E4%BD%BF%E7%94%A8%E6%95%99%E7%A8%8B&spm=1000.2123.3001.4430

httpsblog.csdn.netsinat\_34093604articledetails79713296 DataSource也是一个非常昂贵的操作，所以通常DataSource实例总是作为一个全局变量存储

C3P0

BoneCP

Druid

JdbcUtils：DBCP 数据库连接池 C3P0 数据库连接池 httpsblog.csdn.netqq\_22172133articledetails81266048

9.10.5 使用连接池和传统方式的性能比较（见Test）。

1. 使用传统方式创建100个线程，每个线程都会创建新的连接，通过这个连接向数据库插入1条数据，然后关闭这个连接。

2. 使用数据库连接池HikariCP的方式，创建一个有10条连接的连接池，然后创建100个线程，每个线程都会向连接池借用连接，借用到后，向数据库插入1条数据，然后归还这个连接。

通过时间统计，比较这两种方式的性能表现差异。

2145 2021429

专题十 Spring

Spring是一个基于IOC和AOP的结构J2EE系统的框架

IOC 反转控制 是Spring的基础，Inversion Of Control

简单说就是创建对象由以前的程序员自己new 构造方法来调用，变成了交由Spring创建对象

DI 依赖注入 Dependency Inject. 简单地说就是拿到的对象的属性，已经被注入好相关值了，直接使用即可。

10.1 演示Spring如何获取对象（见MySpring/Test类）

导入额外的jar包，在lib中

对象的属性有seter方法，xml中配置bean

10.2 Spring注入对象（见MySpring/Test类）

xml中bean中property配置ref为另一个bean的名

10.3 使用注解方式完成注入对象（对注入对象行为的注解）（见MySpring/Test类）

在bean前添加contextannotation-config 告诉Spring要用注解进行配置

声明该成员对象前加上@Autowired注解

或是在该成员对象的setter前加上@Autowired

也可用@Resource(name=c) c为xml中的一个对应的bean

10.3.1 对bean进行注解

通过注解进行bean本身对象移出XML配置文件

xml不用bean，只要内容contextcomponent-scan base-package=pojo其作用是告诉Spring，bean都放在pojo这个包下

在pojo包中的类前加上@Component或@Component(xxx)，表明此类是bean，属性初始化放在属性声明上进行

10.4 AOP Aspect Oriented Program 面向切面编程

将功能分为：

核心业务，比如登陆，增加数据，删除数据都叫核心业务

周边功能，比如性能统计，日志，事务管理等等

周边功能在Spring的面向切面编程AOP思想里，即被定义为切面

在面向切面编程AOP的思想里面，核心业务功能和切面功能分别独立进行开发，然后把切面功能和核心业务功能 编织 在一起，这就叫AOP

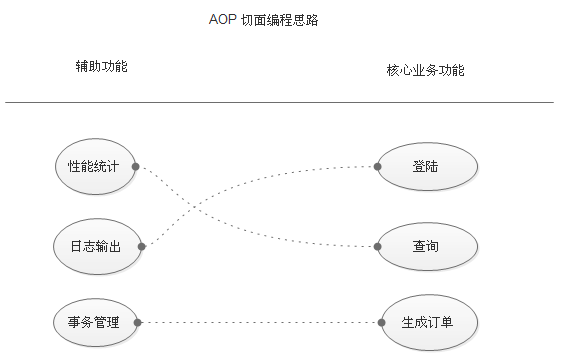
需要额外的jar包在lib中

10.4.1思路图：

a. 辅助功能和核心业务功能彼此独立进行开发，比如登陆功能，即便是没有性能统计和日志输出，也可以正常运行

b. 如果有需要，就把"日志输出" 功能和 "登陆" 功能 编织在一起，这样登陆的时候，就可以看到日志输出了

c. 辅助功能，又叫做切面，这种能够选择性的，低耦合的把切面和核心业务功能结合在一起的编程思想，就叫做切面编程



10.4.2 演示AOP（见MySpring/Test类）

a．准备业务类ProductService，并先调用一次，使用IOC

b. 准备日志切面LoggerAspect

该日志切面的功能是 在调用核心功能之前和之后分别打印日志，切面就是原理图中讲的那些辅助功能。

打印日志：System.out.println("start log:" + joinPoint.getSignature().getName());

Object object = joinPoint.proceed();//就是将来与某个核心功能编织之后，用于执行核心功能的代码

c. xml配置

声明业务对象<bean name="productService" class="pojo.ProductService">

声明日志切面<bean id="loggerAspect" class="pojo.LoggerAspect">

指定核心业务功能 <aop:pointcut id=”loggerCutPoint”expression=”execution(\*pojo.ProductService.\*(..))”/>

execution(\*pojo.ProductService.\*(..))表示对满足任意返回类型（\*）的该类的任意方法（pojo.ProductService.\*）的调用，参数是任意数量的类型（(..)）

指定辅助功能<aop:aspect id=”logAspect” ref=”loggerAspect”><aop:around pointcut-ref=”loggerCutPoint” method=”log”/></aop:aspect>

然后通过aop:config把业务对象和辅助功能编织在一起

10.5 注解方式AOP（见MySpring/Test类）

a.注解配置业务类使用 @Component("s") 注解ProductServices 类

b.注解配置切面@Aspect 注解表示这是一个切面  
@Component 表示这是一个bean,由Spring进行管理  
在切面方法前@Around(value = "execution(\*service.ProductServices.\*(..))") 表示对service.ProductServices 这个类中的所有方法进行切面操作

c.xml配置

去掉原有信息，添加

<context:component-scan package=”service” />

<context:component-scan package=”aspect”/>

扫描这两个包定位bean, 定位业务类和切面类

<aop:aspect-autoproxy/>

找到被注解的切面类，进行切面配置

10.6 注解方式测试（见MySpring/SpringTest类）

需要用到jUnit，需要导入包junit-4.12.jar和hamcrest-all-1.3.jar，在jUnit文件中有

类前添加：

a、@RunWith(SpringJUnit4ClassRunner.class)

表示这是一个Spring的测试类

b、@ContextConfiguration(“classpath:applicationContextAOP2.xml”)

定位Spring的配置文件

c、@Autowired

给这个测试类装配一个bean对象

d、@Test

这是一个进行测试的方法

主题十一 Tomcat

10.1安装使用Tomcat

不使用Tomcat是通过打开本地文件的形式打开的

使用Tomcat是通过http的连接打开，Tomcat是web服务器

10.1.1启动tomcat

点击运行批处理文件：bin\startup.bat，tomcat7最高匹配jdk8，或命令行命令startup

也可用配置到eclipse中启动

10.1.2部署网页

部署一个功能完备的web应用有多种方式

部署一个html只要复制到webapps\ROOT下

10.1.3端口号

8080是tomcat默认使用的端口号

tomcat的端口配置相关信息在conf\server.xml中

<Connector port=”8080” protocol=”HTTP/1.1”connectionTimeout=”20000” redirectPort=”8543” />

80端口是web服务默认的端口号，用这个端口可用不用显式写出链接中的端口。

* 1. tomcat问题排查

针对通过startup.bat独立方式启动tomcat出现的常见问题的排查（IDE中启动tomcat不包括）

* + 1. JAVA\_HOME

现象：点击startup.bat之后，屏幕一闪而过

检验：通过cmd命令进入控制台，然后切换到对应的目录执行startup命令，得到JRE\_HOME environment ....提示

分析：Tomcat本身是JAVA程序，必须要有JDK才可以执行

解决：配置JDK环境变量

10.2.2 CATALINA\_HOME未设置

现象：点击startup.bat之后，屏幕一闪而过

检验：命令行执行startup，出现CATALINA\_HOME environment … 提示

分析：tomcat执行必须依赖CATALINA\_HOME和CATALINA\_BASE这两个环境变量，如果没配置，会自动采取bin目录的父目录作为这两个环境变量的值，如果配置了不正确，就会出现错误

解决：在环境变量中删除CATALINA\_HOME和CATALINA\_BASE，不仅要检测系统变量，还要用户变量。或者配置正确的这两个变量。

* + 1. CATALINA\_HOME错误设置

现象：Tomcat可用成功启动，就是不能访问自己配置的web应用，提示404

检验：在环境变量里发现CATALINA\_HOME设置在另一个合法的tomcat目录上了，所以无论运行哪个startup.bat，都会导致这个目录的程序被启动。

分析：默认的tomcat会优先根据环境变量中CATALINA\_HOME来定位目录来运行

解决：在环境变量中删除CATALINA\_HOME和CATALINA\_BASE的配置，不仅要删除系统变量，还要用户变量。或者，使用纯净的Tomcat，已经取消了对环境变量CATALINA\_HOME和CATALINA\_BASE的依赖。

* + 1. 端口冲突

现象：屏幕一闪而过，或者不会关闭，但有大量错误，会看到Address already in use:JVM\_BIND <null>:80，即端口被占用

检验：在Tomcat目录logs目录里，会有日志文件：catalina:yyyy-mm-dd.log，会记载上述错误

分析：端口是独占式的，可能被其他程序或其他tomcat占用了

解决：修改端口或关闭其他程序

* + 1. localhost:yyyy-mm-dd.log

现象：404

检验：在logs目录的日志文件中有大量错误

分析：当server.xml和web.xml配置错误时，当前web应用会部署失败，并将错误信息输出到localhost:yyyy-mm-dd.log文件中

解决：看是什么错误引起的，可能是配置的context目录不存在等等

* + 1. Error FilterStart

现象：404错误，明明有文件，但就是不能访问

检验：也算配置失败，但不会大量报错，只显示：xxxxx startup failed due to previous errors

分析：过滤器启动失败就会报这个错误

解决：看错误信息输出文件localhost:yyyy-mm-dd.log

* + 1. JDK版本

现象：404错误，明明有文件，但是不能访问

检验：查看java版本

分析：当前部署的wen应用是由高版本JDK编译生成的，而当前tomcat运行使用的JDK是低版本的，就会报错：UnsupportClassVersionError

解决：使用对应版本的jdk

* 1. 404问题排查

404表示File Not Found，文件不存在

10.3.1 正确端口号

注意默认端口号和本tomcat配置的端口号

10.3.2 环境变量

若有多个Tomcat服务器，发现启动其中一个startup.bat时，启动的是另外的服务器。那就是环境变量的问题。

修改或删除环境变量中的CATALINA\_HOME和CATALINA\_BASE的值，系统变量和用户变量都要。

或让tomcat对其无依赖。

* + 1. 确定Web应用是否成功

tomcat可以启动，当web应用无法运行，404。查logs目录下留下的错误信息

* + 1. webapps下的ROOT目录

在server.xml中配置的<context是以“/”为开始路径，同时webapps目录下还存在ROOT目录，这个是告诉tomcat以“/”为开始路径。tomcat通常只选择ROOT目录，那server.xml中配置的<context就无法启动，导致404。

解决办法：删除ROOT目录，或重命名

* + 1. 访问地址的时候是否加了应用的名称

加入部署时，指定了路径，比如：

<Context path=”/j2ee” docBase=”e:\\project\\j2ee\\web” debug=”0”reloadable=”false” />

访问时需要加上j2ee：<http://127.0.0.1:8080/j2ee/hello>

* + 1. 是否访问对资源

检测各个文件的命名和访问的名字是否对应正确

* + 1. 不能访问WEB-INF目录下的资源

从安全角度出发，tomcat不允许访问WEB-INF目录下的HTML，JSP文件。如果访问会404。

* + 1. 访问地址后多加了斜杠

如：http://127.0.0.1:8080/j2ee/login.html/

* 1. 端口被占用

查看端口被哪些程序占用找到进程ID ：netstat –ano|findstr “80”

根据pid（进程id）查询对应的应用程序tasklist|findstr “xxxx”

根据名称结束该线程：taskkill /f /t /im java.exe

或通过pid终止：taskkill /f /pid xxxx

* 1. 持久化异常

现象：启动报错，Exception loading sessions from persistent storage异常

分析：tomcat的session持久化机制引起的。这个功能意义在重启tomcat后保存之前的session，tomcat会把session持久化在本tomcat目录下work/Catalina/localhost/session.ser这个文件中。如果tomcat非正常关闭，则这个文件无法正确的结束（无EOF）

解决办法：删除session.ser文件

长久解决办法：关闭tomcat持久化功能。修改conf目录下的server.xml文件。在项目的context间加入一句代码

<Manager className=”org.apache.catalina.session.PersistenManager” saveOnRestart=”false”/>

重启tomcat就不会报错

10.6 tomcat独立部署

10.6.1 启动：通过bin文件的startup.dat启动，shutdown.dat关闭，过程中不要修改文件可能导致无法关闭

10.6.2 关于浏览器访问web应用的路径配置

在tomcat版本5.5前，在conf/server.xml中配置，Host元素中appBase属性是加载web应用的默认路径文件夹（相对路径的父路径是本tomcat目录）；Host元素中context元素代表一个运行在服务器中的web应用：path属性是要建立的虚拟目录指定访问web应用的URL路口，可以为空，但每个context必须不同；docBase是web应用实际的路径（相对路径的父路径是Host元素中的appBase的值，默认是ROOT应用；reloadable为true时会tomcat监视WEB\_INF/classes和WEB-INF/lib目录class文件的改变，会重新加载web应用。

在tomcat版本5.5之后，说是推荐使用conf/context.xml来配置，因为server.xml是不可动态重新加载的，context.xml是可以的，但是我写入相同配置连屁都没有，但debug又可以识别，说明是加载了的，但docBase的路径就是无效，只认得server.xml中Host的appBase。

路径书写：单斜杠/，单反斜杠\，双反斜杠\\都可以