# Java-note2

## 前端

BOM是**Browser Object Model** BOM 是为了操作浏览器出现的 API，window 是其的一个对象。

DOM是**Document Object Model** DOM 是为了操作文档出现的 API，document 是其的一个对象；

DOM把所有的html都转换为节点

## Java基础 version2

### 线程

**线程是cpu调度的最小单位**

**线程**start方法重复调用的话，会出现java.lang.IllegalThreadStateException异常

实现Runnable接口比继承Thread类所具有的优势：

1）：适合多个相同的程序代码的线程去处理同一个资源

2）：可以避免java中的单继承的限制

3）：增加程序的健壮性，代码可以被多个线程共享，代码和数据独立

4）：线程池只能放入实现Runable或callable类线程，不能直接放入继承Thread的类

synchronized关键字的作用域有二种：

1）是某个对象实例内，synchronized aMethod(){}可以防止多个线程同时访问这个对象的synchronized方法（如果一个对象有多个synchronized方法，只要一个线程访问了其中的一个synchronized方法，其它线程不能同时访问这个对象中任何一个synchronized方法）。这时，不同的对象实例的synchronized方法是不相干扰的。也就是说，其它线程照样可以同时访问相同类的另一个对象实例中的synchronized方法；

2）是某个类的范围，synchronized static aStaticMethod{}防止多个线程同时访问这个类中的synchronized static 方法。它可以对类的所有对象实例起作用。

2、除了方法前用synchronized关键字，synchronized关键字还可以用于方法中的某个区块中，表示只对这个区块的资源实行互斥访问。用法是: synchronized(this){/\*区块\*/}，它的作用域是当前对象；

3、synchronized关键字是不能继承的，也就是说，基类的方法synchronized f(){} 在继承类中并不自动是synchronized f(){}，而是变成了f(){}。继承类需要你显式的指定它的某个方法为synchronized方法；

### IO

InputStream并不是一个接口，而是一个抽象类，它是所有输入流的超类

File f = **new** File("C:\\Windows\\notepad.exe");

注意Windows平台使用\作为路径分隔符，在Java字符串中需要用\\表示一个\。Linux平台使用/作为路径分隔符：

用try ... finally来编写上述代码会感觉比较复杂，更好的写法是利用Java 7引入的新的try(resource)的语法，只需要编写try语句，让编译器自动为我们关闭资源。推荐的写法如下：

public void readFile() throws IOException {

try (InputStream input = new FileInputStream("src/readme.txt")) {

int n;

while ((n = input.read()) != -1) {

System.out.println(n);

}

} // 编译器在此自动为我们写入finally并调用close()

}

实际上，编译器并不会特别地为InputStream加上自动关闭。编译器只看try(resource = ...)中的对象是否实现了java.lang.AutoCloseable接口，如果实现了，就自动加上finally语句并调用close()方法。InputStream和OutputStream都实现了这个接口，因此，都可以用在try(resource)中。InputStream一样，考虑到在发生异常的情况下如何正确地关闭资源。写入过程也会经常发生IO错误，例如，磁盘已满，无权限写入等等。我们需要用try(resource)来保证OutputStream在无论是否发生IO错误的时候都能够正确地关闭：

OutputStream也提供了close()方法关闭输出流，以便释放系统资源。要特别注意：OutputStream还提供了一个flush()方法，它的目的是将缓冲区的内容真正输出到目的地。

为什么要有flush()？因为向磁盘、网络写入数据的时候，出于效率的考虑，操作系统并不是输出一个字节就立刻写入到文件或者发送到网络，而是把输出的字节先放到内存的一个缓冲区里（本质上就是一个byte[]数组），等到缓冲区写满了，再一次性写入文件或者网络。对于很多IO设备来说，一次写一个字节和一次写1000个字节，花费的时间几乎是完全一样的，所以OutputStream有个flush()方法，能强制把缓冲区内容输出。

通常情况下，我们不需要调用这个flush()方法，因为缓冲区写满了OutputStream会自动调用它，并且，在调用close()方法关闭OutputStream之前，也会自动调用flush()方法。

但是，在某些情况下，我们必须手动调用flush()方法。举个栗子：

小明正在开发一款在线聊天软件，当用户输入一句话后，就通过OutputStream的write()方法写入网络流。小明测试的时候发现，发送方输入后，接收方根本收不到任何信息，怎么肥四？

原因就在于写入网络流是先写入内存缓冲区，等缓冲区满了才会一次性发送到网络。如果缓冲区大小是4K，则发送方要敲几千个字符后，操作系统才会把缓冲区的内容发送出去，这个时候，接收方会一次性收到大量消息。

解决办法就是每输入一句话后，立刻调用flush()，不管当前缓冲区是否已满，强迫操作系统把缓冲区的内容立刻发送出去。

实际上，InputStream也有缓冲区。例如，从FileInputStream读取一个字节时，操作系统往往会一次性读取若干字节到缓冲区，并维护一个指针指向未读的缓冲区。然后，每次我们调用int read()读取下一个字节时，可以直接返回缓冲区的下一个字节，避免每次读一个字节都导致IO操作。当缓冲区全部读完后继续调用read()，则会触发操作系统的下一次读取并再次填满缓冲区。

try (Reader reader = new InputStreamReader(new FileInputStream("src/readme.txt"), "UTF-8")) {

// TODO:

}

上述代码实际上就是FileReader的一种实现方式。

使用try (resource)结构时，当我们关闭Reader时，它会在内部自动调用InputStream的close()方法，所以，只需要关闭最外层的Reader对象即可。

PrintStream和OutputStream相比，除了添加了一组print()/println()方法，可以打印各种数据类型，比较方便外，它还有一个额外的优点，就是不会抛出IOException，这样我们在编写代码的时候，就不必捕获IOException

PrintStream最终输出的总是byte数据，而PrintWriter则是扩展了Writer接口，它的print()/println()方法最终输出的是char数据。两者的使用方法几乎是一模一样的：

如果不调用List的contains()、indexOf()这些方法，那么放入的元素就不需要实现equals()方法。前提是泛型是引用类型就需要重写equals方法

public boolean equals(Object o) {

if (o instanceof Person) {

Person p = (Person) o;

return Objects.equals(this.name, p.name) && this.age == p.age;

}

return false;

}

因此，我们总结一下equals()方法的正确编写方法：

1. 先确定实例“相等”的逻辑，即哪些字段相等，就认为实例相等；
2. 用instanceof判断传入的待比较的Object是不是当前类型，如果是，继续比较，否则，返回false；
3. 对引用类型用Objects.equals()比较，对基本类型直接用==比较。

使用Objects.equals()比较两个引用类型是否相等的目的是省去了判断null的麻烦。两个引用类型都是null时它们也是相等的。

### 集合

Java集合使用统一的Iterator遍历 是最高效的

通常直接使用for each遍历 因为底层就是Iterator；

使用enum定义的枚举类是一种引用类型。前面我们讲到，引用类型比较，要使用equals()方法，如果使用==比较，它比较的是两个引用类型的变量是否是同一个对象。因此，引用类型比较，要始终使用equals()方法，但enum类型可以例外。

这是因为enum类型的每个常量在JVM中只有一个唯一实例，所以可以直接用==比较：

### 其他

Java编译器可以帮助我们自动在int和Integer之间转型：

Integer n = 100; *// 编译器自动使用Integer.valueOf(int)*

**int** x = n; *// 编译器自动使用Integer.intValue()*

Java的接口是interface的定义 编程接口泛指接口规范 如网络协议等。

静态方法属于class 不属于实例 固不能用this

StringBuffer是StringBuilder的线程安全版本，现在很少使用。

Java 9引入的模块目的是为了管理依赖

一个类允许访问同一个package的没有public、private修饰的class，以及没有public、protected、private修饰的字段和方法

一个.java文件只能包含一个public类，但可以包含多个非public类

var关键字

浮点数常常无法精确表示

double和float类型在运算中很容易丢失精度,造成数据的不准确性,Java提供我们BigDecimal类可以实现浮点数据的高精度运算

浮点数0.1在计算机中就无法精确表示，因为十进制的0.1换算成二进制是一个无限循环小数，很显然，无论使用float还是double，都只能存储一个0.1的近似值。但是，0.5这个浮点数又可以精确地表示。

由于浮点数存在运算误差，所以比较两个浮点数是否相等常常会出现错误的结果。正确的比较方法是判断两个浮点数之差的绝对值是否小于一个很小的数：

*// 比较x和y是否相等，先计算其差的绝对值:*

**double** r = Math.abs(x - y);

*// 再判断绝对值是否足够小:*

**if** (r < 0.00001) {

*// 可以认为相等*

} **else** {

*// 不相等*

}

强制转型的结果很可能是错的

短路运算，**&&当第一个条件不成之后，后面的条件都不执行了，而&则还是继续执行，直到整个条件语句执行完为止**

需要特别注意，在一个复杂的四则运算中，两个整数的运算不会出现自动提升的情况。例如：

**double** d = 1.2 + 24 / 5; *// 5.2*

计算结果为5.2，原因是编译器计算24 / 5这个子表达式时，按两个整数进行运算，结果仍为整数4。

int opt;

switch (fruit) {

case "apple":

opt = 1;

break;

case "pear":

case "mango":

opt = 2;

break;

default:

opt = 0;

break;

}//这里没有；

使用新的switch语法，不但不需要break，还可以直接返回值。把上面的代码改写如下：

窗体顶端

public class Main {

public static void main(String[] args) {

String fruit = "apple";

int opt = switch (fruit) {

case "apple" -> 1;

case "pear", "mango" -> 2;

default -> 0;

};

System.out.println("opt = " + opt);

}

}

可变参数用类型...定义，可变参数相当于数组类型：

**class Group {**

**private** String[] names;

**public** **void** setNames(String... names) {

**this**.names = names;

}

}

上面的setNames()就定义了一个可变参数。调用时，可以这么写：

Group g = **new** Group();

g.setNames("Xiao Ming", "Xiao Hong", "Xiao Jun"); *// 传入3个String*

g.setNames("Xiao Ming", "Xiao Hong"); *// 传入2个String*

g.setNames("Xiao Ming"); *// 传入1个String*

g.setNames(); *// 传入0个String*

private 和protectd不能修饰一般类

 因为外部类和外部类之间,访问权限就只有 包内包外 所以也自然应该就只有 public 和 default 可以修饰,其他的两位都是多鱼

InstanceOf（）判断一个变量所指的实例是否是同个类型或者是其子类。

在 java 8 之前，接口与其实现类之间的 **耦合度** 太高了（**tightly coupled**），当需要为一个接口添加方法时，所有的实现类都必须随之修改。默认方法解决了这个问题，它可以为接口添加新的方法，而不会破坏已有的接口的实现

# Java笔记

## java基础易忘点 version1

### 多线程

Java多线程编程的特点又在于：

* 多线程模型是Java程序最基本的并发模型；
* 后续读写网络、数据库、Web开发等都依赖Java多线程模型。

多线程程序是乱序执行

main方法其实也是一个线程。在java中所以的线程都是同时启动的，至于什么时候，哪个先执行，完全看谁先得到CPU的资源。

在java中，每次程序运行至少启动2个线程。一个是main线程，一个是垃圾收集线程。因为每当使用java命令执行一个类的时候，实际上都会启动一个ＪＶＭ，每一个ｊＶＭ实习在就是在操作系统中启动了一个进程。

即join()的作用是：“等待该线程终止”，这里需要理解的就是该线程是指的主线程等待子线程的终止。也就是在子线程调用了join()方法后面的代码，只有等到子线程结束了才能执行。

**interrupt():不要以为它是中断某个线程！它只是线线程发送一个中断信号，让线程在无限等待时（如死锁时）能抛出抛出，从而结束线程，但是如果你吃掉了这个异常，那么这个线程还是不会中断的！**

如果类用public修饰，则类名必须与文件名相同。一个文件中只能有一个public修饰的类。

JDK9 不够稳定，特别是Tomcat7 **无法在JDK9 中运行**，

不建议在 Java 程序中使用 char 数据类型

双引号，用来引用字符串，单引号用来表示单个字符

中文编码表

* GB2312
* UNICODE

字符中重要的ASCII码对应关系

* a : 97
* A : 65
* 0 : 48、

### Final

引用类型的变量值为对象地址值，地址值不能更改，但是地址内的对象属性值可以修改。

final Person p = new Person();

Person p2 = new Person();

p = p2; //final修饰的变量p，所记录的地址值不能改变

p.name = "小明";//可以更改p对象中name属性值

### static

开发中，我们想在类中定义一个静态常量，通常使用public static final修饰的变量来完成定义。此时变量名用全部大写，多个单词使用下划线连接

* 接口中的每个成员变量都默认使用public static final修饰。
* 所有接口中的成员变量已是静态常量，由于接口没有构造方法，所以必须显示赋值。可以直接用接口名访问。

interface Inter {

public static final int COUNT = 100;

}

//访问接口中的静态变量

Inter.COUNT

### 匿名类

匿名对象是指创建对象时，只有创建对象的语句，却没有把对象地址值赋值给某个变量。

匿名对象在没有指定其引用变量时，只能使用一次。

new Person().eat(); 创建一个匿名对象，调用eat方法

new Person().eat(); 想再次调用eat方法，重新创建了一个匿名对象

### 内部类

访问方式： 外部类名.内部类名 变量名 = new 外部类名().new 内部类名();

### 匿名内部类

内部类是为了应对更为复杂的类间关系。查看源代码中会涉及到，而在日常业务中很难遇到，这里不做赘述。最常用到的内部类就是匿名内部类，它是局部内部类的一种。  
定义的匿名内部类有两个含义：

* 临时定义某一指定类型的子类
* 定义后即刻创建刚刚定义的这个子类的对象

定义匿名内部类的作用与格式  
作用：匿名内部类是创建某个类型子类对象的快捷方式。  
格式：

new 父类或接口(){

//进行方法重写

};

### System类

在API中System类介绍的比较简单，我们给出定义，System中代表程序所在系统，提供了对应的一些系统属性信息，和系统操作。  
System类不能手动创建对象，因为构造方法被private修饰，阻止外界创建对象。System类中的都是static方法，类名访问即可。在JDK中，有许多这样的类。  
常用方法

currentTimeMillis() 获取当前系统时间与1970年01月01日00:00点之间的毫秒差值

exit(int status) 用来结束正在运行的Java程序。参数传入一个数字即可。通常传入0记为正常状态，其他为异常状态

gc() 用来运行JVM中的垃圾回收器，完成内存中垃圾的清除。

getProperty(String key) 用来获取指定键(字符串名称)中所记录的系统属性信息

arraycopy方法，用来实现将源数组部分元素复制到目标数组的指定位置

练习一：定义一个方法，接收一个数组，数组中存储10个学生考试分数，该方法要求返回考试分数最低的后三名考试分数。

[复制代码](javascript:void(0);)

public static int[] method(double[] arr){

Arrays.sort(arr); //进行数组元素排序（元素值从小到大进行排序）

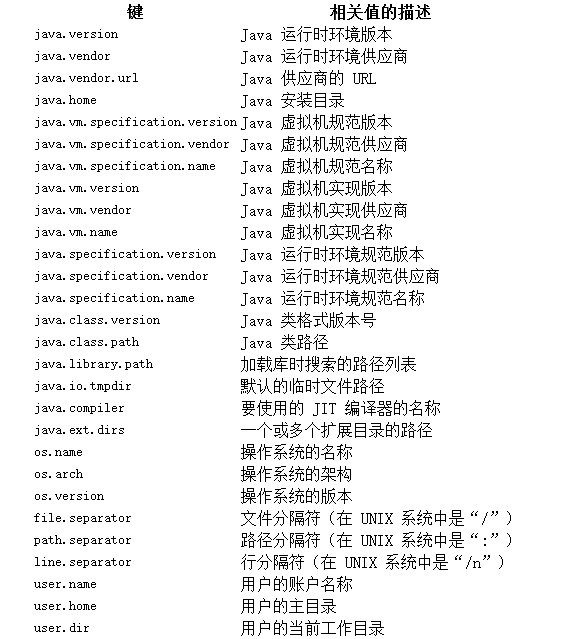
int[] result = new int[3]; //存储后三名考试分数

System.arraycopy(arr, 0, result, 0, 3);//把arr数组前3个元素复制到result数组中

return result;

}

[复制代码](javascript:void(0);)



### 基本包装类

1字符串变成基本类型

parseXXX(String s);其中XXX表示基本类型，参数为可以转成基本类型的字符串，如果字符串无法转成基本类型，将会发生数字转换的问题 NumberFormatException

System.out.println(Integer.parseInt("123") + 2);

//打印结果为 125

2 基本类型变成字符串的三种方式

* 基本类型直接与””相连接即可；34+""
* 调用String的valueOf方法；String.valueOf(34) ；

调用包装类中的toString方法；Integer.toString(34) ；

3 基本类型和对象的转换

自动装箱拆箱 当数值在byte范围之内时，进行自动装箱，不会新创建对象空间而是使用已有的空间

nteger a = new Integer(3);

Integer b = new Integer(3);

System.out.println(a==b);//false

System.out.println(a.equals(b));//true

System.out.println("---------------------");

Integer x = 127;

Integer y = 127;

//在jdk1.5自动装箱时，如果数值在byte范围之内，不会新创建对象空间而是使用原来已有的空间。

System.out.println(x==y); //true

System.out.println(x.equals(y)); //true

[复制代码](javascript:void(0);)

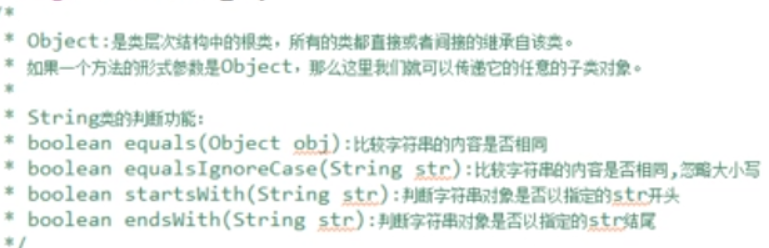
## TCP/UDP

由于使用UDP协议消耗资源小，通信效率高，所以通常都会用于音频、视频和普通数据的传输例如视频会议都使用UDP协议，因为这种情况即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。

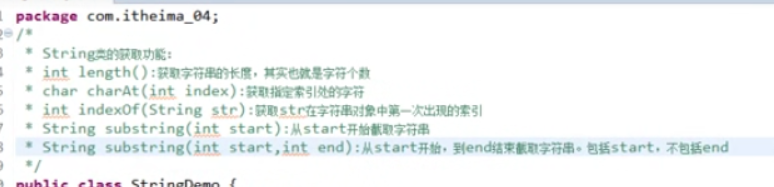
由于TCP协议的面向连接特性，它可以保证传输数据的安全性，所以是一个被广泛采用的协议，例如在下载文件时，如果数据接收不完整，将会导致文件数据丢失而不能被打开，因此，下载文件时必须采用TCP协议。

## 1：String类的常见方法

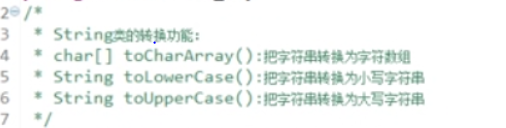
String类的判断功能



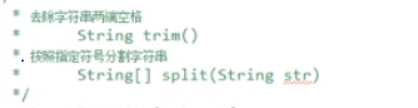
String类的获取功能

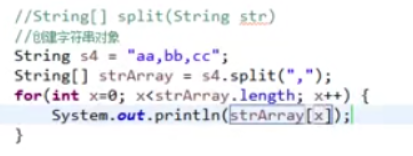


String类的转换功能

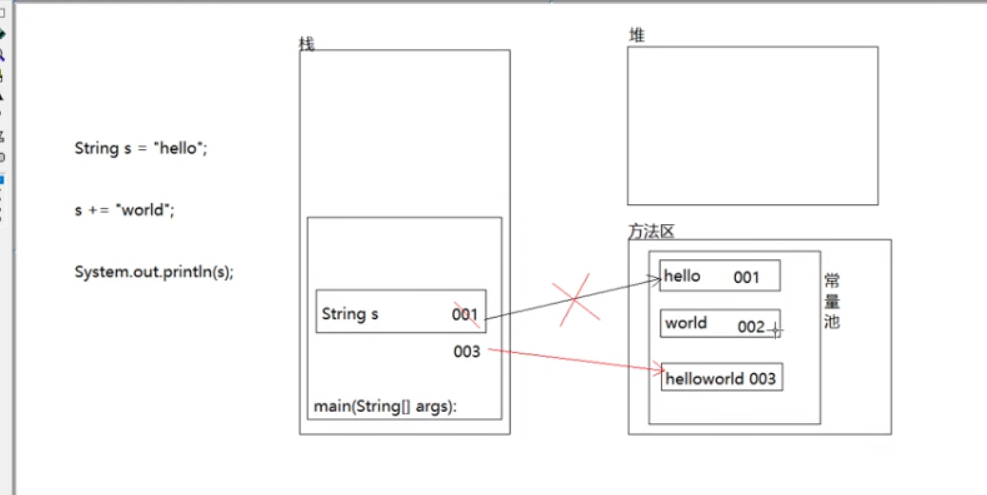


String类的去空格和分割功能





String类拼接的弊端-浪费空间有费时



## 2 StringBuilder的常用方法

添加及反转功能

## 

## 3 集合

//由于元素被存放进集合后全部被提升为Object类型

//当需要使用子类对象特有方法时，需要向下转型

### 迭代器

一直把集合中的所有元素全部取出。这种取出方式专业术语称为迭代。

ArrayList 重写方法 iterator(),返回了Iterator接口的实现类的对象

Iterator接口也可以使用<>来控制迭代元素的类型的。代码演示如下：

[复制代码](javascript:void(0);)

Collection<String> coll = new ArrayList<String>();

coll.add("abc");

coll.add("aabbcc");

coll.add("shitcast");

Iterator<String> it = coll.iterator();

while (it.hasNext()) {

String str = it.next();

//当使用Iterator<String>控制元素类型后，就不需要强转了。获取到的元素直接就是String类型

System.out.println(str.length());

}

### 泛型接口

[复制代码](javascript:void(0);)

/\*

\* 带有泛型的接口

\*

\* public interface List <E>{

\* abstract boolean add(E e);

\* }

\*

\* 实现类,先实现接口,不理会泛型

\* public class ArrayList<E> implements List<E>{

\* }

\* 调用者 : new ArrayList<String>() 后期创建集合对象的时候,指定数据类型

\*

\*

\* 实现类,实现接口的同时,也指定了数据类型

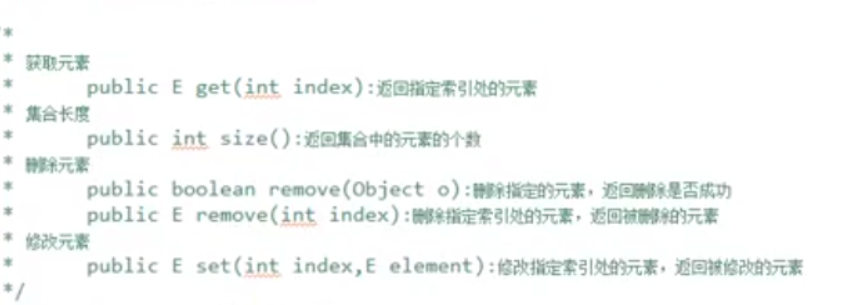
\* public class XXX implements List<String>{

\* }

\* new XXX()

\*/

ArrayList常见方法



### HashSet

使用HashSet存储自定义类型，如果没有重写该类的hashCode与equals方法，则判断重复时，使用的是地址值，如果想通过内容比较元素是否相同，需要重写该元素类的hashcode与equals方法。

### Map静态导入

在导包的过程中我们可以直接导入静态部分，这样某个类的静态成员就可以直接使用了。在源码中经常会出现静态导入。

静态导入格式： import static XXX.YYY;  ，导入后YYY可直接使用。

例如：Map.Entry的访问，简化后为Entry

[复制代码](javascript:void(0);)

import static java.util.Map.Entry;

public class HashMapTest {

public static void main(String[] args) {

//1,创建hashmap集合对象。

Map<Student,String> map = new HashMap<Student,String>();

//取出元素。键值对方式

//Set<Map.Entry<Student, String>> entrySet = map.entrySet();

Set<Entry<Student, String>> entrySet = map.entrySet();

//for (Map.Entry<Student, String> entry : entrySet) {

for (Entry<Student, String> entry : entrySet) {

Student key = entry.getKey();

String value = entry.getValue();

System.out.println(key.toString()+"....."+value);

}

}

}

#### Map集合不能直接使用迭代器或者foreach进行遍历。但是转成Set之后就可以使用了。

当给HashMap中存放自定义对象时，如果自定义对象作为key存在，这时要保证对象唯一，必须复写对象的hashCode和equals方法(如果忘记，请回顾HashSet存放自定义对象)。

如果要保证map中存放的key和取出的顺序一致，可以使用LinkedHashMap集合来存放

## 异常

异常分两种：  
　1：编译时被检查的异常，只要是Exception及其子类都是编译时被检测的异常。  
　2：运行时异常，其中Exception有一个特殊的子类RuntimeException，以及RuntimeException的子类是运行异常，也就说这个异常是编译时不被检查的异常。  
  
编译时被检查的异常和运行时异常的区别：  
　编译被检查的异常在函数内被抛出，函数必须要声明，否编译失败。  
　声明的原因：是需要调用者对该异常进行处理。  
　运行时异常如果在函数内被抛出，在函数上不需要声明。  
　不声明的原因：不需要调用者处理，运行时异常发生，已经无法再让程序继续运行，所以，不让调用处理的，直接让程序停止，由调用者对代码进行修正。

# Io流

//使用缓冲流一定要调用flush方法

1.BufferedOutputStream在close()时会自动flush  
2.BufferedOutputStream在不调用close()的情况下，缓冲区不满，又需要把缓冲区的内容写入到文件或通过网络发送到别的机器时，才需要调用flush.

close（）方法的作用  
1、关闭输入流，并且释放系统资源  
2、BufferedInputStream装饰一个 InputStream 使之具有缓冲功能，is要关闭只需要调用最终被装饰出的对象的 close()方法即可，因为它最终会调用真正数据源对象的 close()方法。因此，可以只调用外层流的close方法关闭其装饰的内层流。

　//一定要记得关闭流，否则文件可能一直被占用而无法操作

缓冲流的速度更快 效率更高

BufferedReader ---》String readLine() 读取一个文本行。

BufferedWriter ---》void newLine() 写入一个行分隔符。

## FileWirter写数据（基本流）

B:FileWriter方法:

构造方法

FileWriter(String fileName) 传入一个文件的路径

成员方法

void write(String str) 向文件中写str

void flush() 将内存中的数据刷新到文件中刷新缓冲区。刷新方法将数据从内存刷到硬盘上

void close() 关流释放系统底层资源先刷新缓冲区，然后通知系统释放资源。

FileWriter其它写方法

void write(String str):写一个字符串数据

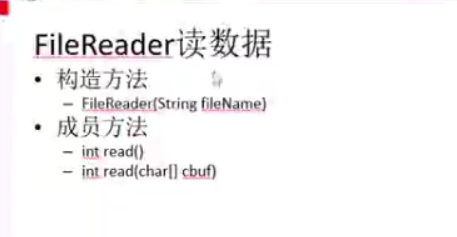
void write(String str,int index,int len):写一个字符串中的一部分数据

void write(int ch):写一个字符数据,这里写int类型的好处是既可以写char类型的数据，也可以写char对应的int类型的值。'a',97

void write(char[] chs):写一个字符数组数据

void write(char[] chs,int index,int len):写一个字符数组的一部分数据

## FileReader类读取数据



输入流读文件的步骤：

\* A:创建输入流对象

\* B:调用输入流对象的读数据方法

\* C:释放资源

第一种方法 int read():一次读取一个字符

第二种方法 一次读取一个数组

/读写数据

char[] chs = new char[1024];

int len;

while((len=fr.read(chs))!=-1) {

fw.write(chs, 0, len);

}

通过测试，我们知道，如果读取数据的返回值是-1的时候，就说明没有数据了，这也是我们循环的结束条件

\*/

int ch;

//1:fr.read()

//2:ch=fr.read()

//3:ch != -1

while((ch=fr.read())!=-1) {

//System.out.println(ch);

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

System.out.print((char)ch);

}

//释放资源

fr.close();

## 缓冲流（比基本流更加高效 固一般都用此）

BufferedWriter

BufferedReader

## 缓冲流的特有方法使用

BufferedWriter

void newLine():写一个换行符，这个换行符由系统决定,不同的操作系统newLine()方法使用的换行符不同

windows:\r\n

linux:\n

mac:\r

BufferedReader

String readLine():一次读取一行数据，但是不读取换行符

## Properties类

Properties类表示了一个持久的属性集。Properties可保存在流中或从流中加载。属性列表中每个键及其对应值都是一个字符串。

1、**序列化**是干什么的？

　　简单说就是为了保存在内存中的各种对象的状态（也就是实例变量，不是方法），并且可以把保存的对象状态再读出来。虽然你可以用你自己的各种各样的方法来保存object states，但是**Java**给你提供一种应该比你自己好的保存对象状态的机制，那就是序列化。

# 异常

Finally的应用场景

\* finally:组合try...catch使用，用于释放资源等收尾工作，无论try...catch语句如何执行，finally的代码一定会执行

\*

\* try {

\* 有可能出现问题的代码;

\*

\* } catch(异常对象) {

\* 处理异常;

\* } finally {

\* 释放资源;

\* 清理垃圾;

\* }

\*

\*/

public class ExceptionDemo5 {

public static void main(String[] args) {

//method();

FileWriter fw = null;

try {

//System.out.println(2 / 0);

fw = new FileWriter("a.txt");

fw.write("hello");

fw.write("world");

System.out.println(2 / 0);

fw.write("java");

//fw.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} finally {

//释放资源

try {

if(fw != null) {

fw.close();

}

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

# File类

## File类创建和删除功能

boolean createNewFile():指定路径不存在该文件时时创建文件,返回true,否则返回false

boolean mkdir():当指定的单级文件夹不存在时创建文件夹并返回true，否则返回false

boolean mkdirs():当指定的多级文件夹某一级文件夹不存在时,创建多级文件夹并返回true,否则返回false

boolean delete():删除文件或者删除单级文件夹

注意：删除一个文件夹，这个文件夹下面不能有其他的文件和文件夹

## File类的判断功能

boolean exists():判断指定路径的文件或文件夹是否存在

boolean isAbsolute():判断当前路路径是否是绝对路径

boolean isDirectory():判断当前的目录是否存在

boolean isFile():判断当前路径是否是一个文件

boolean isHidden():判断当前路径是否是隐藏文件

## File类的获取功能和修改名字功能

File getAbsoluteFile():获取文件的绝对路径,返回File对象

String getAbsolutePath():获取文件的绝对路径,返回路径的字符串

String getParent():获取当前路径的父级路径,以字符串形式返回该父级路径

File getParentFile():获取当前路径的父级路径,以字File对象形式返回该父级路径

String getName():获取文件或文件夹的名称

String getPath():获取File对象中封装的路径

long lastModified():以毫秒值返回最后修改时间

long length():返回文件的字节数

boolean renameTo(File dest): 将当前File对象所指向的路径 修改为 指定File所指向的路径

## File类的其它获取功能

String[] list():以字符串数组的形式返回当前路径下所有的文件和文件夹的名称

File[] listFiles():以File对象的形式返回当前路径下所有的文件和文件夹的名称

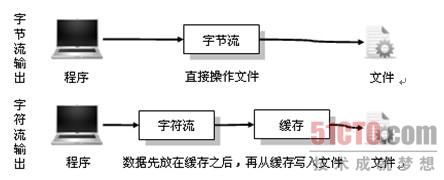
# IO流分类：

流向

输入流 读取数据 FileReader Reader

输出流 写出数据 FileWriter Writer

数据类型



字节流

字节输入流 读取数据 InputStream(一般使用其子类FileInputStream)

标准输入输出流：

public static final InputStream in：字节输入流,用来读取键盘录入的数据

字节输出流 写出数据 OutputStream(一般使用其子类FileOutputStream)

public static final PrintStream out：字节输出流,将数据输出到命令行

字符流

字符输入流 读取数据 Reader(一般使用其子类FileReader)

字符输出流 写出数据 Writer(一般使用其子类FileReader)

1.Java的字节流  
   InputStream是所有字节输入流的祖先，而OutputStream是所有字节输出流的祖先。  
2.Java的字符流  
  Reader是所有读取字符串输入流的祖先，而writer是所有输出字符串的祖先。  
InputStream，OutputStream,Reader,writer都是抽象类。所以不能直接new

**在所有的硬盘上保存文件或进行传输的时候都是以字节的方法进行的，包括图片也是按字节完成，而字符是只有在内存中才会形成的，所以使用字节的操作是最多的。**

 如果要java程序实现一个拷贝功能，应该选用字节流进行操作（可能拷贝的是图片），并且采用边读边写的方式（节省内存）。

## 转换流

### OutputStreamWriter:将字节输出流转换为字符输出流

OutputStreamWriter(System.*out*)字节流输出转换成字符输出流

### InputStreamReader:将字节输入流转换为字符输入流

InputStreamWriter(System.*In*) 将字节输入流转换为字符输入流

为了输入输出流一致 固有时需要转换流

## 打印流PrintStream(字节打印流)

## PrintWriter(字符打印流)

## 注意：只能输出不能输入

## 对象操作流

ObjectInputStream 称为 反序列化流,利用输入流从文件中读取对象

ObjectOutputStream 称为 序列化流,利用输出流向文件中写入对象

readObject()

writeObject();

解决对象输入流读取对象出现异常的问题-就是对象读取完了 继续读取会抛出异常 解决方法——（把对象写进集合 在把集合装载在对象操作流中 那么读取的时候就可以根据集合的大小 准确的读出对象 ）

**public** **class** ObjectOutputStreamDemo3 {

**public** **static** **void** main(String[] args) **throws** IOException, ClassNotFoundException {

//method();

//创建对象输入流的对象

ObjectInputStream ois = **new** ObjectInputStream(**new** FileInputStream("b.txt"));

//读取数据

Object obj = ois.readObject();

//System.out.println(obj);

//向下转型，获取具体的子类对象

ArrayList<Student> list = (ArrayList<Student>) obj;

**for** (Student student : list) {

System.*out*.println(student);

}

//释放资源

ois.close();

}

**private** **static** **void** method() **throws** IOException, FileNotFoundException {

//创建对象输出流的对象

ObjectOutputStream oos = **new** ObjectOutputStream(**new** FileOutputStream("b.txt"));

//创建集合对象

ArrayList<Student> list = **new** ArrayList<Student>();

//添加学生对象

list.add(**new** Student("wangwu",30));

list.add(**new** Student("zhaoliu",28));

//写出集合对象

oos.writeObject(list);

//释放资源

oos.close();

}

}

特点：用于操作对象。可以将对象写入到文件中，也可以从文件中读取对象。

**Serializable:序列号（是根据成员类构成方法一定的算法自动算出的编号），是一个标识接口，只起标识作用，没有方法**

**\* 当一个类的对象需要IO流进行读写的时候，这个类必须实现该接口**

Exception in thread "main" java.io.InvalidClassException

序列化ID 是对象一旦实现seralizeble接口的就会根据自身的成员和构造生成一个id,序列化的时候也会

将这个id序列化到文件中, 如果一旦成员改变,就会导致文件中id和我们类中的id不一致,

解决方案:

将这个id固定写好;

\* 当 Serialization 运行时检测到某个类具有以下问题之一时，抛出此异常。

该类的序列版本号与从流中读取的类描述符的版本号不匹配 （一般是已经写入数据，但文件被更改 固读出来的序列号跟以前不匹配 抛出异常 那么可以固定这个序列号 就不会出现这种情况）

该类包含未知数据类型

该类没有可访问的无参数构造方法 （了解就好）

# Properties集合

Properties 类表示了一个持久的属性集。Properties 可保存在流中或从流中加载。属性列表中每个键及其对应值都是一个字符串。

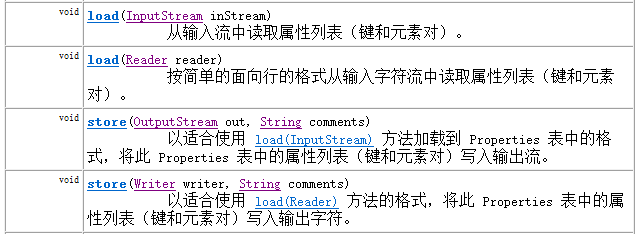
特点：

1、Hashtable的子类，map集合中的方法都可以用。

2、该集合没有泛型。键值都是字符串。

3、它是一个可以持久化的属性集。键值可以存储到集合中，也可以存储到持久化的设备(硬盘、U盘、光盘)上。键值的来源也可以是持久化的设备。

4、有和流技术相结合的方法。



## 编码表的概述

编码表：把计算机底层的二进制数据转换成我们能看到的字符

\* ASCII

\*

\* GB2312 --- GBK

\* Unicode 所有的字符都占2个字节

\* UTF-8 长度可变的码表

\*

\* ANSI：本地编码表 gbk

\* Java中的字符串默认使用的ANSI（gbk）

\*

\* 乱码：编码保持前后一致即可解决

# 多线程

格式:

synchronized(锁对象){

//需要同步的代码

}

注意：锁对象需要被所有的线程所共享

**public** **class** TicketThread **implements** Runnable {

**int** tickets = 100;//火车票数量

Object obj = **new** Object();

@Override

**public** **void** run() {

//出售火车票

**while**(**true**) {

**synchronized** (obj) {

格式:

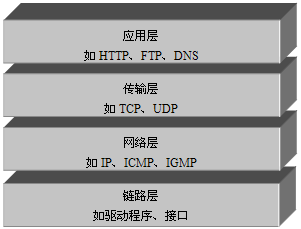
修饰符 synchronized 返回值 方法名(){

}

非静态同步方法的锁对象是this

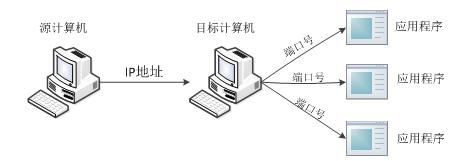
\* 静态的同步方法的锁对象是当前类的字节码对象

# 面向网络编程

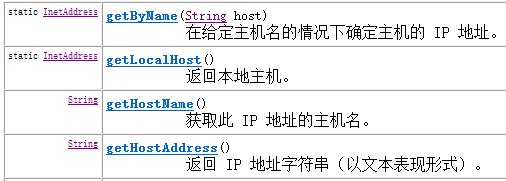


0~1023之间的端口号用于一些知名的网络服务和应用，用户的普通应用程序需要使用1024以上的端口号，从而避免端口号被另外一个应用或服务所占用。

接下来通过一个图例来描述IP地址和端口号的作用，如下图所示。



了解了IP地址的作用，我们看学习下JDK中提供了一个InetAdderss类，该类用于封装一个IP地址，并提供了一系列与IP地址相关的方法，下表中列出了InetAddress类的一些常用方法。



InetAddress address = InetAddress.getByName("itheima");//给定主机名

InetAddress address = InetAddress.*getByName*("192.168.1.107");//ip地址是唯一的

## UDP协议

由于使用UDP协议消耗资源小，通信效率高，所以通常都会用于音频、视频和普通数据的传输例如视频会议都使用UDP协议，因为这种情况即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。

DatagramPacket类，该类的实例对象就相当于一个集装箱，用于封装UDP通信中发送或者接收的数据。

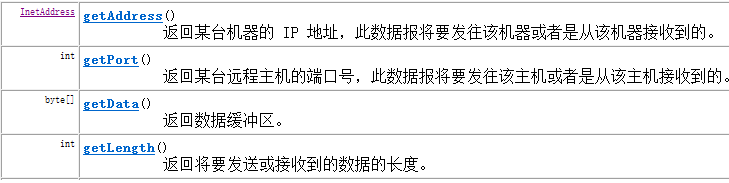
在创建发送端和接收端的DatagramPacket对象时，使用的构造方法有所不同

发送端一定要明确指出数据的目的地(ip地址和端口号)，而接收端不需要明确知道数据的来源，只需要接收到数据即可。



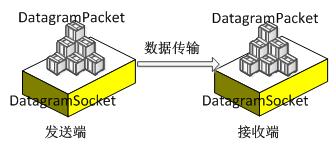


上面我们讲解了DatagramPacket的构造方法，接下来对DatagramPacket类中的常用方法进行详细地讲解，如下表所示。



## DatagramSocket类

JDK中提供的一个DatagramSocket类。DatagramSocket类的作用就类似于码头





该构造方法用于创建发送端的DatagramSocket对象，在创建DatagramSocket对象时，并没有指定端口号，此时，系统会分配一个没有被其它网络程序所使用的端口号。



该构造方法既可用于创建接收端的DatagramSocket对象，又可以创建发送端的DatagramSocket对象，在创建接收端的DatagramSocket对象时，必须要指定一个端口号，这样就可以监听指定的端口。

上面我们讲解了DatagramSocket的构造方法，接下来对DatagramSocket类中的常用方法进行详细地讲解。



发送端：

创建发送端Socket对象

创建数据并打包

发送数据

释放资源

接收端：

创建接收端Socket对象

接收数据

解析数据

输出数据

释放资源

## TCP协议

Udp不能发送大量数据吗？

TCP通信同UDP通信一样，都能实现两台计算机之间的通信，通信的两端都需要创建socket对象。

区别在于，UDP中只有发送端和接收端，不区分客户端与服务器端，计算机之间可以任意地发送数据。

而TCP通信是严格区分客户端与服务器端的，在通信时，必须先由客户端去连接服务器端才能实现通信，服务器端不可以主动连接客户端，并且服务器端程序需要事先启动，等待客户端的连接。

在JDK中提供了两个类用于实现TCP程序，一个是ServerSocket类，用于表示服务器端，一个是Socket类，用于表示客户端。

ServerSocket的构造方法进行逐一地讲解。





，在创建ServerSocket对象后，需要继续调用该对象的accept()方法，接收来自客户端的请求。当执行了accept()方法之后，服务器端程序会发生阻塞，直到客户端发出连接请求，accept()方法才会返回一个Scoket对象用于和客户端实现通信，程序才能继续向下执行。

为此JDK提供了一个Socket类，用于实现TCP客户端程序。

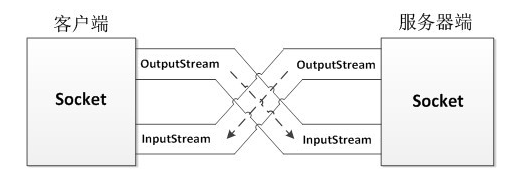


接下来学习一下Socket的常用方法，如表所示。

|  |  |
| --- | --- |
| **方法声明** | **功能描述** |
| int getPort() | 该方法返回一个int类型对象，该对象是Socket对象与服务器端连接的端口号 |
| InetAddress getLocalAddress() | 该方法用于获取Socket对象绑定的本地IP地址，并将IP地址封装成InetAddress类型的对象返回 |
| void close() | 该方法用于关闭Socket连接，结束本次通信。在关闭socket之前，应将与socket相关的所有的输入/输出流全部关闭，这是因为一个良好的程序应该在执行完毕时释放所有的资源 |
| InputStream getInputStream() | 该方法返回一个InputStream类型的输入流对象，如果该对象是由服务器端的Socket返回，就用于读取客户端发送的数据，反之，用于读取服务器端发送的数据 |
| OutputStream getOutputStream() | 该方法返回一个OutputStream类型的输出流对象，如果该对象是由服务器端的Socket返回，就用于向客户端发送数据，反之，用于向服务器端发送数据 |

在Socket类的常用方法中，getInputStream()和getOutStream()方法分别用于获取输入流和输出流。当客户端和服务端建立连接后，数据是以IO流的形式进行交互的，从而实现通信。

接下来通过一张图来描述服务器端和客户端的数据传输，如下图所示。



发送方

使用TCP协议发送数据

创建发送端Socket对象（创建连接）

获取输出流对象

发送数据

释放资源

服务端

使用TCP协议接收数据

创建接收端Socket对象ServerSocket对象

监听（阻塞）

获取输入流对象

获取数据

输出数据

释放资源 //服务端一般不关闭

# JAVA前端

Css中

块级元素会自动在前面和后面加上换行，并且在其上的width和height也能够生效  
  
div默认是块级元素

内联元素前后没有换行，并且在其上的width和height也无效。 其大小由其中的内容决定  
span默认是内联元素

内联是不换行，但是不能指定大小  
块级时能指定大小，但是会换行  
  
有时候，需要元素处于同一行，同时还能指定大小，这个时候，就需要用到**内联-块级** inline-block

完整的javascript由[语言基础](http://how2j.cn/k/javascript/javascript-type/428.html)，[BOM](http://how2j.cn/k/javascript/javascript-bom-window/449.html)和[DOM](http://how2j.cn/k/dom/dom-node/457.html)组成。

　需要注意的是，使用float脱离文档流时，其他盒子会无视这个元素，但其他盒子内的文本依然会为这个元素让出位置，环绕在周围。   
　　而对于使用absolute positioning脱离文档流的元素，其他盒子与其他盒子内的文本都会无视它

<div style="clear:both"></div>的作用清除浮动

# Web

## CSS

块要在同一行就得加浮动

Float时

加一个空白块clear=“both” 下面块元素就可以正常排序

## JS

document.write()//向页面输出

.innerHTML()//向页面输出

document.write是直接将内容写入页面的内容流，会导致页面全部重绘，innerHTML将内容写入某个DOM节点，不会导致页面全部重绘

Document.getElementById(“id的名称”)

Document.getElementByName（“Name的名称”）

### 定时器

1

由于定时器在调用时，都会返回一个整形的数字，该数字代表定时器的序号，即第多少个定时器，所以定时器的清除要借助于这个返回的数字。

var timeid =  setInterval（“方法名或方法”，“延时”）;//创建

clearInterval(timeid); 消除定时器

2

当方法执行完成定时器停止(但是定时器还在,只不过没用了);

var tmid =  setTimeout(“方法名或方法”, “延时”); 只执行一次

clearTimeout(tmid);

### 事件

1、在html中，使用onclick属性 固方法名要双引号 HMTL对大小写不敏感，而JS则会区分大小写。

2、在javascript中，使用onclick属性

注意函数名没有双引号。

Onload（）文档加载完成后能立即触发

表单事件

onfoucs：在对象获得焦点时发生的事件

onblur:在对象失去焦点时发生的事件

onkeyup: 按键抬起事件

onsubmit事件

表单是否提交取决于onsubmit事件的返回值。  
若onsubmit事件返回false，则将阻止表单的提交。  
如果不返回值，则默认为true。固onsubmit="return check();" 是否提交表单取决于check()的返回值

ondblclick（）双击事件； multiple 显示多行；

<select multiple="multiple" id="leftSelect" ondblclick="selectOne()">

<option>华为</option>

<option>小米</option>

<option>锤子</option>

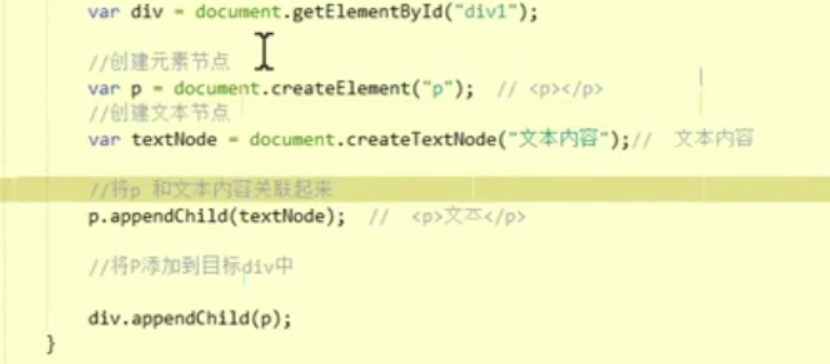
<option>oppo</option>

</select>

图片的显示和隐藏

img.style.display = "block";

img.style.display = "none";

元素节点的创建过程

删除select元素里面的所有option

三种方法：

<select id="city">

<option>湖南</option>

<option>上海</option>

</select>

使用jquery清空

$("#city").find("option").remove();

或者

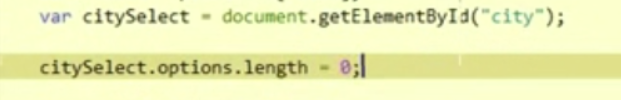
$("#city").empty();

或者

$("#city").html("");

---------------------

还有一种时



## Jquery

$是jQuery的别名

文档加载事件

$(function(){　});是下面的简写 表示当页面加载完立即执行 类是Onload

$(document).ready(function(){

// 在这里写你的代码...

});

## Jq与js的对象转换

1)jQuery对象是一个数据对象，可以通过[index]的方法，来得到相应的js对象。   
  
如：var $v =$("#v") ; //jQuery对象   
  
var v=$v[0]; //js对象   
  
alert(v.checked) //检测这个checkbox是否被选中   
  
(2)jQuery本身提供，通过.get(index)方法，得到相应的js对象   
如：var $v=$("#v"); //jQuery对象   
var v=$v.get(0); //js对象 

**js对象转成jQuery对象:**   
  
  
如：var v=document.getElementById("v"); //js对象   
  
var $v=$(v); //jQuery对象

## 动画

基本

* + [show()](mk:@MSITStore:C:\Users\林志育\Desktop\java笔记\黑马笔记\jQueryAPI1.4.chm::/show.html)
  + [hide()](mk:@MSITStore:C:\Users\林志育\Desktop\java笔记\黑马笔记\jQueryAPI1.4.chm::/hide.html)

滑动

* + [slideDown(speed, [callback])](mk:@MSITStore:C:\Users\林志育\Desktop\java笔记\黑马笔记\jQueryAPI1.4.chm::/slideDown_speed_callback.html)
  + [slideUp(speed, [callback])](mk:@MSITStore:C:\Users\林志育\Desktop\java笔记\黑马笔记\jQueryAPI1.4.chm::/slideUp_speed_callback.html)

淡入淡出

* + [fadeIn(speed, [callback])](mk:@MSITStore:C:\Users\林志育\Desktop\java笔记\黑马笔记\jQueryAPI1.4.chm::/fadeIn_speed_callback.html)
  + [fadeOut(speed, [callback])](mk:@MSITStore:C:\Users\林志育\Desktop\java笔记\黑马笔记\jQueryAPI1.4.chm::/fadeOut_speed_callback.html)

自定义

* + [animate(param,[dur],[e],[fn])](mk:@MSITStore:C:\Users\林志育\Desktop\java笔记\黑马笔记\jQueryAPI1.4.chm::/animate_params_duration_easing_callback.html)

## 选择器

基本选择器

层级选择器

属性选择器

表单选择器

jQuery append(content)

jQuery after(content)

这个操作与对指定的元素执行appendChild方法，将它们添加到文档中的情况类似。

**Content ：**String, Element, jQuery

如 $("p").append("<b>Hello</b>");

## 遍历

第一种方法

$("div").each(function (index, domEle) { }); index 指的是索引domEle指的是内容

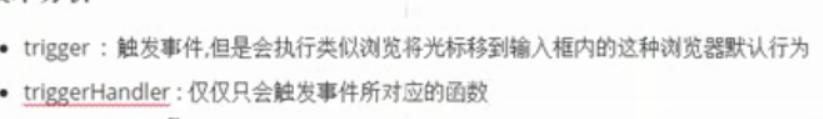
第二种方法

jQuery.each(object, [callback]) object需要例遍的对象或数组。

$.each( [0,1,2], function(i, n){

alert( "Item #" + i + ": " + n );

});



trigger(“触发的函数名”):在每一个匹配的元素上触发某类事件。

triggerHandler(“触发的函数名”):这个特别的方法将会触发指定的事件类型上所有绑定的处理函数。

triggerHandler只会触发JQ对象集合中第一个元素的事件处理函数

# Bootstrap

Bootstrap 是最受欢迎的 HTML、CSS 和 JS 框架，用于开发响应式布局、移动设备优先的 WEB 项目。

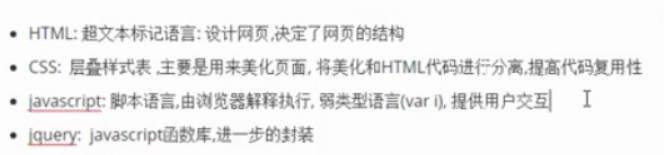
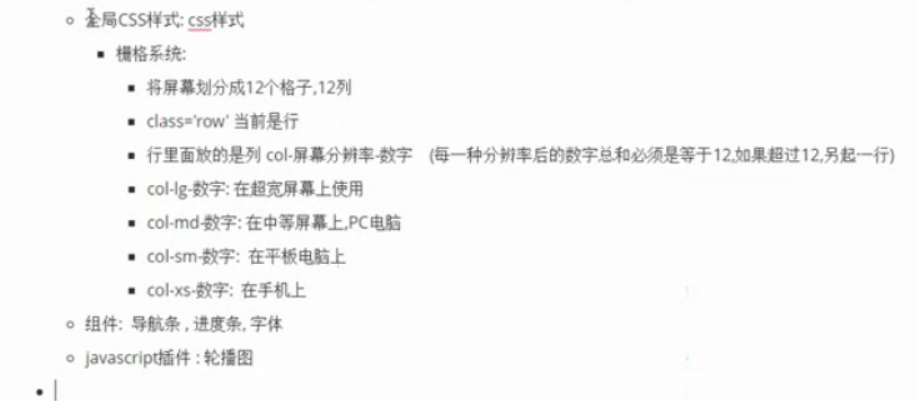
响应式---根据不同的分辨率显示不同的页面布局 提高用户体验

Bootstrap中文网

全局 CSS 样式（栅格系统 表格）

组件

JavaScript 插件



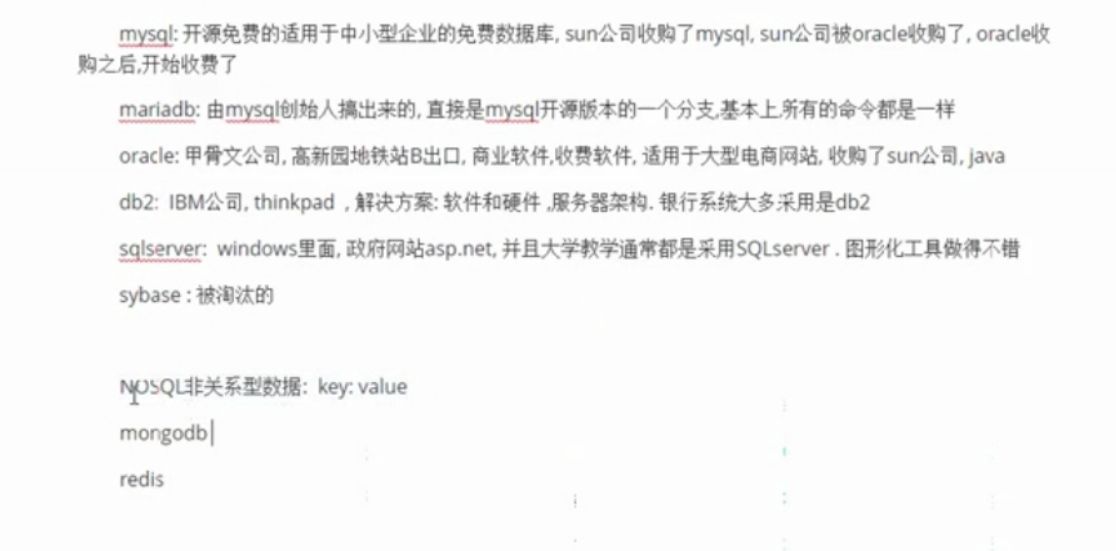
Jquery最重要的是要会使用选择器

# 数据库

1.什么是数据库，数据库有什么作用?

数据库就是存储数据的仓库，用户增删改查进行管理数据

2.常见数据库有哪些?



Mysql服务器的端口为3307 用户名 root 密码123456密码已经改为admin

命令行mysql –uroot –p123456 连接数据库

查看mysql服务

在运行中输入 services.msc

在使用mysql之前，要确保mysql服务开启.

在服务器可以通过右键属性，设置服务手动开启，这样这个服务就不会跟随电脑开机启动。

一种方法可以在命令行上通过命令来开启与关闭服务器

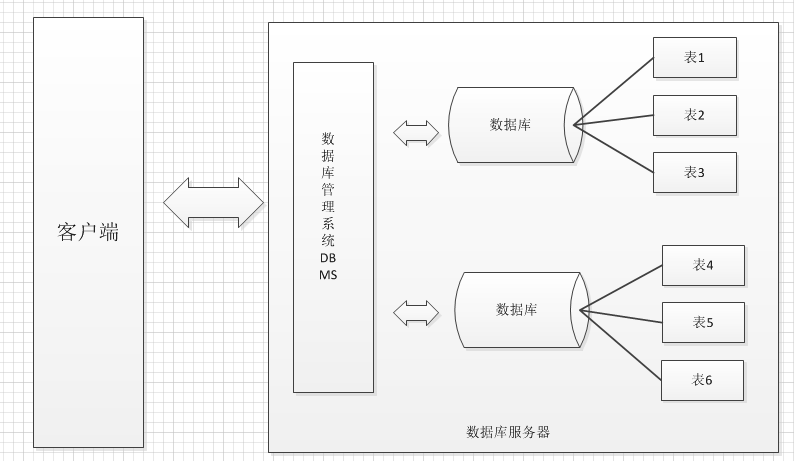
开启 net start mysql 必须以管理员身份启动cmd

关闭 net stop mysql

另一种方法就是服务里启动

Sqlyog在这个目录下

F:\mysql\sqlyog\SQLyog\_Enterprise



## SQL介绍

1.SQL是什么，它的作用是什么

结构化查询语言(Structured Query Language)简称SQL，是一种数据库查询和[程序设计语言](http://baike.baidu.com/view/128511.htm)，用于存取数据以及查询、更新和管理[关系数据库系统](http://baike.baidu.com/view/549699.htm)

SQL的语法有以下要求：

SQL语句可以单行或多行书写，以分号结尾；

可使用空格和缩进来增强语句的可读性；

MySQL数据库的SQL语句不区分大小写，建议使用大写，例如：SELECT \* FROM user。

char与varchar区别?

char是一个定长字符串.指定长度不会随着内容的不足而改变

varchar是一个可变长度的字符串，它根据信息自动改变长度.(只在字符串长度小于指定长度情况下)

数据库中一共有六种约束，而mysql只支持五种

主键是能确定一条记录

主键约束 primary key

非空 not null

唯一 unique

自动增长 auto\_increment

* 在values中列出的数据位置必须与被加入的列的排列位置相对应。

###### .除了数值类型外，其它的字段类型的值必须使用引号引起。

* 如果要插入空值，可以不写字段，或者插入 null.
* 注意:对于自动增长的列在操作时，直接插入null值即可.

###### 关于delete 与truncate的区别?(笔试题)

1.delete是一行一行删除 truncate是将表结构销毁，在重新创建表结构.

如果数据比较多，truncate的性能高。

2.delete是dml语句 truncate dcl语句

delete是受事务控制. 可以回滚数据.

truncate是不受事务控制. 不能回滚.

###### 编码问题

下面我们向user表中插入一条记录

insert into user(username,password) values(‘张三’,’123’);

当执行这条sql时，会报错

ERROR 1366 (HY000): Incorrect string value: '\xD5\xC5\xC8\xFD' for column 'username' at row 1

原因:

因为mysql的客户端编码的问题我们的是utf8,而系统的cmd窗口编码是gbk

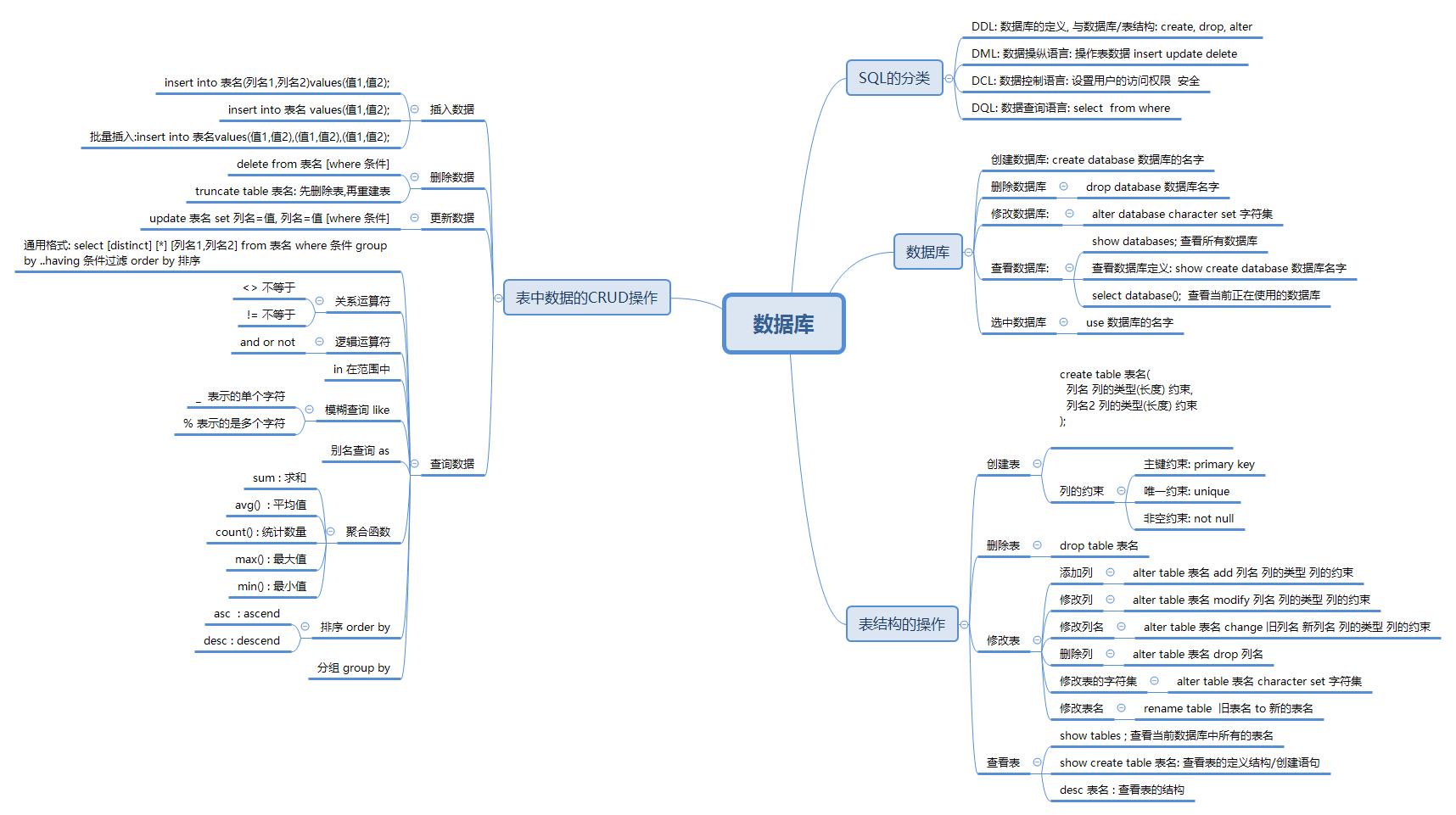
###### 笔试题: having与where的区别:

1.having是在分组后对数据进行过滤.

where是在分组前对数据进行过滤

1. having后面可以使用聚合函数

where后面不可以使用聚合函数。



## 数据库---英语单词

CRUD操作(create 添加数据read读取数据 update 修改数据delete删除数据)

ascend

英 [əˈsend] 美 [əˈsɛnd]

vt.

攀登;继承;占领

vi.

上升;爬坡;追溯

descend

英 [dɪˈsend] 美 [dɪˈsɛnd]

v.

降临;下来，下降 ;下斜;突然造访

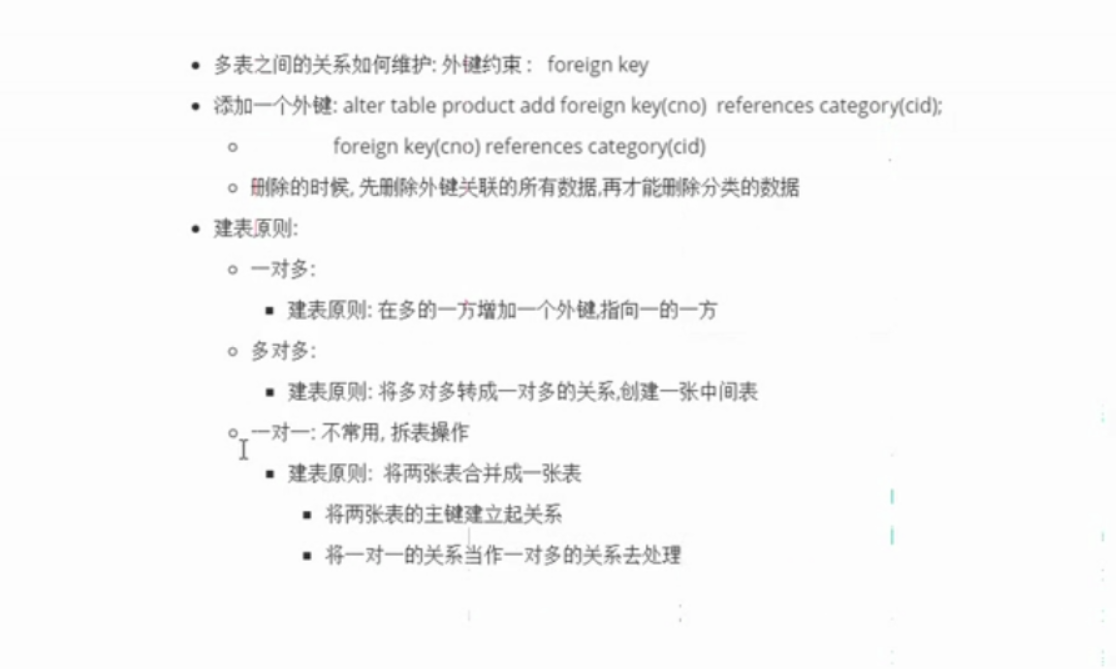
collate

英 [kəˈleɪt] 美 [kəˈlet, ˈkɑlˌet, ˈkoˌlet]

vt.

核对，校对（书、底稿等）;（装钉）整理，检查;[宗]委任牧师职

# 数据库day2



笛卡儿积 select 列 from table1 ，table2 ；

内连接查询

隐式查询 select 列 from table1 ，table2 where table1.col=table2.col

显式查询 select 列 from table1[inner] join table2  on table1.col=table2.col

外连接

左连接 LEFT  JOIN或LEFT OUTER JOIN          on

  select   a.\*,b.\*   from   a   left   join   b     on   a.id=b.parent\_id

右连接RIGHT  JOIN 或 RIGHT  OUTER  JOIN      on

分页查询



子查询

**经过个人实践，证明子查询效率特别低，而一般的子查询都可以由关连查询来实现相同的功能，关联查询的效率要提高很多，所以建议在数据查询时避免使用子查询（尤其是在记录很多时），而最好用关联查询来实现。**

# 你服务器

提供的服务类型不同，分为 文件服务器、[数据库服务器](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E6%9C%8D%E5%8A%A1%E5%99%A8/613818)、应用程序服务器、WEB服务器

常见的服务器端脚本语言，php，jsp，asp。

JSP全名为Java Server Pages，中文名叫java服务器页面，其根本是一个简化的Servlet设计，它是由Sun Microsystems公司倡导、许多公司参与一起建立的一种动态网页技术标准。

ASP即Active Server Pages，是MicroSoft公司开发的服务器端脚本环境，可用来创建动态交互式网页并建立强大的web应用程序。

客户端脚本，对应有 javascript，perl等等。

# JDBC

JAVA Database Connectivity

为什么会出现JDBC

> SUN公司提供的一种数据库访问规则、规范, 由于数据库种类较多，并且java语言使用比较广泛，sun公司就提供了一种规范，让其他的数据库提供商去实现底层的访问规则。 我们的java程序只要使用sun公司提供的jdbc驱动即可。

## ###使用JDBC的基本步骤

在这之前还需在项目下建立一个lib的目录导入驱动

F:\mysql\JDBC的驱动下的mysql-connector-java-5.1.7-bin.jar导进去

点击驱动并右键add to build path 让项目知道它的路径；

二次注册

1. 注册驱动

一次注册

DriverManager.registerDriver(new com.mysql.jdbc.Driver());这种方法不建议；因为会造成二次注册；

驱动防二次注册

Driver 这个类里面有静态代码块里面有（DriverManager.registerDriver(new com.mysql.jdbc.Driver());

），所以等同于我们注册了两次驱动。因为new一次就相当注册一次，再 registerDriver 固相当注册两次其实没这个必要的。

//静态代码块 ---> 类加载了，就执行。 java.sql.DriverManager.registerDriver(new Driver());

最后形成以下代码即可。

Class.forName("com.mysql.jdbc.Driver");//使用 Class 的静态 forName() 方法实现动态加载类 把它加载到内存来

这句话在驱动4版本后就可不用写 也就是不用注册 驱动会自动帮你注册

2. 建立连接

、//DriverManager.getConnection("jdbc:mysql://localhost/test?user=monty&password=greatsqldb");

//2. 建立连接 参数一： 协议 + 访问的数据库 ， 参数二： 用户名 ， 参数三： 密码。

conn = DriverManager.getConnection("jdbc:mysql://localhost：端口号/student", "root", "root");

记得加端口号 我mysql的端口号设置为3307；

3. 创建statement

//3. 创建statement ， 跟数据库打交道，一定需要这个对象

st = conn.createStatement();

4. 执行sql ，得到ResultSet

//4. 执行查询 ， 得到结果集

String sql = "select \* from t\_stu";

ResultSet rs = st.executeQuery(sql);

5. 遍历结果集

//5. 遍历查询每一条记录

while(rs.next()){

int id = rs.getInt("id");

String name = rs.getString("name");

int age = rs.getInt("age");

System.out.println("id="+id + "===name="+name+"==age="+age);

}

6. 释放资源 注意释放资源的顺序

if (rs != null) {

try {

rs.close();

} catch (SQLException sqlEx) { } // ignore

rs = null;

}

还有两个也要释放资源-这里没写出

## 使用properties配置文件

1. 在src底下声明一个文件 xxx.properties ，里面的内容吐下：

driverClass=com.mysql.jdbc.Driver

url=jdbc:mysql://localhost/student

name=root

password=root

2. 在工具类里面，使用静态代码块，读取属性

static{

try {

//1. 创建一个属性配置对象

Properties properties = new Properties();

InputStream is = new FileInputStream("jdbc.properties"); //对应文件位于工程根目录

//使用类加载器，去读取src底下的资源文件。 后面在servlet //对应文件位于src目录底下

//InputStream is = JDBCUtil.class.getClassLoader().getResourceAsStream("jdbc.properties");

//导入输入流。

properties.load(is);

//读取属性

driverClass = properties.getProperty("driverClass");

url = properties.getProperty("url");

name = properties.getProperty("name");

password = properties.getProperty("password");

} catch (Exception e) {

e.printStackTrace();

}

}

## 单元测试是什么？

     简单的说，单元测试就是对你程序中最小的功能模块进行测试

https://blog.csdn.net/weixin\_37380784/article/details/80540574

## ###数据库的CRUD sql

总结 insert ，delect，update 使用executeUpdate(sql) 返回int 类型数据

大于零表示操作成功

查找的话用.executeQuery(sql); 返回的使ResultSet

例如： }

\* delete

String sql = "delete from t\_stu where name='aobama'";

//影响的行数， ，如果大于0 表明操作成功。 否则失败

int result = st.executeUpdate(sql);

if(result >0 ){

System.out.println("删除成功");

}else{

System.out.println("删除失败");

}

## ##Statement安全问题

1. Statement执行 ，其实是拼接sql语句的，然后在一起执行。

固早期登入bug 密码输入后面价格 or 1=1 即可登入成功

SELECT \* FROM t\_user WHERE username='admin' AND PASSWORD='100234khsdf88' or '1=1'

固判断合理 登入成功

## PrepareStatement替换statement。

1. 相比较以前的statement， 预先处理给定的sql语句，对其执行语法检查。 在sql语句里面使用 ? 占位符来替代后续要传递进来的变量。 后面进来的变量值，将会被看成是字符串，不会产生任何的关键字。

String sql = "insert into t\_user values(null , ? , ?)";

ps = conn.prepareStatement(sql);

//给占位符赋值 从左到右数过来，1 代表第一个问号， 永远你是1开始。

ps.setString(1, userName);

ps.setString(2, password);

ps.executeQuery或者ps.executeUpdate;

## Dao模式

Data Access Object 数据访问对象--声明与实现分开

1 建一个接口 一般是组长要求实现的东西

2 建一个实现类 那程序员实现它

# XML和tomcat

笔记在黑马笔记

XML

1. 会写XML ， 看得懂XML

2. 解析

1. 有几种方式

Dom Sax

2. 针对以上两种方式，有如下解决方案 API

Dom4j

Jdom

jaxp sun 自带

Dom4j

//1. 构建解析对象

SAXReader reader = new SAXReader();

//2. 指定到底解析谁。 文件

Document document = reader.read(path|file|inputStream);

//3. 获取根元素

Element rootElement = document.getRootElement();

//获取根元素下的子元素 ，元素名称 ： stu

Element element = rootElement.element("stu").element("name");

String namerootElement.element("stu").element("name").getText();

//map<key ,value> list set

//获取根元素下的所有stu元素

List<Element> list = rootElement.elements("stu");

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

//这里是遍历出来了根标签下的所有子标签 stu

Element element = list.get(i);

element.getText();

}

Xpath : 支持我们在解析xml的时候，快速的定位到具体的元素 path --路径

//dom4j 支持xPath的方法有哪些。

//2. 指定到底解析谁。 文件

Document document = reader.read(path|file|inputStream);

//3. 获取根元素

Element rootElement = document.getRootElement();

现在调用该方法的是Element这种类型的对象

但是定义这个selectNodes的方法是定义在一个接口中 Node

请问这么写有没有错?

//获取文档中所有的stu 体现了java的多态 | 继承

rootElement.selectNodes("//stu");

//获取文档中第一个出现的stu

rootElement.selectSingleNode("//stu")

面试会出现 自己开发的话一般不用自己写

区别 ， 优缺点。

DTD

Schema

Tomcat知道如何跑起来即可

## TomCat启动闪退

1 环境配置问题闪退

2 端口被占用

3 分配的内存不够

4 我尝试了很多办法，一步步分析。到最后还是要归到版本的适配问题上，如果你的JDK版本过高，那你就换个版本高的tomcat,问题肯定迎刃而解，比如最新版JDK-10那你就配置tomcat-9的版本。如果版本适合，不存在环境配置好了还闪退的问题。

像我的电脑需要用tomcat-8的版本 不能用7的版本 ；

# #Http协议&Servlet

笔记在黑马笔记目录

状态码

咱们这次交互到底是什么样结果的一个code.

200 : 成功，正常处理，得到数据。

403 : for bidden 拒绝

404 ： Not Found

500 ： 服务器异常

302 ：重新定位 网页跳转 需要先设置状态码 再设置setheader（“location”，“跳转网页的路劲”）

# ##HttpServletRequest 和 HttpServletResponse

## ###Servlet配置方式

\* 1. 全路径匹配

> 以 / 开始 /a /aa/bb

> localhost:8080/项目名称/aa/bb

\* 2. 路径匹配 , 前半段匹配

> 以 / 开始 ， 但是以 \* 结束 /a/\* /\*

> \* 其实是一个通配符，匹配任意文字

> localhost:8080/项目名称/aa/bb

\* 3. 以扩展名匹配

> 写法： 没有/ 以 \* 开始 \*.扩展名 \*.aa \*.bb

## ###ServletContext

> Servlet 上下文

> 每个web工程都只有一个ServletContext对象。 说白了也就是不管在哪个servlet里面，获取到的这个类的对象都是同一个。

###如何得到对象

//1. 获取对象

ServletContext context = getServletContext();

### ### 有什么作用

1. 获取全局配置参数

2. 获取web工程中的资源 个人理解为其实就是获取文件的路径

3. 存取数据，servlet间共享数据 域对象

####.可以获取全局配置参数

![icon](img/img01.png)

获取全局参数

![icon](img/img02.png)

####. 可以获取Web应用中的资源

1. 获取资源在tomcat里面的绝对路径

先得到路径，然后自己new InpuStream

context.getRealPath("") //这里得到的是项目在tomcat里面的根目录。

D:\tomcat\apache-tomcat-7.0.52\apache-tomcat-7.0.52\wtpwebapps\Demo03\

String path = context.getRealPath("file/config.properties");

D:\tomcat\apache-tomcat-7.0.52\apache-tomcat-7.0.52\wtpwebapps\Demo03\file\config.properties

2. getResourceAsStream 获取资源 流对象

直接给相对的路径，然后获取流对象。

![icon](img/img03.png)

### 通过classloader去获取web工程下的资源

![icon](img/img04.png)

### 使用ServletContext存取数据。

1. 定义一个登陆的html页面， 定义一个form表单

![icon](img/img06.png)

2. 定义一个Servlet，名为LoginServlet

![icon](img/img07.png)

3. 针对成功或者失败，进行判断，然后跳转到不一样的网页

![icon](img/img08.png)

###ServletContext存取值分析

![icon](img/img09.png)

##细节：

<!--

A路径： Servlet的路径

http://localhost:8080/Demo4/login

B路径： 当前这个html的路径：

http://localhost:8080/Demo4/login.html -->

<form action="login" method="get">

账号:<input type="text" name="username"/><br>

密码:<input type="text" name="password"/><br>

<input type="submit" value="登录"/>

</form>

###ServletContext 何时创建， 何时销毁?

服务器启动的时候，会为托管的每一个web应用程序，创建一个ServletContext对象

从服务器移除托管，或者是关闭服务器。

\* ServletContext 的作用范围

> 只要在这个项目里面，都可以取。 只要同一个项目。 A项目 存， 在B项目取，是取不到的？ ServletContext对象不同。

##HttpServletRequest

> 这个对象封装了客户端提交过来的一切数据。

1. 可以获取客户端请求头信息

//得到一个枚举集合

Enumeration<String> headerNames = request.getHeaderNames();

while (headerNames.hasMoreElements()) {

String name = (String) headerNames.nextElement();

String value = request.getHeader(name);

System.out.println(name+"="+value);

}

2. 获取客户端提交过来的数据

String name = request.getParameter("name");

String address = request.getParameter("address");

System.out.println("name="+name);

System.out.println("address="+address);

-------------------------------------------------

//name=zhangsan&name=lisi&name=wangwu 一个key可以对应多个值。

Map<String, String[]> map = request.getParameterMap();

Set<String> keySet = map.keySet();

Iterator<String> iterator = keySet.iterator();

while (iterator.hasNext()) {

String key = (String) iterator.next();

System.out.println("key="+key + "--的值总数有："+map.get(key).length);

String value = map.get(key)[0];

String value1 = map.get(key)[1];

String value2 = map.get(key)[2];

System.out.println(key+" ======= "+ value + "=" + value1 + "="+ value2);

}

3. 获取中文数据

> 客户端提交数据给服务器端，如果数据中带有中文的话，有可能会出现乱码情况，那么可以参照以下方法解决。

\* 如果是GET方式

1. 代码转码

String username = request.getParameter("username");

String password = request.getParameter("password");

System.out.println("userName="+username+"==password="+password);

//get请求过来的数据，在url地址栏上就已经经过编码了，所以我们取到的就是乱码，

//tomcat收到了这批数据，getParameter 默认使用ISO-8859-1去解码

//先让文字回到ISO-8859-1对应的字节数组 ， 然后再按utf-8组拼字符串

username = new String(username.getBytes("ISO-8859-1") , "UTF-8");

System.out.println("userName="+username+"==password="+password);

直接在tomcat里面做配置，以后get请求过来的数据永远都是用UTF-8编码。

2. 可以在tomcat里面做设置处理 conf/server.xml 加上URIEncoding="utf-8"

<Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443" URIEncoding="UTF-8"/>

\* 如果是POST方式

这个说的是设置请求体里面的文字编码。 get方式，用这行，有用吗？ ---> 没用

request.setCharacterEncoding("UTF-8");

这行设置一定要写在getParameter之前。

##HttpServletResponse

> 负责返回数据给客户端。

\* 输出数据到页面上

//以字符流的方式写数据

//response.getWriter().write("<h1>hello response...</h1>");

//以字节流的方式写数据

response.getOutputStream().write("hello response2222...".getBytes());

### 响应的数据中有中文，那么有可能出现中文乱码

\* 以字符流输出

> response.getWriter()

//1. 指定输出到客户端的时候，这些文字使用UTF-8编码

response.setCharacterEncoding("UTF-8");

//2. 直接规定浏览器看这份数据的时候，使用什么编码来看。

response.setHeader("Content-Type", "text/html; charset=UTF-8");

response.getWriter().write("我爱黑马训练营...");

\* 以字节流输出

> response.getOutputStream()

//1. 指定浏览器看这份数据使用的码表

response.setHeader("Content-Type", "text/html;charset=UTF-8");

//2. 指定输出的中文用的码表

response.getOutputStream().write("我爱深圳黑马训练营..".getBytes("UTF-8"));

--------------------------------------------

###不管是字节流还是字符流，直接使用一行代码就可以了。

response.setContentType("text/html;charset=UTF-8");

然后在写数据即可。

###演练下载资源。

1. 直接以超链接的方式下载，不写任何代码。 也能够下载东西下来。

让tomcat的默认servlet去提供下载：<br>

<a href="download/aa.jpg">aa.jpg</a><br>

<a href="download/bb.txt">bb.txt</a><br>

<a href="download/cc.rar">cc.rar</a><br>

> 原因是tomcat里面有一个默认的Servlet -- DefaultServlet 。这个DefaultServlet 专门用于处理放在tomcat服务器上的静态资源。

