**J2EE Notes** ----Emilio

**Lesson 1**

JVM –JRE

java中的**变量名**主要遵循五个**命名规则**：

　　1、只能以**字母、“\_”或“$”**符号作为变量名开头。

　　2、变量中可包含数字，但不能以数字开头。

　　3、除了“\_”和“$”符号以外，变量中不能包含其他特殊字符。

　　4、不能用class, String, public, interface, for, int, instanceof等java**关键字**作为变量名。

　　5、在java中变量名严格区分大小写，例如：name和Name就是两个不同的变量。

**Java 关键字列表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | [assert](http://baike.baidu.com/view/653925.htm) | [boolean](http://baike.baidu.com/view/1229867.htm) | [break](http://baike.baidu.com/view/198315.htm) | [byte](http://baike.baidu.com/view/44243.htm) |
| [case](http://baike.baidu.com/view/103969.htm) | [catch](http://baike.baidu.com/view/1908899.htm) | [char](http://baike.baidu.com/view/1006519.htm) | [class](http://baike.baidu.com/view/76711.htm) | [const](http://baike.baidu.com/view/1065598.htm) |
| [continue](http://baike.baidu.com/view/1725657.htm) | [default](http://baike.baidu.com/view/1109945.htm) | [do](http://baike.baidu.com/view/126968.htm) | [double](http://baike.baidu.com/view/860124.htm) | [else](http://baike.baidu.com/view/2249891.htm) |
| [enum](http://baike.baidu.com/view/827326.htm) | [extends](http://baike.baidu.com/view/745501.htm) | [final](http://baike.baidu.com/view/2116821.htm) | [finally](http://baike.baidu.com/view/137061.htm) | [float](http://baike.baidu.com/view/797539.htm) |
| [for](http://baike.baidu.com/view/124948.htm) | [goto](http://baike.baidu.com/view/1363088.htm) | [if](http://baike.baidu.com/view/127156.htm) | [implements](http://baike.baidu.com/view/2424683.htm) | [import](http://baike.baidu.com/view/2117022.htm) |
| [instanceof](http://baike.baidu.com/view/1989052.htm) | [int](http://baike.baidu.com/view/804413.htm) | [interface](http://baike.baidu.com/view/334756.htm) | [long](http://baike.baidu.com/view/270868.htm) | [native](http://baike.baidu.com/view/1935765.htm) |
| [new](http://baike.baidu.com/view/571284.htm) | [package](http://baike.baidu.com/view/1006568.htm) | [private](http://baike.baidu.com/view/51615.htm) | [protected](http://baike.baidu.com/view/833512.htm) | [public](http://baike.baidu.com/view/1022383.htm) |
| [return](http://baike.baidu.com/view/1350512.htm) | [**strictfp**](http://baike.baidu.com/view/1866622.htm) | [short](http://baike.baidu.com/view/981206.htm) | [static](http://baike.baidu.com/view/536145.htm) | [super](http://baike.baidu.com/view/496937.htm) |
| [switch](http://baike.baidu.com/view/600161.htm) | [synchronized](http://baike.baidu.com/view/1207212.htm) | [this](http://baike.baidu.com/view/626297.htm) | [throw](http://baike.baidu.com/view/3019126.htm) | throws |
| [transient](http://baike.baidu.com/view/1698173.htm) | [try](http://baike.baidu.com/view/388893.htm) | [void](http://baike.baidu.com/view/1004734.htm) | [volatile](http://baike.baidu.com/view/608706.htm) | [while](http://baike.baidu.com/view/1455003.htm) |

**50个关键字描述列表**

abstract //抽象方法，抽象类的修饰符

assert //断言条件是否满足

boolean //布尔数据类型

break //跳出循环或者label代码段

byte //8-bit 有符号数据类型

case //switch语句的一个条件

catch //和try搭配捕捉异常信息

char //16-bit Unicode字符数据类型

class //定义类

const //未使用，保留字以便扩展

continue //不执行循环体剩余部分

default //switch语句中的默认分支

do //循环语句，循环体至少会执行一次

double //64-bit双精度浮点数

else //if条件不成立时执行的分支

enum //枚举类型

extends //表示一个类是另一个类的子类，单一继承

final //表示一个值在初始化之后就不能再改变了

//表示方法不能被重写，或者一个类不能有子类

finally //try 语句不管是否发生异常都要执行的语句块

float //32-bit单精度浮点数

for //for循环语句

goto //未使用，保留字以便以后扩展

if //条件语句

implements //表示一个类实现了接口

import //导入类

instanceof //测试一个对象是否是某个类的实例

int //32位整型数

interface //接口，一种抽象的类型，仅有方法和常量的定义

long //64位整型数

native //表示方法用非java代码实现

new //分配新的类实例

package //一系列相关类组成一个包

private //表示私有字段，或者方法等，只能从类内部访问

protected //表示字段只能通过类或者其子类访问

//子类或者在同一个包内的其他类

public //表示共有属性或者方法

return //方法返回值

short //16位数字

static //表示在类级别定义，所有实例共享的

strictfp //strict float point浮点数严格使用的规则，它所修饰的类、接口、方法所 //对应的所有代码中的float,double都是严格按照 IEEE 754 算法计算的， //不会因为硬件平台的不同而导致结果的不一致

super //表示基类

switch //选择语句

synchronized //表示同一时间只能由一个线程访问的代码块

this //表示调用当前实例或者调用另一个构造函数

throw //抛出异常（动作，直接抛出）

throws //**定义**方法可能抛出的异常（定义，不一定抛出）

transient //修饰**不要序列化**的字段

try //表示代码块要做异常处理或者和finally配合

//不管是否抛出异常都执行finally中的代码

void //标记方法不返回任何值

volatile //标记字段可能会被多个线程同时访问，而不做同步，线程共享变量

while //while循环

成员变量虚拟机加载该类时自动初始化

**引用变量** 指向某个变量 只能通过引用变量来访问对象，大小4字节(类似于指针)

Java 中只有间接访问（引用） new 一个对象，即新建一个引用（地址）指向该对象

垃圾对象 –---- 不指向任何对象，值为空

一切**对象都在堆中**（动态生成，销毁）

数组 int [] array=new int[100]; //创建100个引用变量

for(int i=0;i<100;i+=)

array[i]=new int();//每次创建一个int变量

java不允许数组越界，会抛出异常，每次对数组操作都会有越界检查

java语言区分大小写，关键字都是小写

byte 1个字节

double 默认为浮点类型 float f=1.23f 以f 作为标志，不允许高向低转换**（强制转换可以）**

模运算只与被除数的符号有关

Instanceof 关系运算符 是否是实例

位运算符号：&按位与 |按位或 ^ 按位异或

逻辑运算符：&&与（短路运算符，若前面为假，则后面不执行，程序运行快）

二元运算符结果最少是Int，不可能是byte

byte b1=4,b2=3;

b1=b1+b2;//语法错误，少于 int ，不符合加法运算的定义

b1=(byte)(b1+b2);

b1+=b2;//不会出现语法错误，隐含了一个强制类型转换，将计算结果转换成左边的类型

三元运算符：a ? b : c

移位运算：

算数移位：>> ,<< 逻辑移位 >>> ,<<< (无符号移位)

算数运算符>关系运算符>逻辑运算符

**Lesson 2**

世间万物皆对象

复杂数据类型：

Date 日期类型 8个字节 long型

System

String

封装

良好的封装，所有的成员变量都定义成私有，修改和读取都是通过本类公开函数进行

**继承**

“ A **is a** B” , A **extends** B

提高复用性，符合客观世界的关系 **单一继承** extends （不要超过7层）

所有类都是默认继承根类Object（除了8个数据类型，一切数据都是类）

元类Class (meta-class) 根类Object（root-class）

构造函数第一句一定是是super() ，不写则添加，写了则调用，而且*必须是父类含有的 构造函数*，否则会报错

多态：

Overload重载，函数名一致，形参不一致 override覆盖，派生类覆盖父类函数

苹果是水果，水果不是苹果 只允许低类型给高类型充值，不允许反过来

可以通过downcast强制类型转换，promotion 向上提升

类型检查时，动态绑定，对象只能调用他自身拥有的方法和父类的方法

只要父类中的函数被子类重载

**接口：**

两个系统之间的通信标准

**Interface** 全是抽象函数，不用写**abstract**关键字，不用写实现。

其他类要实现该接口就使用implements，一个类可以实现多个接口，逗号隔开

Abstract 修饰抽象类，修饰抽象函数，不用写实现，便于搭建框架让后续开发者实现

abstract void display();

**抽象类不一定**需要包含抽象函数，可以包含具体函数但是**接口**必须只包含抽象函数，而且抽象函数里面没有函数体

**抽象类和接口的区别：**

1. abstract class 在 Java 语言中表示的是一种继承关系，一个类**只能使用一次继承关系**。但是，一个类却**可以实现多个interface**。
2. 在abstract class 中可以有自己的数据成员，也**可以有非abstarct的成员方法**，而在interface中，只能有静态的不能被修改的数据成员（也就是必须是static final的，不过在 interface中一般不定义数据成员），所有的成员方法默认都是 public abstract 类型的，只有函数名没有函数体。
3. abstract class和interface所反映出的设计理念不同。其实abstract class表示的是"is-a"关系，interface表示的是"like-a"关系。
4. 继承抽象类和实现接口的类**必须实现其中的所有的抽象方法**。抽象类中可以有非抽象方法，这个子类可以不重写。但接口中全部是抽象方法，实现接口的类必须全部实现。
5. 接口中定义的变量默认是public static final 型，且必须给其初值，所以实现类中不能重新定义，也不能改变其值。抽象类中的变量默认是 friendly 型，其值可以在子类中重新定义，也可以重新赋值。   
     
     
   **结论**  
   　　abstract class 和 interface 是 Java语言中的两种定义抽象类的方式，它们之间有很大的相似性。但是对于它们的选择却又往往反映出对于问题领域中的概念本质的理解、对于设计意图的反映是否正确、合理，因为它们表现了概念间的不同的关系（虽然都能够实现需求的功能）。

2015.1.15 Plus: Actually, only two different points between abstract class and interface:

**1. Design concept: is-A VS has-A**

**2. Default type: no mandatory type VS public abstract methods & public static final fields**

abstract method must be implemented by sub-class

线程建立联系，构造函数使用参数传递实际的对象，该对象实现Runable接口

Ps:编码风格：驼峰标识，getter/setter 索引参数

**Lessson3**

包package 用于限定变量的命名空间

javac –d将编译后的文件指定放在某个目录下，如果有包，则按照包的结构构造目录

package 语句 是第一句 用户定义的包的名字不要以Java和javax开头，因为这是系统的包

一般一个源文件对应一个类的声明，这是较好的编程习惯

包的命名都是小写，从大往小写

源文件只能包含一个公开类，而且文件名就是公开类的类名

Java.lang 自动导入，无须显示表示

Java.util 中包含各种集合

访问权限：

默认为同包访问权限default

public（所有都可访问，只要导入了该包）

private（只能在此类体中访问）

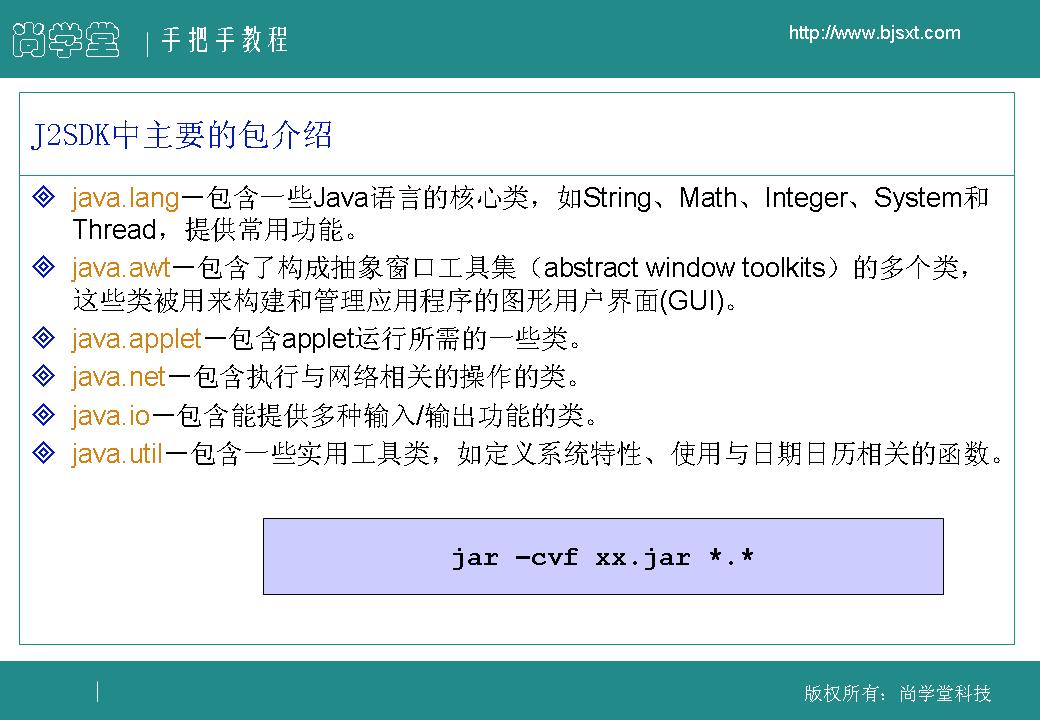
protected(不同包中的派生类也可以访问，只能通过派生类访问不能通过父类访问)

static 静态成员，由此类所拥有，不是由某个对象所拥有，它是公有的

静态函数不能出现this，super，实例函数，因为静态函数为类所拥有的，不能访问某个 特定的对象

main主函数必须是static，这样才能进入该类进行操作，而且必须是public，虚拟机才能访 问到该主函数，进入此类进行操作

public static void main（String []args）//命令行的参数传入主函数



instanceof 判断前者是否是后者（类或者派生类）的对象

抽象类不能被继承，构造函数是私有的也不能被构造（继承），全是静态

异常：

可避免异常——运行异常RuntimeException

异常类基类Throwable

构造函数中有一个Throwable（Throwable cause）由一个异常产生另一个异常，是异常 的链，产生连锁反应

Error 机器无法处理的错误，不用捕捉错误

常见异常处理手段：

extends Exception 而不是RuntimeException（运行异常是程序写的不好而造成的，可以避免）

有可能产生异常 throws Exception 调用者进行捕获或者向上抛出，直到主函数进行处理， 或者直接用虚拟机处理，并down 机

包装类：

Float，Integer，Double, Character, Boolean，Byte，Short，long

doubleValue()//取Double中的double值

valueOf(double d) //返回Double实例对象,值为d

valueOf(String s)//返回Double实例对象，值为s代表的Double值

Double.parseDouble(String s)//从字符串中抽取double类型的值

Integer.parseInt(String s)//从字符串中抽取int 基本数据类型的值

|  |  |
| --- | --- |
| static int | [**parseInt**](file:///D:\PPT\J2EE\docs\api\java\lang\Integer.html#parseInt(java.lang.String))([**String**](file:///D:\PPT\J2EE\docs\api\java\lang\String.html) s)  Parses the string argument as a signed decimal integer. |

valueOf返回一个对象

|  |  |
| --- | --- |
| static [**Double**](file:///D:\PPT\J2EE\docs\api\java\lang\Double.html) | [**valueOf**](file:///D:\PPT\J2EE\docs\api\java\lang\Double.html#valueOf(double))(double d)  Returns a Double instance representing the specified double value. |

String s=String.valueOf(double d)

|  |  |
| --- | --- |
| static [**String**](file:///D:\PPT\J2EE\docs\api\java\lang\String.html) | [**valueOf**](file:///D:\PPT\J2EE\docs\api\java\lang\String.html#valueOf(double))(double d)  Returns the string representation of the double argument. |

Integer.toString() //转换成字符串

int i=””+I;//fastest way to change int to a string representation of it

Integer 对象中的函数，使用方法String bin= Integer.toBinaryString(int i);

|  |  |
| --- | --- |
| static [String](file:///D:\PPT\J2EE\docs\api\java\lang\String.html) | [toBinaryString](file:///D:\PPT\J2EE\docs\api\java\lang\Integer.html#toBinaryString(int))(int i)  Returns a string representation of the integer argument as an unsigned integer in base 2. |

Date ,类java.util.Date

DateFormat,SimplDateFormat包 java.text.DateFormat

ANSI 汉字两个字节

UTF-8 汉字四个字节

Unicode 汉字两个字节

查找**对象相等**都是**判断类型、属性相等**，**并不是**指该对象地址一致

Object中实现了equals(Object )判断两个对象值相等

每个对象都必须实现equals函数

public boolean equals(Object o)

{

if(o instanceof Student)

{

Student s=(Student)o;

return this.age==s.age&&this.name.equals(o.name);

}

else

return false;

}

Set set =new TreeSet;

Int i=0

While(i<7)

{

Set.add(Math.random()\*35+1);

} **Object**

* + - The general contract of hashCode is:
      * Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
      * If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.
      * It is *not* required that if two objects are unequal according to the [equals(java.lang.Object)](file:///D:\PPT\J2EE\docs\api\java\lang\Object.html#equals(java.lang.Object)) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

**Lesoon4**

任何一个元素都有迭代器iterator,无序集合遍历只能通过iterator

Iterator it=set.iterator();

While(it.hasNext()){

Object element =it.next();

System.out.println(element);

}

有序集合表List->ArrayList,LinkedList

集合Set->HashSet,TreeSet(自动升序排列);HashSet

TreeSet 必须是包含可以比较大小的元素，不同类型会抛出 ClassCastException

任何对象需要比较大小必须实现Comparable接口，加入HashSet的对象必须实现compareTo

Public int compareTo(Object o){

Student s=(Student)o;

Return age-s.age ;//或者按学号排序return id.compareTo(s.id);

}

*此函数存在风险，将异常交给调用者处理*

HashSet 可以添加不同类型的

映射Map

主键唯一，相同键值，进行覆盖 **二维**

put（key,value）键值对出现，get(Object key),没找到则返回null

Set set=map.keySet();//Collection c=map.valueCollection

HashMap//接受Null值，未同步集合 Hashtable//拒绝Null值，同步集合

Iterator it=set.iterator();

While(it.hasNext()){

Object key =it.next();//获得主键

Object value=map.get(key);//以主键获取值

}

**JDK5 以后更方便的遍历方法**

1. **for** (String key : map.keySet()) {
2. System.out.println(key + " : " + map.get(key));
3. }

使用Map.Entry 这个类可以获得一条记录即： Entry <key,value>

1. **private** **void** a(){
2. Map<String, String> values = **new** HashMap();
3. **for** (Map.Entry entry : values.entrySet()) {
4. Object key = entry.getKey( );
5. Object value = entry.getValue();
6. }
7. }

泛型(尖括号中必须是对象Object)

Set<Integer> set=new TreeSet<Integer>();//使类型转换更加严谨，更安全，指定类型

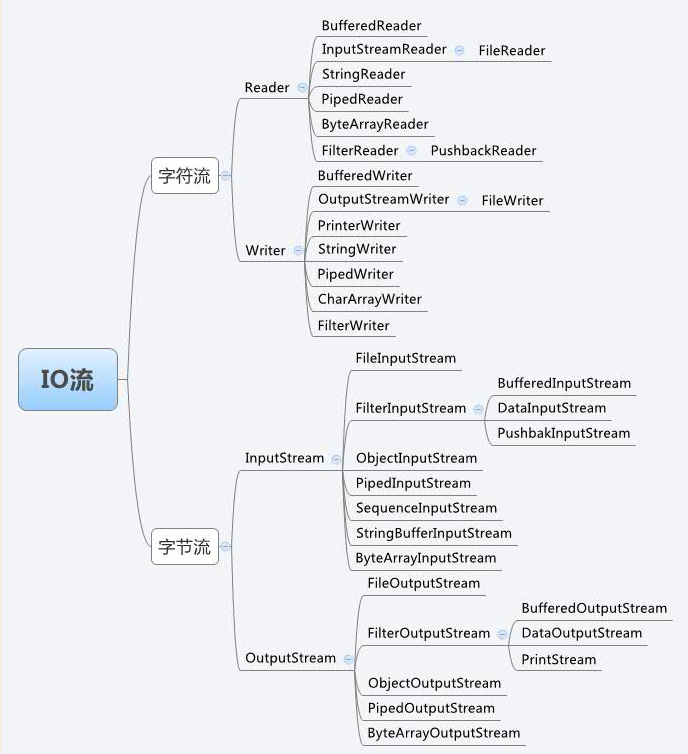
Public class A<T>{

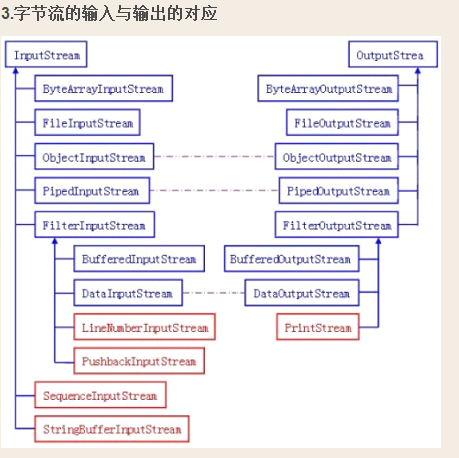
}//更加抽象地实现类

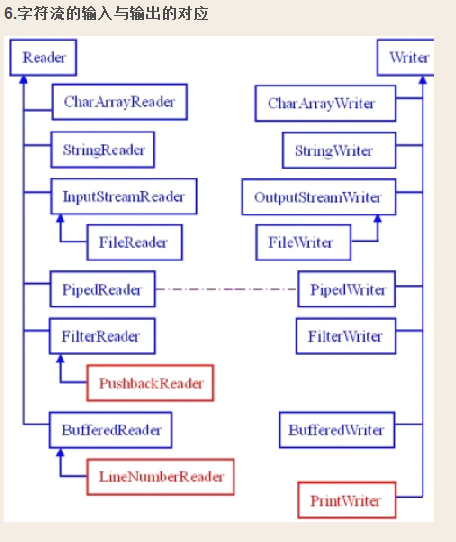
可以 A a=new A();也可以 A<Integer> a=new A<Integer>();

|  |
| --- |
| [**HashMap**](file:///D:\PPT\J2EE\docs\api\java\util\HashMap.html#HashMap())()  Constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75). |
| [**HashMap**](file:///D:\PPT\J2EE\docs\api\java\util\HashMap.html#HashMap(int))(int initialCapacity)  Constructs an empty HashMap with the specified initial capacity and the default load factor (0.75). |

I/O //java.io.\*;







7.转换流的特点：

1. 其是字符流和字节流之间的桥梁
2. 可对读取到的字节数据经过指定编码转换成字符
3. 可对读取到的字符数据经过指定编码转换成字节

何时使用转换流？

1. 当字节和字符之间有转换动作时；
2. 流操作的数据需要编码或解码时。

具体的对象体现：

1. InputStreamReader:*字节到字符的桥梁*
2. OutputStreamWriter:*字符到字节的桥梁*

这两个流对象是字符体系中的成员，它们有转换作用，本身又是字符流，所以在构造的时候需要传入字节流对象进来。

**管道PipedInputStream/PipedOutputStream类：**

当需要在两个线程中读写数据的时候，由于线程的并发执行，读写的同步问题可能会发生困难，这时候可以使用管道，管道事实上是一个队列。

管道是由系统维护的一个缓冲区，当然程序员也可以自己直接指定该缓冲区的大小（只需要设置管道流类中的PIPE\_SIZE属性的值）。当生产者生产出数据后，只需要将数据写入管道中，消费者只需要从管道中读取所需要的数据。利用管道的这种机制，可以将一个线程的输出结果直接连接到另一个线程的输入端口，实现两者之间的数据直接传送。

线程1

线程2

临时文件

管道

1．管道的连接：

方法之一是通过构造函数直接将某一个程序的输出作为另一个程序的输入，在定义对象时指明目标管道对象

PipedInputStream pInput=new PipedInputStream();

PipedOutputStream pOutput= new PipedOutputStream(pInput);

方法之二是利用双方类中的任一个成员函数 connect()相连接

PipedInputStream pInput=new PipedInputStream();

PipedOutputStream pOutput= new PipedOutputStream();

**pInput.connect(pOutput);**

2．管道的输入与输出：

输出管道对象调用write()成员函数输出数据（即向管道的输入端发送数据）；而输入管道对象调用read()成员函数可以读起数据（即从输出管道中获得数据）。这主要是借助系统所提供的缓冲机制来实现的。

实例：Java的管道的输入与输出

import java.io.\*;

public class PipedIO //程序运行后将sendFile文件的内容拷贝到receiverFile文件中

{

public static void main(String args[]){

try

{//构造读写的管道流对象

PipedInputStream pis=new PipedInputStream();

PipedOutputStream pos=new PipedOutputStream();

//实现关联

pos.connect(pis);

//构造两个线程，并且启动。

new Sender(pos,"c:\\text2.txt").start();

new Receiver(pis,"c:\\text3.txt").start();

}

catch(IOException e)

{System.out.println("Pipe Error"+ e);}}

}

//线程发送

class **Sender** extends Thread

{

PipedOutputStream pos;

File file;

//构造方法

Sender(PipedOutputStream pos, String fileName)

{

this.pos=pos;

file=new File(fileName);

}

//线程运行方法

public void run()

{

try

{

//读文件内容

FileInputStream fs=new FileInputStream(file);

int data;

while((data=fs.read())!=-1)

{//写入管道始端

**pos.write(data);**

}

**pos.close();**

}

catch(IOException e)

{System.out.println("Sender Error" +e);}}

}

//线程读

class **Receiver** extends Thread

{

PipedInputStream pis;

File file;

//构造方法

Receiver(PipedInputStream pis, String fileName)

{

this.pis=pis;

file=new File(fileName);

}

//线程运行

public void run()

{

try

{//写文件流对象

FileOutputStream fs=new FileOutputStream(file);

int data;

//从管道末端读

while((data=**pis.read()**)!=-1)

{

//写入本地文件

fs.write(data);}

**pis.close();}**

catch(IOException e)

{

System.out.println("Receiver Error" +e);}}

}

**随机文件读写：RandomAccessFile类**

它直接继承于Object类而非InputStream/OutputStream类，从而可以实现读写文件中任何位置中的数据（只需要改变文件的读写位置的指针）。

编程步骤：

① 生成流对象并且指明读写类型；

② 移动读写位置；

③ 读写文件内容；

④ 关闭文件。

另外由于RandomAccessFile类实现了DataOutput与DataInput接口，因而利用它可以读写Java中的不同类型的基本类型数据（比如采用readLong()方法读取长整数，而利用readInt()方法可以读出整数值等）。

程序实例：

利用随机数据流RandomAccessFile类来实现记录用户在键盘的输入，每执行一次，将用户的键盘输入存储在指定的UserInput.txt文件中。

import java.io.\*;

public class RandomFileRW

{

public static void main(String args[])

{

StringBuffer buf=new StringBuffer();

char ch;

try

{while( (ch=(char)System.in.read()) !='\n')

{buf.append(ch);}

//读写方式可以为"r" or "rw"

RandomAccessFile myFileStream=new RandomAccessFile("c:\\UserInput.txt","rw");

myFileStream.seek(myFileStream.length()) ;

myFileStream.writeBytes(buf.toString());

//将用户从键盘输入的内容添加到文件的尾部

myFileStream.close();}

catch(IOException e)

{}}

}

**DataInput/DataOutput接口：**

实现与机器无关的各种数据格式读写（如readChar() 、readInt()、readLong()、readFloat()，而readLine()将返回一个String）。其中RandomAccessFile类实现了该接口，具有比FileInputStream或FileOutputStream类更灵活的数据读写方式。

**File Operation**

File f=new File(String filename);//create a file object in the memory, not a file,regardless wheather it exists

相对路径是编程好习惯，便于移植“../a/b.txt”

自动备份，监督文件更改时间，只要修改了就上传long lastModified();

[javateacher@163.com](mailto:javateacher@163.com)

String[] list();//get all children’s filename

文件内容操作:

流操作——单向读写，不能两次踏入同一条河

Data flow

RandomAccessFile(File name, String operationMode)

Int abstract read();//-1 EOF, 放到整数的最低位

Int read(byte[] b);// read a lot of data, return the number of character being read

Int read(byte[],offset,length)

流链接

FileInputStream 节点流，参数是节点设备（硬盘，声卡，显卡，键盘）

BufferInputStream 处理流，参数是节点流，缓存节点流

DataInputStream 处理流，参数是节点流，**数据组装**等操作处理

dis.readInt();dis.readDouble();dis.readString()//UTF-8 coding

ZipInputStream 处理流，压缩文件

flush 清空缓存，强制输出

ObjectInputStream // stream operation of serializable object

Class student implements java.io.Serializable{}//object serialize, exchange object through byte

Object.clone()//clone 浅复制，新建一个对象，对于其内部的数据，重新建立变量，其内部 聚合的对象仅仅复制地址，不会新建其内部对象

**序列化：对象转换成有序的字节该对象的所有数据和对象都会被序列化，递归序列化 ，深 复制**

文本流：

Reader, Writer

FileReader fr=new FileReader(“ ”);

Int i=0;

While((i=fr.read())!=-1){

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

}

改进：

FileInputStream fis= new FileInputStream(“a.txt”);

InputStreamReader isr=new InputStreamReader (fis,”GBK”);//internationalize

上面功能等价于FileReader，但是实现了国际化 （台湾编码 ”BIG-5”）

BufferedReader br=new BufferedReader(isr)

String str=br.readLine();

有需要就用split分割

**Java coding method: Unicode**

PrintStream，PrintWriter

自动将数据转换成字符串，分行输出

二者都是处理流，但是前者链接在字节流，后者链接在字符流

PipedInputStream PipedOutputStream //线程安全，可用于多线程并发访问同一个文件

将输入输出流链接起来，管道操作，此类的输出作为另一个的输入pis.connect(pos)

System.in;

Scanner in=new Scanner (System.in);

***Assignments:***研究 int, double组装函数（依次读8个字节通过移位组装成源数据）

Scanner, Date，FileInputStream, DataInputStream，ObjectInputStream,split(“,”) API

**Lesson5**

**多线程编程 java.lang.Tread**

Runable 接口 负责连接普通类跟线程

Class task implements Runable{

Public void run(){

　　....

　　}

}

Thread t1=new Thread(new task());// constructor

t1.start(); //add it to ready queue

T1.stop(); //force the thread to stop, die, not recommended

T1.interrupt() //request the thread to stop a while, it’s up to the thread whether to stop

Class myThread extends Thread{

Pulic void run(){ .... }

}

MyThread().start();

start();//instance member, only

sleep(t); //static method, make the thread in execution sleep t ms , free cpu running->block

yield();//static method, willing to free cpu, running->ready

join();//wait a thread to die

resume();//as opposed to suspend(); block->ready->block

**匿名类：**

Object a=new Object(){ ... }//只能通过父类访问，没有类名，声明与实现同时出现

引用是访问类的唯一方式，而且只能访问自己的函数，类比较短，没必要再重写一个类

成员变量无顺序，内部类声明可以

内部类：

final 可以修饰类、变量

final class 此类不能被继承

final method 此方法不能被修改、重写

final=5,常量

*虚拟机退出：当前程序没有任何用户线程在执行*

多线程并发程序实现结果是不确定的，这跟操作系统的调度顺序有关

线程：用户线程user-thread 守护线程daemon-thread (后台程序)

t1.setDaemon(true); //set a thread to be run in background

线程继承父类的线程属性，父类是用户线程，则子类默认是用户线程

线程并发访问：同步

**synchronized** public char pop(){….}

每个对象隐含都有一个lock，用于控制同步并发，同步代码获得对象锁才能执行该代码，Lock 状态 free->busy

synchronized(Object o){ code block… }

synchronized(A a ){…}//任意创建一个对象，只要用它的对象锁即可

有**synchronized的地方不一定要有notify() notifyAll(),但有notify() notifyAll()的地方一定有synchronized，先使用notify后使用wait 推荐使用notifyAll()**

Unicode ‘\u0000’ 4位16进制的数 64bit 8byte

wait();//只能出现在**同步代码**中，free cpu, free lock

notify();//唤醒当前对象锁对应的等待队列中的线程，最先进入等待队列的线程

notifyAll();//唤醒所有此对象锁锁住的对象,建议使用，虽然效率较低

while(index<1){

wait(); //在进行下一步之前，判断index，可能另一个进程已经抢占

}

**Synchronize & notifyAll**

public final void notifyAll()

唤醒在此对象监视器上等待的所有线程。线程通过调用其中一个 wait 方法，在对象的监视器上等待。

直到当前的线程放弃此对象上的锁定，才能继续执行被唤醒的线程。被唤醒的线程将以常规方式与在该对象上主动同步的其他所有线程进行竞争；例如，唤醒的线程在作为锁定此对象的下一个线程方面没有可靠的特权或劣势。

此方法只应由作为此对象监视器的所有者的线程来调用。请参阅 notify 方法，了解线程能够成为监视器所有者的方法的描述。

抛出：

IllegalMonitorStateException - 如果当前的线程不是此对象监视器的所有者。

另请参见：

notify(), wait()

public static synchronized ApplicationContextUtil getInstance() {}

**synchronized 关键字**，它包括两种用法：synchronized 方法和 synchronized 块。

1. **synchronized 方法：**通过在方法声明中加入 synchronized关键字来声明 synchronized 方法。如：

public synchronized void accessVal(int newVal);

synchronized 方法控制对类成员变量的访问：每个类实例对应一把锁，每个 synchronized 方法都必须获得调用该方法的类实例的锁方能执行，否则所属线程阻塞，方法一旦执行，就独占该锁，直到从该方法返回时才将锁释放，此后被阻塞的线程方能获得 该锁，重新进入可执行状态。这种机制确保了同一时刻对于每一个类实例，其所有声明为 synchronized 的成员函数中至多只有一个处于可执行状态（因为至多只有一个能够获得该类实例对应的锁），从而有效避免了类成员变量的访问冲突（只要所有可能访问类成员变 量的方法均被声明为 synchronized）。

在 Java 中，**不光是类实例，每一个类也对应一把锁**，这样我们也可将类的静态成员函数声明为 synchronized ，以控制其对类的静态成员变量的访问。

synchronized 方法的缺陷：若将一个大的方法声明为synchronized 将会大大影响效率，典型地，若将线程类的方法 run() 声明为 synchronized ，由于在线程的整个生命期内它一直在运行，因此将导致它对本类任何 synchronized 方法的调用都永远不会成功。当然我们可以通过将访问类成员变量的代码放到专门的方法中，将其声明为 synchronized ，并在主方法中调用来解决这一问题，但是 Java 为我们提供了更好的解决办法，那就是 synchronized 块。

**2. synchronized 块：**通过 synchronized关键字来声明synchronized 块。语法如下：

synchronized(syncObject) {

//允许访问控制的代码

}

synchronized 块是这样一个代码块，其中的代码必须获得对象 syncObject （如前所述，可以是类实例或类）的锁方能执行，具体机制同前所述。由于可以针对任意代码块，且可任意指定上锁的对象，故灵活性较高。

对synchronized(this)的一些理解

一、当两个并发线程访问同一个对象object中的这个synchronized(this)同步代码块时，一个时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。

二、然而，当一个线程访问object的一个synchronized(this)同步代码块时，另一个线程仍然可以访问该object中的非synchronized(this)同步代码块。

三、尤其关键的是，当一个线程访问object的一个synchronized(this)同步代码块时，其他线程对object中所有其它synchronized(this)同步代码块的访问将被阻塞。

四、第三个例子同样适用其它同步代码块。也就是说，当一个线程访问object的一个synchronized(this)同步代码块时，它就获得了这个object的对象锁。结果，其它线程对该object对象所有同步代码部分的访问都被暂时阻塞。

五、以上规则对其它对象锁同样适用

**网络编程：**

Socket, ServerSocket

Public class Tclient{

Main(){

Socke s=new Socket(“127.0.0.1”,8080);//向服务器某端口发送连接请求

InputStream in=s.getInputStream();

OutputStream os=s.getOutputStream();

BufferedInputStream bis

DataInputStream

Dos.flush();//清空本地缓存，推送本地数据至服务器

//ObjectInputStream 可以用来替代DataInputStream，而且还可以写

}

}

ServerSocket ss=new ServerSocket(8080);//监听客户连接请求

Socket s = ss.accept();

DataInputStream dis=new DataInputStream(s.getInputStream);

Int command;

While(true){

command=dis.readInt();//先接收用户要执行的动作

switch(command)

{

case 0:

case 1:

}

}

Public void upload(File file){

dos.writUTF(“upload”);

dos.writUTF(file.getName());

dos.writUTF(file.length());

byte[] buffer

while(){

}

}

**Assignments: Thread API**

**Lesson6**

String**字符串对象只读**，字符串常量，其内容不能改变，新生成一个对象

StringBuffer 长度可变 StringBuilder //单线程使用

解决冲突：同步控制、复制数据

RPC remote protocol call

远程对象调用，就像调用本地对象一样

Event delegation model事件委托模型

事件源 getSource(); //当某个事件发生时，触发一个事件处理对象

**Lesson7**

GUI

awt同级件机制，租借机制，与操作系统的组件对象有关

swing 不同平台移植是一致的, java语言画出来的

swt IBM

javafx //sun公司最近推出的强大的GUI插件

MVC ：Model View Controller

Number Guess

Frame f=new Frame();

Button

TextField

TextArea

Label //输出较短文本

或者TextArea.setEditable(false);

Ta.setText(“..”);

**视图负责与用户进行交互，产生事件，交给Controller执行**

**视图中一定要定义一个模型类，因为这是关联数据的关键**

**控制器负责将模型中的数据映射到视图中的某个显示部件，将视图的某个动作映射到模型中某些功能的执行**

提供一个模型->设置模型值->将组件加入到view中->监听视图动作变化->交给控制器进行相应动作的执行（调用模型中的函数）

Dimension setSize(Dimension);

setBounds(x,y,width,height);

Layout: FlowLayout, BorderLayout,GridLayout, CardLayout

布局管理器自动设置各个组件的位置

流布局：按照顺序放

边界布局：BorderLayout.SOUTH

Panel 面板 （中间容器），可以放在各个地方，包含一些小组件

网格布局：new GridLayout(2,3);//2 rows ,3 columns

GridBagLayout : 行列高度可能不一样

CardLayout 卡片切换，遍历组件，显示在最前面

注册听众：

addActionListener(new ActionListener(){

Public void actionPerformed(ActionEvent ae)

Process();

}

)

**Lesson8**

Javac –d .. a.java

Javac –classpath d:\

Jar java archive file归档文件 jar cvf a.jar E.class

自动创建META-INF/MENIFEST.MF配置文件

安全、下载快（只需建立一个HTTP连接）、封装、可移植性强、压缩

-c 创建文件

-v 详细输出解压过程 verbose

-x 解压extract

-f 确定压缩的文件

-t 列出归档目录

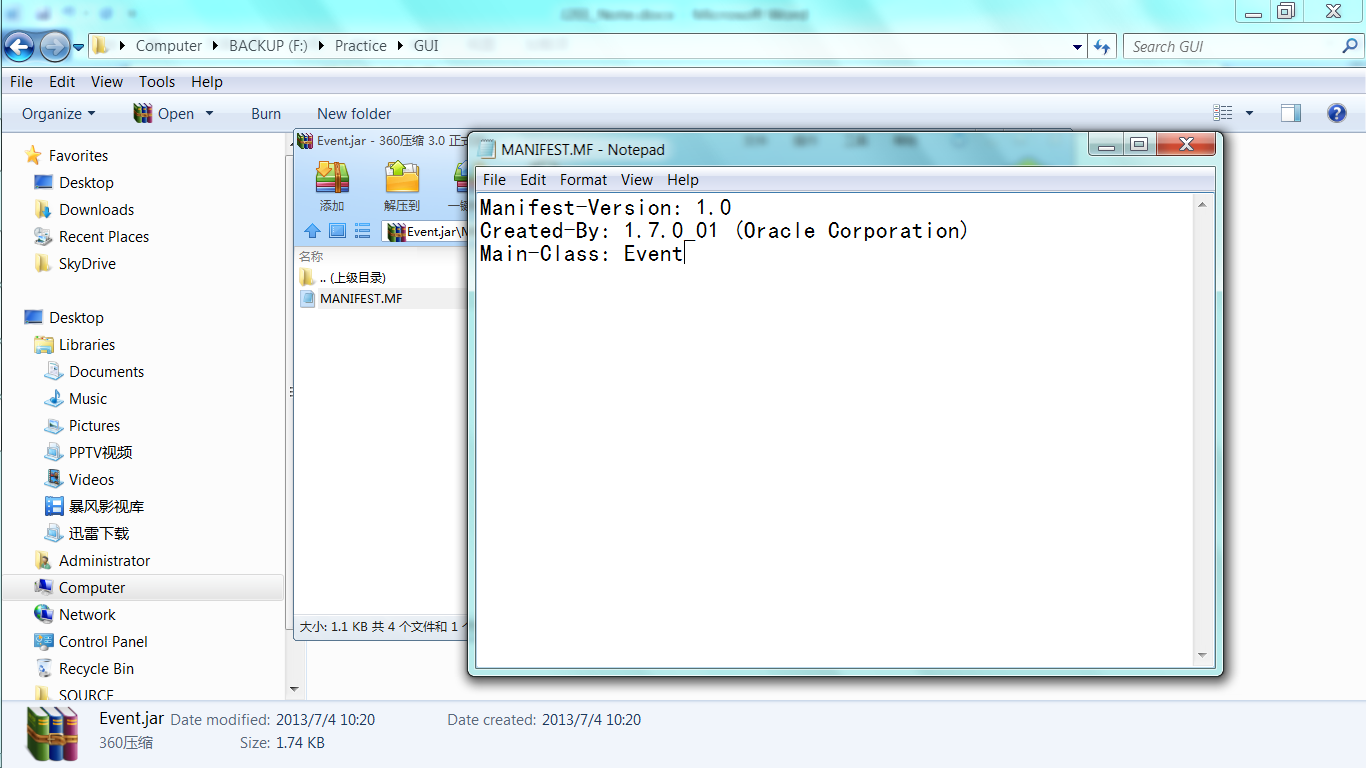
-m manifest 声明文件

Main-class: com.abac.test.Test;//制定jar包中的主类，运行jar文件就会

-u update

jar cvf a.jar test //将当前目录得到test文件夹压缩成一个包，并在输出流显示过程

jar cvfe Event.jar Event \*.class //创建Event.jar包，包含当前目录的所有class文件， 并制定Event.class为主类，即程序入口，自动生成一个META-INF目录，其中包含 MANIFEST.MF配置文件：其内容如下：



**运行该包：java –jar Event.jar 或javaw Event.jar**

如果要自定义主类，必须遵循格式： Main-Class:(空格)Event(换行)

Jar tf Event.jar > readme.txt //output Event.jar details to a file

-C （大写C）d:\java 改变根目录，因为当前目录与根目录不在一个

Java命令严格区分大小写

C:administer\> jar cvf a.jar –C D:\ java\com\\*.class;//包名映射成目录名

保存需要使用的目录，

Java –jar abc.jar; //manifest

双击文件能够直接运行的jar文件，必须含有图形界面

新建lib文件，将所有需要的jar文件复制到此文件下，方便移植

ODBC: open database connection (统一标准)

JDBC: java database connection

CRUD：create, insert, update, delete

Class.forName(“com.mysql.jdbc.Driver”);

**数据库连接池：**

DataSource:

首先在context.xml中添加：

<Resource name="jdbc/student" auth="Container"  
           type="javax.sql.DataSource"  
           username="root" password="123"  
           driverClassName="com.mysql.jdbc.Driver"  
           url="jdbc:mysql://localhost:3306/student"/>

InitialContext ctx =new InitialContext();

DataSource ds= ctx.lookup(“java:comp/env/student”);

Connection con =ds.getConnection();

JDBC:

Connection con=DriverManager.getConnection(“jdbc:mysql://localhost/student”,username，password);

Statement : 标准SQL（sequel）语句

Statement stat= conn.createStatement();

ResultSet:

ResultSet rs=stmt.executeQuery();

While(rs.next()){

String name=rs.getString(“name”);

}

rs.close();

stmt.close(); //及时关闭

conn.close();

连接池 提前建立连接，用完放回连接池

Properties p=new Properties();

driverClass=p.getProperty(“driverClass”);

username= p.getProperty(“username”);

password= p.getProperty(“password”);

properties

properties 字符串化的名值对key –value

**Assignments: properties api**

**Lesson9**

Xml extensible markup language//可扩展的标记语言

HTML5 high text markup language

Html 大小写不区分，结构不严谨

Xml meta-language //元语言，可以用它创建其他语言,区分大小写，起始与结束标记一致

与平台、语言无关，可以用来在各个平台之间通信，因为标准一样

<?xml version=”1.0” encoding=”utf-8”> //头标记，处理语句

<name color=”red” id=”007”> //属性写在尖括号中

<c>my name is Tom</c>

</name>// 嵌套，最外层元素只有一个，构成文档树（标记树）

简写空元素<c/> 只包含空格、Tab、回车都是空元素

元素的属性和名字都必须是合法的定义

wel-formated document //格式良好，物理定义：符合格式，

//逻辑定义：应用程序的逻辑 ，比如age>18

利用**包名区分重名的局部变量**，为避免命名空间冗余，可以定义一个前缀来代替该命名

<book xmlns:p1=”www.oracle.com” xmlns:p2=”www.ibm.com”>

<p1:title></p1:title>

<p2:title></p2:title>

</book>

默认命名空间<xmlns=”www.taobao.com”>

属性没有默认命名空间，要使用必须加前缀

解析器：

SAX simple api for xml 基于与流、事件方式，节省内存。调用用户程序

DOM document object model

节点对象：node –>elements

javax java extension //扩展包

enum WEATHER{SPRING,SUMMER,AUTUMN,WINTER };

//www.apache.org.com //最大的开源网站

import org.dom4j.\*;

import org.dom4j.io.\*;//dom for java

**javadoc 程序API文档自动生成**

**/\*\***

**\*@....文档注释**

**\*@author**

**\*@version**

**\*@deprecated**

**\*@param 方法的参数说明**

**\*@return**

**\*/**

javadoc -d API -windowtitile MyDocument -doctitle Event Study API -header Emilio -authorEmilio Smith \*.java

**DTD SCHEME**

Student.dtd

<!ELEMENT students(student)+>//包含至少一个student

<!ELEMENT student(id,name,gender,age)> //包含他的子元素，逗号表示顺序

<!ELEMENT id #PCDATA>//

<!ELEMENT name #PCDATA>

SCHEME

包含64种数据类型，可以自己定义数据类型

<!DOCTYPE system =“students.dtd”>//定义逻辑格式

Java lsr 2222 3 1111 2 3333 4 5555 1 last

本端口 相邻节点数 相邻端口 距离 相邻端口 距离 相邻端口 距离 最后一个标记

# [在eclipse下建立一个项目，没有web.xml的原因](http://www.cnblogs.com/newlooker/archive/2012/11/16/2772876.html)

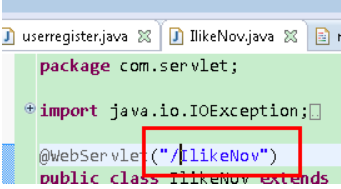
创建Dynamic web module version这个选项默认成了3.0，

按照老规范，应该是在eclipse的WebContent \ WEB-INF \ 目录下创建web.xml的

而新规范是可以不用web.xml的，如tomcat 7.0就支持新规范，

这样相关的servlet配置会以注解的形式直接写到代码里面了，比如@WebServlet。

当然如果你用老的项目导入，可以选择Dynamic web module version这个选项设成2.5，这样就自动有web.xml生成了。   
  
或者看下地址栏是不是地址不正确的。有时地址不正确也会这样。  
  
今天在做一个实训的时候，也发现打开JSP页面显示404.找了好久，才发现是一个servlet中少了一个“/”  
加上后，一切正常



**Assignment：设计模式：工厂模式 ；Xpath**

**Lesson10**

targetNamespace 指定目标命名空间

pattern 利用正则表达式定义模式

dataLoader

标签接口，空接口

Serializable， Cloneable

compareTo(Object o){

return ((Student)o.java-this.java);//强制类型转换，风险交给调用者

}

equals(Object o){

if(o instanceOf(Student))

}

Public interface DataLoaderII extends DataLoader// 版本升级，继承加覆盖

字符串trim();//去除前后空格

If(s.length()==0) continue;

String st=new StringTokenizer(s,”,”);

S1=st.nextToken().trim();

getName();//get item’s name

getText();//get item’s value

xml的注释前面加#， 使某一行失效

sax处理xml文件，记得加标记，便于遍历跟属性组织，流文件一去不复返

程序尽量不要更改，推荐通过修改配置文件来修改软件的执行行为

**通过实例化动态生成类：**

Object a=Beans.instantiate(,”com.abc.A”);//必须含有公开的无参数构造函数,未指定ClassLoder就会调用系统默认的类加载器 ClassLoader进行类的加载

将一个类加载到内存中，并且进行实例化，链接ClassLoader

Objecet a= Beans.instantiate (this.getClass().getClassLoader,a1);

Class.forName(“com.abc.A”);//类加载，再链接

### Method Detail

#### instantiate

* public static [Object](file:///D:\PPT\J2EE\docs\api\java\lang\Object.html) instantiate([ClassLoader](file:///D:\PPT\J2EE\docs\api\java\lang\ClassLoader.html) cls,
* [String](file:///D:\PPT\J2EE\docs\api\java\lang\String.html) beanName)
* throws [IOException](file:///D:\PPT\J2EE\docs\api\java\io\IOException.html),

[ClassNotFoundException](file:///D:\PPT\J2EE\docs\api\java\lang\ClassNotFoundException.html)

Instantiate a JavaBean.

Parameters:

cls - the class-loader from which we should create the bean. If this is null, then the system class-loader is used.

beanName - the name of the bean within the class-loader. For example "sun.beanbox.foobah"

Throws:

[ClassNotFoundException](file:///D:\PPT\J2EE\docs\api\java\lang\ClassNotFoundException.html) - if the class of a serialized object could not be found.

[IOException](file:///D:\PPT\J2EE\docs\api\java\io\IOException.html) - if an I/O error occurs.

#### instantiate

* public static [Object](file:///D:\PPT\J2EE\docs\api\java\lang\Object.html) instantiate([ClassLoader](file:///D:\PPT\J2EE\docs\api\java\lang\ClassLoader.html) cls,
* [String](file:///D:\PPT\J2EE\docs\api\java\lang\String.html) beanName,
* [BeanContext](file:///D:\PPT\J2EE\docs\api\java\beans\beancontext\BeanContext.html) beanContext)
* throws [IOException](file:///D:\PPT\J2EE\docs\api\java\io\IOException.html),

[ClassNotFoundException](file:///D:\PPT\J2EE\docs\api\java\lang\ClassNotFoundException.html)

Instantiate a JavaBean.

Parameters:

cls - the class-loader from which we should create the bean. If this is null, then the system class-loader is used.

beanName - the name of the bean within the class-loader. For example "sun.beanbox.foobah"

beanContext - The BeanContext in which to nest the new bean

Throws:

[ClassNotFoundException](file:///D:\PPT\J2EE\docs\api\java\lang\ClassNotFoundException.html) - if the class of a serialized object could not be found.

[IOException](file:///D:\PPT\J2EE\docs\api\java\io\IOException.html) - if an I/O error occurs.

Properties p=new Properties();

prop.load(new FileInputStream (“demo.properties”))

String pc=p.getproperty(“name”);

Player player=(Player)Beans.instantiate(getClass().geClassLoader(),pc);

Assignment: JTable, JfreeChart , java.beans.Beans

**Lesson 11**

doGet //get information from database

doPost //update database

MIME type : define type of service text/html(server tell browser request’s type)

application/x-download

PrintWriter out=response.getWriter();//get writer of response

OutputStream os=response.getOutputStream();

Javac –classpath d:\java\apache-tomcat-7.0.40\lib\servlet-api.jar ….java

Unix //stable

md helloworld // md WEB-INF // md classes ,md lib

cd WEB-INF deployment descriptor

web.xml

<web-app></>

ROOT directory, Context path=”/myProject” docBase=”d:\first\hello” //file position

request getParameter(“username”); //key-value submit and get

response.setContentType(“text/html”);

PrintWriter out =response.getWriter();

**RequestDispacher rd=request.getRequestdispacher(“/hello”);**

**rd.forward(request,response);//handle request stop, transmit this task to another page**

**rd.include(request,response);//include the process result of another page, continue processing**

**RequestDispacher.include(req,res);//动态包含html、Servlet或者Jsp，调用者决定页面的最终走向，决定response的页面头信息，include的页面不能改变这些**

**<%jsp:include file=”..”%>//最终解释就是上面的include**

jsp java servlet page //dynamic page , html associated with java

jsp->servlet->class->JVM //server process

JavaScript //client process , most programmer

HTML5 css

Java //logical process

<%@ page import=”java.util.\*, java.io.\*” contentType=”text/html; charset=gbk ”

Session=”true” errorPage=”/error.jsp” isErrorPage=”false” buffer=”1kb”

%>

<%@ include file=”a.html” %> //include a page that usually visited %>

Expression:

Num: <%=5+6 %> //code piece will be transported to service of a servlet

Declaration :

<%! Int i=0; double x=1.0; public void a() {

Synchronized(this){

I,x operation

}

}%>

One servlet , one object

JSP 指令：告诉JSP引擎对JSP页面如何编译。（无控制逻辑，无可见输出）

Instance: <%@ 指令类别 属性1=“。。”属性2=“。。”%>

**Page:**

<%@page import=”java.io.\*”, pageEncoding=”utf-8”, errorPage=”error.jsp”,isErrorPage=”false”%>

**Include: 静态包含，translation time include**

<%@ include file=”top.jsp”%>//文件只能是文本类型，实现文件的复用

**Taglib:**

JSP 动作：使用XML语法格式的标记来控制服务器的动作

< jsp:include **page**=”top.jsp”/>//当页面被请求时加载一个文件,**运行时包含，动态**

区别于指令include，

1. jsp:include **只会包含该文件的输出结果**，不会有变量重名问题
2. jsp:include **自动检测包含文件的变化**，实时更新页面内容

表单Form：

<form action=”a.jsp” method=”get”/”post”>//将表单提交到当前页面所在目录下的名为a.jsp的文件中，提交方式默认为get

解决中文乱码问题：

1、无法显示中文：<%@page contentType=”text/html;charset=gbk”%>

2、提交数据乱码: String s=request.getParameter(“s”);

s=new String(s.getBytes(“ISO-8859-1”),”gbk”);

or

request.setCharacterEncoding(“gbk”);

String s=request.getParameter(“s”);

Or

使用过滤器

404：Not Found

500: Internal Server Error

400: Bad Request

401: Unauthorized

403: Forbidden

Jsp:forward 区别于 sendRedirect 转发和重定向的区别：

forward: 同一个web应用程序内部的文件转发，实现数据的共享，效率高，浏览器地 址不变，因为是服务器去请求资源，与用户无关

sendRedirect: 重定向，相当于客户重新发送一个新的请求，无法共享数据，效率较低， 可以转向其他应用程序

**Lesson12**

多个组件需要交流信息，需要通过对request setAttribute, getAttribute，实现数据更新

通过requestDispacher派发request, response给其他页面

request.getRequestDispacher(“/judge.jsp”).forward(request,response);

一系列连续的操作，导致有意义的服务，这一次过程称为会话 Session 在此之中，Session一直存在于内存中。要支持并发访问，必须先估计Session内存的消耗

Session中部分的数据保存到数据库中完成持久化保存

HttpSession ses=request.getSession(); //if login success , session start

Ses.setAttribute()

Ses.invalidate();//session stop

Ses.setMaxInactiveInterval(int ..seconds);// free session in the memory,30min as usual

isNew();//session first been created

cookie //a handful of data transmitted between client and server to identify different session

request.getSession();//get cookie, and acquire session id query map to get session,set cookie //Time to live in the browser cookie

several request persist session data of user

cookie->session

if cookie been blocked by user, we must rewrite url and attached session id to url item

Session priority: cookie-> map->url->getParameter(“sessionId”);

HttpServletRequest

HttpSession

HttpServlet implements Servlet， ServletConfig

ServletContext任何一个组件都含有ServletConfig

ServletContext sc=getServletConfig().getContext();//函数冗余

ServletContext sc= getContext();//更加方便，因为Servlet实现了ServletConfig对象

<servlet>

<servlet-name>s1></servlet-name>

<servlet-class>a.java</servlet-class>

</servlet>

<init-parameter>

<parameter- name>name</ parameter- name >

<parameter-value>vlae</ parameter-value >

<servlet-mapping>

<ur-pattern>

www.baid.com

</ur-pattern>

</ servlet-mapping >

Init(ServletcConfig)//same as constructor

Implicit definition

HttpRequest request ,

HttpResponse response,

ServletConfig application,

ServletSession session,out,page,pageContext,exception(isErrorPage)

MIME

response.setContentType(“application/x-download”);

response.setContentType(“application-Type”,”application/x-download”);

<=>response.setHeader(“application/octed-stream”);

response.setHeader(“”);

java –d d:\abc A.java (put A.class to d:\abc)

root directory

WEB-INF 保存web工程，对浏览器屏蔽本地文件的引用地址

String file=”/WEB-INF/user.txt”

String filename=getServletContext().getRealPath(file);//get absolute address of file

最终发布会将项目置于服务器的根目录下,对用户屏蔽文件细节

<form action=”<%=request.getContextPath()%> /login.jsp” >//获得服务器的根路径，提交至根目录

requestDispacher(“/user.jsp”).forward(req,res)//获得当前工程的Web Root 路径，只有程序员知道此路径在哪儿，用户**不会**在url地址栏获得信息，这属于内部转发

如果表单要转发给某个servlet，使用相对路径时应该用xml文件中该类的url-pattern地址

在servlet**的构造函数中不要使用getServletContext() 因**为该对象还没有被构造出来故没有上下文，也就无法获得某个文件的地址了，所以要间接访问某个地址可以在service函数中再调用getServletContext()

<servlet>

<servlet-name>名</servlet-name>

<servlet-class>类路径</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>名</servlet-name>

<url-pattern>/映射位置</url-pattern>

</servlet-mapping>

注意映射位置是从你的站点根目录开始

action="映射位置" form表单提交的位置是服务器的根目录，如果需要提交给某个工程的某个servlet，**一定要在其前面加上工程名！！！** 然后才是web.xml 中url-pattern地址

**Lesson13**

Interface Filter: //预处理、后处理

public void doFilter(ServletRequest req,ServletResponse res,FilterChain){

//预处理request code

Request.setCharacterEncoding(“gbk”);

fc.doFileter(req,res);//穿过此过滤器

//后处理response code

}

public void init(FilterConfig fc){

config=fc;

}

destroy()

*过滤器用途：*

*1、安全保障：根据上下文获取注册信息，当信息不合法时，拒绝请求，同时可以根据 用户的不同身份给予不同的访问权限*

*2、性能保障：在过滤器中进行一些操作使得服务的效率提高，例如，客户端发送请求 之前将内容压缩以减少传输的时间，过滤器就可以在请求到达JSP页面之前将其解 压缩，经服务端处理之后又将响应结果压缩，发给客户端。又如，对于响应比较高 的内容可以从过滤器可以为之设置缓存，应用直接从缓存提取内容，提高应用效率。*

*3、会话处理 对会话处理集中管理，一同显示，使页面和代码分离*

*4、获得请求数据进行操作 如转换编码方式，获得请求处理时间、浏览器类型进行日志 操作。*

如果不想在servlet中修改Filter的值，可以在xml文件中修改，那么实现就要稍微麻烦一点：

<filter>

<filter-name>SetEncoding</filter-name>

<filter-class>com.file.SetEncoding</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>GBK</param-value>

</init-param>

<init-param>

<param-name>enable</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>SetEncoding</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

public void init(FilterConfig fc) throws ServletException

{

this.filterConfig=fc;

this.encoding=filterConfig.getInitParameter(“encoding”);//get encoding type

String isEnable=filterConfig.getInitParameter(“enable”);

/////assure whether we should enable this filter according to configuration//////

if(isEnable==null)//we must examinize whether this string is null

this.enable=true;

else if(isEnable.equalsIgnorCase(“true”)) //after ‘if’ there is no NullPointException

this.enable=true;

else

this.enable=false;

}

public void doFilter(ServletRequest request, ServletResponse response,

FilterChain filterChain) throws IOException, ServletException

{

if(enable==true||request.getCharacterEncoding==null)

{

String enc=this.encoidng;

if(enc!=null)

{

request.setCharacterEncoding(this.encoding);

response.setCharacterEncoding(this.encoding);

}

}

filterChain.doFilter(request,response);

}

adapter pattern //功能适配，适配器类,接口的转换，函数调用的协调

public class AAdapter {

A a;

Adapter()

Void a(string)

}

覆盖函数不允许抛出被覆盖函数未声明的**检查**异常（运行异常可以，编译器不检查）

覆盖函数的访问权限不能比被覆盖函数的权限低>=

Runtime Exception //NullPointerException ,可以避免的，通过软件编程可以

HttpClient, HttpComponent OCR//光学字符识别, TTS//文本朗读

Java.awt.robot //自动运行鼠标，键盘

Annotation 注解，类似于接口

<url-pattern>/servlet.\*</url-pattern> //限制访问的文件，需要经过过滤器的文件

ContentType=”text/html;charset=gbk” 不建议使用，因为页面都是html的

推荐使用pageEncoding=”gbk”

<jsp:useBean> <jsp:getProperty/><jsp:setProperty/>

Java bean

公开类，无参构造函数,放在包中

propertyChangeEvent

<jsp:useBean id=”a” type=”com.abc.A” scope> //id 是局部对象名且关联绑定的key，引用时如果存在a对象则使用该对象，如果不存在则新建一个A a=new A();

<jsp:getProperty name=”a” property=”num”/>

<jsp:setProperty name=”a” property=”num” value=”3”/>

<jsp:setProperty name=”a” property=”num” parm=”num”/>

//从request对象中getParameter(“num”); 然后给property 充值

<jsp:setProperty name=”a” property=”\*”/>//将用户的属性一一匹配

**EL expression language**

${1>2}

${student.age} ==${student[“age”]}==${param[“age”]}

${paramValues.hobby[0]}

Implicit object

pageContext

pageScope **Map <key,vlaue>**

requestScope , sessionScope, applicationScope

param

paramVlaues value->String[] 多个值

headerValues

init

cookie

访问集合的能力超强，根据key得到value，访问数组只能通过方括号，下标

中文乱码问题：

1. jsp 页面乱码 contenType=”text/html;charset=gbk”，windows->preference->contentType->jsp->encodingType->utf-8 update
2. html 乱码 修改浏览器的页面编码方式设置
3. url 传值乱码 更改Tomcat配置文件，connector中添加一项URLEncoding=utf-8
4. 表单提交乱码

**Username=request.getParameter(“username”);**

Username=new String(Username.getBytes(“ISO-8859-1”),”gbk”);

Or

doFilter(req,res,fc){

req.setCharacterEncoding(“gbk”);

fc.doFilter(req,res);

}

1. 文件下载乱码

response.setCharacterEncoding(“utf-8”);

response.setCharacterEncoding(“utf-8”);

String fName=request.getParameter(“file”);

fName=java.net.URLEncoder.encode(fName,”utf-8”);

String realName=new String(fName.getBytes(“ISO-8859-1”),”UTF-8”);

response.setHeader(“Content-Disposition”,”attachment;filename=realName+”.doc”)

response.setContentType(“application/x-download”);

1. 文件上传乱码

 DiskFileItemFactory factory = new DiskFileItemFactory();  
 9http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif            String path = request.getRealPath("/upload");  
10http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif//            设置文件存储位置  
11http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif            factory.setRepository(new File(path));  
12http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif//      设置大小，如果文件小于设置大小的话，放入内存中，如果大于的话则放入磁盘中  
13http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif            factory.setSizeThreshold(1024\*1024);      
15http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif            ServletFileUpload upload = new ServletFileUpload(factory);  
16http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif//            这里就是中文文件名处理的代码，其实只有一行，setHeaderEncoding就可以了  
17http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif            upload.setHeaderEncoding("utf-8");//文件名乱码解决关键

//上传文件必须是以post方式提交才会有效，而且enctype=”multi-part/form-data”

**Lesson14**

JDBC program

Class.forName(“com.mysql.jdbc.Driver”);//加载某一个类,需要引入包名

{ }//相当于无参构造函数

Static{ }//加载某一个类时已经执行，与对象无关，仅执行一次

Class.forName(“..”)//向内存中加载一个类，静态初始化，创建一个driver对象，加载到DriverManager之中

jdbc

接口访问：最少访问原则，隐藏细节

编译时不需要加载类，运行时才需要加载

占位符 ? //未知某个具体的数据，先占着，参数从1开始计数

预编译sql PreparedStatement ps=conn.prepareStatement(“selet \* from user where age>? and age<?”)//先进行sql语句语法检查，效率更高  **ps.setInt(1,90);**设置预编译第一个？（占位符）的数据值，动态设置数据

数据库数据类型到java语言的类型，最好按照类型转换规范标准

Varchar,String numeric,BigDecimal or double

sql 注入，PreparedStatement定义预定义sql语句格式 防止sql注入

jsp 之中不需要抛出异常，因为它编译之后会放在servlet中的 service 中的try {}语句块之中

**数据库连接池**，预先创建，用完立即释放,小网站连接池中15条足矣，大网站100条顶天

import javax.naming.\*;

DataSource ds=new InitialContext().lookup(“java:comp/env/student/”);

Connection conn=(DataSource)ds.getConnection();

将对服务器的访问写在配置文件中，避免应用程序的修改和编译，具有强可移植性。

名字服务器 naming

Context c;

c.bind(String,Object);//绑定名字与对象到服务器

c.rebind(“cart” ,new Cart());

Cart c=(Cart)c.lookup(“cart”);

<Context path=”…” docBase=” ..”>

<Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"

maxActive="100" maxIdle="30" maxWait="10000" username="javauser"password="javadude"

driverClassName="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost:3306/javatest"/>

</Context>

**数据池配置方法：Datasource**

[**http://hi.baidu.com/sunny\_kaka/item/e2e7f8d2dff22790260ae7ee**](http://hi.baidu.com/sunny_kaka/item/e2e7f8d2dff22790260ae7ee)

**较好：** [**http://blog.sina.com.cn/s/blog\_8417657f0100xhnf.html**](http://blog.sina.com.cn/s/blog_8417657f0100xhnf.html)

1. 配置Tomcat/conf 目录下的 server.xml

<GlobalNamingResources>

<Resource

name="jdbc/DBPool"

type="javax.sql.DataSource"

password=""

driverClassName="com.mysql.jdbc.Driver"

maxIdle="2"

maxWait="5000"

username="root"

url="jdbc:mysql://localhost:3306/student"

maxActive="3"

/>

</GlobalNamingResources>

name:指定连接池的名称

type:指定连接池的类，他负责连接池的事务处理

url:指定要连接的数据库

driverClassName:指定连接数据库使用的驱动程序

username:数据库用户名

password:数据库密码

maxWait:指定最大建立连接等待时间，如果超过此时间将接到异常

maxIdle:指定连接池中连接的最大空闲数

maxActive:指定连接池最大连接数

1. 配置WEB-INF/web.xml

<web-app>

<resource-ref>

<description>mysql数据库连接池配置</description>

<res-ref-name>jdbc/DBPool</res-ref-name>

<res-type>javax.sql.DataSource</res-type>

<res-auth>Container</res-auth>

<res-sharing-scope>Shareable</res-sharing-scope>

</resource-ref>

</web-app>

1. 配置 META-INF/context.xml

与server.xml文件所在的位置相同。

<Context>

<ResourceLink

name="jdbc/DBPool"

type="javax.sql.DataSource"

global="jdbc/DBPool"

/>

</Context>

4、测试

DataSource pool = null;

Context env = null;

Connection conn = null;

Statement st = null;

ResultSet rs = null;

try{

env = (Context)new InitialContext().lookup("java:comp/env");

//检索指定的对象，返回此上下文的一个新实例

pool = (DataSource)env.lookup("jdbc/DBPool");

//获得数据库连接池

if(pool==null){out.printl("找不到指定的连接池！");}

con = pool.getConnection();

st = con.createStatement();

rs = st.executeQuery("select \* from student");

}catch(Exception ex){out.printl(ne.toString());}

mysqld//start mysql daemon

type TestJdbc.java //显示该类的代码

java –classpath .: d：[\\first\WebContent\WEB-INF\lib\mysql-connector-5.1.20.jar](file:///\\first\WebContent\WEB-INF\lib\mysql-connector-5.1.20.jar) TestJdbc

create a user:

grant all on student.\* to ‘Emilio’ @ ’%’ identified by ‘1234’//super user has privilege of operating database student

Web 应用根目录下 getContextPath();//获得根目录

/WEB-INF(屏蔽网站信息，不暴露给用户)

/image

/css

/classes

/lib

<Context path=””//发布到当前服务器的根目录下，webapps/ ROOT,省去用户敲

docBase=”d:\\student”

/>

//可以设置主页，敲服务器域名直接进入主页

<web-app>

<listener>

<listener-class>MyAppListener</listener-class>

</listener>

application.getAttribute(“mylistener”);

import javax.servlet.annotation.\*;//不用部署,声明即可

@WebListener @WebServlet @WebFilter

Eclipse 下面没有xml文件，若需要使用自己得配置一下，进入Tomcat安装目录下面的conf\Catalina\localhost 里面，自己新建一个与当前项目同名的XML文件即可

import javax.servlet.jsp;

PageContext //一个页面的上下文 getOut,getRequest,getResponse

tag 编程

import javax.servlet.jsp.tagext.\*;

将java程序转换成标签，变成标签语言，简化了web页面的开发，便于美工设计页面

doStartTag()

doEndTag();

public void setPageContext(PageContext){}

public int doStartTag(){

pageContext.getOut().print(new Date());

}

adapter

TagSupport //适配器类，只要覆盖需要用的函数

{

protected PageContext pageContext;//子类可以访问

public int StartPage() throws JspException{

pagecontext.getOut().print();

//do EndPage return EVAL\_PAGE SKIP\_PAGE

return EVAL\_BODY\_INCLUDE; SKIP\_BODY;

}

}

1. tld //define format of tag file

taglib指令

<%@ taglib uri=”/WEB-INF/a.tld” prefix=”a”%>

组件化页面开发，全部是标记语言

File f=new File(“a.txt”)

f.createNewFile();//当且仅当此文件不存在时新建一个此文件

f.exist()//判断文件是否存在

f.mkdir();//新建一个文件夹，是一个目录不是文件

如果不指定路径，Tomcat会将用户新建的文件存储在bin目录下，如果需要将其放置在自己工程下面，可以getServletContext().getRealPath(“/WEB-INF”) （此函数会屏蔽文件分隔符，即写出”WEB-INF”也可以到每个工厂所必需的WEB-INF目录，再将文件放置在它下面即可，通过File.seperator隔开

**Lesson 15**

tag的配置xml

指定uri

<%@taglib uri=”” prefix=””%>

<tag>

<name></name> **记住：此处是name不是tag-name不要想当然地乱写**

<tag-class></tag-class>

<tei-class>myTei</tei-class>

<body-content>JSP</body-content> //或者是empty

jsp通过标记属性给标记类传值，默认值是字符串

<attribute>

**<name></name>**

<rtexprvalue>true</rtexprvalue> //属性值是否支持动态表达式的值

<required></required>

</attribut>

</tag>

标记处理器的set 函数在jsp页面会被调用

1. setPageContext(pageContext);

JspWriter out=pageContext.getOut(); try {}catch IOException

JspWriter与printWriter大体一致，只是会抛出异常

因为tagSurpport没有声明IOException 所以只能用try{}catch捕捉该异常

子类不能抛出父类没有声明的检查异常

release(){属性值复原}//将tag放回tag池之中，以便以后使用

BodyTagSupport

bodyContent类似于pageContext,隐含在BodyTag，protected变量

doStartTag

return EVAL\_BODY\_BUFFERED; //创建一个BodyContent

//处理体内容必须要让doStartTag返回此值才能不略过体内容

getPreviousOut().println(bodyContent.getString());//否则只能输出到bodyContent

return AT\_BEGIN /NESTED/AT\_END

pageContext.setAttribute(“d”,new Date());

**自定义一个脚本变量 实现tld向jsp传值**

public class MyTei extends TagExtraInfo{

public VariableInfo[] getVariableInfo(TagData td){

return VariableInfo[]

{

new VariableInfo(td.getAttributeString(“varName”), td.getAttributeString(“varName”))

}

}

}

doStartTag()

pageContext. java.beans.Bean.instantiate()

**Assignment: Class , Beans,Method 类的API**

**//java.beans.Beans java.lang.reflect.Method**

**分页显示->复用**

**Lesson16**

simpleTag

void doTag() throws IOException

setJspContext(JspContext) //传入pageContext，处理标记

setJspBody(JspFragment)//处理体内容的jsp片段

取消了数据池，

加入缓存，可以对大容量数据处理加速 **通用缓存技术**

[**www.apache.org**](http://www.apache.org) **开源项目,选择感兴趣的深究**

simpleTagSupport

JspContext 高度抽象的对象，高于pageContext

void doTag(){

PageContext pc=(PageContext)getJspContext();

pc.getSession().getAttribute(“dd”);

JspFragment jf=getJspBody();

jf.invoke(null);//输出到jsp out 对象中

StringWriter sw=new StringWriter();//缓冲区

jf.invoke(sw);//将缓冲区的

String s=sw.toString();//获得writer中的数据

pc.getOut().println(s);//输出它

}

<[%@taglib uri=”/WEB-INF/A.tld” prefix=”s”%](mailto:%25@taglib%20uri=)>//自己写tag标记的时候，可以这么写

<[%@taglib uri=”http://www.abc.com/my/mytags” prefix=”s”%](mailto:%25@taglib%20uri=)>//**打包时，为了唯一定位,借鉴url**

使用标记语言时不用考虑类型转换，Jsp页面自动转换类型

减少空指针的异常，多进行可能出现空指针的判断

getJspContext().getOut().write(s);//可以循环使用

**tld文件打包的时候放META-INF目录下**

**displaytag.jar //开源的tag包，来自第三方的组件库，拿来就用**

**实现jsp的第三种复用——标记处理器**

**menu.jsp ->menu.tag /WEB-INF/tags**

**通过包名、前缀来区分重名的不同组件**

**<%@taglib tagdir=”/WEB-INF/tags/com/tags” prefix=”a”%>**

**<%@attribute name=”menuItems” required=”true”%>**

**jsp->tag 通过attribute传递 tag->jsp tei文件传递信息**

**<%@ variable name-given=”d” type=”dd” %>//脚本变量，不建议用**

**建议使用request,response,session实现数据的共享**

**Lesson17**

**可靠、异步通信 分布式计算 不同应用程序之间通信**

**javax.jms**

**进程之间通信。**

**jms Java Message Service**

**异步的，可靠的**

**一个进程将消息给消息服务器，然后另一个进程从消息服务器取消息。**

**消息服务器（tomcat不支持，可以下载 jboss(j2ee服务器)，glassfish,weblogic） 高吞吐量**

**消息传递的方式：**

**一：**

**消息队列： 一对一，一个进程取了消息后，消息生命周期结束**

**二：**

**主题模式： 一对多，多个进程都得到了消息后，消息生命周期结束**

**message-oriented middleware(MOM)**

**QueueSession createQueueSession(boolean transacted, int acknowledgeMode)**

**transaction 交易，事物**

**ACID Atomic原子性 Consistency隔离性 Integrity一致性（不应该有中间状态，只能是成功或者失败 commit rollback） Durability持久性**

**acknowledgeMode**

**AUTO\_ACKNOWLEDGE**

**CLIENT\_ACKNOLEDGE**

**DUPS\_OK\_ACKNOLEDGE**

**java message service： 配置步骤**

**1、创建一个连接工厂**

**2、为消息定义存储方式（可选，主要为了消息永久存储）**

**3、创建JMS服务器，此服务器是队列和主题的容器，并把这些队列和主题映射到存储当中**

**4、创建队列或者主题，映射到JMS服务器**

**任何一个事物都有ACID 属性（automatic（原子性），隔离性、移植、持久性）**

**commit（执行） rollback（事物回滚）**

**javax.jms**

createQueueSession(boolean transctedd int acknowlegeMode)

//创建一个会话队列，是否加入事物，确认模式是什么

**j2ee server include web,message,naming service**

PTP mode

QueueSession createReceiver

QueueSender

QueueReceiver

Context //abastract production

Hashtable ht

ht.put(Context.INITIAL)

Context ctx=new InitialContext(ht);//代理模式 proxy

MessageListener

onMessage(Message m)//Listener{

//接受消息

}

r.setMessageListener

Java Mail

MIME multipurpose internet mail extension

base64编码

class MimeMessage

receiverType： To CC BCC

smtp pop3//服务器，需要用户名密码

大部分邮件服务器都使用http->WebService->邮件服务器 （）

JAMES //启动服务其，配置邮件服务器，就可以搭建邮件服务器

Properties prop=new Properties（）；

prop.setProperty（）；

步骤： 创建属性文件；设定服务器、用户、密码、是否验证；创建MimeMessage；设置接受者、标题、日期； 发送（transport.send()）

pop3 有发送限制

Authenticator、Store、Folder、FetchProfile、web开源系统

**Lesson17**

前端控制器url-pattern \*.do 完成预处理、业务逻辑控制器

将用户动作映射到对应功能的执行

DAO->service

加载struts程序

myForm extends ActionForm

验证的实现全部封装在validate()中

reset

软件设计服务于实际，来源于实际，不要约束实际

myAction extends Action

前端控制器处理request,将request、response包装在form bean中传至后端控制器

前端控制器可以将String类型自动转换成bean中的类型，这样做减少了后端的压力，但是就减少了验证，数据类型换成String验证放在后端，可以对验证进行人性化操作

mapping.findForwrd(“result”);

taglib tags-logic tags-html tags-bean

<html:form action=”/calculate”>

<html:text property=”num1”>

<html:text property=”num2”>

<html:submit property=”submit” value=”add”>

configuration

<form name=”..” type=”com.abc.calcu”>

<action-mappings>

<action path=”/calc” type=”com.abc.calc” name=”calcForm” input=”/input.jsp”>

<forward name=”result” path=”/result”>

</action-mappings>

<bean:write name=”calcForm” property=””>

DynaActionForm

get(String propertyName)

set(String name,Object value)

<property name=”num1” type=String/>

写一个Action,

将所有标签都写在配置文件properties中，支持国际化

resource buddle// 读取本地操作系统的语言类型

将功能相近的一组动作写在action中，减少action的数量

耦合

struts2

业务逻辑不访问前端，只依赖于模型

**assignment：动态代理、反射机制**

**Servlet 路径匹配**

**一、Servlet容器对url的匹配过程：**

当一个请求发送到Servlet容器的时候，容器先会将请求的url减去当前应用上下文的路径作为Servlet的映射url，比如我访问的是 http://localhost/test/aaa.html，我的应用上下文是test，容器会将http://localhost/test去掉， 剩下的/aaa.html部分拿来做Servlet的映射匹配。这个映射匹配过程是有顺序的，而且当有一个Servlet匹配成功以后，就不会去理会剩下 的Servlet了（filter不同，后文会提到）。其匹配规则和顺序如下：

1. **精确路径匹配**。例子：比如ServletA 的url-pattern为 /test，ServletB的url-pattern为 /\* ，这个时候，如果我访问的url为http://localhost/test ，这个时候容器就会先进行精确路径匹配，发现/test正好被ServletA精确匹配，那么就去调用ServletA，也不会去理会其他的 Servlet了。

2. **最长路径匹配**。例子：ServletA的url-pattern为/test/\*，而ServletB的url-pattern为/test/a/\*，此 时访问http://localhost/test/a时，容器会选择路径最长的Servlet来匹配，也就是这里的ServletB.

3. **扩展匹配**，如果url最后一段包含扩展，容器将会根据扩展选择合适的Servlet.例子：ServletA的url-pattern：\*.action

4. 如果前面三条规则都没有找到一个Servlet，容器会根据url选择对应的请求资源。如果应用定义了一个**default Servlet**，则容器会将请求丢给default Servlet（什么是default Servlet？后面会讲）。

根据这个规则表，就能很清楚的知道Servlet的匹配过程，所以定义Servlet的时候也要考虑url-pattern的写法，以免出错。

对于filter，不会像Servlet那样只匹配一个Servlet，因为filter的集合是一个链，所以只会有处理的顺序不同，而不会出现只选择一个filter.Filter的处理顺序和filter-mapping在web.xml中定义的顺序相同。

**二、url-pattern详解**

在web.xml文件中，以下语法用于定义映射：

以“/‘开头和以”/\*“结尾的是用来做路径映射的。

以前缀“\*.”开头的是用来做扩展映射的。

“是用来定义default Servlet映射的。

剩下的都是用来定义详细映射的。比如： /aa/bb/cc.action

所以，为什么定义“/\*.action”这样一个看起来很正常的匹配会错？因为这个匹配即属于路径映射，也属于扩展映射，导致Servlet容器无法判断。

**Eclipse中用Tomcat发布的Web项目，更改其部署路径**

**D:\eclipse\workspace**

**在Eclipse中配置完Tomcat后，发布到的路径是：**

**D:\eclipse\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp1\wtpwebapps**

**这个路径太深了，所以我想使用Tomcat的默认路径，如何在Eclipse中进行修改呢？**

**1、找到Server面板，右击当前的那个Tomcat，先remove掉其中所有的工程。**

**2、再右击那个Tomcat，选择Clean，清空一下。**

**3、双击那个Tomcat，会打开属性面板，找到左边第二个Server Locations。**

**选择第二个会在Tomcat路径下创建一个新的wtpwebapps文件，存放Tomcat发布的web项目；**

**选择第三个，你可以在Server Path中输入你想要的路径，保存即可。**

## Address alreadly in use :JVM\_Bind:8080

**原因：8080端口被占用**

**Solutions:**

**1,独立运行的Tomcat没有关闭.自安装的tomcat程序设置开机自动运行,或者在之前运行过,先关闭myecplipse或jbuilder,在任务管理器中找到Tomcat的进程,将其kill掉(javaw.EXE 内存大的是EyEclipse，小的是tomcat ，关闭小的)即可.有时候Tomcat非法关闭时,在进程中,仍然存在,仍然占用8080端口.所以只要将其进程杀掉.就可以解决.**

**2,安装了其他的软件占用了Tomcat的端口.用可以查看端口的工具查看一下,或者,直接改掉tomcat的默认端口.即可**

**3. 在service.XML中直接该端口**

Unexpected RuntimeException:

Last cause: Class org.apache.wicket.session.DefaultPageFactory can not access a member of class wicket.helloWorld.HelloWorldPage with modifiers "public"  
WicketMessage: Can't instantiate page using constructor 'public wicket.helloWorld.HelloWorldPage()'. An exception has been thrown during construction!

**原因：直接定义某类class xxx**

**解决：改成public class xxx**

**Wicket学习笔记：**

1. **引用wicket:id 的html文件必须和处理该标记的Java文件名称一模一样，必须放置在同一个目录下，<span>…</span>标记中间的内容是不会显示的，所以它可以作为一个注释。**
2. **各个需要的包都要引入到，宁可多也不能少，否则会出乱七八糟的错误很BT**
3. **Html文件必须引用定义了的wicket标记，格式是严格遵守XHTML的规范的，不规范很容易出错；控制页面的java文件必须继承WebPage；application文件必须继承WebApplication ，必须覆盖getHomePage(),返回一个类；XML文件中要配置<servlet><servlet-mapping> or <filter> <filter-mapping>， <servlet-class>必须是wicket定义好的某一个父类**
4. **Wicket只是对每个控件植入一些程序员的操作，具体的类型还是直接通过html语法来定义**

**使用框架时引用Library（包含所需jar包，不报错）但运行时找不到类但是复制到WEB-INF/lib才可以的原因：**

eclipse工程下的library是用来编译里面的src中java文件的  
实际发布到tomcat时,仅仅只复制了WEB-INF/lib里面的jar包,所以出现eclipse可以正常编译但tomcat运行是找不到类,  
如果你是用的eclipse j2ee IDE,你去WorkSpace\.metadata\.plugins\\*\*.server.core\wtpwebapps找到运行时发布的工程就可以明白了

首先，比如你现在用TOMCAT做服务器，那么需要依赖的JAR包就应该考到WEB-INF/lib中，TOMCAT会自动把它编译然后放入WEB- INF/classes目录（当然你src的代码编译以后也会放到这里）。就应该这么做，这是规定（甲鱼的屁股），你用这个server就应该服从其规 定，不然就去看看TOMCAT如何实现的，再修改原码吧。  
其次，如果将依赖的jar包考到WEB-INF/LIB下，又把这个JAR包的源代码考到SRC下，你觉得会怎么样？先编译谁？谁有效谁无效？你去试试（去看看类路径加载的顺序就知道了）。如果你想修改框架的源代码，你就应该这样做

**终极解决方案：** <http://www.360doc.com/content/13/0912/12/13861594_313931829.shtml>

wicket项目实战体会：

1. 数据库连接用连接池；可以用多表查询来简化数据库操作；用完记得在finally块中关闭连接。
2. hibernate可以简化很多对多属性实体的操作

**resultset的游标的起始位置是第一行前面，如果在这之前没有next（）一下，就会出现如下异常：**

Exception in thread "main" java.sql.SQLException: Before start of result set  at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:1055)

严重: IOException while loading persisted sessions: java.io.EOFException  
严重: Exception loading sessions from persistent storage  
  
原因是tomcat对硬盘的session读取失败，彻底解决办法一下：将work下面的文件清空，主要是\*.ser文件，或者只是删除掉session.ser即可以解决。

主类型   大小   最小值   最大值   封装器类型  
boolean   1   位   -   -   Boolean  
char   16   位   Unicode   0   Unicode   2   的16   次方-1   Character  
byte   8   位   -128   +127   Byte（注释①）  
short   16   位   -2   的15   次方   +2   的15   次方-1   Short（注释①）  
int   32   位   -2   的31   次方   +2   的31   次方-1   Integer  
long   64   位   -2   的63   次方   +2   的63   次方-1   Long  
float   32   位   IEEE754   IEEE754   Float  
double   64   位   IEEE754   IEEE754   Double

# [关于 MySQL 的 boolean 和 tinyint(1)](http://www.cnblogs.com/xiaochaohuashengmi/archive/2011/08/25/2153011.html)

boolean类型  
MYSQL保存BOOLEAN值时用1代表TRUE,0代表FALSE，boolean在MySQL里的类型为tinyint(1),  
MySQL里有四个常量：true,false,TRUE,FALSE,它们分别代表1,0,1,0，  
mysql> select true,false,TRUE,FALSE;  
+------+-------+------+-------+  
| TRUE | FALSE | TRUE | FALSE |  
+------+-------+------+-------+  
|    1 |     0 |    1 |     0 |  
+------+-------+------+-------+  
可以如下插入boolean值：insert into [xxxx(xx)] values(true)，当然也可以values(1);  
举例如下：  
mysql> alter table test add isOk boolean;  
Query OK  
mysql> desc test;  
+-------+-------------+------+-----+---------+----------------+  
| Field | Type        | Null | Key | Default | Extra          |  
+-------+-------------+------+-----+---------+----------------+  
| id    | int(11)     | NO   | PRI | NULL    | auto\_increment |  
| isOk  | tinyint(1)  | YES  |     | NULL    |                |  
+-------+-------------+------+-----+---------+----------------+  
mysql> insert into test(isOk) values(true);  
Query OK  
mysql> select isOk from test ;  
+------+  
| isOk |  
+------+  
|    1 |  
+------+  
=================

 MySQL没有boolean类型。这也是比较奇怪的现象。例：

|  |
| --- |
| create table xs (    id int primary key,    bl boolean ) |

这样是可以创建成功，但查看一下建表后的语句，就会发现，mysql把它替换成tinyint(1)。也就是说**mysql把boolean=tinyInt了**，但POJO类要定义成什么类型呢？  
因 为惯性思维，在java类中也把它定义成type。然后在Struts中使用<s:check/>标签。这就产生一个严重的问题 了。<s:check>是boolean，而POJO去定义成byte。这样数据永远也无法提交，被struts的intercept拦截掉 了。**解决办法是在POJO类中定义成boolean，在mysql中定义成tinyint(1)。**

------

 TINYINT(1) or ENUM( 'true' , 'false')

-------

 设计模式：

工厂模式，批量生产某个类；

建造者模式，封装构建某个复杂的类

spring=bean container 类特别多，记忆和搜索较难，用配置文件来管理这些类

ApplicationContext extends Beans getBean();

Method m = a.getClass().getDeclaredMethod();

m.invoke();

**java classes of tool**

1. **MD5加密**

**使用java.security.MessageDigest 包对信息进行MD5加密**

**MessageDigest md= MessageDigest.getInstance(“MD5”);**

**byte[] result=md.digest(tempString.getBytes()); //tempString is the string to be encrypted**

**//normally we’re going to change byte array to a string type for easily saving**

**//when we verify the password, just need to judge (password==encrypFunction(tempString))**

**properties file record // user .home, user.dir //define the file where we save the program**

**ExtJs(JS framework)**

处理中文乱码代码实现部分：

jsp 页面：

<%@ page language = "java" pageEncoding = "UTF-8" %>

<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >

< html >

< head >

< script type = "text/javascript" >

// 开启网页对话脚本

function openDialog(url,width,height)

{

var property= "status:no;center:yes;resizable:yes;scroll:yes;dialogWidth:" +width+ "px;dialogHeight:" +height+ "px;" ;

var ret=window.showModalDialog(url, 'modalDialogwin' ,property);

if (ret == null || ret == "" )

{return false ; }

return true ;

}

// 查看待办工作脚本

function showork(url)

{

// 将 url 地址进行编码

url=encodeURI(url);

//alert(url);

// 调用 openDialog 方法开启网页对话框

openDialog(url,900,500)

}

</ script >

< title > 待办工作查看 </ title >

</ head >

< body >

<!-- 部分代码略 -->

< table border = "1" align = "center" >

< tr >

< td >

< a href = "#" onclick = "javascript:showork('leavesp?work= 部门主管审批 ') "> 查看待办工作 </ a >

</ td >

</ tr >

</ table >

</ body >

</ html >

页面部分就一个超连接，用来开启一个网页对话框，只是开启的这个对话框中请求的不是一个物理的页面，而是请求的一个 servlet （ leavesp ），而且 url 中带有中文参数值（ ?work= 部门主管审批 ）。

为了处理中文乱码部分能够在整个 WEB 系统中都能够使用到，所以将中文乱码处理写成了一个 Filter( 过滤器 ) ，并在 web.xml 中配置所有的请求都将经过这个 Filter 进行过滤。 Filter 部分的代码如下：

文件名： ProFilter.java

**public class** ProFilter **implements** Filter

{

**protected** FilterConfig filterConfig ;

/\*\*

\* 初始化

\*/

**public void** init(FilterConfig filterConfig) **throws** ServletException

{

**this** . filterConfig =filterConfig;

}

/\*\*

\* 将 inStr 转为 UTF - 8 的编码形式

\* **@param** inStr 输入字符串

\* **@return** UTF - 8 的编码形式的字符串

\* **@throws** UnsupportedEncodingException

\*/

**private** String toUTF(String inStr) **throws** UnsupportedEncodingException

{

String outStr = "" ;

**if** (inStr != **null** )

{

//outStr=java.net.URLDecoder.decode(inStr);

// 不用 decode 了 , 到这的时候就已经自动 decode 过了

// 将字符串转为 UTF-8 编码形式

outStr = **new** String(inStr.getBytes( "iso-8859-1" ), "UTF-8" );

}

**return** outStr;

}

/\*\*

\* 中文乱码过滤处理

\*/

**public void** doFilter(ServletRequest svlrequest, ServletResponse svlresponse,

FilterChain chain) **throws** IOException, ServletException

{

// 将 Servlet 请求与响应对象转换成 HttpServlet 请求与响应对象

HttpServletRequest request=(HttpServletRequest)svlrequest;

HttpServletResponse response=(HttpServletResponse)svlresponse;

// 获得请求的方式 (1.post or 2.get), 根据不同请求方式进行不同处理

String method = request.getMethod();

//1. 以 post 方式提交的请求 , 直接设置编码为 UTF-8

**if** (method.equalsIgnoreCase( "post" ))

{

**try**

{

request.setCharacterEncoding( "UTF-8" );

} **catch** (UnsupportedEncodingException e)

{

e.printStackTrace();

}

}

//2. 以 get 方式提交的请求

**else**

{

// 取出客户提交的参数集

Enumeration<String> paramNames = request.getParameterNames();

// 遍历参数集取出每个参数的名称及值

**while** (paramNames.hasMoreElements())

{

String name = paramNames.nextElement(); // 取出参数名称

String values[] = request.getParameterValues(name); // 根据参数名称取出其值

// 如果参数值集不为空

**if** (values != **null** )

{

// 如果参数值集中只有一个值

**if** (values. length == 1)

{

**try**

{

// 调用 toUTF(values[0]) 函数 ,(values[0] 即第一个参数值 ) 方法转换参数值的字元编码

String vlustr=toUTF(values[0]);

// 并将该值以属性的形式藏在 request

request.setAttribute(name, vlustr);

} **catch** (UnsupportedEncodingException e)

{

e.printStackTrace();

}

}

// 如果参数值集中有多个值

**else**

{

// 遍历参数值集

**for** ( **int** i=0;i<values. length ;i++)

{

**try**

{

// 回去依次将每个值调用 toUTF(values[i]) 方法转换参数值的字元编码

String vlustr=toUTF(values[i]);

values[i] = vlustr;

} **catch** (UnsupportedEncodingException e)

{

e.printStackTrace();

}

}

// 将该值以属性的形式藏在 request

request.setAttribute(name, values);

}

}

}

}

// 设置响应方式和支持中文的字元集

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

// 继续执行下一个 filter, 无一下个 filter 则执行请求

chain.doFilter(request, response);

}

/\*\*

\* 销毁方法

\*/

**public void** destroy()

{

}

}

过滤器部分，可用来处理页面提交的 post 和 get 方法产生的中文乱码问题。

Post 方式提交的数据（ form 表单中提交的数据）只需要进行请求字符编码设置 request.setCharacterEncoding( "UTF-8" ); 和响应设置 response.setContentType( "text/html;charset=UTF-8" ); 设置就可以了。

Get 方式提交的数据会包含在 url 中，（如： leavesp?work= 部门主管审批），则需要取出参数名： Enumeration<String> paramNames = request.getParameterNames(); （如： work ）和取出参数值： String values[] = request.getParameterValues(name);( 如：部门主管审批 ), 然后再将参数值进行编码转换： outStr = **new** String(inStr.getBytes( "iso-8859-1" ), "UTF-8" );

过滤器写好这后就需要到 web.xml 中 < web-app > 与 </ web-app > 之间进行配置了， web.xml 中配置如下：

<!-- 配置过滤器 -->

< filter >

< description > 处理中文乱码过滤器 </ description >

< filter-name > ProFilter </ filter-name >

< filter-class > com.util.filters.ProFilter </ filter-class >

</ filter >

< filter-mapping >

< filter-name > ProFilter </ filter-name >

< url-pattern > \* </ url-pattern ><!-- 过滤所有请求，注： tomcat5 请用 /\* -->

</ filter-mapping >

这样配置之后只要页面有任何请求都会通过 ProFilter 进行中文处理了，就不会再发生中文乱码问题了。

那一串通过 get 传递过来的中文处理过程如下：

1.        在 jsp 页面的 JavaScript 里面进行编码处理：

url=**encodeURI**(‘ leavesp?work= 部门主管审批 ’)

编码后结果： url= leavesp?work=%E9%83%A8%E9%97%A8%E4%B8%BB%E7%AE%A1%E5%AE%A1%E6%89%B9

2.        因为在 web.xml 过滤器配置为 < url-pattern > \* </ url-pattern > 所以所有请求将会通过过滤器进行处理，此过滤器处理过程如下：

2.1 获得参数名集合：

Enumeration<String> paramNames = request.getParameterNames();

String name = paramNames.nextElement();

2.1 结果中有 name=’work’ ；一值。

2.2 通过参数名获得参数值：

String values[] = request.**getParameterValues**(name);

因 work 的值只有一个所以 2.2 的结果为： values[0]=’ é ¨é¨ä¸»ç®¡å®¡æ¹’;( 此处得到的是字符串：“部门主管审批”的 iso-8859-1 编码格式字符 )

2.3 获得字符串的 UTF-8 编码格式字符：

outStr = **new** String(inStr.getBytes( "iso-8859-1" ), "UTF-8" );

2.3 中获得的结果为 outStr= “部门主管审批” .

至此中文已经正常了。

不过过滤器还没处理结束，还得将参数名对应的值 ( 已经处理的值 ) 再重新藏到请求中去。

2.4 将参数名对应结果藏入请求中：

request.setAttribute(name, outStr );

通过 2.4 处理后，请求中的 work 的值就变成了：部门主管审批

2.5 继续执行下一个 filter, 无一下个 filter 则执行请求

chain.**doFilter**(request, response);

3.        通过过滤器之后，就可以进入请求中对应 leavesp 的 servlet 了（ servlet 就不做说明了），此时在 servlet 中通过 String work=request.getParameter(“work”); 就可以获得 url 中传递过来的中文参数值了，结果为 work=” 部门主管审批 ” 。