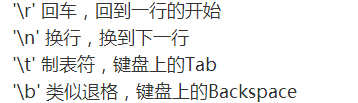
# 程序语言基础

## 变量

float f = 133.66f //float数值类型后要加上f结尾，否则默认为double型



char cout="a"+1 //b

char cout="a"+2 //c 参照的是Unicode表 移动的是位置

final int price;//声明常量 值固定不会被覆盖 只能赋值一次

int price;//声明变量 值可以被覆盖

int a=(int)(b+c) //bc为float型 需强制转换

float d=b+a //有一个为float型 就不需要转换

## 运算

三目运算符：

int sex=1;

String sexLabel=(sex==0?"男":"女"); //结果为女

Boolean b1= A & B //运算过程中AB都执行

Boolean b2= A && B //运算过程中先执行A 若A false则不执行B

## 数组

int[] nums={12,13,12,16,76,222};

Arrays.sort(nums);

System.out.println(Arrays.toString(nums));

//结果为[12, 12, 13, 16, 76, 222]

int[] nums={12,13,16,66,76,200};

int idx = Arrays.binarySearch(nums, 16);

System.out.println(idx);

//结果为2

String[] names = {"zhangsan","david","lisi"};

Arrays.*sort*(names);

System.***out***.println(Arrays.*toString*(names));

//结果为[david, lisi, zhangsan]

## 字符串

String str1="hello";//存放在常量池里面

String str2="hello";

String str3=new String("hello");//new出来的都是新对象 生成不同的内存地址 存放在堆里面

String str4=new String("hello");

System.out.println(str1.equals(str2));//比较的是内容 true

System.out.println(str1==str2);//比较的是内存地址 true

System.out.println(str1==str3);//false

System.out.println(str3==str4);//false

String str="helloworld";//起始位为0

str=str.substring(0,5);//截取字符串 前闭后开 结果为hello

int index=str.indexOf("o");//搜索子字符串所在位置 找不到返回-1

String str="abc"

str=str.concat("def");//拼接

str=str.compareTo(other);//比较大小

str=str.replacefirst(old,new);//替换第一个

str=str.replaceAll(old,new);//替换全部

str.startsWith("")//以..开头 找到返回true，找不到返回false

str.endsWith("")//以..结尾

String s="Abc";

System.***out***.println(s.toLowerCase());//转化为小写

System.***out***.println(s.toUpperCase());//转化为大写

String stat = "We tend to think of a bout of good";

String[] stats=stat.split(" ");

System.***out***.println(stats.length);//输出为9

System.***out***.println(String.*format*("%02d:%03d:%04d", 9,8,6)); //结果为09:008：0006 不足几位补几个0

## 语句

break; //跳出本次循环

continue; //跳出当前循环的这个值 直接进行下一个值的循环

return; //跳出的是当前的方法 函数

# 面向对象编程

## UUID

**private** UUID uuid;

**public** name(){

uuid=UUID.*randomUUID*();

}

## 类型强制装换

//String转化为int

String b="1000";

int a=Integer.parseInt(b);

//String转化为Double

double=Double.parseDouble(b);

//int强制转化为String

**int** a=123456;

String b=String.*valueOf*(a);

## 接口

public interface Animal{

void eat();

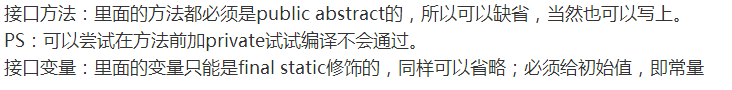
}

public class H implements Animal{

public void eat(){

}

}



## 内部类

public class Outer{

class Inner{

}

}

实例化对象：Outer.Inner inner=new Outer().new Inner()

如果将接口转化为内部类引入下面代码

Animal animal=new Animal(){

public void eat(){

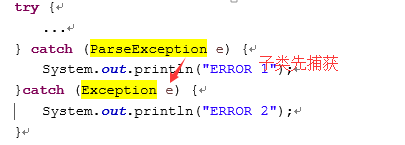
}

};

animal.eat();//引用里面的函数 须在;外执行

# 常用对象

## 异常



//自定义的异常

public class PersonalException extends Exception{

public PersonalException(){

super();

}

public PersonalException(String msg){

super(msg);

}

}

//将自定义异常抛出

public boolean Check()throws PersonalException{

throw new PersonalException("长度不为10")

}

//try catch中接收自定义异常

try{

//此处调用上述函数

}catch(PersonalException e){

System.out.println(e.getMessage());//结果输出为 长度不为10

}

## Math

**Int** num=(**int**)(Math.*random*()\*(max-min+1))+min;

[min,max]随机数

String number="18351845667";

System.***out***.println(Pattern.*matches*("(86)\*0\*1\\d{10}", number)); //判断手机的格式是否正确

## StringBuilder

StringBuilder builder=**new** StringBuilder("abc");

builder.append("def");

System.***out***.println("builder拼接结果"+builder);

StringBuilder builder = **new** StringBuilder("abc");

// 定义拼接次数

**int** times = 1000;

// 计时开始毫秒数

**long** start = System.*currentTimeMillis*();

// 循环拼接字符串

**for** (**int** i = 0; i < times; i++) {

builder.append("def");

}

// 计时结束毫秒数

**long** end = System.*currentTimeMillis*();

System.***out***.println("用时：" + (end - start) + "毫秒");

String str1 = "abc";

**long** start1 = System.*currentTimeMillis*();

// 循环拼接字符串

**for** (**int** i = 0; i < times; i++) {

str1 += "def";

}

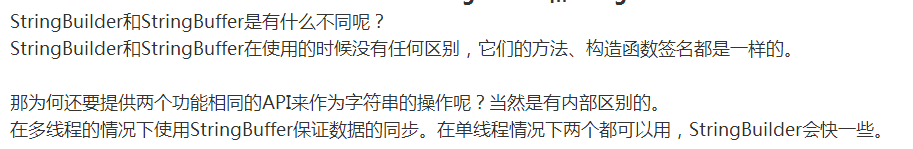
// 计时结束毫秒数

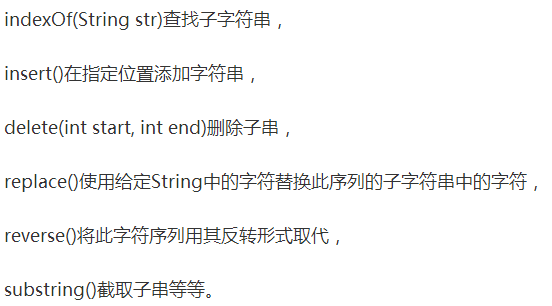
**long** end1 = System.*currentTimeMillis*();

System.***out***.println("用时：" + (end1 - start1) + "毫秒");

}

//StringBuilder 的拼接速度跟更快





## List Set Map

使用集合定义数组

List<Employee> emps=new ArrayList<Employee>();

Employee emp=new Employee();

emps.add(emp);

for(Employee e:emps){

}

有序列表

List<String> strs=new ArrayList<String>();

strs.add(int index,E element);//指定位置添加元素

strs.remove(int index);//移除指定位置的元素

strs.set(index,E element);//替换指定位置的元素

strs.sublist(int fromIndex,int toIndex);//截取列表

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

strs.get(i);//读取i位置的元素

}

for(String str:strs){

}

无序列表

Set<String> str=new HashSet<String>();//元素不重复

Map映射容器

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

//按照键值存储

map.put("key",value);

//按照键值取值

map.get("key"); //键值里key是唯一的 后存储的key会将之前的key覆盖

//是否包含键值

map.containsKey("key");

//取出所有key值

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

//取出所有value值

Collection<Double> values=map.values();

List集合

//排序

Collections.sort(list);

//二分查找

Collections.binarySearch(list,value);//返回值int

//最大最小值

Collections.max(list);

Collections.min(list);

## 排序接口

///////有时不可用Collection是需要实现Comparable接口

比如在汽车类Car中

public class Car implements Comparable<Car>{

public int compareTo(Car c){

return (this.getWeight().compareTo(c.getWeight()))

////按重量进行的排序

}

}

//////如果是自定义的类实现接口

public class PersonalSet implements Comparator<Car>{

public int compare(Car c1,Car c2){

return c1.getWeight()-c2.getWeight();//

//两个都可用

return c1.getWeight().compareTo(c2.getWeifht());

}

}

//////引用时的代码为 Collections.sort(cars,new PersonalSet());

## List的冒泡

**for**(**int** i=1;i<=courses.size()-1;i++){

**for**(**int** j=1;j<=courses.size()-i;j++){

**if**(courses.get(j-1).getGrade()<courses.get(j).getGrade()){

courses.add(j-1,courses.get(j));

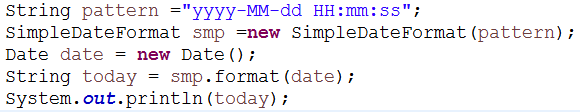
courses.remove(j+1);

}

}

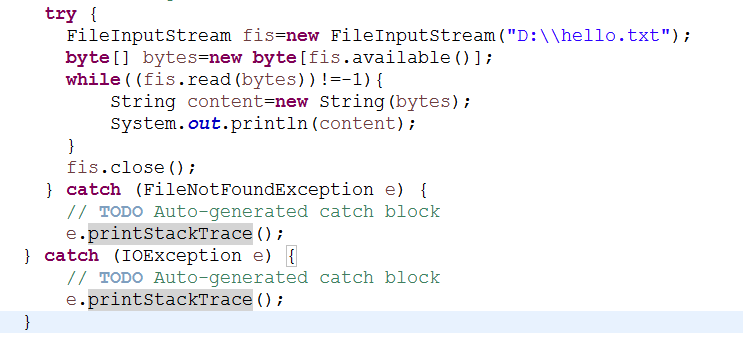
}

## 日期



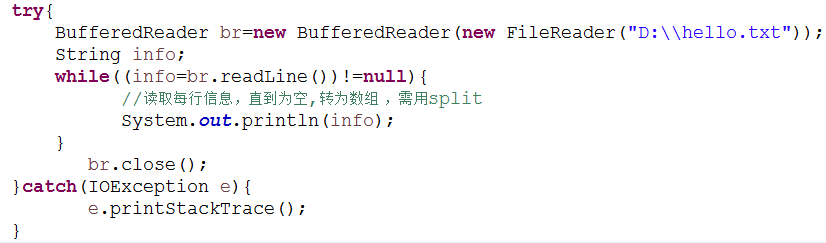
# 高级对象

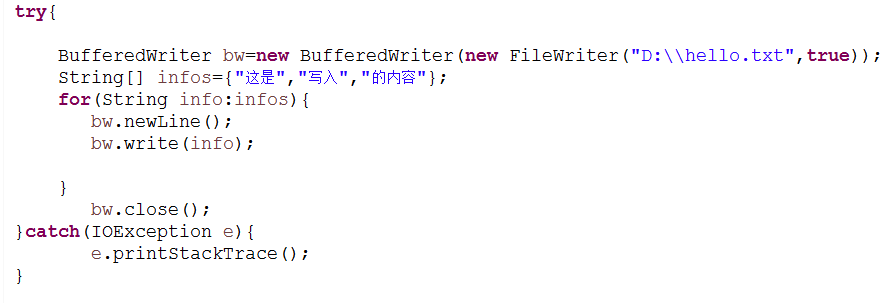
## 字节流的应用



## QQ截图20161017084718

## 字符流的应用





## 解析XML（了解）

//将bookstore.xml解析到Document中

String path=FileReader.class.getResource("/").getFile();

String xmlName=path+"/bookstore.xml";

try{

///解析bookstore.xml

DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();

DocumentBuilder docBuilder=dbf.newDocumentBuilder();

Document doc=docBuilder.parse(xmlName);

Element ele=doc.getDocumentElement();获取根节点元素bookstore

System.out.println(ele.getTagName());输出根节点标签名bookstore

NodeList booknodes=ele.getElementsByTagName("book"); //可选择根节点下任意的子标签

System.out.println(booknodes.getLength()); //输出根元素bookstore下 子元素book的个数

for(int i=0;i<booknodes.getLength();i++){

Node node=booknodes.item(i);///获取子元素

System.out.println(node.getNodeName());//输出book名

System.out.println(node.getNodeValue());//获取元素的value

System.out.println(node.getTextContent());//输出book节点下 所有内容

NamedNodeMap nodeMap=booknodes[i].getAttributes();///返回节点的属性

Attr attr=(Attr)nodeMap.item(i);

attr.getName();//获取到属性的名称name

attr.getValue();//获取到属性的值value

}

NodeList nodeList=ele.getChildNodes();///获取根节点下 子节点的列表///获取父节点的列表是ele.getParentNode()

System.out.println(nodeList.getLength());//输出根节点下 子节点的个数

}catch(SAXException e){

e.printStackTrace();

}catch(IOException e){

e.printStackTrace();

}catch(ParserConfigurationException e){

e.printStackTrace();

}

## 线程

### 单继承

**public** **class** MyThread **extends** Thread{

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

**for**(**int** i=0;i<10;i++){

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("MyThread"+i);

}

}

}

**public** **class** ThreadDemo {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

MyThread mt=**new** MyThread();

mt.start();

MyThread mt2=**new** MyThread();

mt2.start();

**for**(**int** i=0;i<10;i++){

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("主线程"+i);

}

}

}

### 多实现（Runnable）

**public** **class** MyThread **implements** Runnable{

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

**for**(**int** i=0;i<10;i++){

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("MyThread"+i);

}

}

}

**public** **class** RunnableDemo {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Thread t=**new** Thread(**new** MyThread());

t.start();

**for**(**int** i=0;i<10;i++){

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("主线程"+i);

}

}

}

### 优先级

**public** **class** MyRunnable1 **implements** Runnable{

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

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

System.***out***.println("+");

}

}

}

**public** **class** MyRunnable2 **implements** Runnable{

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

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

System.***out***.println("\*");

}

}

}

**public** **class** Test {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

MyRunnable1 r1=**new** MyRunnable1();

MyRunnable2 r2=**new** MyRunnable2();

Thread t1=**new** Thread(r1);

Thread t2=**new** Thread(r2);

System.***out***.println("t1的优先级"+t1.getPriority());

System.***out***.println("t2的优先级"+t2.getPriority());

//给T1设置最高的优先级

t1.setPriority(Thread.***MAX\_PRIORITY***);

System.***out***.println("t1改变后的优先级是："+t1.getPriority());

//给t2设置最低的优先级

t2.setPriority(Thread.***MIN\_PRIORITY***);

System.***out***.println("t2改变后的优先级是："+t2.getPriority());

t1.start();

t2.start();

}

}

### Join

**public** **class** TestJoin {

**public** **static** **void** main(String[] args) {

MyThread mt=**new** MyThread();

mt.start();

**for**(**int** i=0;i<10;i++){

**if**(i==5){

**try** {

mt.join();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

System.***out***.println("++++++");

}

}

}

**class** MyThread **extends** Thread{

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

**for**(**int** i=0;i<10;i++){

System.***out***.println("\*\*\*\*\*\*");

}

}

}

### Sleep

**public** **class** TestSleep {

**public** **static** **void** main(String[] args) {

**for**(**int** i=0;i<10;i++){

System.***out***.println(i);

**try** {

Thread.*sleep*(200);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

### Yield

**public** **class** TestYield {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Thread t1=**new** Thread(**new** MyRunnable1());

Thread t2=**new** Thread(**new** MyRunnable2());

t1.start();

t2.start();

}

}

**class** MyRunnable1 **implements** Runnable{

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

**for**(**int** i=0;i<200;i++){

System.***out***.println("+");

Thread.*yield*();

}

}

}

**class** MyRunnable2 **implements** Runnable{

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

**for**(**int** i=0;i<200;i++){

System.***out***.println("\*");

Thread.*yield*();

}

}

}

### 同步方法

**public** **synchronized** **void** printScore(){

}

### 同步块

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

**synchronized** (p) {

p.printScore();

}

}

一旦获得对象锁 则其他线程不能再执行被锁对象的其他任何同步方法 只有在同步方法执行完毕之后释放了锁 其他程序才能执行

**public** **static** **void** main(String[] args) {

Source s=**new** Source();

MyThread t1=**new** MyThread("method1", s);

MyThread t2=**new** MyThread("method2",s);

t1.start();

t2.start();

}

}

**public** **class** Source {

**synchronized** **void** method1(){

System.***out***.println("进入method1方法，获得锁");

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("method1执行完毕，释放锁");

}

**synchronized** **void** method2(){

System.***out***.println("进入method1方法，获得锁");

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("method1执行完毕，释放锁");

}

}

**public** **class** MyThread **extends** Thread{

String name;

Source s;

MyThread(String name,Source s){

**this**.name=name;

**this**.s=s;

}

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

**if**("method1".equals(**this**.name)){

**this**.s.method1();

}**else**{

**this**.s.method2();

}

}

## 通信

### URL

**public** **class** TestURL {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**try** {

URL url=**new** URL("http://learn.iflysse.com:8003/Pages/TrainingProgram.aspx#");

System.***out***.println("主机："+url.getHost());

System.***out***.println("端口："+url.getPort());

System.***out***.println("路径："+url.getPath());

System.***out***.println("协议："+url.getProtocol());

System.***out***.println("文件名："+url.getFile());

System.***out***.println("内容为"+url.getContent());

} **catch** (MalformedURLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**public** **class** URLReader {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**try** {

URL url=**new** URL("http://learn.iflysse.com:8003/Pages/TrainingProgram.aspx#");

InputStreamReader isr=**new** InputStreamReader(url.openStream());

BufferedReader br=**new** BufferedReader(isr);

String line=**null**;

**while**((line=br.readLine())!=**null**){

System.***out***.println(line);

}

} **catch** (MalformedURLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

### InetAddress

**public** **class** InetAddress\_test {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**try**{

InetAddress baidu=InetAddress.*getByName*("www.baidu.com");

System.***out***.println(baidu);

InetAddress[] baidus=InetAddress.*getAllByName*("www.baidu.com");

**for**(InetAddress b:baidus){

System.***out***.println(b);

}

InetAddress my=InetAddress.*getLocalHost*();

System.***out***.println(my);

**byte**[] bs=**new** **byte**[]{(**byte**)172,(**byte**)16,104,(**byte**)32};

InetAddress.*getByAddress*(bs);

}**catch**(UnknownHostException e){

e.printStackTrace();

}

}

}

### Socket通信

**public** **class** Client {

**public** **static** **void** main(String[] args) {

**try** {

//1.创建客户端Socket 指定服务器地址和端口

Socket socket=**new** Socket("localhost",8888);

//2.获取输出流，向服务端发送信息

OutputStream os=socket.getOutputStream();//字节输出流

PrintWriter pw=**new** PrintWriter(os);//将输出流包装成打印流

System.***out***.println("请向服务端发送内容：");

Scanner sc=**new** Scanner(System.***in***);

String content=sc.next();

pw.write(content);

pw.flush();

socket.shutdownOutput();//关闭输出流

//3.获取输入流 用来读取服务端的响应

InputStream is=socket.getInputStream();

BufferedReader br=**new** BufferedReader(**new** InputStreamReader(is));

String info=**null**;

**while**((info=br.readLine())!=**null**){//循环读取服务端的信息

System.***out***.println("服务端："+info);

}

//4.关闭资源

br.close();

is.close();

pw.close();

os.close();

socket.close();

} **catch** (UnknownHostException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**public** **class** Server {

**public** **static** **void** main(String[] args) {

**try** {

//1.创建一个服务器Socket,绑定指定的端口 来监听此端口

ServerSocket serverSocket=**new** ServerSocket(8888);

Socket socket=**null**;

//记录客户端数量

**int** count=0;

System.***out***.println("\*\*\*服务器即将启动，等待客户端的连接\*\*\*");

//循环监听等待客户端的连接

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

//2.调用accept()方法开始监听，等待客户端的连接

socket=serverSocket.accept();

//创建一个新的线程

ServerThread serverThread=**new** ServerThread(socket);

//启动线程

serverThread.start();

count++;

System.***out***.println("客户端的数量为："+count);

InetAddress address=socket.getInetAddress();

System.***out***.println("当前客户端的IP："+address.getHostAddress());

}

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**public** **class** ServerThread **extends** Thread{

//和本线程相关的Socket

Socket socket=**null**;

**public** ServerThread(Socket socket){

**this**.socket=socket;

}

//执行线程的操作，响应客户端的请求

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

InputStream is=**null**;

InputStreamReader isr=**null**;

BufferedReader br=**null**;

OutputStream os=**null**;

PrintWriter pw=**null**;

**try** {

//3.获取输入流，读取客户端所发送的信息

is=socket.getInputStream();//字节输入流

isr=**new** InputStreamReader(is);//将字节流转化为字符流

br=**new** BufferedReader(isr);//为输入流添加缓冲

String info=**null**;

**while**((info=br.readLine())!=**null**){//循环读取客户端的信息

System.***out***.println("客户端："+info);

}

//4.获取输出流 响应客户端请求

os=socket.getOutputStream();

pw=**new** PrintWriter(os);

pw.write("欢迎您!");

pw.flush();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}**finally**{

**try** {

//5.关闭资源

**if**(pw!=**null**)

pw.close();

**if**(os!=**null**)

os.close();

**if**(br!=**null**)

br.close();

**if**(isr!=**null**)

isr.close();

**if**(is!=**null**)

is.close();

**if**(socket!=**null**)

socket.close();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

### UDP通信

**public** **class** UDPServer {

**public** **static** **void** main(String[] args) {

**try** {

/\*

\* 接收客户端发送的数据

\* \*/

//1.窗机服务器端UDP，用于接收数据

DatagramSocket socket=**new** DatagramSocket(8800);

//2.创建数据报，用于接收客户端发送的数据

**byte**[] data=**new** **byte**[1024];

DatagramPacket packet=**new** DatagramPacket(data, data.length);

//3.接收客户端发送的数据

System.***out***.println("\*\*\*服务器端已经启动，等待客户端发送数据\*\*\*");

socket.receive(packet);//此方法在接收到数据报之前会阻塞

//4.读取数据报

String info=**new** String(data,0,packet.getLength());

System.***out***.println("客户端："+info);

/\*

\* 向客户端响应数据

\* \*/

//1.定义客户端的地址、地址、数据

InetAddress address=packet.getAddress();

**int** port=packet.getPort();

**byte**[] data2="欢迎您!".getBytes();

//2.创建数据包包含响应的数据信息

DatagramPacket packte2=**new** DatagramPacket(data2, data2.length,address,port);

//3.响应客户端

socket.send(packte2);

//4.关闭资源

socket.close();

} **catch** (SocketException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**public** **class** UDPClient {

**public** **static** **void** main(String[] args) {

**try** {

/\*

\* 向服务器端发送数据

\* \*/

//1.定义服务器的地址、端口号、数据

InetAddress address=InetAddress.*getByName*("localhost");

**int** port=8800;

**byte**[] data="用户名：admin,密码：123".getBytes();

//2.创建数据包 包括发送的信息

DatagramPacket packet=**new** DatagramPacket(data, data.length,address,port);

//3.创建DatagramSocket对象

DatagramSocket socket=**new** DatagramSocket();

//4.向服务器端 发送数据包

socket.send(packet);

/\*

\* 向服务器端接收数据

\* \*/

//1.创建数据报 用于接收服务器端响应的数据

**byte**[] data2=**new** **byte**[1024];

DatagramPacket packet2=**new** DatagramPacket(data2, data2.length);

//2.接受服务器响应的数据

socket.receive(packet2);

//3.读取服务器端响应的信息

String reply=**new** String(data2,0,packet2.getLength());

System.***out***.println("服务器："+reply);

//4.关闭资源

socket.close();

} **catch** (UnknownHostException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (SocketException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

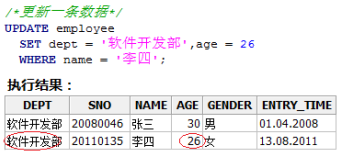
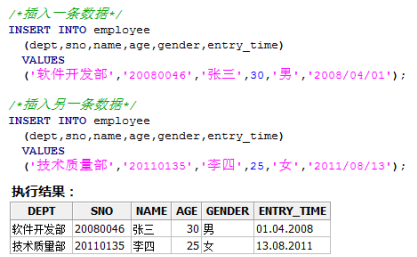
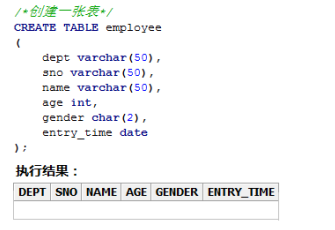
e.printStackTrace();

}

}

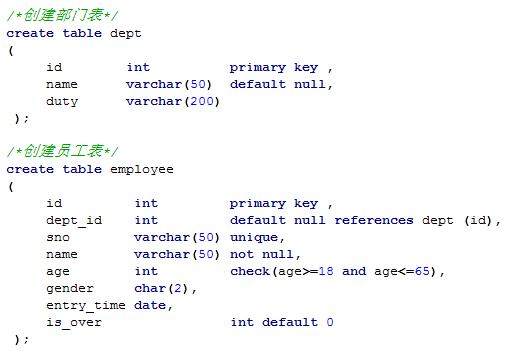
}

# 数据库



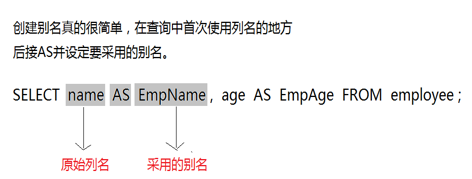
## 完整性约束



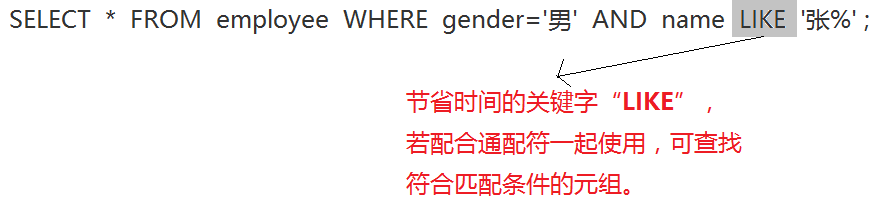
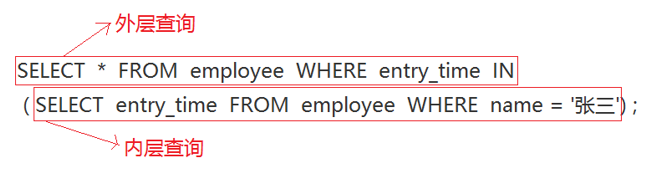


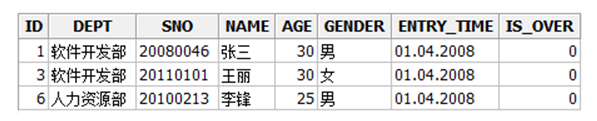
## 查询

### 投影查询

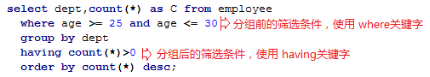
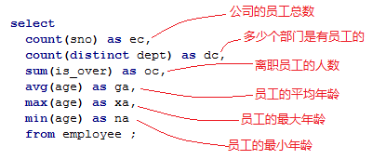


### 选择查询

6360506060892017461135302

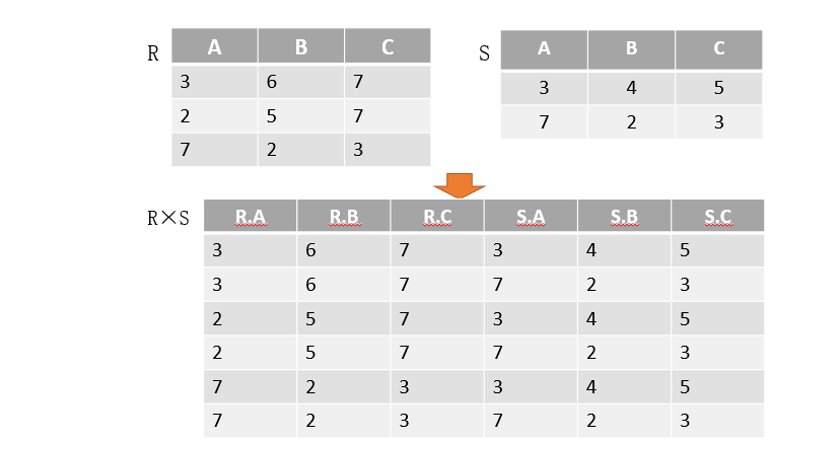


### 聚合查询



### 多表查询

笛卡尔积



=> select \* from R cross join S;

Where R.A=S.A;

查询A和B的笛卡尔积

select \* from A cross join B;

查询A和B的内连接 即等值连接

select \* from A cross join B where A.xx=B.xx;

select \* from A inner join B on A.xx=B.xx;

左外连接

select dept.name as D,employee.name as E

from dept left outer join employee on employee.dept\_id=dept.id

where 1=1;

### 合并查询

6360505779349461971371548

//union all 不会删除重复行也不会自动增序

6360505803577547122148716

### 创建视图

create view v(Sno,Cno)

as

select student.no,course.no

from student cross join course

where student.Id=Course.STUDENT\_ID and ...;

select \* from v;

select student.no as Sno,course.no as Cno

from student cross join course

where student.Id=Course.STUDENT\_ID and ...;

## JDBC

**public** **class** JDBC {

**public** **static** **void** main(String[] args) {

**try** {

Class.*forName*("com.mysql.jdbc.Driver");

Connection conn=DriverManager.*getConnection*("jdbc:mysql://localhost:3306/init","root","123");

// PreparedStatement pstat=conn.prepareStatement("SELECT kehu.`name`FROM kehu WHERE kehu.`name`=? OR kehu.`name`=?;");

// pstat.setObject(1, "q");

// pstat.setObject(2, "w");

// ResultSet rs=pstat.executeQuery();

/\* 如果批量添加信息的话 在for循环内最后要加上stat.addBatch();

循环出来使用stat.executeBatch(); \*/

// pstat.executeUpdate(); //除查询外 都用executeUpdate()

Statement stat = conn.createStatement();

ResultSet rs=stat.executeQuery("SELECT kehu.`name`FROM kehu WHERE kehu.`name`='q' OR kehu.`name`='w';");

**while**(rs.next()){

// System.out.println(rs.getObject("name"));

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

System.***out***.println(name);

}

rs.close();

stat.close();

// pstat.close();

conn.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

# 正则表达式

Pattern p=Pattern.*compile*("")*; //pattern正则表达式*

Matcher m=p.matcher(info);//mather匹配目标

**while**(m.find()){ //判断是否找到

String email=m.group();//输出匹配内容

System.***out***.println(email);

}

# JavaScript

## 解析

document.write(parseInt(numString,radix));

## 字符串和Math

var word="aSd";

document.write(word.toSupperCase());//转化为大写 结果为ASD

document.write(word.toLowerCase());//转化为小写 结果为asd

document.write(word.replace("aSd","qwe"));//替换

document.write(typeof word);//输出数据类型

var num=Math.random()\*(n-m)+m;//m~n之间随机赋值

var num=Math.round();//四舍五入

var num=Math.random();//0~1之间随机赋值

var num=Math.max(a,b,c);//取最大值

document.write(Math.abs(-1));//绝对值

## Undefined和Null



## Boolean型的判定

var a=null;

document.write(Boolean(a));

QQ截图20161017103917

## 运算

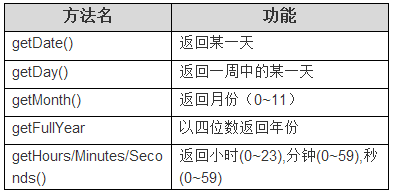
document.write("2"+3+4); //234

document.write(2+"3"+4); //234

document.write(2+3+"4"); //54

## Date

var mydate=new Date();



document.write(mydate.getMonth()+1);

## 循环

var i=0;

while(i<6){

document.write(i);//结果为：012345//先判断 后循环

i++;

}

var i=1;

do{

document.write(i); //结果为12345//先循环 后判断

i++;

}while(i<6)

## 数组

var arr=new Array();

var arr=new Array(6);

var arr=new Array("","","");

var arr=["","",""];

arr.push("");在数组最后一位添加元素

var pop=arr.pop();//移除数组最后一个元素 并返回该元素

arr.sort();//对数组进行升序排序

arr.join("-");返回一个字符串 将数组用"-"连接起来

## 函数

1.函数声明的方式

Function myfunction(){

document.write();

}

myfunction();//函数调用//可放在之前 也可放在之后

2函数表达式的方式

var myfun=function(){

document.write();

}

myfun();//只能放在函数表达式之后

3.传参函数（已知参数个数）

function sum(a,b){

return a+b;

}

document.write(sum(1,2));

4.传参函数（未知参数个数）

function sum(){

var a=arguments[0];//1

var b=arguments[1];//2

var c=arguments[2];//3

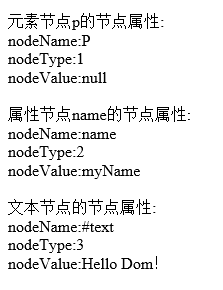
return a+b+c;

}

document.write(sum(1,2,3));

# JavaScriptDOM

## 节点



var content=document.getElementById("").innerHTML;//获取节点里的所有内容

var content=document.getElementById("").parentNode;//父亲节点

var content=document.getElementById("").childNode;//所有的直接子节点

var content=document.getElementById("").firstChild;//第一个直接子节点

var oH=document.createElement("h4");//创建一个节点

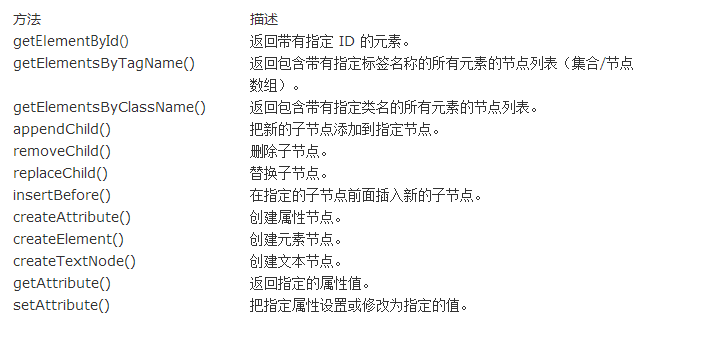
oH.innerHTML="Dom节点";//添加节点的内容

var oB=document.getElementById("");//获取需要添加节点的位置

oB.insertBefore(oH,oB.firstChild);//放到第一个子节点之前

oB.removeChild(oB.firstNode);//删除第一个子节点

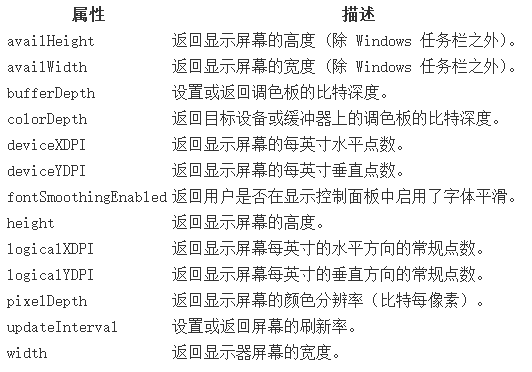
oB.replaceChild(oH,oB.firstChild);//替换第一个子节点



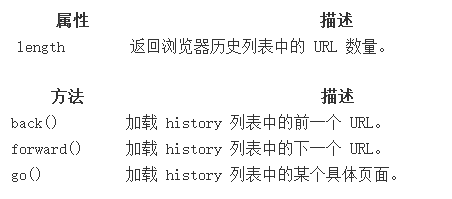
## navigator.



## screen.



## history.



function next(){

window.history.forward();

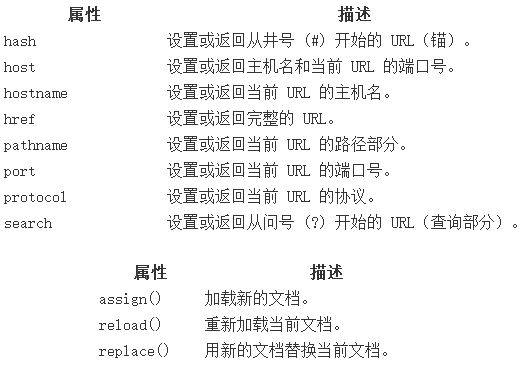
}

function before(){

window.history.back();

}

## location.



## 窗口控制

function \_alert(){

alert("这是alert方法，用于弹出一个窗口");

}

function \_prompt(){

var Info=window.prompt("这是prompt方法，用于获取输入内容");

alert("你输入了"+Info);

}

function \_confirm(){

var flag=confirm("这是confirm方法，用于选择确认或取消");

if(flag==true){

alert("你选择了确认");

}else{

alert("你选择了取消");

}

}

## 时间定时器

var mytimer=window.setTimeout("function()",1000);

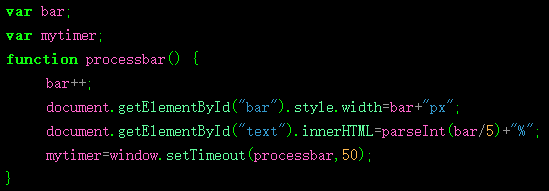
//一秒后执行function()

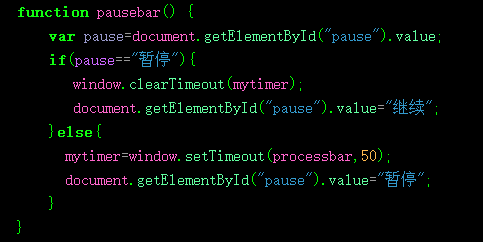
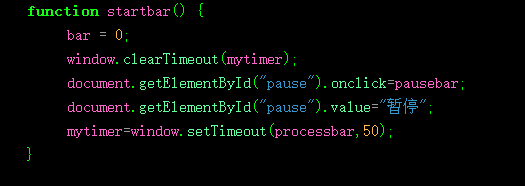
var mytimer=window.setInterval("function()",1000);

//每隔一秒后执行function()

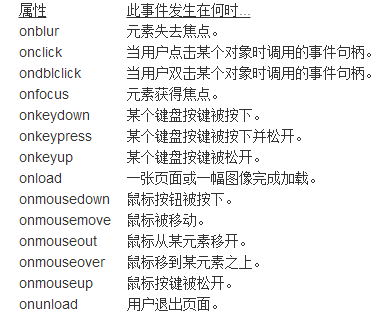
window.clearTimeout(mytimer);

window.clearInterval(mytimer);

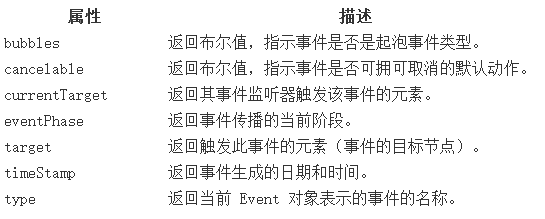




## Event



## event.



## 键盘事件

function keyEvent(event){

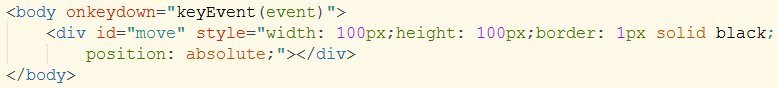
alert("你按下了"+event.keyCode+"号键")

}

<body onkeydown="keyEvent(event)">

</body>

## 控制div移动



# JavaScript面向对象

## 创建对象

1.0

var Hy={

name : hy,

age : 23,

info:function(){

return Hy.age;

}

};

document.write(Hy.name+Hy.age);

document.write(Hy["name"]+Hy["age"]);

alert(Hy.info());

2.0

var Hy=new Object();

Hy.name=Hy;

Hy.age=29;

Hy.info=function(){

return Hy.name;

}

document.write(Hy.name+Hy.age);

alert(Hy.info());

3.0

function createPerson(name,age){

var Person = new Object();

Person.name=name;

Person.age=age;

return Person;

}

var PersonA=createPerson("ZhangSan","18");

4.0

function Person(name,age){

this.name=name;

this.age=age;

this.sayName=function(){ //此行可写成Person.prototype.sayName=function(){

document.write(this.name);

};

}

var personA=new Person("ZhangSan","18");

personA.sayName();

alert(personA instanceof Person);//判断personA是否是Perosn的实例

4.1

function Person(name,age){

this.name=name;

this.age=age;

this.sayName=sayName;

}

function sayName(){

document.write(this.name);

}

var personA=new Person("ZhangSan","18");

personA.sayName();

alert(personA instanceof Person);//判断personA是否是Perosn的实例

## 继承

function Animal(color){

this.color=color;

this.getAreas=function(){

return 0;

}

};

function Cat(height){

this.Ani=Animal;

this.Ani("blue");

//等价于this.color=blue;

//等价于Animal.call(this,"blue");

delete this.Ani;

this.height=height;

}

var c=new Cat(5);

document.write(c.color);//结果为blue

## 对象冒充方法

function add(a,b){

alert(a+b);

}

function sub(a,b){

alert(a-b);

}

add.call(sub,8,5) == add(8,5);//结果为13

## 原型链继承

function Person(name,age){

this.name=name;

this.age=age;

}

Person.prototype.say=function(){

document.write("hello,my name is"+this.name);

};

function Man(){ }

Man.prototype=new Person("Jack",30);

var man = new Man();

man.say();

# JQuery

## Baidu CDN:

<head>  
<script src="http://libs.baidu.com/jquery/1.10.2/jquery.min.js">  
</script>  
</head>

## 文档就绪事件

$(document).ready(function(){  
  
   ***// 开始写 jQuery 代码...***  
  
});

$(function(){  
  
   ***// 开始写 jQuery 代码...***  
  
});

## 选择器

id选择器 $("#test")

$(document).ready(function(){  
  $("button").click(function(){  
    $("#test").hide();  
  });  
});

class选择器 $(".test")

$(document).ready(function(){  
  $("button").click(function(){  
    $(".test").hide();  
  });  
});

更多实例

|  |  |  |
| --- | --- | --- |
| **语法** | **描述** | **实例** |
| $("\*") | 选取所有元素 | [在线实例](http://www.w3cschool.cn/tryrun/showhtml/tryjquery_sel_all2" \t "http://www.w3cschool.cn/jquery/_blank) |
| $(this) | 选取当前 HTML 元素 | [在线实例](http://www.w3cschool.cn/tryrun/showhtml/tryjquery_sel_this" \t "http://www.w3cschool.cn/jquery/_blank) |
| $("p.intro") | 选取 class 为 intro 的 <p> 元素 | [在线实例](http://www.w3cschool.cn/tryrun/showhtml/tryjquery_sel_pclass" \t "http://www.w3cschool.cn/jquery/_blank) |
| $("p:first") | 选取第一个 <p> 元素 | [在线实例](http://www.w3cschool.cn/tryrun/showhtml/tryjquery_sel_pfirst" \t "http://www.w3cschool.cn/jquery/_blank) |
| $("ul li:first") | 选取第一个 <ul> 元素的第一个 <li> 元素 | [在线实例](http://www.w3cschool.cn/tryrun/showhtml/tryjquery_sel_ullifirst" \t "http://www.w3cschool.cn/jquery/_blank) |
| $("ul li:first-child") | 选取每个 <ul> 元素的第一个 <li> 元素 | [在线实例](http://www.w3cschool.cn/tryrun/showhtml/tryjquery_sel_ullifirstchild" \t "http://www.w3cschool.cn/jquery/_blank) |
| $("[href]") | 选取带有 href 属性的元素 | [在线实例](http://www.w3cschool.cn/tryrun/showhtml/tryjquery_sel_hrefattr" \t "http://www.w3cschool.cn/jquery/_blank) |
| $("a[target='\_blank']") | 选取所有 target 属性值等于 "\_blank" 的 <a> 元素 | [在线实例](http://www.w3cschool.cn/tryrun/showhtml/tryjquery_sel_hrefattrblank" \t "http://www.w3cschool.cn/jquery/_blank) |
| $("a[target!='\_blank']") | 选取所有 target 属性值不等于 "\_blank" 的 <a> 元素 | [在线实例](http://www.w3cschool.cn/tryrun/showhtml/tryjquery_sel_hrefattrnotblank" \t "http://www.w3cschool.cn/jquery/_blank) |
| $(":button") | 选取所有 type="button" 的 <input> 元素 和 <button> 元素 | [在线实例](http://www.w3cschool.cn/tryrun/showhtml/tryjquery_sel_button2" \t "http://www.w3cschool.cn/jquery/_blank) |
| $("tr:even") | 选取偶数位置的 <tr> 元素 | [在线实例](http://www.w3cschool.cn/tryrun/showhtml/tryjquery_sel_even" \t "http://www.w3cschool.cn/jquery/_blank) |
| $("tr:odd") | 选取奇数位置的 <tr> 元素 |  |

## 事件



$("p").click(function(){  
  $(this).hide();  
});

****hover()****

hover()方法用于模拟光标悬停事件。

当鼠标移动到元素上时，会触发指定的第一个函数(mouseenter);当鼠标移出这个元素时，会触发指定的第二个函数(mouseleave)。

$("#p1").hover(function(){  
  alert("You entered p1!");  
  },  
  function(){  
  alert("Bye! You now leave p1!");  
});

****focus()****

当元素获得焦点时，发生 focus 事件。

当通过鼠标点击选中元素或通过 tab 键定位到元素时，该元素就会获得焦点。

$("input").focus(function(){  
  $(this).css("background-color","#cccccc");  
});

****blur()****

当元素失去焦点时，发生 blur 事件。

$("input").blur(function(){  
  $(this).css("background-color","#ffffff");  
});

## 隐藏和显示

Show() hide()

$("#hide").click(function(){  
  $("p").hide();  
});  
  
$("#show").click(function(){  
  $("p").show();  
});

$(*selector*).hide(*speed,callback*);  
  
$(*selector*).show(*speed,callback*);

可选的 speed 参数规定隐藏/显示的速度，可以取以下值："slow"、"fast" 或毫秒。

可选的 callback 参数是隐藏或显示完成后所执行的函数名称。

$("button").click(function(){  
  $("p").hide(1000);  
});

Toggle()

通过 jQuery，您可以使用 toggle() 方法来切换 hide() 和 show() 方法。

显示被隐藏的元素，并隐藏已显示的元素：

$("button").click(function(){  
  $("p").toggle();  
});

$(*selector*).toggle(*speed,callback*);

## 淡入和淡出

fadeIn()

$("button").click(function(){  
  $("#div1").fadeIn();  
  $("#div2").fadeIn("slow");  
  $("#div3").fadeIn(3000);  
});

$(*selector*).fadeIn(*speed,callback*);

fadeOut()

$("button").click(function(){  
  $("#div1").fadeOut();  
  $("#div2").fadeOut("slow");  
  $("#div3").fadeOut(3000);  
});

$(*selector*).fadeOut(*speed,callback*);

fadeToggle()

jQuery fadeToggle() 方法可以在 fadeIn() 与 fadeOut() 方法之间进行切换。

如果元素已淡出，则 fadeToggle() 会向元素添加淡入效果。

如果元素已淡入，则 fadeToggle() 会向元素添加淡出效果。

$(*selector*).fadeToggle(*speed,callback*);

可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。

可选的 callback 参数是 fading 完成后所执行的函数名称。

$("button").click(function(){  
  $("#div1").fadeToggle();  
  $("#div2").fadeToggle("slow");  
  $("#div3").fadeToggle(3000);  
});

## 滑动

slideDown()

jQuery slideDown() 方法用于向下滑动元素。

**语法:**

$(*selector*).slideDown(*speed,callback*);

可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。

可选的 callback 参数是滑动完成后所执行的函数名称。

$("#flip").click(function(){  
  $("#panel").slideDown();  
});

slideUp()

jQuery slideUp() 方法用于向上滑动元素。

**语法:**

$(*selector*).slideUp(*speed,callback*);

$("#flip").click(function(){  
  $("#panel").slideUp();  
});

slideToggle()

jQuery slideToggle() 方法可以在 slideDown() 与 slideUp() 方法之间进行切换。

如果元素向下滑动，则 slideToggle() 可向上滑动它们。

如果元素向上滑动，则 slideToggle() 可向下滑动它们。

$(*selector*).slideToggle(*speed,callback*);

$("#flip").click(function(){  
  $("#panel").slideToggle();  
});

## 动画

animate()

$(***selector***).animate({***params***}***,speed,callback***);

必需的 params 参数定义形成动画的 CSS 属性。

可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。

默认地，jQuery 提供针对动画的队列功能。

这意味着如果您在彼此之后编写多个 animate() 调用，jQuery 会创建包含这些方法调用的"内部"队列。然后逐一运行这些 animate 调用。

$("button").click(function(){  
  var div=$("div");  
  div.animate({height:'300px',opacity:'0.4'},"slow");  
  div.animate({width:'300px',opacity:'0.8'},"slow");  
  div.animate({height:'100px',opacity:'0.4'},"slow");  
  div.animate({width:'100px',opacity:'0.8'},"slow");  
});

下面的例子把 <div> 元素往右边移动了 100 像素，然后增加文本的字号：

$("button").click(function(){  
  var div=$("div");  
  div.animate({left:'100px'},"slow");  
  div.animate({fontSize:'3em'},"slow");  
});

## 停止动画

jQuery stop() 方法用于停止动画或效果，在它们完成之前。

stop() 方法适用于所有 jQuery 效果函数，包括滑动、淡入淡出和自定义动画。

$(*selector*).stop(*stopAll,goToEnd*);

可选的 stopAll 参数规定是否应该清除动画队列。默认是 false，即仅停止活动的动画，允许任何排入队列的动画向后执行。

可选的 goToEnd 参数规定是否立即完成当前动画。默认是 false。

$("#stop").click(function(){  
  $("#panel").stop();  
});

## 捕获

text() html()

下面的例子演示如何通过 jQuery text() 和 html() 方法来获得内容：

$("#btn1").click(function(){  
  alert("Text: " + $("#test").text());  
});  
$("#btn2").click(function(){  
  alert("HTML: " + $("#test").html());  
});

Val()

下面的例子演示如何通过 jQuery val() 方法获得输入字段的值：

$("#btn1").click(function(){  
  alert("Value: " + $("#test").val());  
});

attr()

jQuery attr() 方法用于获取属性值。

下面的例子演示如何获得链接中 href 属性的值：

$("button").click(function(){  
  alert($("#w3s").attr("href"));  
});

## 设置

text()、html() 以及 val()

我们将使用前一章中的三个相同的方法来设置内容：

text() - 设置或返回所选元素的文本内容

html() - 设置或返回所选元素的内容（包括 HTML 标记）

val() - 设置或返回表单字段的值

下面的例子演示如何通过 text()、html() 以及 val() 方法来设置内容：

$("#btn1").click(function(){  
  $("#test1").text("Hello world!");  
});  
$("#btn2").click(function(){  
  $("#test2").html("<b>Hello world!</b>");  
});  
$("#btn3").click(function(){  
  $("#test3").val("Dolly Duck");  
});

text()、html() 以及 val() 的回调函数

上面的三个 jQuery 方法：text()、html() 以及 val()，同样拥有回调函数。回调函数由两个参数：被选元素列表中当前元素的下标，以及原始（旧的）值。然后以函数新值返回您希望使用的字符串。

下面的例子演示带有回调函数的 text() 和 html()：

$("#btn1").click(function(){  
  $("#test1").text(function(i,origText){  
    return "Old text: " + origText + " New text: Hello world!  
    (index: " + i + ")";   
  });  
});  
  
$("#btn2").click(function(){  
  $("#test2").html(function(i,origText){  
    return "Old html: " + origText + " New html: Hello <b>world!</b>  
    (index: " + i + ")";   
  });  
});

attr()

jQuery attr() 方法也用于设置/改变属性值。

下面的例子演示如何改变（设置）链接中 href 属性的值：

$("button").click(function(){  
  $("#w3s").attr("href","http://www.w3cschool.cn/jquery");  
});

attr() 方法也允许您同时设置多个属性。

下面的例子演示如何同时设置 href 和 title 属性：

$("button").click(function(){  
  $("#w3s").attr({  
    "href" : "http://www.w3cschool.cn/jquery",  
    "title" : "jQuery 教程"  
  });  
});

attr() 的回调函数

jQuery 方法 attr()，也提供回调函数。回调函数由两个参数：被选元素列表中当前元素的下标，以及原始（旧的）值。然后以函数新值返回您希望使用的字符串。

下面的例子演示带有回调函数的 attr() 方法：

$("button").click(function(){  
  $("#w3s").attr("href", function(i,origValue){  
    return origValue + "/jquery";   
  });  
});

## 添加元素

append() - 在被选元素的结尾插入内容(指在元素内)

prepend() - 在被选元素的开头插入内容

after() - 在被选元素之后插入内容

before() - 在被选元素之前插入内容

append()

jQuery append() 方法在被选元素的结尾插入内容。

$("p").append("Some appended text.");

prepend()

jQuery prepend() 方法在被选元素的开头插入内容。

$("p").prepend("Some prepended text.");

after() 和 before()

jQuery after() 方法在被选元素之后插入内容。

jQuery before() 方法在被选元素之前插入内容。

$("img").after("在后面添加文本");  
  
$("img").before("在前面添加文本");

## 删除元素

remove()

jQuery remove() 方法删除被选元素及其子元素。

$("#div1").remove();

empty()

jQuery empty() 方法删除被选元素的子元素。

$("#div1").empty();

过滤被删除的元素

jQuery remove() 方法也可接受一个参数，允许您对被删元素进行过滤。

该参数可以是任何 jQuery 选择器的语法。

下面的例子删除 class="italic" 的所有 <p> 元素：

$("p").remove(".italic");

## CSS类

addClass()

下面的例子展示如何向不同的元素添加 class 属性。当然，在添加类时，您也可以选取多个元素：

$("button").click(function(){  
  $("h1,h2,p").addClass("blue");  
  $("div").addClass("important");  
});

您也可以在 addClass() 方法中规定多个类：

$("button").click(function(){  
  $("#div1").addClass("important blue");  
});

removeClass()

下面的例子演示如何在不同的元素中删除指定的 class 属性：

$("button").click(function(){  
  $("h1,h2,p").removeClass("blue");  
});

toggleClass()

下面的例子将展示如何使用 jQuery toggleClass() 方法。该方法对被选元素进行添加/删除类的切换操作：

$("button").click(function(){  
  $("h1,h2,p").toggleClass("blue");  
});

## css()方法

设置多个 CSS 属性

如需设置多个 CSS 属性，请使用如下语法：

css({"*propertyname*":"*value*","*propertyname*":"*value*",...});

下面的例子将为所有匹配元素设置 background-color 和 font-size：

$("p").css({"background-color":"yellow","font-size":"200%"});

## 尺寸

width() 不包括内边距、边框或外边距

height()

innerWidth() 包括内边距

innerHeight()

outerWidth() 包括内边距和边框

outerHeight()

## 遍历

### 祖先

parent() 方法返回被选元素的直接父元素。

$(document).ready(function(){  
  $("span").parent();  
});

parents() 方法返回被选元素的所有祖先元素，它一路向上直到文档的根元素 (<html>)。

$(document).ready(function(){  
  $("span").parents();  
});

下面的例子返回所有 <span> 元素的所有祖先，并且它是 <ul> 元素：

$(document).ready(function(){  
  $("span").parents("ul");  
});

parentsUntil() 方法返回介于两个给定元素之间的所有祖先元素。

下面的例子返回介于 <span> 与 <div> 元素之间的所有祖先元素：

$(document).ready(function(){  
  $("span").parentsUntil("div");  
});

### 后代

children() 方法返回被选元素的所有直接子元素。

下面的例子返回每个 <div> 元素的所有直接子元素：

$(document).ready(function(){  
  $("div").children();  
});

您也可以使用可选参数来过滤对子元素的搜索。

下面的例子返回类名为 "1" 的所有 <p> 元素，并且它们是 <div> 的直接子元素：

$(document).ready(function(){  
  $("div").children("p.1");  
});

下面的例子返回属于 <div> 后代的所有 <span> 元素：

$(document).ready(function(){  
  $("div").find("span");  
});

下面的例子返回 <div> 的所有后代：

$(document).ready(function(){  
  $("div").find("\*");  
});

### 同胞

siblings()

siblings() 方法返回被选元素的所有同胞元素。

$(document).ready(function(){  
  $("h2").siblings();  
});

下面的例子返回属于 <h2> 的同胞元素的所有 <p> 元素：

$(document).ready(function(){  
  $("h2").siblings("p");  
});

next()

next() 方法返回被选元素的下一个同胞元素。

$(document).ready(function(){  
  $("h2").next();  
});

nextAll()

nextAll() 方法返回被选元素的所有跟随的同胞元素。

$(document).ready(function(){  
  $("h2").nextAll();  
});

nextUntil()

下面的例子返回介于 <h2> 与 <h6> 元素之间的所有同胞元素：

$(document).ready(function(){  
  $("h2").nextUntil("h6");  
});

### 过滤

first() 方法返回被选元素的首个元素。

下面的例子选取首个 <div> 元素内部的第一个 <p> 元素：

$(document).ready(function(){  
  $("div p").first();  
});

last() 方法返回被选元素的最后一个元素。

下面的例子选择最后一个 <div> 元素中的最后一个 <p> 元素：

$(document).ready(function(){  
  $("div p").last();  
});

eq() 方法返回被选元素中带有指定索引号的元素。

索引号从 0 开始，因此首个元素的索引号是 0 而不是 1。下面的例子选取第二个 <p> 元素（索引号 1）：

$(document).ready(function(){  
  $("p").eq(1);  
});

filter() 方法允许您规定一个标准。不匹配这个标准的元素会被从集合中删除，匹配的元素会被返回。

下面的例子返回带有类名 "intro" 的所有 <p> 元素：

$(document).ready(function(){  
  $("p").filter(".intro");  
});

not()下面的例子返回不带有类名 "intro" 的所有 <p> 元素：

$(document).ready(function(){  
  $("p").not(".intro");  
});

## 鼠标拖拽

jQuery(document).ready(

function () {

$('.layer').mousedown(

function (event) {

var isMove = true;

var abs\_x = event.pageX - $('.layer').offset().left;

var abs\_y = event.pageY - $('.layer').offset().top;

$(document).mousemove(function (event) {

if (isMove) {

var obj = $('.layer');

obj.css({'left':event.pageX - abs\_x, 'top':event.pageY - abs\_y});

}

}

).mouseup(

function () {

isMove = false;

}

);

}

);

}

);

# JavaWeb

## JSP

<%! %> 相当于class类 声明变量和方法

<%= %> 调用有返回值的变量和方法并将其输出

<% %> 相当于main 先将内容输出在HTML中 通过HTML显示在网页中

<%! //使用jsp方法

String NineNine(){

String s="";

**for**(**int** i=1;i<=9;i++){

**for**(**int** j=1;j<=i;j++){

s=s+(i+"\*"+j+"="+i\*j+" ");

}

s=s+"<br>";

}

**return** s;

}

//jsp内置out对象，使用脚本方式打开

**void** NineNine2(JspWriter out)**throws** Exception{

**for**(**int** i=1;i<=9;i++){

**for**(**int** j=1;j<=i;j++){

out.println(i+"\*"+j+"="+i\*j+" ");

}

out.println("<br>");

}

}

%>

<h1>九九乘法表</h1><%= NineNine() %> <br>

<h1>九九乘法表</h1><% NineNine2(out); %> <br>

### Out

<head>

<base href=*"*<%=basePath%>*"*>

<title>My JSP 'out.jsp' starting page</title>

<meta http-equiv=*"pragma"* content=*"no-cache"*>

<meta http-equiv=*"cache-control"* content=*"no-cache"*>

<meta http-equiv=*"expires"* content=*"0"*>

<meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>

<meta http-equiv=*"description"* content=*"This is my page"*>

<!--

<link rel="stylesheet" type="text/css" href="styles.css">

-->

</head>

<body>

<h1>out内置对象</h1>

<hr>

<%

out.println("<h2>静夜思</h2>");

out.println("床前明月光<br>");

out.println("疑是地上霜<br>");

out.flush();//这里不会抛出异常

//out.clear();//这里会抛出异常

out.clearBuffer();

out.println("举头望明月<br>");

out.println("低头思故乡<br>");

%>

缓冲区大小：<%=out.getBufferSize() %>byte <br>

缓冲区剩余大小：<%=out.getRemaining() %>byte <br>

是否自动清空缓冲区：<%=out.isAutoFlush() %>byte <br>

</body>

### Request

<body>

<h1>用户注册</h1>

<hr>

<form action=*"request.jsp"* name=*"regform"* method=*"post"*>

<table>

<tr><td>用户名：</td>

<td><input type=*"text"* name=*"username"*/></td>

</tr>

<tr><td>爱好：</td>

<td>

<input type=*"checkbox"* name=*"fav"* value=*"read"*/>读书

<input type=*"checkbox"* name=*"fav"* value=*"music"*/>音乐

<input type=*"checkbox"* name=*"fav"* value=*"movie"*/>电影

<input type=*"checkbox"* name=*"fav"* value=*"internet"*/>上网

</td>

</tr>

<tr><td colspan=*"2"*><input type=*"submit"* value=*"提交"*/></td>

</tr>

</table>

</form>

<a href=*"request.jsp?username=张三"*>测试URL传参</a>

</body>

</head>

<body>

<h1>request内置对象</h1>

<% request.setCharacterEncoding("utf-8") ;//解决中文乱码问题

request.setAttribute("password","123456");//暂时存储

%>

<hr>

用户名：<%=request.getParameter("username") %>

爱好：<%

**if**(request.getParameterValues("fav")!=**null**){

String[] favs=request.getParameterValues("fav");

**for**(String fav:favs){

out.println(fav+" ");

}

}

%>

密码：<%=request.getAttribute("password") %><br>

请求体的MIME类型：<%=request.getContentType() %><br>

请求类的协议类型和版本号：<%=request.getProtocol() %><br>

接受请求的服务器主机名：<%=request.getServerName() %><br>

服务器端口号：<%=request.getServerPort() %><br>

请求文件的长度：<%=request.getContentLength() %><br>

请求客户端的IP地址<%=request.getRemoteAddr() %><br>

请求的真实路径：<%=request.getRealPath("request.jsp") %><br>

请求的山下文路径：<%=request.getContextPath() %><br>

</body>

### Response（面试题）

<%@ page language=*"java"* import=*"java.util.\*,java.io.\*"* contentType=*"text/html; charset=utf-8"*%>

<%

response.setContentType("text/html; charset=utf-8");//设置响应的

MIME类型

out.println("<h1>response内置对象</h1>");

out.println("<hr>");

//out.flush();

PrintWriter outer=response.getWriter();//获得输出流对象

outer.println("大家好，我是response对象形成的输出流outer对象");

//response.sendRedirect("login.jsp");//请求重定向

//请求重定向

//response.sendRedirect("request.jsp");

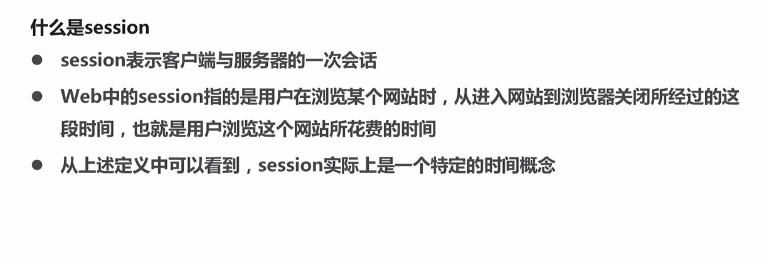
//请求转发

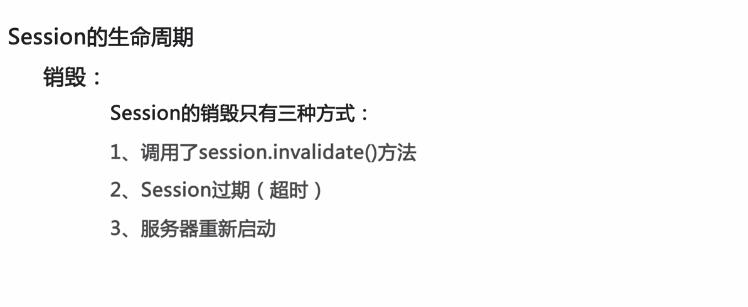
request.getRequestDispatcher("request.jsp").forward(request,response);

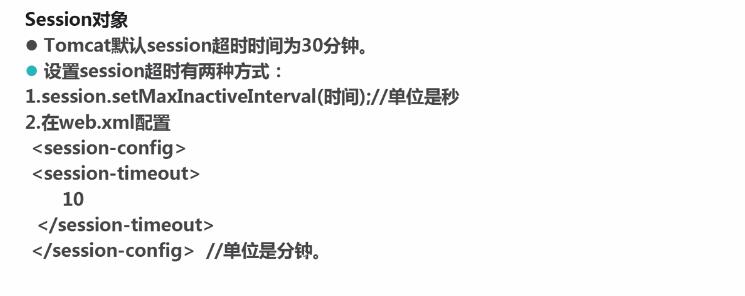
%>



### Session







<body>

<h1>session内置对象</h1>

<hr>

<%

SimpleDateFormat sdf=**new** SimpleDateFormat("yyyy年MM月dd日");

Date d=**new** Date(session.getCreationTime());

session.setAttribute("username","admin");

session.setAttribute("password","123456");

session.setAttribute("age",20);

//设置当前session最大生存期限 单位是秒

session.setMaxInactiveInterval(10);//10秒钟后当前session过期

%>

Session创建时间：<%=sdf.format(d) %><br>

Session的ID编号：<%=session.getId() %><br>

从Session中获取用户名：<%=session.getAttribute("username") %><br>

<a href=*"session2.jsp"* target=*"\_blank"*>跳转到session2中</a>

</body>

<body>

<h1>session内置对象</h1>

<hr>

<%

//SimpleDateFormat sdf=new SimpleDateFormat("yyyy年MM月dd日");

//Date d=new Date(session.getCreationTime());

//session.setAttribute("username","admin");

%>

Session的ID编号：<%=session.getId() %><br>

从Session中获取用户名：<%=session.getAttribute("username") %><br>

session中保存的属性有：<%

String[] names=session.getValueNames();

**for**(String name:names){

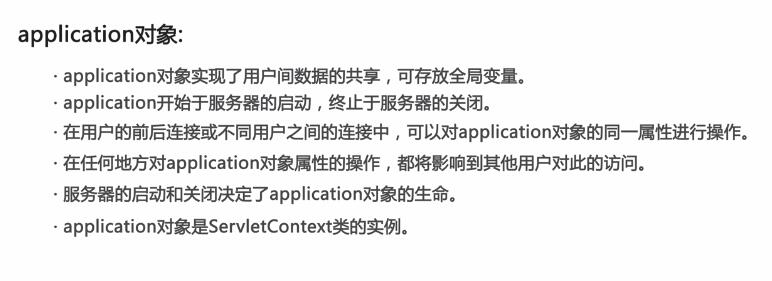
out.println(name+" ");

}

%>

</body>

### Application



<body>

<h1>application内置对象</h1>

<hr>

<%

application.setAttribute("city","北京");

application.setAttribute("postcode","10000");

application.setAttribute("email","lisi@126.com");

%>

所在城市是：<%=application.getAttribute("city") %><br>

application中的属性有：

<%

Enumeration attributes=application.getAttributeNames();

**while**(attributes.hasMoreElements()){

out.println(attributes.nextElement()+"<br>");

}

%><br>

JSP(SERVLET)引擎及版本号：<%=application.getServerInfo() %><br>

</body>

### Page

Include.jsp

<%

Date date=**new** Date();

SimpleDateFormat sdf=**new** SimpleDateFormat("yyyy年MM月dd日");

String s=sdf.format(date);

out.println(s+"<br>");

%>

pageContext.jsp

<body>

<h1>pageContext内置对象</h1>

<hr>

用户名是：<%=pageContext.getSession().getAttribute("username") %>

<%

//跳转到注册页面

//pageContext.forward("register.jsp");

pageContext.include("include.jsp");//嵌入include.jsp

%>

</body>

### Exception

exception\_test.jsp

<%@ pagelanguage=*"java"*contentType=*"text/html; charset=utf-8"* errorPage=*"exception.jsp"*%>

<body>

<h1>测试异常的对象</h1>

<hr>

<%

out.println(100/0);

%>

</body>

exception.jsp

<%@ page language=*"java"*contentType=*"text/html;charset=utf-8"* isErrorPage=*"true"*%>

<body>

<h1>exception内置对象</h1>

<hr>

异常消息是：<%=exception.getMessage() %><br>

异常的字符串描述：<%=exception.toString() %>

</body>

### Include

<jsp:include page=*"Login.jsp"* flush=*"false"*></jsp:include>

<%@ include file=*"Login.jsp"* %>

pageContext.include("include.jsp");



### Forward

<jsp:forward page=*"Register.jsp"*></jsp:forward>

==>

request.getRequestDispatcher("request.jsp").forward(request,response);

### Param

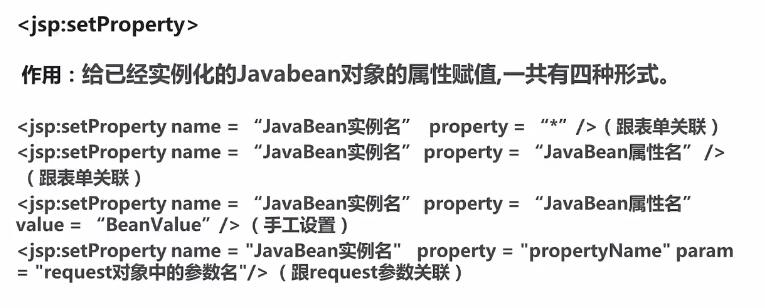
<jsp:forward page=*"Register.jsp"*>

<jsp:param value=*"newValue"* name=*"newName"*></jsp:param>

传参 可覆盖

</jsp:forward>

## JavaBeans





users.java

**package** com.po;

**public** **class** users {

**private** String username;

**private** String password;

**public** users(){ }

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

}

login.jsp

<body>

<form name=*"loginForm"* method=*"post"* action=*"dologin.jsp?pass=11"*>

<table>

<tr>

<td>用户名：</td>

<td><input type=*"text"* name=*"username"* value=*""*/></td>

</tr>

<tr><td>密码：</td>

<td><input type=*"password"* name=*"password"*

value=*""*/></td>

</tr>

<tr><td colspan=*"2"* align=*"center"*><input type=*"submit"*

value=*"登录"*/></td>

</tr>

</table>

</form>

</body>

dologin.jsp

<body>

<h1>setProperty动作元素</h1>

<!-- 将users类实例化为 myusers对象-->

<jsp:useBean id=*"myusers"* class=*"com.po.users"* scope=*"page"*></jsp:useBean>

<hr>

<!-- 根据表单自动匹配所有的属性 -->

<%--<jsp:setProperty property="\*" name="myusers"/>--%>

<!-- 根据表单匹配所有部分的属性 -->

<%--<jsp:setProperty property="username" name="myusers"/>--%>

<%--跟表单无关,通过手工赋值给属性 --%>

<%--<jsp:setProperty property="username" name="myusers" value="lisi"/>--%>

<%--<jsp:setProperty property="password" name="myusers" value="88888"/>--%>

<!--通过URL传参给属性赋值 -->

<jsp:setProperty property=*"username"* name=*"myusers"*/>

<jsp:setProperty property=*"password"* name=*"myusers"* param=*"mypass"*/>

<!-- 用传统的方式获取用户名和密码 -->

<%--

用户名：<%=myusers.getUsername() %> <br>

密码：<%=myusers.getPassword() %> <br>--%>

<!-- 用getProperty方式获取用户名和密码 -->

用户名：<jsp:getProperty property=*"username"* name=*"myusers"*/><br>

密码： <jsp:getProperty property=*"password"* name=*"myusers"*/><br>

<a href=*"testscope.jsp"*>测试JavaBean的四个作用域范围</a>

<%

request.getRequestDispatcher("testscope.jsp").

forward(request,response);

%>

</body>

Testscope.jsp

<body>

<h1>JavaBean的四个作用域范围</h1>

<jsp:useBean id=*"myusers"* class=*"com.po.users"* scope=*"page"*></jsp:useBean>

<hr>

用户名：<jsp:getProperty property=*"username"* name=*"myusers"*/><br>

密码： <jsp:getProperty property=*"password"* name=*"myusers"*/><br>

<!-- 使用内置对象获取用户名和密码 -->

<%--

用户名：<%=((users)application.getAttribute("myusers")).getUsername() %><br>

密码： <%=((users)application.getAttribute("myusers")).getPassword() %><br>

--%>

<%--

用户名：<%=((users)session.getAttribute("myusers")).getUsername() %><br>

密码： <%=((users)session.getAttribute("myusers")).getPassword() %><br>

--%>

<%--

用户名：<%=((users)request.getAttribute("myusers")).getUsername() %><br>

密码： <%=((users)request.getAttribute("myusers")).getPassword() %><br>

--%>

<%

String username=**null**;

String password=**null**;

**if**(pageContext.getAttribute("myusers")!=**null**){

username=((users)pageContext.getAttribute("myusers")).getUsername();

password=((users)pageContext.getAttribute("myusers")).getPassword();

}

%>

用户名：<%=username%><br>

密码： <%=password%><br> page作用域只能在当前页面获取！！

</body>

1：Class class1=Foo.class; ------->实际在告诉我们，任何一个类都有一个隐含的静态成员变量  
2: Class class2=foo1.getClass()---->已知该类的对象，通过该类的对象通过getClass方法  
class1和class2表示Foo类的类 类型（class type） Foo这个类本身就是一个对象，Foo类的对象：  
3:Class class3=null; class3=Class.forName();  
class1==class2==class3