## 1.Jsp中request传递参数汉字乱码

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

< !-- 告诉浏览器该jsp格式为utf-8，此时注意网页本身(右击->properties)的编码是否为utf-8 -->  
  
在head内加入：  
<!-- 告诉服务器,这里采用的是utf-8 -->  
< meta http-equiv="Content-Type" content="text/html; charset=utf-8">  
  
3.1对于get请求：  
String name2 = new String(name.getBytes("iso-8859-1"),"UTF-8");  
  
3.2 对于post请求  
 request.setCharacterEncoding("utf-8");//对于post方法有效，对get无效.一定要放在解析的第一行，否则不行。  
 String name = request.getParameter("name");

<%@ **page** language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  
<%@ **page** import="java.io.\*,java.util.\*" %>  
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  
<html>  
<head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">  
 <title>菜鸟教程(runoob.com)</title>  
</head>  
<body>  
<h1>使用 GET 方法读取数据</h1>  
<ul>  
 <li><p><b>站点名:</b>  
 **<%** String name =request.getParameter("name");  
 String name2 = new String(name.getBytes("iso-8859-1"),"UTF-8");  
 **%>  
 <%=** name2**%>** </p></li>  
 <li><p><b>网址:</b>  
 **<%=** request.getParameter("url")**%>** </p></li>  
</ul>  
</body>  
</html>

在浏览器中输入：

<http://localhost:8080/mavenWebDemo/test.jsp?name=%E8%8F%9C%E9%B8%9F%E6%95%99%E7%A8%8B&url=http://ww.runoob.com>

输出：



## 2. java统计字符串中每个字符出现的次数

例如String str = “abcaaaefdabbhg”;   
统计该字符串中每个字符出现的次数，输出：   
a====5   
b====3   
c====1   
d====1   
e====1   
f====1   
g====1   
h====1

public static void count(String str){  
//将字符串转化为字符数组  
char[] chars = str.toCharArray();  
//创建一个HashMap名为hm  
HashMap<Character,Integer> hm = new HashMap();  
  
//定义一个字符串c，循环遍历遍历chars数组  
for(char c : chars){  
//containsKey(c),当c不存在于hm中  
if(!hm.containsKey(c)){  
hm.put(c,1);  
}else{  
//否则获得c的值并且加1  
hm.put(c, hm.get(c)+1);  
}  
  
//或者上面的if和else替换成下面这一行  
/\* hm.put(c,hm.containsKey(c) ? hm.get(c)+1:1);\*/  
}  
  
  
for(Character key: hm.keySet()){  
//hm.keySet()代表所有键的集合,进行格式化输出  
System.out.println(key + "====" + hm.get(key));  
}  
}  
  
public static void main(String[] args) {  
// TODO 自动生成的方法存根  
String str = "abcaaaefdabbhg";  
count(str);  
}  
}

## 3.Java中字符串indexof() 的使用方法

Java中字符串中子串的查找共有四种方法(indexof())

indexOf 方法返回一个整数值，指出 String 对象内子字符串的开始位置。如果没有找到子字符串，则返回-1。

如果 startindex 是负数，则 startindex 被当作零。如果它比最大的字符位置索引还大，则它被当作最大的可能索引。

Java中字符串中子串的查找共有四种方法，如下：

1、int indexOf(String str) ：返回第一次出现的指定子字符串在此字符串中的索引。

2、int indexOf(String str, int startIndex)：从指定的索引处开始，返回第一次出现的指定子字符串在此字符串中的索引。

3、int lastIndexOf(String str) ：返回在此字符串中最右边出现的指定子字符串的索引。

4、int lastIndexOf(String str, int startIndex) ：从指定的索引处开始向后搜索，返回在此字符串中最后一次出现的指定子字符串的索引。

private static void testIndexOf() {

String string = "aaa456ac";

//查找指定字符是在字符串中的下标。在则返回所在字符串下标；不在则返回-1.

System.out.println(string.indexOf("b"));//indexOf(String str)；返回结果：-1，"b"不存在

// 从第四个字符位置开始往后继续查找，包含当前位置

System.out.println(string.indexOf("a",3));//indexOf(String str, int fromIndex)；返回结果：6

//（与之前的差别：上面的参数是 String 类型，下面的参数是 int 类型）参考数据：a-97,b-98,c-99

// 从头开始查找是否存在指定的字符

System.out.println(string.indexOf(99));//indexOf(int ch)；返回结果：7

System.out.println(string.indexOf('c'));//indexOf(int ch)；返回结果：7

//从fromIndex查找ch，这个是字符型变量，不是字符串。字符a对应的数字就是97。

System.out.println(string.indexOf(97,3));//indexOf(int ch, int fromIndex)；返回结果：6

System.out.println(string.indexOf('a',3));//indexOf(int ch, int fromIndex)；返回结果：6

//这个就是灵活运用String类提供的方法，拆分提供的字符串。

//String s = "D:\\Android\\sdk\\add-ons";

//System.out.println(s);

//while (s.lastIndexOf("\\") > 0) {

// s = s.substring(0, s.lastIndexOf("\\"));

// System.out.println(s);

//}

}

## 4.java Random.nextInt()方法的使用

public int nextInt(int n)

该方法的作用是生成一个随机的int值，该值介于[0,n)的区间，也就是0到n之间的随机int值，包含0而不包含n。

import java.util.Random;

import org.junit.Test;

public class RandomDemo {

@Test

public void Demo(){

Random rnd = new Random();

int code = rnd.nextInt(8999) + 1000;

System.out.println("code:"+code);

}

@Test

public void Demo1(){

Random r = new Random();

int nextInt = r.nextInt();

Random r1 = new Random(10);

int nextInt2 = r1.nextInt();

System.out.println("nextInt:"+nextInt);

System.out.println("nextInt2:"+nextInt2);

}

/\*\*

\* 生成[0,1.0)区间的小数

\*

\*/

@Test

public void Demo2(){

Random r = new Random();

double d1 = r.nextDouble();

System.out.println("d1:"+d1);

}

/\*\*

\* 生成[0,5.0)区间的小数

\*

\*/

@Test

public void Demo3(){

Random r = new Random();

double d2 = r.nextDouble()\* 5;

System.out.println("d1:"+d2);

}

/\*\*

\* 生成[1,2.5)区间的小数

\*

\*/

@Test

public void Demo4(){

Random r = new Random();

double d3 = r.nextDouble() \* 1.5 + 1;

System.out.println("d1:"+d3);

}

/\*\*

\* 生成任意整数

\*

\*/

@Test

public void Demo5(){

Random r = new Random();

int n1 = r.nextInt();

System.out.println("d1:"+n1);

}

/\*\*

\* 生成[0,10)区间的整数

\*

\*/

@Test

public void Demo6(){

Random r = new Random();

int n2 = r.nextInt(10);

int n3 = Math.abs(r.nextInt() % 10);

System.out.println("n2:"+n2);

System.out.println("n3:"+n3);

}

/\*\*

\* 生成[0,10]区间的整数

\*

\*/

@Test

public void Demo7(){

Random r = new Random();

int n3 = r.nextInt(11);

int n4 = Math.abs(r.nextInt() % 11);

System.out.println("n3:"+n3);

System.out.println("n4:"+n4);

}

/\*\*

\* 生成[-3,15)区间的整数

\*

\*/

@Test

public void Demo8(){

Random r = new Random();

int n4 = r.nextInt(18) - 3;

int n5 = Math.abs(r.nextInt() % 18) - 3;

System.out.println("n4:"+n4);

System.out.println("n5:"+n5);

}

}

## 5.java中charAt()方法的使用

public charcharAt(int index)

charAt(int index)方法是一个能够用来检索特定索引下的字符的String实例的方法.

charAt()方法返回指定索引位置的char值。索引范围为0~length()-1.

如: str.charAt(0)检索str中的第一个字符,str.charAt(str.length()-1)检索最后一个字符.

public static void main(String[] args) {

String s = "bejing welcome you";

System.out.println(s.charAt(1)); //e

System.out.println(s.charAt(5)); //g

System.out.println(s.charAt(15)); //y

}

## 6.BigDecimal加减乘除运算

使用BigDecimal要用String来够造，要做一个加法运算，需要先将两个浮点数转为String，然后够造成BigDecimal。

在其中一个上调用add方法，传入另一个作为参数，然后把运算的结果（BigDecimal）再转换为浮点数。

public static double add(double v1,double v2)

public static double sub(double v1,double v2)

public static double mul(double v1,double v2)

public static double div(double v1,double v2)

public static double div(double v1,double v2,int scale)

public static double round(double v,int scale)

import java.math.BigDecimal;

/\*\*

\* 由于Java的简单类型不能够精确的对浮点数进行运算，这个工具类提供精

\* 确的浮点数运算，包括加减乘除和四舍五入。

\*/

public class Arith{ //默认除法运算精度

private static final int DEF\_DIV\_SCALE = 10; //这个类不能实例化

private Arith(){

}

/\*\*

\* 提供精确的加法运算。

\* @param v1 被加数

\* @param v2 加数

\* @return 两个参数的和

\*/

public static double add(double v1,double v2){

BigDecimal b1 = new BigDecimal(Double.toString(v1));

BigDecimal b2 = new BigDecimal(Double.toString(v2));

return b1.add(b2).doubleValue();

}

/\*\*

\* 提供精确的减法运算。

\* @param v1 被减数

\* @param v2 减数

\* @return 两个参数的差

\*/

public static double sub(double v1,double v2){

BigDecimal b1 = new BigDecimal(Double.toString(v1));

BigDecimal b2 = new BigDecimal(Double.toString(v2));

return b1.subtract(b2).doubleValue();

}

/\*\*

\* 提供精确的乘法运算。

\* @param v1 被乘数

\* @param v2 乘数

\* @return 两个参数的积

\*/

public static double mul(double v1,double v2){

BigDecimal b1 = new BigDecimal(Double.toString(v1));

BigDecimal b2 = new BigDecimal(Double.toString(v2));

return b1.multiply(b2).doubleValue();

}

/\*\*

\* 提供（相对）精确的除法运算，当发生除不尽的情况时，精确到

\* 小数点以后10位，以后的数字四舍五入。

\* @param v1 被除数

\* @param v2 除数

\* @return 两个参数的商

\*/

public static double div(double v1,double v2){

return div(v1,v2,DEF\_DIV\_SCALE);

}

/\*\*

\* 提供（相对）精确的除法运算。当发生除不尽的情况时，由scale参数指

\* 定精度，以后的数字四舍五入。

\* @param v1 被除数

\* @param v2 除数

\* @param scale 表示表示需要精确到小数点以后几位。

\* @return 两个参数的商

\*/

public static double div(double v1,double v2,int scale){

if(scale<0){

throw new IllegalArgumentException(

"The scale must be a positive integer or zero");

}

BigDecimal b1 = new BigDecimal(Double.toString(v1));

BigDecimal b2 = new BigDecimal(Double.toString(v2));

return b1.divide(b2,scale,BigDecimal.ROUND\_HALF\_UP).doubleValue();

}

/\*\*

\* 提供精确的小数位四舍五入处理。

\* @param v 需要四舍五入的数字

\* @param scale 小数点后保留几位

\* @return 四舍五入后的结果

\*/

public static double round(double v,int scale){

if(scale<0){

throw new IllegalArgumentException("The scale must be a positive integer or zero");

}

BigDecimal b = new BigDecimal(Double.toString(v));

BigDecimal one = new BigDecimal("1");

return b.divide(one,scale,BigDecimal.ROUND\_HALF\_UP).doubleValue();

}

};

## 7.数组转换成List集合

方法一 ：通过add把数组中的数据循环添加到List集合中

方法二 ：采用java中集合自带的asList()方法完成转换

public static void ArrayToList(Object[] obj){  
 System.*out*.println("方法一-----------------------------------------");  
 List<Object> list = new ArrayList<Object>();  
 //String数组转List集合  
 for(Object str : obj){  
 list.add(str);  
 }  
 // 输出List集合  
 for(Object li : list){  
 System.*out*.println(li.toString());  
 }  
  
 System.*out*.println("方法二-----------------------------------------");  
  
 // String数组转List集合  
 List<Object> list1 = Arrays.*asList*(obj);  
 // 输出List集合  
 for(Object li : list1){  
 System.*out*.println(li.toString());  
 }  
}

## 8.List集合转换成数组

方法一 ：把List中的数据循环添加到数组中

方法二 ：采用集合的toArray()方法直接把List集合转换成数组

public static void ListToArray(List<Object> list){  
 System.*out*.println("方法一-----------------------------------------");  
 Object[] strArray = new Object[list.size()];  
 // List转换成数组  
 for(int i=0;i<strArray.length;i++){  
 strArray[i]=list.get(i);  
 }  
  
 // 输出List集合  
 for(Object s : strArray){  
 System.*out*.println(s.toString());  
 }  
  
  
 System.*out*.println("方法二-----------------------------------------");  
 // List转换成数组  
 Object[] str = list.toArray(new Object[list.size()]);  
 // 输出List集合  
 for(Object s:str){  
 System.*out*.println(s.toString());  
 }  
}

不管是数组转换成集合，还是集合转换成数组，都要注意转换类型的一致性，String[]数组转String类型的集合，当需要使用int，double等集合的时候，需要使用对应的对象   
如：数组int[]用Integer[]，double[]用Double[]   
因为List集合是对象的集合，而int、double等不是对象，所以需要用字段的对应对象类

## 9.Map的遍历

### 通过获取所有的key按照key来遍历

public static void getMapvalue1(Map<Object,Object> map){  
 System.*out*.println("Map 遍历方法一:");  
 for(Object o : map.keySet()){  
 System.*out*.println("key:"+o.toString()+" value:"+map.get(o).toString());  
 }  
}

### 通过Map.entrySet使用iterator遍历key和value

public static void getMapValue4(Map<Object,Object> map){  
 System.*out*.println("Map 遍历方法四:");  
 Iterator<Map.Entry<Object, Object>> iterator =map.entrySet().iterator();  
 while (iterator.hasNext()){  
 Map.Entry<Object,Object> entry = iterator.next();  
 System.*out*.println("key:"+entry.getKey().toString()+" value:"+entry.getValue().toString());  
 }  
}

### 通过Map.entrySet遍历key和value

public static void getMapValue2(Map<Object,Object> map){  
 System.*out*.println("Map 遍历方法二:");  
 for(Map.Entry<Object,Object> entry : map.entrySet()){  
 System.*out*.println("key:"+entry.getKey().toString()+" value:"+entry.getValue().toString());  
 }  
}

### 通过Map.values()遍历所有的value，但不能遍历key

public static void getMapValue3(Map<Object,Object> map){  
 System.*out*.println("Map 遍历方法三:");  
 for(Object o : map.values()){  
 System.*out*.println("value:"+o.toString());  
 }  
}

测试方法：

public static void main(String[] args) {  
 Map<Object,Object> map = new HashMap<Object,Object>();  
 for(int i=1;i<=20;i++){  
 map.put(i,"mapValue"+i);  
 }  
 *getMapvalue1*(map);  
 *getMapValue2*(map);  
 *getMapValue3*(map);  
 *getMapValue4*(map);  
}

## 10.生成唯一Token

*/\*\*  
 \* 生成Token  
 \* Token：gBu2bAZ7oC4eKxpVrWtI6A==  
 \** ***@return*** *\*/*public static String makeToken(){ //checkException  
 // 7346734837483 834u938493493849384 43434384  
 String token = (System.*currentTimeMillis*() + new Random().nextInt(999999999)) + "";  
 //数据指纹 128位长 16个字节 md5  
 try {  
 MessageDigest md = MessageDigest.*getInstance*("md5");  
 byte md5[] = md.digest(token.getBytes());  
 //base64编码--任意二进制编码明文字符 adfsdfsdfsf  
 BASE64Encoder encoder = new BASE64Encoder();  
 return encoder.encode(md5);  
 } catch (NoSuchAlgorithmException e) {  
 throw new RuntimeException(e);  
 }  
}  
  
public static void main(String[] args) {  
 String s = *makeToken*();  
 System.*out*.println(s);//wkra+s2rBtLDRAhTgBuP4w==  
}