**字符串的常见操作**

                   a、charAt方法

该方法的作用是按照索引值(规定字符串中第一个字符的索引值是0，第二个字符的索引值是1，依次类推)，获得字符串中的指定字符。例如：

         String s = “abc”;

         char c = s.chatAt(1);

则变量c的值是’b’。

                   b、compareTo方法

该方法的作用是比较两个字符串的大小，比较的原理是依次比较每个字符的字符编码。首先比较两个字符串的第一个字符，如果第一个字符串的字符编码大于第二个的字符串的字符编码，则返回大于0的值，如果小于则返回小于0的值，如果相等则比较后续的字符，如果两个字符串中的字符编码完全相同则返回0。

                            例如：

                                     String s = “abc”;

                                     String s1 = “abd”;

                                     int value = s.compareTo(s1);

                            则value的值是小于0的值，即-1。

                            在String类中还存在一个类似的方法compareToIgnoreCase，这个方法是忽略字符的大小写进行比较，比较的规则和compareTo一样。例如：

                                     String s = “aBc”;

                                     String s1 = “ABC”;

                                     int value = s. compareToIgnoreCase (s1);

                            则value的值是0，即两个字符串相等。

                   c、concat方法

该方法的作用是进行字符串的连接，将两个字符串连接以后形成一个新的字符串。例如：

                                     String s = “abc”;

                                     String s1 = “def”;

                                     String s2 = s.concat(s1);

则连接以后生成的新字符串s2的值是”abcdef”，而字符串s和s1的值不发生改变。如果需要连接多个字符串，可以使用如下方法：

                                     String s = “abc”;

                                     String s1 = “def”;

                                     String s2 = “1234”;

         String s3 = s.concat(s1).concat(s2);

则生成的新字符串s3的值为”abcdef1234”。

其实在实际使用时，语法上提供了一种更简单的形式，就是使用“+”进行字符串的连接。例如：

         String s = “abc” + “1234”;

则字符串s的值是”abc1234”，这样书写更加简单直观。

而且使用“+”进行连接，不仅可以连接字符串，也可以连接其他类型。但是要求进行连接时至少有一个参与连接的内容是字符串类型。而且“+”匹配的顺序是从左向右，如果两边连接的内容都是基本数字类型则按照加法运算，如果参与连接的内容有一个是字符串才按照字符串进行连接。

例如：

         int a = 10;

         String s = “123” + a + 5;

则连接以后字符串s的值是“123105”，计算的过程为首先连接字符串”123”和变量a的值，生成字符串”12310”，然后使用该字符串再和数字5进行连接生成最终的结果。

而如下代码：

         int a = 10;

         String s = a + 5 + “123”;

则连接以后字符串s的值是”15123”，计算的过程为首先计算a和数字5，由于都是数字型则进行加法运算或者数字值15，然后再使用数字值15和字符串”123”进行连接获得最终的结果。

而下面的连接代码是错误的：

         int a = 12;

         String s = a + 5 + ‘s’;

因为参与连接的没有一个字符串，则计算出来的结果是数字值，在赋值时无法将一个数字值赋值给字符串s。

                   d、endsWith方法

该方法的作用是判断字符串是否以某个字符串结尾，如果以对应的字符串结尾，则返回true。

例如：

         String s = “student.doc”;

         boolean b = s.endsWith(“doc”);

则变量b的值是true。

                   e、equals方法

该方法的作用是判断两个字符串对象的内容是否相同。如果相同则返回true，否则返回false。例如：

         String s = “abc”;

         String s1 = new String(“abc”);

         boolean b = s.equals(s1);

而使用“==”比较的是两个对象在内存中存储的地址是否一样。例如上面的代码中，如果判断：

         boolean b = (s == s1);

则变量b的值是false，因为s对象对应的地址是”abc”的地址，而s1使用new关键字申请新的内存，所以内存地址和s的”abc”的地址不一样，所以获得的值是false。

在String类中存在一个类似的方法equalsIgnoreCase，该方法的作用是忽略大小写比较两个字符串的内容是否相同。例如：

         String s = “abc”;

         String s1 =”ABC”;

         boolean b = s. equalsIgnoreCase (s1);

则变量b的值是true。

                   f、getBytes方法

该方法的作用是将字符串转换为对应的byte数组，从而便于数据的存储和传输。例如：

         String s = “计算机”;

         byte[] b = s.getBytes();   //使用本机默认的字符串转换为byte数组

         byte[] b = s.getBytes(“gb2312”); //使用gb2312字符集转换为byte数组

                            在实际转换时，一定要注意字符集的问题，否则中文在转换时将会出现问题。

                   g、indexOf方法

该方法的作用是查找特定字符或字符串在当前字符串中的起始位置，如果不存在则返回-1。例如：

                                     String s = “abcded”;

                                     int index = s.indexOf(‘d’);

                                     int index1 = s.indexOf(‘h’);

则返回字符d在字符串s中第一次出现的位置，数值为3。由于字符h在字符串s中不存在，则index1的值是-1。

                            当然，也可以从特定位置以后查找对应的字符，例如：

                                     int index = s.indexOf(‘d’,4);

则查找字符串s中从索引值4(包括4)以后的字符中第一个出现的字符d，则index的值是5。

由于indexOf是重载的，也可以查找特定字符串在当前字符串中出现的起始位置，使用方式和查找字符的方式一样。

另外一个类似的方法是lastIndexOf方法，其作用是从字符串的末尾开始向前查找第一次出现的规定的字符或字符串，例如：

                                     String s = “abcded”;

                                     int index = s. lastIndexOf(‘d’);

                            则index的值是5。

                   h、length方法

该方法的作用是返回字符串的长度，也就是返回字符串中字符的个数。中文字符也是一个字符。例如：

         String s = “abc”;

         String s1 = “Java语言”;

         int len = s.length();

         int len1 = s1.length();

则变量len的值是3，变量len1的值是6。

                   i、replace方法

该方法的作用是替换字符串中所有指定的字符，然后生成一个新的字符串。经过该方法调用以后，原来的字符串不发生改变。例如：

         String s = “abcat”;

         String s1 = s.replace(‘a’,’1’);

该代码的作用是将字符串s中所有的字符a替换成字符1，生成的新字符串s1的值是”1bc1t”，而字符串s的内容不发生改变。

如果需要将字符串中某个指定的字符串替换为其它字符串，则可以使用replaceAll方法，例如：

         String s = “abatbac”;

         String s1 = s.replaceAll(“ba”,”12”);

该代码的作用是将字符串s中所有的字符串”ab”替换为”12”，生成新的字符串”a12t12c”，而字符串s的内容也不发生改变。

如果只需要替换第一个出现的指定字符串时，可以使用replaceFirst方法，例如：

         String s = “abatbac”;

         String s1 = s. replaceFirst (“ba”,”12”);

该代码的作用是只将字符串s中第一次出现的字符串”ab”替换为字符串”12”，则字符串s1的值是”a12tbac”，字符串s的内容也不发生改变。

                   j、split方法

该方法的作用是以特定的字符串作为间隔，拆分当前字符串的内容，一般拆分以后会获得一个字符串数组。例如：

         String s = “ab,12,df”;

         String s1[] = s.split(“,”);

该代码的作用是以字符串”,”作为间隔，拆分字符串s，从而得到拆分以后的字符串数字s1，其内容为：{“ab”,”12”,”df”}。

该方法是解析字符串的基础方法。

如果字符串中在内部存在和间隔字符串相同的内容时将拆除空字符串，尾部的空字符串会被忽略掉。例如：

         String s = “abbcbtbb”;

         String s1[] = s.split(“b”);

则拆分出的结果字符串数组s1的内容为：{“a”,””,”c”,”t”}。拆分出的中间的空字符串的数量等于中间间隔字符串的数量减一个。例如：

         String s = “abbbcbtbbb”;

         String s1[] = s.split(“b”);

则拆分出的结果是：{“a”,””,””,”c”,”t”}。最后的空字符串不论有多少个，都会被忽略。

如果需要限定拆分以后的字符串数量，则可以使用另外一个split方法，例如：

         String s = “abcbtb1”;

         String s1[] = s.split(“b”,2);

该代码的作用是将字符串s最多拆分成包含2个字符串数组。则结果为：{“a”,”cbtb1”}。

         如果第二个参数为负数，则拆分出尽可能多的字符串，包括尾部的空字符串也将被保留。

                   k、startsWith方法

该方法的作用和endsWith方法类似，只是该方法是判断字符串是否以某个字符串作为开始。例如：

         String s = “TestGame”;

         boolean b = s.startsWith(“Test”);

则变量b的值是true。

                   l、substring方法

该方法的作用是取字符串中的“子串”，所谓“子串”即字符串中的一部分。例如“23”是字符串“123”的子串。

字符串“123”的子串一共有6个：”1”、”2”、”3”、”12”、”23”、”123”。而”32”不是字符串”123”的子串。

例如：

         String s = “Test”;

         String s1 = s.substring(2);

则该代码的作用是取字符串s中索引值为2(包括)以后的所有字符作为子串，则字符串s1的值是”st”。

如果数字的值和字符串的长度相同，则返回空字符串。例如：

         String s = “Test”;

         String s1 = s.substring(4);

则字符串s1的值是””。

如果需要取字符串内部的一部分，则可以使用带2个参数的substring方法，例如：

         String s = “TestString”;

         String s1 = s.substring(2,5);

则该代码的作用是取字符串s中从索引值2(包括)开始，到索引值5(不包括)的部分作为子串，则字符串s1的值是”stS”。

下面是一个简单的应用代码，该代码的作用是输出任意一个字符串的所有子串。代码如下：

                                     String s = “子串示例”;

                                     int len = s.length(); //获得字符串长度

                                     for(int begin = 0;begin < len – 1;begin++){ //起始索引值

                                               for(int end = begin + 1;end <= len;end++){ //结束索引值

                                                        System.out.println(s.substring(begin,end));

                                               }

                                     }

在该代码中，循环变量begin代表需要获得的子串的起始索引值，其变化的区间从第一个字符的索引值0到倒数第二个字符串的索引值len -2，而end代表需要获得的子串的结束索引值，其变化的区间从起始索引值的后续一个到字符串长度。通过循环的嵌套，可以遍历字符串中的所有子串。

                   m、toCharArray方法

该方法的作用和getBytes方法类似，即将字符串转换为对应的char数组。例如：

         String s = “abc”;

         char[] c = s.toCharArray();

则字符数组c的值为：{‘a’,’b’,’c’}。

                   n、toLowerCase方法

                            该方法的作用是将字符串中所有大写字符都转换为小写。例如：

                                     String s = “AbC123”;

                                     String s1 = s.toLowerCase();

                            则字符串s1的值是”abc123”，而字符串s的值不变。

类似的方法是toUpperCase，该方法的作用是将字符串中的小写字符转换为对应的大写字符。例如：

                                     String s = “AbC123”;

                                     String s1 = s. toUpperCase ();

则字符串s1的值是”ABC123”，而字符串s的值也不变。

                   o、trim方法

该方法的作用是去掉字符串开始和结尾的所有空格，然后形成一个新的字符串。该方法不去掉字符串中间的空格。例如：

                                     String s = “   abc abc 123 “;

                                     String s1 = s.trim();

                            则字符串s1的值为：” abc abc 123”。字符串s的值不变。

                   p、valueOf方法

该方法的作用是将其它类型的数据转换为字符串类型。需要注意的是，基本数据和字符串对象之间不能使用以前的强制类型转换的语法进行转换。

另外，由于该方法是static方法，所以不用创建String类型的对象即可。例如：

         int n = 10;

         String s = String.valueOf(n);

则字符串s的值是”10”。虽然对于程序员来说，没有发生什么变化，但是对于程序来说，数据的类型却发生了变化。

介绍一个简单的应用，判断一个自然数是几位数字的逻辑代码如下：

         int n = 12345;

         String s = String.valueOf(n);

         int len = s.length();

则这里字符串的长度len，就代表该自然数的位数。这种判断比数学判断方法在逻辑上要简单一些。

关于String类的使用就介绍这么多，其它的方法以及这里到的方法的详细声明可以参看对应的API文档。