2、具体内容

在实际的项目开发过程之中，只要是项目都一定会存在有String类的定义，所在掌握这个类之中的常用处理方法对我们开发而言是非常重要的。

■JavaDoc简介

在以后进行开发的过程之中肯定要大量的去使用Java的API文档（JavaDoc），这个文档可以直接通过oracle的在线访问进行查看。地址：https://docs.oracle.com/javase/9/docs/api/overview-summary.html；

在JDK1.9之前，所有的Java中的常用类库都会在JVM启动的时候进行全部的加载，这样实际上性能会有所下降，所在在JDK1.9开始提供的模块化设计，将一些程序类放在了不同的模块里面。

Java SE

Module	Description
java.activation	Defines the JavaBeans Activation Framework (JAF) API.
java.base	Defines the foundational APIs of the Java SE Platform.
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
java.corba	Defines the Java binding of the OMG CORBA APIs, and the RMI-IIOP API.
java.datatransfer	Defines the API for transferring data between and within applications.
java.desktop	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.
java.instrument	Defines services that allow agents to instrument programs running on the JVM.

在模块之中会包含有大量的程序开发包：

Packages

Exports

Package	Description
java.io	Provides for system input and output through data streams, serialization and the file system.
java.lang	Provides classes that are fundamental to the design of the Java programming language.
java.lang.annotation	Provides library support for the Java programming language annotation facility.
java.lang.invoke	The `java.lang.invoke` package contains dynamic language support provided directly by the Java core class libraries and virtual machine.

如果现在想要去看String类的相关定义，则可以打开java.lang这个包，String是一个系统提供的较为标准的类，所以现在以这个类的文档结构进行说明，一般文档里面的组成会有如下几个部分：

- 类的完整定义：

Module java.base
Package java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

类相关说明信息：

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

成员属性摘要：

Field Summary

Fields

Modifier and Type	Field	Description
static Comparator<String>	CASE_INSENSITIVE_ORDER	A Comparator that orders String objects as by compareToIgnoreCase.

构造方法摘要：如果看见有“Deprecated”描述的方法表示不建议使用

Constructors

Constructor	Description
String()	Initializes a newly created String object so that it represents an empty character sequence.
String(byte[] bytes)	Constructs a new String by decoding the specified array of bytes using the platform's default charset.
String(byte[] ascii, int hibyte)	**Deprecated.** This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the String constructors that take a Charset, charset name, or that use the platform's default charset.
String(byte[] bytes, int offset, int length)	Constructs a new String by decoding the specified subarray of bytes using the platform's default charset.

方法摘要：左边为返回值，右边为方法名称和相应的参数

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods | Deprecated Methods

Modifier and Type	Method	Description
char	charAt(int index)	Returns the char value at the specified index.
IntStream	chars()	Returns a stream of int zero-extending the char values from this sequence.
int	codePointAt(int index)	Returns the character (Unicode code point) at the specified index.
int	codePointBefore(int index)	Returns the character (Unicode code point) before the specified index.
int	codePointCount(int beginIndex, int endIndex)	Returns the number of Unicode code points in the specified text range of this String.

最后的部分就是对于方法和成员的详细解释了。

详细的说明：

charAt

```
public char charAt(int index)
```

Returns the char value at the specified index. An index ranges from 0 to length() - 1. The first char value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

If the char value specified by the index is a surrogate, the surrogate value is returned.

Specified by:
charAt in interface CharSequence

Parameters:
index - the index of the char value.

Returns:
the char value at the specified index of this string. The first char value is at index 0.

Throws:
IndexOutOfBoundsException - if the index argument is negative or not less than the length of this string.

文档一般都会有一些“假”的中文翻译版（自动翻译的），对于这种翻译版最好别使用，整个Java的开发涉及到的文档有几十份，没有中文，如果现在都没习惯于阅读英文文档，后面的文档你就看不懂了。

■字符串与字符数组

在JDK1.9以前，所有的String都利用了字符数组实现了包装的处理，所以在String类里面是提供有相应的转换处理方法的，这些方法包含构造方法与普通方法两类。

NO.	方法名称	类型	描述
01	public String(char[] value)	构造	将传入的字符数组变为字符串
02	public String(char[] value,int offset,int count)	构造	将部分字符数组变为字符串
03	public char charAt(int index)	普通	获取指定索引位置的字符
04	public char[] toCharArray()	普通	将字符串中的数据以字符数组的形式返回

范例：观察charAt()方法

```
public class StringDemo {
	public static void main(String args[]) {
		String str = "www.mldn.cn" ;
		char c = str.charAt(5) ;
		System.out.println(c) ;
	}
}
```

利用charAt()是可以获取某一个指定索引位置的字符，但是程序之中的索引下标都是从0开始的。

在程序语言之中，最早一直强调的就是字符串应该使用字符数组进行描述，所在这一操作在String类的方法中也是有所体现的。

范例：实现字符串与字符数组的转换

```
public class StringDemo {
	public static void main(String args[]) {
		String str = "helloworld" ;
		char [] result = str.toCharArray() ; // 将字符串变为字符数组
		for (int x = 0 ;x < result.length ; x ++) {
			result[x] -= 32 ;	// 编码减少32
		}
		// 将处理后的字符数组交给String变为字符串
		String newStr = new String(result) ;
		System.out.println(newStr) ;
		System.out.println(new String(result,0,5)) ;
	}
}
```

现在假设说要求做一个验证功能呢，判断某一个字符串中的数据是否全部由数字所组成，那么这个时候可以采用如下操作：

- 如果要想判断字符串中的每一位最好的做法是将字符串变为字符数组；
- 可以判断每一个字符是否在数字的范围之内（‘0’~‘9’）；
- 如果有一维不是数字则表示验证失败

范例：实现字符串的数据检查

```
public class StringDemo {
	public static void main(String args[]) {
		String str = "helloworld" ;
		System.out.println(isNumber(str) ? "由数字所组成" : "不是由数字所组成") ;
		System.out.println(isNumber("123") ? "由数字所组成" : "不是由数字所组成") ;
	}
	// 该方法主要是判断字符串是否由数字所组成
	public static boolean isNumber(String str) {
		char [] result = str.toCharArray() ; // 将字符串变为字符数组
		for (int x = 0 ; x < result.length ; x ++) { // 依次判断
			if (result[x] < '0' || result[x] > '9') {
				return false ;	// 后面不再判断
			}
		}
		return true ;
	}
}
```

在实际开发之中处理中文的时候往往使用char类型，因为其可以包含中文数据。

■字符串与字节数组

字符串与字节数组之间也可以实现转换的处理操作，但是需要提醒一下，当进行了字符串与字节转换时，其主要目的是为了进行二进制的数据传输，或者是进行编码转换。

NO.	方法名称	类型	描述
01	public String(byte[] bytes)	构造	将全部的字节数组变为字符串
02	public String(byte[] bytes,int offset,int length)	构造	将部分的字节数组变为字符串
03	public byte[] getBytes()	普通	将字符串转为字节数组
04	public byte[] getBytes(String charsetName) throws UnsupportedEncodingException	普通	编码转换

范例：观察字节与字符串的转换

```
public class StringDemo {
	public static void main(String args[]) {
		String str = "helloworld" ;
		byte data[] = str.getBytes() ;	// 将字符串变为字节数组
		for (int x = 0 ; x < data.length ; x ++) {
			data[x] -= 32 ;
		}
		System.out.println(new String(data)) ;
```

```
        System.out.println(new String(data,0,5)) ;
    }
}
```

字节本身是有长度限制的，一个字节最多可以保存的范围是：-128~127之间。

■字符串比较

字符串比较里面最为常用的方法就是equals()方法，但是这个方法需要注意的是其实进行大小写区分的，而除了equals()之外还有许多的比较方法：

NO.	方法名称	类型	描述
01	public boolean equals(Object anObject)	普通	区分大小写的相等判断
02	public boolean equalsIgnoreCase(String anotherString)	普通	不区分大小写比较
03	public int compareTo(String anotherString)	普通	进行字符串大小比较,该方法范湖一个int数据，该数据由三种取值：大于（>0）、小于（<0）、等于（=0）
04	public int compareToIgnoreCase(String str)	普通	不区分大小写进行字符串大小比较

范例：观察大小写比较

```
public class StringDemo {
    public static void main(String args[]) {
        String strA = "mldn" ;
        String strB = "MLDN" ;
        System.out.println(strA.equals(strB)) ;
    }
}
```

范例：不区分大小写比较

```
public class StringDemo {
    public static void main(String args[]) {
        String strA = "mldn" ;
        String strB = "MLDN" ;
        System.out.println(strA.equalsIgnoreCase(strB)) ;
    }
}
```

范例：进行大小比较

```
public class StringDemo {
    public static void main(String args[]) {
        String strA = "mldn" ;
        String strB = "mldN" ;
        System.out.println(strA.compareTo(strB)) ;    // 32
        System.out.println(strB.compareTo(strA)) ;    // -32
        System.out.println("Hello".compareTo("Hello")) ; // 0
    }
}
```

compareTo()方法后面还会有更加详细的解释，对于此方法很重要，而后为了可以实现忽略大小写的比较也可以使用compareToIgnoreCase()方法实现。

范例：忽略大小写比较

```
public class StringDemo {
    public static void main(String args[]) {
        String strA = "mldn" ;
        String strB = "mldN" ;
        System.out.println(strA.compareToIgnoreCase(strB)) ; // 0
    }
}
```

由于此时的内容一样，所以在不计较大小写的情况下，两者的比较结果就是相同的。

■字符串查找

从一个完整的字符串之中查找子字符串的存在就属于字符串查找操作，在String类里面一共提供有如下的几个查找方法：

NO.	方法名称	类型	描述
01	public boolean contains(String s)	普通	判断此字符串是否存在
02	public int indexOf(String str)	普通	从头查找指定字符串的位置，找不到返回0
03	public int indexOf(String str,int fromIndex)	普通	从指定位置查找指定字符串的位置
04	public int lastIndexOf(String str)	普通	由后向前查找指定字符串的位置
05	public int lastIndexOf(String str,int fromIndex)	普通	从指定位置由后向前查找指定字符串的位置
06	public boolean startsWith(String prefix)	普通	判断是否以指定的字符串开头
07	public boolean startsWith(String prefix,int toffset)	普通	由指定位置判断是否以指定的字符串开头
08	public boolean endsWith(String suffix)	普通	判断是否以指定的字符串结尾

范例：判断子字符串是否存在

```
public class StringDemo {
    public static void main(String args[]) {
        String str = "www.mldn.cn" ;
        System.out.println(str.contains("mldn")) ;      // true
        System.out.println(str.contains("hello")) ;     // false
    }
}
```

此方法的操作相对而言比较直观一些，但是这个方法是从JDK1.5之后开始追加到程序之中的功能。在JDK1.5之前如果进行数据的查询往往只能依靠indexOf()方法来完成

范例：判断字符串是否存在

```
public class StringDemo {
    public static void main(String args[]) {
        String str = "www.mldn.cn" ;
        System.out.println(str.indexOf("mldn")) ;
```

```
        System.out.println(str.indexOf("hello")) ;      // -1
        if (str.indexOf("mldn") != -1) {
            System.out.println("数据存在。") ;
        }
    }
}
```

indexOf()是为了进行子字符串位置的查询，在一些开发之中就可以利用此形式来进行一些索引的确定。indexOf()是由前向后查找的，也可以由后向前查找。

范例：利用lastIndexOf()查找

```
public class StringDemo {
    public static void main(String args[]) {
        String str = "www.mldn.cn" ;
        System.out.println(str.lastIndexOf(".")) ;
    }
}
```

```
public class StringDemo {
    public static void main(String args[]) {
        String str = "www.mldn.cn" ;
        System.out.println(str.lastIndexOf(".")) ;
        System.out.println(str.lastIndexOf(".",5)) ;
    }
}
```

在进行字符串查找的时候也需要判断开头或结尾操作。

范例：判断是否以指定的字符串开头或结尾

```
public class StringDemo {
    public static void main(String args[]) {
        String str = "**@@www.mldn.cn##" ;
        System.out.println(str.startsWith("**")) ;
        System.out.println(str.startsWith("@@",2)) ;
        System.out.println(str.endsWith("##")) ;
    }
}
```

后面的许多设计都需要它的支持。

■字符串替换

所谓的字符串替换指的是可以通过一个指定的内容进行指定字符串的替换显示，对于替换方法主要由两个：

NO.	方法名称	类型	描述
01	public String replaceAll(String regex,String replacement)	普通	全部替换
02	public String replaceFirst(String regex,String replacement)	普通	替换首个

范例：实现替换处理

```
public class StringDemo {
```

```
    public static void main(String args[]) {
        String str = "helloworld" ;
        System.out.println(str.replaceAll("l","X")) ;
        System.out.println(str.replaceFirst("l","X")) ;
    }
}
```

替换后面会有更加详细的讲解，并且在开发之中，尤其是涉及的时候替换很重要。

■字符串拆分

在字符串处理的时候还提供一种字符串拆分的处理方法，字符串的拆分操作主要是可以根据一个指定的字符串或者是一些表达式实现字符串的拆分，并且拆分完成的数据将以字符串数组的形式返回。

NO.	方法名称	类型	描述
01	public String[] split(String regex)	普通	按照指定的字符串全部拆分
02	public String[] split(String regex,int limit)	普通	按照指定的字符串拆分为指定个数，后面不拆了

范例：观察字符串拆分处理

```
public class StringDemo {
    public static void main(String args[]) {
        String str = "hello world hello mldn" ;
        String result [] = str.split(" ") ; // 空格拆分
        for (int x = 0 ; x < result.length ; x ++) {
            System.out.println(result[x]) ;
        }
    }
}
```

除了可以全部拆分之外，也可以拆分为指定的个数。

范例：拆分指定个数

```
public class StringDemo {
    public static void main(String args[]) {
        String str = "hello world hello mldn" ;
        String result [] = str.split(" ",2) ; // 空格拆分
        for (int x = 0 ; x < result.length ; x ++) {
            System.out.println(result[x]) ;
        }
    }
}
```

但是在进行拆分的时候有可能会预见拆不了的情况，这个时候最简单的理解就是使用“\\”进行转义。

```
public class StringDemo {
    public static void main(String args[]) {
        String str = "192.168.1.2" ;
        String result [] = str.split("\\.") ; // 空格拆分
        for (int x = 0 ; x < result.length ; x ++) {
```

```
            System.out.println(result[x]) ;
        }
    }
}
```

对于拆分与替换的更多操作后续才会进行说明。

■字符串截取

从一个完整的字符串之中截取出子字符串，对于截取操作有两个方法：

NO.	方法名称	类型	描述
01	public String substring(int beginIndex)	普通	从指定索引截取到结尾
02	public String substring(int beginIndex,int endIndex)	普通	截取指定索引范围中的子字符串

范例：观察子字符串截取操作

```
public class StringDemo {
    public static void main(String args[]) {
        String str = "www.mldn.cn" ;
        System.out.println(str.substring(4)) ;
        System.out.println(str.substring(4,8)) ;
    }
}
```

在实际的开发之中，有些时候的开始或结束索引往往都是通过indexOf()方法计算得来的。

范例：观察截取

```
public class StringDemo {
    public static void main(String args[]) {
        // 字符串结构： “用户id-photo-姓名.后缀”
        String str = "mldn-photo-张三.jpg" ;
        int beginIndex = str.indexOf("-",str.indexOf("photo")) + 1;
        int endIndex = str.lastIndexOf(".") ;
        System.out.println(str.substring(beginIndex,endIndex)) ;
    }
}
```

在以后的实际开发之中，这种通过计算来确定索引的情况非常常见。

■格式化字符串

从JDK1.5开始为了吸引更多的传统开发人员，Java提供了格式化数据的处理操作，类似于C语言之中的格式化输出语句，可以利用占位符实现数据的输出，对于占位符而言，常用的：字符串（%s）、字符（%c）、整数（%d）等来描述。

NO.	方法名称	类型	描述
01	public static String format(String format,各种类型... args)	普通	根据指定结构进行文本格式化显示

范例：观察格式化处理

```
public class StringDemo {
	public static void main(String args[]) {
		String name = "张三" ;
		int age = 18 ;
		double score = 98.765321 ;
		String str = String.format("姓名：%s、年龄：%d、成绩：%5.2f。",name,age,score) ;
		System.out.println(str) ;
	}
}
```

这种格式化的输出操作算是String的一个附加功能。

■其它方法

在String类里面还有一些比较小的方法提供给开发者使用，这些方法如下：

范例：观察字符串连接

```
public class StringDemo {
	public static void main(String args[]) {
		String strA = "www.mldn.cn" ;
		String strB = "www.".concat("mldn").concat(".cn") ;
		System.out.println(strB) ;
		System.out.println(strA == strB) ;  // false
	}
}
```

从整体的运行结果来讲，虽然内容相同，但是发现最终的结果是一个false，证明此操作并没有实现静态的定义。

在字符串定义的时候“""”和“null”不是一个概念，一个表示有实例化对象，一个表示没有实例化对象，而isEmpty()主要是判断字符串的内容，所以一定要在有实例化对象的时候进行调用。

范例：判断空字符串

```
public class StringDemo {
	public static void main(String args[]) {
		String str = "" ;
		System.out.println(str.isEmpty()) ;  // true
		System.out.println("mldn".isEmpty()) ;  // false
	}
}
```

范例：观察length()与trim()

```
public class StringDemo {
	public static void main(String args[]) {
		String str = "   Hello World   " ;
		System.out.println(str.length()) ;    // 长度
		String trimStr = str.trim() ;
```

```
        System.out.println(str) ;
        System.out.println(trimStr) ;
        System.out.println(trimStr.length()) ;
    }
}
```

在进行一些数据输入的时候（用户名和密码）很难保证输入的数据没有空格，有空格的时候数据的查找就会出现错误，那么就必须对输入的数据进行处理，使用trim()。

在String类里面提供有大小写转换，但是这种转换的特征是可以避免非字母的转换。

范例：观察大小写转换

```
public class StringDemo {
    public static void main(String args[]) {
        String str = "Hello World !!!" ;
        System.out.println(str.toUpperCase()) ;
        System.out.println(str.toLowerCase()) ;
    }
}
```

用这样的方式进行转换可以节约我们的开发成本，毕竟如果要自己去写，还需要去判断字母范围，而后再进行转换。

虽然在Java之中String类以及提供有大量的方法了，但是缺少一个首字母大写的方法，这个方法可以由开发者自行设计，利用方法的组合即可。

范例：自定义一个实现首字母大写的方法

```
class StringUtil {
    public static String initcap(String str) {
        if (str == null || "".equals(str)) {
            return str ;      // 原样返回
        }
        if (str.length() == 1) {
            return str.toUpperCase() ;
        }
        return str.substring(0,1).toUpperCase() + str.substring(1) ;
    }
}
public class StringDemo {
    public static void main(String args[]) {
        System.out.println(StringUtil.initcap("hello")) ;
        System.out.println(StringUtil.initcap("m")) ;
    }
}
```

此代码是日后实际开发中必定要使用的程序。