常用类 - Day02

今日学习内容

- StringBuilder、StringBuffer
- Math类
- Random类
- UUID类
- Date类
- SimpleDateFormat类
- Calendar类
- 正则表达式

学习目标

- 掌握StringBuilder的操作
- 掌握String、StringBuilder、StringBuffer三者的区别
- 掌握Math类常用方法，随机数的生成和UUID的使用
- 掌握日期的转换操作（格式化和解析）
- 了解日历类获取年月日和增加天数操作
- 了解正则表达式
- 熟悉查看API，熟悉方法调用

1.1 StringBuffer和StringBuilder类(掌握)

1.1.1 可变字符串概述(了解)

Sting是不可变的，每次内容改变都会在内存中创建新的内存区域，如果现在要把N个字符串连接起来，此时需要在内存中创建N块内存空间，性能很低。此时要解决多个字符串做拼接性能低的问题，此时，**需要对同一字符串对象内容进行修改操作，就需要可变字符串。**

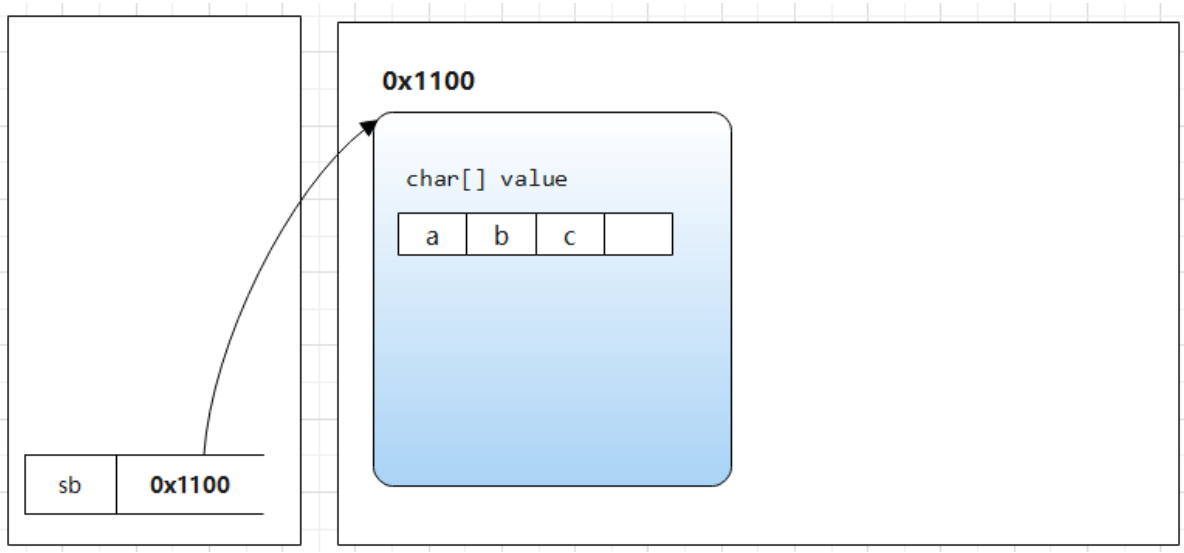
java中提供了两类可变字符串的类型StringBuffer、StringBuilder,位于 java.lang 包中。

1.1.2 StringBuffer(掌握)

StringBuffer封装了一个字符数组，并提供了对该字符数组进行增加、删除、修改、查询的方法。

我们完全可以把StringBuffer看成一个"**字符或者字符串的容器**"!

可变字符串内存图



StringBuffer构造方法

- StringBuffer() : 初始化默认容量是16的可变字符串
- StringBuffer(int capacity) 初始化容量是capacity的可变字符串

```
public static void main(String[] args) {
    // 创建一个可变字符串容器，默认容量是16个字符
    StringBuffer sb = new StringBuffer();
    // 创建一个可变字符串容器，容量是100个字符
    StringBuffer sb2 = new StringBuffer(100);
}
```

StringBuffer常用方法

- **append**
- delete
- setCharAt / replace

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

    StringBuffer sb = new StringBuffer();
    // 追加操作(在末尾添加)
    sb.append("hello");
    sb.append(17);
    System.out.println(sb.toString());  // hello17

    // 删除操作
    //sb.delete(0,5); // 17

    // 修改操作
    // sb.setCharAt(0,'H');  // Hello17
    sb.replace(0,5,"HELLO"); // HELLO17
    System.out.println(sb.toString());

    // 查询（字符个数，容量,打印容器中的字符）操作
    System.out.println("容器中的字符个数：" + sb.length());  // 7
    System.out.println("容器的容量：" + sb.capacity());     // 16
}
```

1.1.4 StringBuilder(掌握)

StringBuffer在源代码级别上已经做到了线程安全，所以StringBuffer非常适合多线程环境。
如果在单线程条件下使用可变字符串序列时，一定优先考虑StringBuilder, 实际开发过程中，**更多建议使用StringBuilder**

面试题: StringBuffer和StringBuilder的区别 (了解)

相同点：都是可变字符串，提供了相同的增删改查和自动拓容操作。
不同点:
StringBuffer 线程安全，效率低；StringBuilder线程不安全，效率高。
StringBuffer jdk1.0; StringBuilder jdk1.5

1.2. Math（掌握）

Math 类包含用于执行简单的数学运算的方法，如随机数、最大值最小值、指数等，该类的方法都是static修饰的，在开发中其实运用并不是很多，里面有一个求随机数的方法，偶尔会用到。

Math类位于java.lang包中。

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

        // [1] 取整操作 ceil/floor
        float a = 9.2f;
        // Math.ceil(a) 比9.2大或等于的最小整数：向上取整
        System.out.println(Math.ceil(a));
        // Math.floor(a) 比9.2小或等于的最大整数：向下取整
        System.out.println(Math.floor(a));

        // [2]求最大值和最小值
        System.out.println(Math.max(10, 9));
        System.out.println(Math.min(10, 9));

        // [3]随机数 取值返回[0.0,1.0)
        System.out.println(Math.random());
}
```

1.3. Random（掌握）

Random类用于生产一个伪随机数（通过相同的种子，产生的随机数是相同的），Math类的random方法底层使用的就是Random类的方式，Random位于`java.util`包中。

常用方法

- nextInt(int n) : 产生范围在 [0 , n) 的随机整数

```
public static void main(String[] args) {
    // 创建一个Random对象
    Random r = new Random();
    // val的范围[0,100)之间
    int val = r.nextInt(100);
    System.out.println(val);
}
```

需求:随机产生4位的小写英文字母的验证码

```
/**
  * 分析:
  * [1]. 产生一个a-z的随机字符c => 'a' + [0,25]
  * [2]. 把产生的随机字符存入字符数组?  还是可变字符串?
  */
public static void main(String[] args) {
    Random rand = new Random();

    StringBuilder codes = new StringBuilder(4);
    char c;
    for(int i = 0;i < codes.capacity();i++){
        c = (char)('a' + rand.nextInt(26));
        codes.append(c);
    }

    String validCode = codes.toString();
    System.out.println(validCode);
}
```

1.4. UUID （掌握）

UUID表示通用**唯一**标识符 (Universally Unique Identifier)，其算法通过电脑的网卡、当地时间、随机数等组合而成，优点是真实的唯一性，缺点是字符串太长了。

UUID位于`java.util`包中，jdk1.5 才出现的类

常用方法

- randomUUID() : 产生一个随机的唯一标识符，格式：678f9568-8967-4637-a48e-f0eae30faf43

```
public static void main(String[] args) {
    // 获取UUID对象
    UUID uuid = UUID.randomUUID();
    String str = uuid.toString();
    System.out.println("str = " + str);

    // 获取UUID中的前5个字母作为验证码
    String validCode = uuid.substring(0, 5);
    System.out.println(validCode);
}
```

> 提示:
>
> 在java中不通过new构建一个对象的静态方法就称为静态工厂方法，类似地，Integer integer = Integer.valueOf()

1.5. 日期时间

1.5.1 日期时间概述 (了解)

问题1：计算机如何表示时间?

时间戳（time stamp）：具体时间（特定的瞬间）距离历元(1970年01月01日 00:00:00:000) 经过的毫秒数，用long类型存储。

计算机很容易存储long类型数据，所以计算机通过时间戳存储并表示日期时间。

问题2：为什么时间戳一样，时间却不同?

中国时间和国外时间的时间戳都一样，但时区不同。时区导致时间不同。

计算机以格林尼治所在地的标准时间作为时间统一协调时，这个时间在民间称为格林尼治时间（GMT），为了统一国际用法，也称世界协调时(UTC)

问题3：如何计算当地时间?

- 当地时间 = UTC + 时区偏移
- 中国位于东八区，时区偏移为(+8:00)
- 中国时间 = UTC + 8:00
- 日本时间 = UTC + 9:00

1.5.2 Date（掌握）

Date类，日期时间类，表示特定的瞬间，可以解释为年、月、日、小时、分钟和秒值。

注意：我们使用的是java.util.Date类，而不是java.sql.Date。

Date类中的大量方法都标记为已经过时的，即官方不建议使用。在开发中，我们要表示日期（年月日）或时间（时分秒）类型都使用Date类来表示。

常用方法

- Date()
- getTime()

```
public class DateDemo {
    public static void main(String[] args) {

        // 根据系统当前时区，当前日期时间构建一个Date对象
        Date now = new Date();
        // Wed Aug 11 10:52:20 CST(chinese standard time) 2021
        System.out.println(now.toString());

        // 获取Date对象的时间戳(例如:161231345464)
        long ts = now.getTime();
        System.out.println(ts);
    }
}
```

15.3 SimpleDateFormat（掌握）

打印Date对象时，默认打印的是欧美人的日期时间风格，如果需要输出自定义的时间格式，比如2020年12月12日 12:12:12格式或者2020-12-12 12:12:12，此时可以使用SimpleDateFormat类。

SimpleDateFormat类，顾名思义是日期的格式化类，主要包括两个功能的方法：

- 格式化（format）：Date类型转换为String类型：String format(Date date)
- 解析（parse）：String类型转换为Date类型：Date parse(String source)

无论是格式化还是解析都需要设置日期时间的模式，所谓模式就是一种格式(xxxx年xx月xx日 xx时xx分xx秒)。

字母	日期或时间元素	表示	示例
G	Era 标志符	Text	AD
y	年	Year	1996; 96
M	年中的月份	Month	July; Jul; 07
w	年中的周数	Number	27
W	月份中的周数	Number	2
D	年中的天数	Number	189
d	月份中的天数	Number	10
F	月份中的星期	Number	2
E	星期中的天数	Text	Tuesday; Tue
a	Am/pm 标记	Text	PM
H	一天中的小时数（0-23）	Number	0
k	一天中的小时数（1-24）	Number	24
K	am/pm 中的小时数（0-11）	Number	0
h	am/pm 中的小时数（1-12）	Number	12
m	小时中的分钟数	Number	30
s	分钟中的秒数	Number	55
S	毫秒数	Number	978
z	时区	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	时区	RFC 822 time zone	-0800

日期时间模式举例：

```
yyyy-MM-dd                  如2020-12-12
HH:mm:ss                    如20：12：12
yyyy-MM-dd HH:mm:ss         如2020-12-12 20：12：12
yyyy/MM/dd HH:mm:ss         如2020/12/12 20：12：12
yyyy年MM月dd日 HH时mm分ss秒    如2020年12月12日 20时12分12秒
```

格式化和解析代码如下：

```java
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class SimpleDateFormatDemo {
    public static void main(String[] args) throws ParseException {

        Date date = new Date();
        // 创建SimpleDateFormat对象
        SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

        // 格式化（format）：Date类型转换为String类型：String format(Date date)
        String dateStr = df.format(date);
```

```
        System.out.println(dateStr);

        // 解析（parse）：String类型转换为Date类型：Date parse(String source)
        Date date2 = df.parse(dateStr);
        System.out.println(date2.toString());
    }
}
```

代码中public static void main(String[] args) throws Exception表示抛出异常，在main方法中不作任何处理，在异常章节再细讲。

#1-5-3-0

1.5.4 Calendar

Calendar 日历类，也可以用来存储日期和时间，更重要地是，Calendar还提供了**用来对日期时间做相加减，重新设置日期时间功能。**

Calendar本身是一个抽象类，通过getInstance方法获取对象，其底层创建的是Calendar的子类对象。

常用方法

- get(日历字段)：获取日历字段对应的信息
- add / set

```
public class CalendarDemo {
    public static void main(String[] args) throws Exception {
        // 1> 根据当前地区，当前语言环境，当前时间构造一个通用的日历对象
        Calendar cal = Calendar.getInstance();
        System.out.println(cal);

        // 2> 获取日历字段信息
        // 获取日历中的年月日和时分秒信息
        System.out.println(cal.get(Calendar.YEAR));
        // 月从0开始，0表示1月，1表示2月...
        System.out.println(cal.get(Calendar.MONTH));
        System.out.println(cal.get(Calendar.DATE));
        // System.out.println(cal.get(Calendar.DAY_OF_MONTH));
        System.out.println(cal.get(Calendar.HOUR));          // 12小时制
        System.out.println(cal.get(Calendar.HOUR_OF_DAY));   // 24小时制
        System.out.println(cal.get(Calendar.MINUTE));
        System.out.println(cal.get(Calendar.SECOND));


        // 3> 对日历做相加减，重新设置日期时间的操作
        cal.add(Calendar.YEAR, 1);                  // 在当前年份上增加1
        System.out.println(c.get(Calendar.YEAR));   // 2023
        cal.set(Calendar.YEAR,1997);                // 将年份设置为1997
    }
}
```

需求1：查询某个时间最近一周的信息，如何表示最近这一周的开始时间和结束时间

假如给出时间为：2018-05-18 15:05:30，那么最近一周的开始和结束时间分别为：

开始时间：2018-05-12 00:00:00

结束时间：2018-05-18 23:59:59

```
public class CalendarDemo2 {
    public static void main(String[] args) throws Exception {
        // [1] 通过字符串解析出日期时间对象
        String input = "2018-05-18 15:05:30";
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date d = sdf.parse(input);

        // [2] 调整日历时间
        Calendar c = Calendar.getInstance();
        c.setTime(d);// 把当前输入时间转换为Calendar对象

        // [3] 计算结束时间
        c.set(Calendar.HOUR_OF_DAY, 23);
        c.set(Calendar.MINUTE, 59);
        c.set(Calendar.SECOND, 59);
        Date endDate = c.getTime();
        System.out.println(endDate.toLocaleString());

        // [4] 计算开始时间
        c.add(Calendar.SECOND, 1);// 秒数增加1
        c.add(Calendar.DAY_OF_MONTH, -7);// 天数减去7
        Date beginDate = c.getTime();
        System.out.println(beginDate.toLocaleString());
    }
}
```

1.7. 正则表达式（了解）

正则表达式，简写为regex和RE/Re。

正则表达式用来判断某一个字符串是不是符合某一种规则，在开发中通常用于判断检测操作、替换操作、分割操作等。

19.1. 正则表达式规则

正则表达式匹配规则一：元字符（正则表达式中已定义好的符号，这些字符有特定的含义）

No.	规范	描述	No.	规范	描述
1	\\	表示反斜线（\）字符	2	\t	表示制表符
3	\n	表示换行	4	[abc]	字符a、b或c
5	[^abc]	除了a、b、c之外的任意字符	6	[a-zA-Z0-9]	表示由字母、数字组成
7	\d	表示数字	8	\D	表示非数字
9	\w	表示字母、数字、下划线	10	\W	表示非字母、数字、下划线
11	\s	表示所有空白字符（换行、空格等）	12	\S	表示所有非空白字符
13	^	行的开头	14	$	行的结尾
15	.	匹配除换行符之外的任意字符			

正则表达式匹配规则二：量词

数量表示（X表示一组规范）

No.	规范	描述	No.	规范	描述
1	X	必须出现一次	2	X?	可以出现0次或1次
3	X*	可以出现0次、1次或多次	4	X+	可以出现1次或多次
5	X{n}	必须出现n次	6	X{n,}	必须出现n次以上
7	X{n,m}	必须出现n~m次			

逻辑运算符（X、Y表示一组规范）

No.	规范	描述	No.	规范	描述
1	XY	X规范后跟着Y规范	2	X|Y	X规范或Y规范
3	(X)	做为一个捕获组规范			

19.2. 正则表达式练习

判断一个字符串是否全部有数字组成

判断一个字符串是否是手机号码

判断一个字符串是否是18位身份证号码（只要满足18位即可）

判断一个字符串是否6到16位，且第一个字必须为字母

```
public class REDemo {
    public static void main(String[] args) throws Exception {
        // 判断一个字符串是否全部有数字组成
        System.out.println("12345678S".matches("\\d"));// false
        System.out.println("12345678".matches("\\d"));// false
        System.out.println("12345678".matches("\\d*"));// true
        System.out.println("1234".matches("\\d{5,10}"));// false
        System.out.println("12345678".matches("\\d{5,10}"));// true

        // 判断一个字符串是否是手机号码
        String regex1 = "^1[3|4|5|7|8][0-9]{9}$";
        System.out.println("12712345678".matches(regex1));// false
        System.out.println("13712345678".matches(regex1));// true

        // 判断一个字符串是否是18位身份证号码( 不考虑地域问题 )
        String regex2 = "\\d{17}[0-9X]";
        System.out.println("511123200110101234".matches(regex2));// true
        System.out.println("51112320011010123x".matches(regex2));// true
        System.out.println("511123200110101235".matches(regex2));// false

        // 判断一个字符串是否6到16位，且第一个字必须为字母
        String regex3 = "^[a-zA-Z]\\w{5,15}$";
        System.out.println("will".matches(regex3));// false
        System.out.println("17will".matches(regex3));// false
        System.out.println("will17willwillwill".matches(regex3));// false
        System.out.println("will17".matches(regex3));// true
    }
}
```