day08【常用API】

今日内容

- Object类
- 时间日期类
- Math类
- System类

教学目标

- [ ] 能够说出Object类的特点
- [ ] 能够重写Object类的toString方法
- [ ] 能够重写Object类的equals方法
- [ ] 能够使用日期类输出当前日期
- [ ] 能够使用将日期格式化为字符串的方法
- [ ] 能够使用将字符串转换成日期的方法
- [ ] 能够使用Calendar类的get、set、add方法计算日期
- [ ] 能够使用Math类对某个浮点数进行四舍五入取整
- [ ] 能够使用System类获取当前系统毫秒值

第一章 Object类

1.1 概述

`java.lang.Object`类是Java语言中的根类，即所有类的父类。它中描述的所有方法子类都可以使用。在对象实例化的时候，最终找的父类就是Object。

如果一个类没有特别指定父类， 那么默认则继承自Object类。例如：

```
public class MyClass /*extends Object*/ {
    // ...
}
```

根据JDK源代码及Object类的API文档，Object类当中包含的方法有11个。今天我们主要学习其中的2个：

- `public String toString()`：返回该对象的字符串表示。
- `public boolean equals(Object obj)`：指示其他某个对象是否与此对象“相等”。

1.2 toString方法

方法摘要

- `public String toString()`：返回该对象的字符串表示。

toString方法返回该对象的字符串表示，其实该字符串内容就是：对象的类型名+@+内存地址值。

由于toString方法返回的结果是内存地址，而在开发中，经常需要按照对象的属性得到相应的字符串表现形式，因此也需要重写它。

覆盖重写

如果不希望使用toString方法的默认行为，则可以对它进行覆盖重写。例如自定义的Person类：

```
public class Person {
    private String name;
    private int age;

    @Override
    public String toString() {
        return "Person{" + "name='" + name + '\'' + ", age=" + age + '}';
    }

    // 省略构造器与Getter Setter
}
```

在IntelliJ IDEA中，可以点击 Code 菜单中的 `Generate...`，也可以使用快捷键 `alt+insert`，点击 `toString()` 选项。选择需要包含的成员变量并确定。

> 小贴士： 在我们直接使用输出语句输出对象名的时候,其实通过该对象调用了其toString()方法。

> 小结：toString方法可以将对象转成字符串。

1.3 equals方法

方法摘要

- `public boolean equals(Object obj)`：指示其他某个对象是否与此对象“相等”。

调用成员方法equals并指定参数为另一个对象，则可以判断这两个对象是否是相同的。这里的“相同”有默认和自定义两种方式。

默认地址比较

如果没有覆盖重写equals方法，那么Object类中默认进行 `==` 运算符的对象地址比较，只要不是同一个对象，结果必然为false。

对象内容比较

如果希望进行对象的内容比较，即所有或指定的部分成员变量相同就判定两个对象相同，则可以覆盖重写equals方法。例如：

```
import java.util.Objects;

public class Person {
    private String name;
    private int age;
```

```
    @Override
    public boolean equals(Object o) {
        // 如果对象地址一样，则认为相同
        if (this == o)
            return true;
        // 如果参数为空，或者类型信息不一样，则认为不同
        if (o == null || getClass() != o.getClass())
            return false;
        // 转换为当前类型
        Person person = (Person) o;
        // 要求基本类型相等，并且将引用类型交给java.util.Objects类的equals静态方法取用结
果
        return age == person.age && Objects.equals(name, person.name);
    }
}
```

这段代码充分考虑了对象为空、类型一致等问题，但方法内容并不唯一。大多数IDE都可以自动生成equals方法的代码内容。在IntelliJ IDEA中，可以使用`Code`菜单中的`Generate...`选项，也可以使用快捷键`alt+insert`，并选择`equals() and hashCode()`进行自动代码生成。

> tips：Object类当中的hashCode等其他方法，今后学习。

> 小结：equals方法可以判断两个对象是否相同，如果要定义自己的比较规则，需要进行重写。

1.4 native本地方法

在Object类的源码中定义了 **native** 修饰的方法， native 修饰的方法称为本地方法。

- 本地方法的作用： 就是Java调用非Java代码的接口。方法的实现由非Java语言实现，比如C或C++。
- 当我们需要访问C或C++的代码时，或者访问操作系统的底层类库时，可以使用本地方法实现。

  也就意味着Java可以和其它的编程语言进行交互。

> Object类源码(部分)：

```
package java.lang;
/**
 * Class {@code Object} is the root of the class hierarchy.
 * Every class has {@code Object} as a superclass. All objects,
 * including arrays, implement the methods of this class.
 *
 * @author  unascribed
 * @see     java.lang.Class
 * @since   JDK1.0
 */
public class Object {
    //本地方法
    private static native void registerNatives();
    //静态代码块
    static {
        registerNatives();
    }
    ......
    ......
}
```

1.5 Objects类

在刚才IDEA自动重写equals代码中，使用到了`java.util.Objects`类，那么这个类是什么呢?

在**JDK7**添加了一个Objects工具类，它提供了一些方法来操作对象，它由一些静态的实用方法组成，这些方法是null-save（空指针安全的）或null-tolerant（容忍空指针的），用于计算对象的hashCode、返回对象的字符串表示形式、比较两个对象。

在比较两个对象的时候，Object的equals方法容易抛出空指针异常，而Objects类中的equals方法就优化了这个问题。方法如下：

- `public static boolean equals(Object a, Object b)`:判断两个对象是否相等。

我们可以查看一下源码，学习一下：

```
public static boolean equals(Object a, Object b) {
    return (a == b) || (a != null && a.equals(b));
}
```

第二章 Date类

2.1概述

`java.util.Date`类 表示特定的瞬间，精确到毫秒。

继续查阅Date类的描述，发现Date拥有多个构造函数，只是部分已经过时，我们重点看以下两个构造函数

- `public Date()`：从运行程序的此时此刻到时间原点经历的毫秒值,转换成Date对象，分配Date对象并初始化此对象，以表示分配它的时间（精确到毫秒）。
- `public Date(long date)`：将指定参数的毫秒值date,转换成Date对象，分配Date对象并初始化此对象，以表示自从标准基准时间（称为“历元（epoch）”，即1970年1月1日00:00:00 GMT）以来的指定毫秒数。

> tips: 由于中国处于东八区（GMT+08:00）是比世界协调时间/格林尼治时间（GMT）快8小时的时区，当格林尼治标准时间为0:00时，东八区的标准时间为08:00。

简单来说：使用无参构造，可以自动设置当前系统时间的毫秒时刻；指定long类型的构造参数，可以自定义毫秒时刻。例如：

```
import java.util.Date;

public class Demo01Date {
    public static void main(String[] args) {
        // 创建日期对象，把当前的时间
        System.out.println(new Date()); // Tue Jan 16 14:37:35 CST 2020
        // 创建日期对象，把当前的毫秒值转成日期对象
        System.out.println(new Date(0L)); // Thu Jan 01 08:00:00 CST 1970
    }
}
```

> tips:在使用println方法时，会自动调用Date类中的toString方法。Date类对Object类中的toString方法进行了覆盖重写，所以结果为指定格式的字符串。

2.2 常用方法

Date类中的多数方法已经过时，常用的方法有：

- `public long getTime()` 把日期对象转换成对应的时间毫秒值。
- `public void setTime(long time)` 把方法参数给定的毫秒值设置给日期对象

示例代码

```
public class DateDemo02 {
    public static void main(String[] args) {
        //创建日期对象
        Date d = new Date();

        //public long getTime():获取的是日期对象从1970年1月1日 00:00:00到现在的毫秒值
        //System.out.println(d.getTime());
        //System.out.println(d.getTime() * 1.0 / 1000 / 60 / 60 / 24 / 365 +
"年");

        //public void setTime(long time):设置时间，给的是毫秒值
        //long time = 1000*60*60;
        long time = System.currentTimeMillis();
        d.setTime(time);

        System.out.println(d);
    }
}
```

> 小结：Date表示特定的时间瞬间，我们可以使用Date对象对时间进行操作。

第三章 DateFormat类

3.1概述

`java.text.DateFormat` 是日期/时间格式化子类的抽象类，我们通过这个类可以帮我们完成日期和文本之间的转换,也就是可以在Date对象与String对象之间进行来回转换。

- **格式化**：按照指定的格式，把Date对象转换为String对象。
- **解析**：按照指定的格式，把String对象转换为Date对象。

3.2 构造方法

由于DateFormat为抽象类，不能直接使用，所以需要常用的子类`java.text.SimpleDateFormat`。这个类需要一个模式（格式）来指定格式化或解析的标准。构造方法为：

- `public SimpleDateFormat(String pattern)`：用给定的模式和默认语言环境的日期格式符号构造SimpleDateFormat。参数pattern是一个字符串，代表日期时间的自定义格式。

3.3 格式规则

常用的格式规则为：

标识字母（区分大小写）	含义
y	年
M	月
d	日
H	时
m	分
s	秒

> 备注：更详细的格式规则，可以参考SimpleDateFormat类的API文档。

3.4 常用方法

DateFormat类的常用方法有：

- `public String format(Date date)`：将Date对象格式化为字符串。
- `public Date parse(String source)`：将字符串解析为Date对象。

```
public class SimpleDateFormatDemo {
    public static void main(String[] args) throws ParseException {
        //格式化：从 Date 到 String
        Date d = new Date();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日
HH:mm:ss");
        String s = sdf.format(d);
        System.out.println(s);
        System.out.println("--------");

        //从 String 到 Date
        String ss = "2048-08-09 11:11:11";
        //ParseException
        SimpleDateFormat sdf2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date dd = sdf2.parse(ss);
        System.out.println(dd);
    }
}
```

> 小结：DateFormat可以将Date对象和字符串相互转换。

第四章 Calendar类

4.1 概述

- java.util.Calendar类表示一个“日历类”，可以进行日期运算。它是一个抽象类，不能创建对象，我们可以使用它的子类：java.util.GregorianCalendar类。
- 有两种方式可以获取GregorianCalendar对象：
  - 直接创建GregorianCalendar对象；
  - 通过Calendar的静态方法getInstance()方法获取GregorianCalendar对象【本次课使用】

4.2 常用方法

方法名	说明
public static Calendar getInstance()	获取一个它的子类GregorianCalendar对象。
public int get(int field)	获取某个字段的值。field参数表示获取哪个字段的值，可以使用Calender中定义的常量来表示： Calendar.YEAR : 年 Calendar.MONTH ： 月 Calendar.DAY_OF_MONTH：月中的日期 Calendar.HOUR：小时 Calendar.MINUTE：分钟 Calendar.SECOND：秒 Calendar.DAY_OF_WEEK：星期
public void set(int field,int value)	设置某个字段的值
public void add(int field,int amount)	为某个字段增加/减少指定的值

4.3 get方法示例

```
public class Demo {
    public static void main(String[] args) {
        //1.获取一个GregorianCalendar对象
        Calendar instance = Calendar.getInstance();//获取子类对象

        //2.打印子类对象
        System.out.println(instance);

        //3.获取属性
        int year = instance.get(Calendar.YEAR);
        int month = instance.get(Calendar.MONTH) + 1;//Calendar的月份值是0-11
        int day = instance.get(Calendar.DAY_OF_MONTH);

        int hour = instance.get(Calendar.HOUR);
        int minute = instance.get(Calendar.MINUTE);
        int second = instance.get(Calendar.SECOND);
```

```
        int week = instance.get(Calendar.DAY_OF_WEEK);//返回值范围：1--7，分别表示："星期日","星期一","星期二",...,"星期六"

        System.out.println(year + "年" + month + "月" + day + "日" +
                           hour + ":" + minute + ":" + second);
        System.out.println(getWeek(week));

    }

    //查表法，查询星期几
    public static String getWeek(int w) {//w = 1 --- 7
        //做一个表(数组)
        String[] weekArray = {"星期日", "星期一", "星期二", "星期三", "星期四", "星期五", "星期六"};
        //              索引       [0]      [1]       [2]      [3]       [4]      [5]      [6]
        //查表
        return weekArray[w - 1];
    }
}
```

4.4 set方法示例

```
public class Demo {
    public static void main(String[] args) {
        //设置属性--set(int field,int value):
        Calendar c1 = Calendar.getInstance();//获取当前日期

        //计算班长出生那天是星期几(假如班长出生日期为：1998年3月18日)
        c1.set(Calendar.YEAR, 1998);
        c1.set(Calendar.MONTH, 3 - 1);//转换为Calendar内部的月份值
        c1.set(Calendar.DAY_OF_MONTH, 18);

        int w = c1.get(Calendar.DAY_OF_WEEK);
        System.out.println("班长出生那天是：" + getWeek(w));


    }
    //查表法，查询星期几
    public static String getWeek(int w) {//w = 1 --- 7
        //做一个表(数组)
        String[] weekArray = {"星期日", "星期一", "星期二", "星期三", "星期四", "星期五", "星期六"};
        //              索引       [0]      [1]       [2]      [3]       [4]      [5]      [6]
        //查表
        return weekArray[w - 1];
    }
}
```

4.5 add方法示例

```
public class Demo {
    public static void main(String[] args) {
        //计算200天以后是哪年哪月哪日，星期几？
        Calendar c2 = Calendar.getInstance();//获取当前日期
        c2.add(Calendar.DAY_OF_MONTH, 200);//日期加200

        int y = c2.get(Calendar.YEAR);
        int m = c2.get(Calendar.MONTH) + 1;//转换为实际的月份
        int d = c2.get(Calendar.DAY_OF_MONTH);

        int wk = c2.get(Calendar.DAY_OF_WEEK);
        System.out.println("200天后是：" + y + "年" + m + "月" + d + "日" +
getWeek(wk));

    }
    //查表法，查询星期几
    public static String getWeek(int w) {//w = 1 --- 7
        //做一个表(数组)
        String[] weekArray = {"星期日", "星期一", "星期二", "星期三", "星期四", "星期
五", "星期六"};
        //            索引      [0]      [1]       [2]      [3]       [4]
[5]      [6]
        //查表
        return weekArray[w - 1];
    }
}
```

第五章 Math类

5.1 概述

- java.lang.Math(类): Math包含执行基本数字运算的方法。
- 它不能创建对象，它的构造方法被“私有”了。因为他内部都是“静态方法”，通过“类名”直接调用即可。

5.2 常用方法

方法名	说明
public static int abs(int a)	获取参数a的绝对值：
public static double ceil(double a)	向上取整
public static double floor(double a)	向下取整
public static double pow(double a, double b)	获取a的b次幂
public static long round(double a)	四舍五入取整

5.3 示例代码

```
public class Demo {
    public static void main(String[] args) {
        System.out.println("-5的绝对值：" + Math.abs(-5));//5
        System.out.println("3.4向上取整：" + Math.ceil(3.4));//4.0
        System.out.println("3.4向下取整：" + Math.floor(3.4));//3.0
        System.out.println("2的8次幂：" + Math.pow(2, 8));//256.0
        System.out.println("3.2四舍五入：" + Math.round(3.2));//3
        System.out.println("3.5四舍五入：" + Math.round(3.5));//4

    }
}
```

第六章 System

6.1 概述

`java.lang.System`类中提供了大量的静态方法，可以获取与系统相关的信息或系统级操作。

6.2 常用方法

方法名	说明
public static void exit(int status)	终止当前运行的 Java 虚拟机，非零表示异常终止
public static long currentTimeMillis()	返回当前时间(以毫秒为单位)

6.3 练习

在控制台输出1-10000，计算这段代码执行了多少毫秒

```
import java.util.Date;
//验证for循环打印数字1-9999所需要使用的时间（毫秒）
public class SystemDemo {
    public static void main(String[] args) {
        //获取当前时间毫秒值
       System.out.println(System.currentTimeMillis());
      //计算程序运行时间
       long start = System.currentTimeMillis();
        for (int i = 1; i <= 10000; i++) {
            System.out.println(i);
        }
        long end = System.currentTimeMillis();
        System.out.println("共耗时毫秒：" + (end - start));
```

```
    }
}
```