day05 【常用API】

今日内容

- 权限修饰符
- 代码块
- Object类
- 时间日期类
- Math类
- System类
- BigInteger类
- BigDecimal类
- 包装类

教学目标

- [ ] 能够说出每种权限修饰符的作用
- [ ] 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
- [ ] 能够说出BigDecimal可以解决的问题
- [ ] 能够说出自动装箱、自动拆箱的概念
- [ ] 能够将基本类型转换为对应的字符串
- [ ] 能够将字符串转换为对应的基本类型

第一章 权限修饰符

知识点--权限修饰符

目标:

- 权限修饰符的使用

路径:

- 概述
- 不同权限修饰符的访问能力

讲解:

3.1 概述

在Java中提供了四种访问权限，使用不同的访问权限修饰符修饰时，被修饰的内容会有不同的访问权限，

- public：公共的
- protected：受保护的
- (空的)：默认的
- private：私有的

3.2 不同权限的访问能力

	public	protected	(空的)	private
同一类中	√	√	√	√
同一包中(子类与无关类)	√	√	√	
不同包的子类	√	√		
不同包中的无关类	√			

```
包:com.itheima.demo9_权限修饰符
public class AAA {
    public void method1(){}
    protected void method2(){}
     void method3(){}
    private void method4(){}

    // 同一个类中
    public void method(){
        method1();
        method2();
        method3();
        method4();
    }
}
public class Test {
    public static void main(String[] args) {
        AAA a = new AAA();
        a.method1();
        a.method2();
        a.method3();
        // a.method4(); 私有方法 编译报错
    }
}

包:com.itheima.demo10_权限修饰符
public class Zi extends AAA {
    public void show(){
        method1();
        method2();
        //method3();编译报错
        //method4();编译报错
    }
```

```
}
public class Test {
    public static void main(String[] args) {
        AAA a = new AAA();
        a.method1();
        //a.method2();// 编译报错
        //a.method3();// 编译报错
        //a.method4();// 编译报错
    }
}
```

可见，public具有最大权限。private则是最小权限。

编写代码时，如果没有特殊的考虑，建议这样使用权限：

- 成员变量使用`private`，隐藏细节。
- 构造方法使用`public`，方便创建对象。
- 成员方法使用`public`，方便调用方法。

小结

略

第二章 代码块

目标:

- 掌握构造代码块和静态代码块的使用

路径:

- 构造代码块
- 静态代码块
- 局部代码块

讲解:

4.1 构造代码块

```
格式：{}
位置：类中,方法外
执行：每次在调用构造方法的时候,就会执行
使用场景：统计创建了多少个该类对象

例如：
public class Person{
    {
        构造代码块执行了
    }
}
```

4.2 静态代码块

格式：

```
格式:static{}
位置: 类中,方法外
执行: 当类被加载的时候执行,并只执行一次
使用场景: 例如加载驱动,这种只需要执行一次的代码就可以放在静态代码块中

public class Person {
    private String name;
    private int age;
 //静态代码块
    static{
        System.out.println("静态代码块执行了");
    }
}
```

4.3 局部代码块

```
格式:{}
位置: 方法中
执行: 调用方法,执行到局部代码块的时候就执行
使用场景: 节省内存空间,没有多大的意义
例如:
public static void main(String[] args) {
     int a = 10;
     //局部代码块
     //局部代码块的作用就是限制变量的作用域,早早的从内存中消失, 节约内存
     //但是现在内存不值钱,所以局部代码块没有使用意义
     {
         int b = 20;
     }

     //int a = 30;     //在同一个区域不能定义重名变量
     //不报错, 以为作用域不同
     int b = 40;
 }
```

小结

第三章 Object类

知识点-- Object类概述

目标:

- 了解Object的概述和常用方法

路径:

- Object类的概述
- Object类中常用的2个方法

讲解:

Object类的概述

- `java.lang.Object`类是Java语言中的根类，即所有类的父类。

  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”。

小结

- Object类是java中的根类
- java中所有的类都是直接或者间接继承Object类,也就意味着,java中所有的类都拥有Object类中的那11个方法

知识点-- toString方法

目标:

- 能够正确使用toString方法

路径:

- toString方法的概述
- 重写toString方法

讲解:

toString方法的概述

- `public String toString()`：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。

重写toString方法

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;

    // alt + insert -> toString() 回车
    @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方法默认返回的字符串内容格式: 对象的类型+@+十六进制数的地址值
- 打印对象的时候,其实就是打印该对象调用toString方法返回的字符串内容
- 如果打印对象的时候不希望打印的是地址值的形式,那么就可以去重写toString方法,指定返回的字符串内容格式 ---->一般开发中,重写toString方法---alt+insert--->toString() 回车

知识点-- equals方法

目标:

- 掌握equals方法的使用

路径:

- equals方法的概述
- equals方法的使用

讲解:

equals方法的概述

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

equals方法的使用

默认地址比较

Object类的equals()方法默认是==比较,也就是比较2个对象的地址值,对于我们来说没有用

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

- 概述: equals方法可以判断两个对象是否相同
- 1.Object类的equals方法默认比较的是2个对象的地址值是否相同 ==一样的效果
  2.如果在开发中,希望调用equals方法比较的不是2个对象的地址值,那么就需要重写equals方法,指定比较规则
  一般开发中,重写toString方法,使用idea的快捷键alt+insert--->equals and hashCode 回车

知识点-- Objects类

目标:

- Objects类

路径:

- Objects类的概述
- Objects类中的equals方法的使用

讲解:

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) {
      // 如果比较的2个对象的地址值相同,就直接返回true
      // 如果2个对象的地址值不同:
      //      如果第一个对象a等于null,就直接返回false
      //      只有第一个对象a不等于null,才会执行a.equals(b)比较这2个对象是否相等
       return (a == b) || (a != null && a.equals(b));
}
```

小结

略

知识点-- native方法

目标

- 理解什么是native方法,以及native方法的作用

路径

- native本地方法的概述

讲解

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。这种方法是没有方法体的,我们查看不了它的实现,所以大家不需要关心该方法如何实现的

- 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。
- 本地方法的作用： 就是当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++。

> 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();
    }
```

```
    ......
    ......
}
```

小结

略

第四章 Date类

知识点--Date类

目标:

- Date类的使用

路径:

- Date类的概述
- Date类中的构造方法
- Date类中的常用方法

讲解:

Date类的概述

`java.util.Date`类 表示一个日期和时间，内部精确到毫秒。

Date类中的构造方法

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) {
        // 创建当前系统日期对象
        Date date1 = new Date();
        System.out.println("date1:"+date1);

        // 创建以标准基准时间为基准 指定偏移毫秒数 对应的日期对象
        Date date2 = new Date(2000);// 1970年1月1日 08:00:02
        System.out.println("date2:"+date2);
    }
}
```

> 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。

Date类中的常用方法

Date类中的多数方法已经过时，常用的方法有：

- ```
  - public long getTime() 获取当前日期对象距离标准基准时间的毫秒值。
  - public void setTime(long time) 设置当前日期对象距离标准基准时间的毫秒值.也就意味着改变了当前日期对象
  boolean after(Date when)  测试此日期是否在指定日期之后。
  boolean before(Date when)  测试此日期是否在指定日期之前。
  ```

示例代码

```
public class Test {
    public static void main(String[] args) {
        /*
            - Date类的概述: java.util.Date类表示日期,精确到毫秒值
            - Date类中的构造方法
                    Date(): 用来创建当前系统时间对应的日期对象
                    Date(long date):用来创建以标准基准时间为基准 指定偏移毫秒数 对应的日期对象  参数单位:毫秒
                                标准基准时间:
                                    0时区: 1970年01月01日 00:00:00
                                    东八区: 1970年01月01日 08:00:00
            - Date类中的常用方法
                long getTime()  获取当前日期对象距离标准基准时间的毫秒数
                void setTime(long time)  设置当前日期对象距离标准基准时间的毫秒数
                boolean after(Date when)  测试此日期是否在指定日期之后。
                boolean before(Date when)  测试此日期是否在指定日期之前。
         */
        // 创建一个当前系统时间的日期对象
        Date date1 = new Date();
        System.out.println("date1:"+date1);// Sat Jun 13 10:56:47 CST 2020

        // 创建一个与标准基准时间偏移了1秒的日期对象
        Date date2 = new Date(1000);
        System.out.println("date2:"+date2);// Thu Jan 01 08:00:01 CST 1970

        System.out.println("========================");
        // 获取:
```
```

```
        System.out.println(date1.getTime());// 1592017331523
        System.out.println(date2.getTime());// 1000

        // 设置:
        date1.setTime(2000);
        System.out.println(date1);// Thu Jan 01 08:00:02 CST 1970

        date2.setTime(3000);
        System.out.println(date2);// Thu Jan 01 08:00:03 CST 1970

        System.out.println("========================");
        System.out.println(date1.after(date2));// false
        System.out.println(date1.before(date2));// true

    }
}
```

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。

第五章 DateFormat类

知识点--DateFormat类

目标:

- 能够掌握DateFormat的使用

路径:

- DateFormat类的概述
- 格式规则
- DateFormat类中的常用方法

讲解:

DateFormat类的概述

`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。

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。

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。

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。

实操--日期类练习

需求

- 键盘输入一个字符串类型的时间，打印你来到世界多少天?

分析

- 从控制台接收用户的生日——String类型，例如："1998-03-18"
- 计算生日距离标准基准时间的毫秒值

- 计算现在时间距离标准基准时间的毫秒值
- 计算出生的天数并打印(2个毫秒值相减---转换为天数)

讲解

```
  public static void main(String[] args) throws Exception{
        //从控制台接收用户的生日--String类型，例如："1998-03-18"
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入你的生日(yyyy-MM-dd)：");
        String birthDay = sc.next();

        //1.计算生日的毫秒值
        //将birthDay转换为Date对象
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        Date birthDate = sdf.parse(birthDay);
        //取毫秒值
        long m1 = birthDate.getTime();

        //2.计算现在的毫秒值
        long m2 = new Date().getTime();

        System.out.println("你出生到现在一共经历了：" + ((m2 - m1) / 1000 / 3600 /
24) + " 天");

    }
```

小结

略

第六章 Calendar类

知识点--Calendar类

目标:

- Calendar类的概述以及常用方法

路径:

- Calendar类的概述
- Calendar类的常用方法

讲解:

Calendar类的概述

- 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】

Calendar类的常用方法

- public static Calendar getInstance() 获取当前日期的日历对象
- public int get(int field) 获取某个字段的值。
  - 参数field: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
- 额外扩展2个方法:
  - void setTime(Date date) 使用给定的 Date 设置此 Calendar 的时间。
  - boolean before(Object when) 判断此 Calendar 表示的时间是否在指定 Object 表示的时间之前，返回判断结果。
    - 调用before方法的日历对象是否在参数时间对象之前,
      - 如果在之前就返回true 例如: 2017年11月11日 2019年12月18日 true
      - 如果不在之前就返回false 例如: 2019年12月18日 2017年11月11日 false
  - boolean after(Object when) 判断此 Calendar 表示的时间是否在指定 Object 表示的时间之后，返回判断结果。

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

        // 创建当前时间的日历对象
        Calendar cal = Calendar.getInstance();
        System.out.println(cal);

        // 获取日历对象中的数据
        // public int get(int field) 获取某个字段的值。
        int year = cal.get(Calendar.YEAR);
        System.out.println(year);// 2020
        System.out.println(cal.get(Calendar.MONTH));// 5
        System.out.println(cal.get(Calendar.WEEK_OF_YEAR));// 24
        System.out.println(cal.get(Calendar.WEEK_OF_MONTH));// 2

        System.out.println("====================");

        // public void set(int field,int value) 设置某个字段的值
        // 需求:修改cal对象中的年份字段的值为 1999
        cal.set(Calendar.YEAR,1999);
        System.out.println(cal.get(Calendar.YEAR));// 1999

        // public void add(int field,int amount)为某个字段增加/减少指定的值
        // 需求: 为年份字段的值增加2年
        cal.add(Calendar.YEAR,2);
        System.out.println(cal.get(Calendar.YEAR));// 2001

        // 需求: 为年份字段的值增加-2年
```

```
        cal.add(Calendar.YEAR,-2);
        System.out.println(cal.get(Calendar.YEAR));// 1999

        System.out.println("====================");
        // void setTime(Date date)  使用给定的 Date 设置此 Calendar 的时间。
        // 获取当前日历对象
        Calendar cal2 = Calendar.getInstance();
        System.out.println(cal2);// 当前日期对应的日历对象

        // 需求: 获取距离标准基准时间1小时的日期对应的日历对象
        // 创建一个距离标准基准时间1小时的日期对象
        Date date = new Date(1*60*60*1000);
        // 调用日历对象的setTime方法,设置日历对象表示的日期
        cal2.setTime(date);
        System.out.println(cal2);// 距离标准基准时间1小时的日期对应的日历对象

        System.out.println("====================");
        // boolean before(Object when) 判断此 Calendar 表示的时间是否在指定 Object 表
示的时间之前，返回判断结果。
        Calendar cal3 = Calendar.getInstance();
        // cal2: 1970-01-01 09:00:00 时间的日历对象
        // cal3: 当前时间的日历对象
        System.out.println(cal2.before(cal3));// true
        System.out.println(cal3.after(cal2));// true
    }

    // Calendar类的构造方法
    private static void method01() {
        // 1.通过构造方法创建日历对象
        GregorianCalendar gc1 = new GregorianCalendar();
        // 2.通过Calendar类的静态方法创建日历对象
        Calendar cal = Calendar.getInstance();

        System.out.println(gc1);
        System.out.println(cal);
    }
}
```

注意

- 1.中国人:一个星期的第一天是星期一,外国人:一个星期的第一天是星期天
- 2.日历对象中的月份是: 0-11

小结

略

第七章 Math类

知识点-- Math类

目标:

- Math工具类的使用

步骤:

- Math类的概述
- Math类的常用方法
- 案例代码

路径:

Math类的概述

- java.lang.Math(类): Math类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。

Math类的常用方法

```
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  例如:3.14 向上取整4.0
public static double floor(double a) 向下取整 例如:3.14 向下取整3.0
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 例如:3.14 取整3  3.56 取整4
public static int max(int a, int b)  返回两个 int 值中较大的一个。
public static int min(int a, int b)  返回两个 int 值中较小的一个。
```

案例代码

```
public class Demo {
    public static void main(String[] args) {
        System.out.println("10的绝对值:"+Math.abs(10));// 10
        System.out.println("-10的绝对值:"+Math.abs(-10));// 10

        System.out.println("3.14向上取整:"+Math.ceil(3.14));// 4.0
        System.out.println("3.54向上取整:"+Math.ceil(3.54));// 4.0
        System.out.println("-3.54向上取整:"+Math.ceil(-3.54));// -3.0

        System.out.println("==================================");
        System.out.println("3.14向下取整:"+Math.floor(3.14));// 3.0
        System.out.println("3.54向下取整:"+Math.floor(3.54));// 3.0
        System.out.println("-3.54向下取整:"+Math.floor(-3.54));// -4.0

        System.out.println("==================================");
        System.out.println("2的3次幂:"+Math.pow(2,3));// 8.0

        System.out.println("==================================");
        System.out.println("3.14四舍五入取整:"+Math.round(3.14));// 3
        System.out.println("3.54四舍五入取整:"+Math.round(3.54));// 4
        System.out.println("-3.54四舍五入取整:"+Math.round(-3.54));// -4


        System.out.println("==================================");
        System.out.println("获取10和20之间的最大值:"+Math.max(10,20));// 20
        System.out.println("获取10和20之间的最小值:"+Math.min(10,20));// 10

    }
}
```

小结

略

第八章 System

知识点--System类

目标:

- System类的概述和常用方法

路径:

- System类的概述
- System类的常用方法
- 案例代码

讲解:

System类的概述

`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。

System类的常用方法

```
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() 返回当前时间距离标准基准时间的毫秒值
static void arraycopy(Object src, int srcPos, Object dest, int destPos, int
length) 拷贝数组中的元素到另一个数组
      参数1src: 源数组
      参数2srcPos:源数组要拷贝的元素的起始索引(从哪个索引位置开始拷贝)
      参数3dest: 目标数组
      参数4destPos:目标数组接收拷贝元素的起始索引(从哪个索引位置开始接收)
      参数5length:需要拷贝多少个元素(拷贝多少个)
```

案例代码

```
public class Demo {
    public static void main(String[] args) {
       /* System.out.println("Hello World!1");
        System.out.println("Hello World!2");
        // 结束程序运行
        System.exit(0);
        System.out.println("Hello World!3");*/


        System.out.println("==========================================");
        long time = System.currentTimeMillis();
        System.out.println("time:"+time);

        long time2 = new Date().getTime();
        System.out.println("time2:"+time2);
```

```
        System.out.println("========================================");
        int[] src = {1,2,3,4,5,6,7};
        int[] dest = {10,20,30,40,50,60,70};

        // 需求:把src数组中的2,3,4拷贝到dest目标数组中,使得目标数组的元素为:
{10,20,30,2,3,4,70};
        System.arraycopy(src,1,dest,3,3);

        // System.out.println("源数组:"+Arrays.toString(src));
        // System.out.println("目标数组:"+Arrays.toString(dest));
        for (int i = 0; i < dest.length; i++) {
            System.out.println(dest[i]);
        }
    }
```

小结

略

实操--练习

需求

- 在控制台输出1-10000，计算这段代码执行了多少毫秒

分析

- 获取循环开始前的毫秒值
- 循环在控制台输出1-10000
- 获取循环结束后的毫秒值
- 计算时间差

讲解

```
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
       long start = System.currentTimeMillis();
        // 循环在控制台输出1-10000
        for (int i = 1; i <= 10000; i++) {
            System.out.println(i);
        }
        // 获取循环结束后的毫秒值
        long end = System.currentTimeMillis();
        // 计算程序运行时间
        System.out.println("共耗时毫秒：" + (end - start));
    }
}
```

小结

略

第九章 BigInteger类

知识点--BigInteger类

目标

- 掌握BigInteger类的使用

路径

- BigInteger类的概述
- BigInteger类的构造方法
- BigInteger类成员方法

讲解

BigInteger类的概述

java.math.BigInteger 类，不可变的任意精度的整数。如果运算中，数据的范围超过了long类型后，可以使用 BigInteger类实现，该类的计算整数是不限制长度的。

BigInteger类的构造方法

- BigInteger(String value) 将 BigInteger 的十进制字符串表示形式转换为 BigInteger。超过long类型的范围，已经不能称为数字了，因此构造方法中采用字符串的形式来表示超大整数，将超大整数封装成BigInteger对象。

BigInteger类成员方法

BigInteger类提供了对很大的整数进行加add、减subtract、乘multiply、除divide的方法，注意：都是与另一个BigInteger对象进行运算。

方法声明	描述
add(BigInteger value)	返回其值为 (this + val) 的 BigInteger，超大整数加法运算
subtract(BigInteger value)	返回其值为 (this - val) 的 BigInteger，超大整数减法运算
multiply(BigInteger value)	返回其值为 (this * val) 的 BigInteger，超大整数乘法运算
divide(BigInteger value)	返回其值为 (this / val) 的 BigInteger，超大整数除法运算，除不尽取整数部分

【示例】

```
public class Test {
    public static void main(String[] args) {
        // 创建BigInteger对象
        BigInteger b1  = new BigInteger("22123456782212345678");
        BigInteger b2  = new BigInteger("10000000000000000000");

        // 加
```

```
        BigInteger res1 = b1.add(b2);
        System.out.println(res1);// 32123456782212345678

        // 减
        BigInteger res2 = b1.subtract(b2);
        System.out.println(res2);//  12123456782212345678

        // 乘
        BigInteger res3 = b1.multiply(b2);
        System.out.println(res3);// 221234567822123456780000000000000000000000

        // 除
        BigInteger res4 = b1.divide(b2);
        System.out.println(res4);// 2    类似：  10/3 = 3

        // 问题:
        // int num = 2212345678;// 编译报错,因为超过了int所能表示的数据范围
        // long numL = 22123456782212345678L;// 编译报错,因为超过了long所能表示的数据范围
    }
}
```

小结

略

第十章 BigDecimal类

知识点-- BigDecimal类

目标:

- 浮点数做运算会有精度问题,如何处理呢

路径:

- BigDecimal类的概述
- BigDecimal类构造方法
- BigDecimal类常用方法

讲解:

BigDecimal类的概述

使用基本类型做浮点数运算精度问题;

看程序说结果:

```
public static void main(String[] args) {
      // 小数运算精度问题:
       System.out.println(0.09 + 0.01);
    //   期望:0.10     很遗憾,结果并不是期望值,而是一个无限趋近期望值的小数
       System.out.println(1.0 - 0.32);
    //    期望:0.68     很遗憾,结果并不是期望值,而是一个无限趋近期望值的小数
     System.out.println(1.015 * 100);
    //   期望:101.5     很遗憾,结果并不是期望值,而是一个无限趋近期望值的小数
     System.out.println(1.301 / 100);
    //   期望:0.01301     很遗憾,结果并不是期望值,而是一个无限趋近期望值的小数
}
```

- **对于浮点运算，不要使用基本类型，而使用"BigDecimal类"类型**
- **java.math.BigDecimal(类):提供了更加精准的数据计算方式。**

BigDecimal类构造方法

构造方法名	描述
BigDecimal(double val)	将double类型的数据封装为BigDecimal对象
BigDecimal(String val)	将 BigDecimal 的字符串表示形式转换为 BigDecimal

注意：推荐使用第二种方式，第一种存在精度问题；

```
    // 构造方法
    private static void method01() {
        // 创建BigDecimal对象
        BigDecimal b1 = new BigDecimal("0.09");
        BigDecimal b2 = new BigDecimal("0.01");

        System.out.println(b1.add(b2));//  0.10 没有精度问题

        // 创建BigDecimal对象
        // BigDecimal b1 = new BigDecimal(0.09);
        // BigDecimal b2 = new BigDecimal(0.01);

        // 由此可发现使用BigDecimal(double val)创建出来的对象,发生数学运算还是有精度问题
        // System.out.println(b1.add(b2));
        // 0.09999999999999999968774977432417472300585359334945678710937 5
    }
```

BigDecimal类常用方法

BigDecimal类中使用最多的还是提供的进行四则运算的方法，如下：

方法声明	描述
public BigDecimal add(BigDecimal value)	加法运算
public BigDecimal subtract(BigDecimal value)	减法运算
public BigDecimal multiply(BigDecimal value)	乘法运算
public BigDecimal divide(BigDecimal value)	除法运算

注意：对于divide方法来说，如果除不尽的话，就会出现java.lang.ArithmeticException异常。此时可以使用divide方法的另一个重载方法；

> BigDecimal divide(BigDecimal divisor, int scale, RoundingMode roundingMode): divisor：除数对应的BigDecimal对象；scale:精确的位数；roundingMode取舍模式

RoundingMode枚举: RoundingMode.HALF_UP 四舍五入

```
public class Test {
    public static void main(String[] args) {
        // 加法:
        // 创建BigDecimal对象
        BigDecimal b1 = new BigDecimal("0.09");
        BigDecimal b2 = new BigDecimal("0.01");

        BigDecimal res1 = b1.add(b2);
        System.out.println(res1);// 0.10

        System.out.println("============================");

        // 减法:
        // 创建BigDecimal对象
        BigDecimal b3 = new BigDecimal("1.0");
        BigDecimal b4 = new BigDecimal("0.32");

        BigDecimal res2 = b3.subtract(b4);
        System.out.println(res2);// 0.68

        System.out.println("============================");

        // 乘法:
        // 创建BigDecimal对象
        BigDecimal b5 = new BigDecimal("1.015");
        BigDecimal b6 = new BigDecimal("100");

        BigDecimal res3 = b5.multiply(b6);
        System.out.println(res3);// 101.500

        System.out.println("============================");

        // 除法:
        // 创建BigDecimal对象
        BigDecimal b7 = new BigDecimal("1.301");
        BigDecimal b8 = new BigDecimal("100");

        BigDecimal res4 = b7.divide(b8);
        System.out.println(res4);// 0.01301
```

```
    }
}


// BigDecimal类的注意事项:
public class Test2 {
    public static void main(String[] args) {
        // 创建BigDecimal对象
        BigDecimal b1 = new BigDecimal("10");
        BigDecimal b2 = new BigDecimal("3");

        //BigDecimal res = b1.divide(b2);// 报ArithmeticException运算异常
        BigDecimal res = b1.divide(b2, 2, RoundingMode.HALF_UP);
        System.out.println(res);// 3.33
    }
}
```

小结:

- Java中小数运算有可能会有精度问题，如果要解决这种精度问题，可以使用BigDecimal

第十一章 Arrays类

知识点-- Arrays类

目标

- 掌握Arrays类的使用

路径

- Arrays类概述
- Arrays类常用方法

讲解

3.1 Arrays类概述

java.util.Arrays类：该类包含用于操作数组的各种方法（如排序和搜索）

3.2 Arrays类常用方法

- public static void sort(int[] a)：按照数字顺序排列指定的数组
- public static String toString(int[] a)：返回指定数组的内容的字符串表示形式
- 示例代码：

```
public static void main(String[] args) {
    int[] arr = {432, 53, 6, 323, 765, 7, 254, 37, 698, 97, 64, 7};
    //将数组排序
    Arrays.sort(arr);
    //打印数组
    System.out.println(Arrays.toString(arr));
}
```

打印结果：

```
[6, 7, 7, 37, 53, 64, 97, 254, 323, 432, 698, 765]
```

小结

略

第十二章 包装类

知识点-- 包装类的概述

目标

- 知道什么是包装类,包装类有哪些

路径

- 包装类的概述

讲解

包装类的概述

Java提供了两个类型系统，基本类型与引用类型，使用基本类型在于效率，然而很多情况，会创建对象使用，因为对象可以做更多的功能，如果想要我们的基本类型像对象一样操作，就可以使用基本类型对应的包装类，如下：

基本类型	对应的包装类（位于java.lang包中）
byte	Byte
short	Short
int	**Integer**
long	Long
float	Float
double	Double
char	**Character**
boolean	Boolean

小结

略

知识点-- Integer类

目标

- 了解Integer类的使用

路径

- Integer类概述
- Integer类构造方法及静态方法

讲解

Integer类概述

包装一个对象中的原始类型 int 的值

Integer类构造方法及静态方法

方法名	说明
public Integer(int value)	根据 int 值创建 Integer 对象(过时)
public Integer(String s)	根据 String 值创建 Integer 对象(过时)
public static Integer valueOf(int i)	返回表示指定的 int 值的 Integer 实例
public static Integer valueOf(String s)	返回保存指定String值的 Integer 对象

示例代码

```
public class IntegerDemo {
    public static void main(String[] args) {
        // 基本数据类型--对应--引用数据类型
        // 创建Integer对象
        Integer i1 = new Integer(10);// i1:表示整数10
        Integer i2 = new Integer("20");// i2: 表示整数20

        Integer i3 = Integer.valueOf(10);// i3:表示整数10
        Integer i4 = Integer.valueOf("20");// i4:表示整数20

        // 引用数据类型--对应--基本数据类型
        int num1 = i1.intValue(); // num1: 10

        // String---int类型
        int age = Integer.parseInt("18");
        System.out.println(age);// 18
    }
}
```

小结

略

知识点--装箱与拆箱

目标:

- 理解什么是装箱什么是拆箱,掌握自动拆箱和自动装箱

路径:

- 装箱与拆箱的概述
- 自动装箱与自动拆箱

讲解:

装箱与拆箱的概述

基本类型与对应的包装类对象之间，来回转换的过程称为"装箱"与"拆箱"：

- **装箱**：从基本类型转换为对应的包装类对象。
- **拆箱**：从包装类对象转换为对应的基本类型。

自动装箱与自动拆箱

由于我们经常要做基本类型与包装类之间的转换，从Java 5（JDK 1.5）开始，基本类型与包装类的装箱、拆箱动作可以自动完成。例如：

```
Integer i = 4;//自动装箱。相当于Integer i = Integer.valueOf(4);
int a = i + 5;//等号右边：将i对象转成基本数值(自动拆箱) i.intValue() + 5;
```

小结

- 可以将"包装类"和"基本类型"混合使用，比较方便

知识点--基本类型与字符串之间的转换

目标:

- 基本类型与字符串之间的转换

步骤:

- 基本类型与字符串之间的转换

讲解:

基本类型转换为String

- 转换方式
- 方式一：直接在数字后加一个空字符串
- 方式二：通过String类静态方法valueOf()
- 示例代码

```
public class IntegerDemo {
    public static void main(String[] args) {
        // 基本数据类型--->字符串:
        String str1 = 10 + "";// "10"
        String str2 = String.valueOf(10);// "10"
    }
}
```

String转换成基本类型

除了Character类之外，其他所有包装类都具有parseXxx静态方法可以将字符串参数转换为对应的基本类型：

- `public static byte parseByte(String s)`：将字符串参数转换为对应的byte基本类型。
- `public static short parseShort(String s)`：将字符串参数转换为对应的short基本类型。
- `public static int parseInt(String s)`：将字符串参数转换为对应的int基本类型。
- `public static long parseLong(String s)`：将字符串参数转换为对应的long基本类型。
- `public static float parseFloat(String s)`：将字符串参数转换为对应的float基本类型。
- `public static double parseDouble(String s)`：将字符串参数转换为对应的double基本类型。
- `public static boolean parseBoolean(String s)`：将字符串参数转换为对应的boolean基本类型。

代码使用（仅以Integer类的静态方法parseXxx为例）如：

- 转换方式
  - 方式一：先通过Integer类的valueOf方法将字符串数字转成Integer，再调用xxxValue方法得到基本数据类型
  - 方式二：通过Integer静态方法parseInt()进行转换
- 示例代码

```
public class IntegerDemo {
    public static void main(String[] args) {
        // 字符串--->基本数据类型
        int num1 = Integer.parseInt(str1); // 10
        double numD = Double.parseDouble("3.14"); // 3.14
        // ...

        // 把String--->对应的包装类类型 ----> 基本数据类型
        int num2 = Integer.valueOf(str1);
    }
}
```

> 注意:如果字符串参数的内容无法正确转换为对应的基本类型，则会抛出`java.lang.NumberFormatException`异常。

小结

略

总结

- 能够说出每种权限修饰符的作用

public: 本类　同包　不同包子类　不同包无关类
protected:本类　同包　不同包子类
默认(空): 本类　同包
private: 本类

- 能够说出Object类的特点
  所有类的父类
- 能够重写Object类的toString方法
  默认返回的是地址值
  如果不想返回地址值,就重写
- 能够重写Object类的equals方法
  默认比较的是2个对象的地址值
  如果有其他比较规则,就重写
- 能够使用日期类输出当前日期
  public Date()
- 能够使用将日期格式化为字符串的方法
  DateFormat: String format(Date date) 格式化
- 能够使用将字符串转换成日期的方法
  DateFormat: Date parse(String str) 解析
- 能够使用Calendar类的get、set、add方法计算日期
  get(int feild)
  set(int field,int value)
  add(int field,int value)
- 能够使用Math类对某个浮点数进行四舍五入取整
  Math.round()
- 能够使用System类获取当前系统毫秒值
  System.currentTimeMillis()
- 能够说出BigDecimal可以解决的问题
  小数运算精度问题
- 能够说出自动装箱、自动拆箱的概念
  自动装箱: 基本数据类型 自动转换为 包装类类型
  自动拆箱: 包装类类型　自动转换为 基本数据类型
- 能够将基本类型转换为对应的字符串
  方式一: 数据 + 空字符串　记忆
  方式二: String类的静态方法valueOf(任意数据类型)
- 能够将字符串转换为对应的基本类型
  方式一: 包装类的parseXXX(Stirng str)方法
  方式二: 包装类的valueOf(String str)方法