day03 【权限修饰符、代码块、常用API】

今日内容

- 权限修饰符
  - 概述
  - 权限修饰符的访问权限
  - 权限修饰符的使用
- 代码块
  - 构造代码块
  - 静态代码块
- Object类
  - 概述
  - toString方法
  - equals方法
- 时间日期类
  - Date类
  - DateFormat类
  - Calendar类
- Math类
  - 概述
  - 常用方法
- System类
  - 概述
  - 常用方法
  - 案例-获取一段时间的运行时间
- BigDecimal类
  - 使用原因
  - 概述
  - 构造方法
  - 常用方法
- 包装类
  - 概述
  - Integer类
  - 装箱与拆箱
  - 自动装箱与自动拆箱
  - 基本类型与字符串之间转换

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:

- 理解权限修饰符的使用

路径:

- 概述
- 访问权限
- 演示权限修饰符使用

讲解:

1.1 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的内容，会有不同的访问权限。

四中权限修饰符

```
public ：公共的。
protected ：受保护的
(default) ：默认的
private   ：私有的
```

1.2 权限修饰符的访问权限

范围	public	protected	default（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	√			

> public具有最大权限。private则是最小权限。

1.3演示权限修饰符的使用

需求：根据如下分类演示不同权限修饰符修饰变量的使用

```
包A
    Fu类
    ZiA类
    OthoerA类
包B
    ZiB类
    OthoerB类
```

//父类

```
public class Fu {
    private int privateNum = 10;
    int defaulteNum = 20;
    protected int protectedNum = 30;
    public int publicNum = 40;
    //本类访问成员
    public void show() {
        System.out.println("privateNum:" + privateNum);
        System.out.println("defaulteNum:" + defaulteNum);
        System.out.println("protectedNum:" + protectedNum);
        System.out.println("publicNum:" + publicNum);
    }
}
```

//同包子类

```
public class ZiA extends Fu {
    //同包子类访问成员
    public void show() {
        //System.out.println("privateNum:" + privateNum);
        System.out.println("defaulteNum:" + defaulteNum);
        System.out.println("protectedNum:" + protectedNum);
        System.out.println("publicNum:" + publicNum);
    }
}
```

//同包无关类

```
public class OtherA {
  //同包无关类访问成员
  public void show() {
    Fu f = new Fu();
    //System.out.println("privateNum:" + f.privateNum);
    System.out.println("defaulteNum:" + f.defaulteNum);
    System.out.println("protectedNum:" + f.protectedNum);
    System.out.println("publicNum:" + f.publicNum);
  }
}
```

//不同包子类

```
public class ZiB extends Fu {
  //不同包子类访问成员
  public void show() {
    //System.out.println("privateNum:" + privateNum);
    //System.out.println("defaulteNum:" + defaulteNum);
    System.out.println("protectedNum:" + protectedNum);
    System.out.println("publicNum:" + publicNum);
  }
}
```

//不同包无关类

```
public class OtherB {
  // 不同包无关类访问成员
  public void show() {
    Fu f = new Fu();
    //System.out.println("privateNum:" + f.privateNum);
    //System.out.println("defaulteNum:" + f.defaulteNum);
    //System.out.println("protectedNum:" + f.protectedNum);
    System.out.println("publicNum:" + f.publicNum);
  }
}
```

//测试类代码

```
// private class Test {//类不能使用private修饰
// class Test {//类可以使用默认修饰
// default class Test {类不能使用default修饰
public class Test {
  public static void main(String[] args) {
    Fu f = new Fu();
    f.show();
    System.out.println("----------------");
    ZiA za = new ZiA();
    za.show();
    System.out.println("----------------");
    OtherA oa = new OtherA();
    oa.show();
    System.out.println("----------------");
    ZiB zb = new ZiB();
    zb.show();
    System.out.println("----------------");
    OtherB ob = new OtherB();
    ob.show();
  }
}
```

1.4权限修饰符常见使用规则

```
类：`public`   `默认`，一般用public。
成员都可使用`public`   `protected`   `默认`   `private`
     成员内部类，一般用 private，隐藏细节。
     修饰成员变量 ：一般用 private，隐藏细节。
     修饰成员方法：一般用public，方便调用方法
     修饰构造方法：一般用public ，方便创建对象
```

> 小贴士：不加权限修饰符，即默认权限修饰符。

小结:

```
```

第二章 代码块

知识点--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
- 演示代码块的使用

讲解:

2.1 构造代码块

构造代码块：定义在成员位置的代码块{}

执行：每次创建对象都会执行构造代码块,优先于构造方法执行

格式

```
{
    //执行语句
}
```

2.2 静态代码块

- **静态代码块**：定义在成员位置，使用static修饰的代码块{ }。
- 执行：随着类的加载而执行且执行一次，优先构造方法和构造代码块的执行。

格式：

```
static{
    //执行语句
}
```

2.3演示代码块的使用

需求：在Person类中定义构造代码块和静态代码块，观察执行效果

//Person类代码

```
public class Person {
  {
    System.out.println("构造代码块...");
  }

  static {
    System.out.println("静态代码块...");
  }

  public Person() {
    System.out.println("构造方法...");
  }
}
```

//测试类代码

```
public class Test {
  public static void main(String[] args) {
    Person p = new Person();
    System.out.println("------------------");
    Person p2 = new Person();
  }
}
```

小结:

```

```

第三章 Object类

知识点-- 概述

目标:

- 理解Object类的在类体系中的位置和作用

路径:

- Object类的概述
- Object中常用方法介绍
- 演示Object中常用方法

讲解:

3.1.1Object类的概述

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

如果一个类没有指定父类，该类默认继承Object类，即`所有类直接或者间接继承Object类`。

在对象实例化的时候，最终找的父类都是Object。

Object类中描述的所有方法子类都可以使用。

```
//当我们定义Person类，如果没有继承类，隐含继承Object
public class Person /*省略 extends Object*/ {
}
```

3.1.2Object中常用方法继承使用

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。主要学习其中的2个。

- `public String toString()`：返回该对象的字符串表示。
    - 返回该对象的字符串表示。
      默认返回：对象的类型名+@+内存地址值字符串形式。
- `public boolean equals(Object obj)`：指示其他某个对象是否与此对象“相等”。
    - 指示其他某个对象是否与此对象“相等”。
      默认返回:通过==运算符比较两个对象地址值是否相同的布尔结果。

3.1.3 演示Object中常用方法继承使用

需求：通过定义人类和学生类，展示toString方法和equals方法的直接与间接继承
//Teacher类代码(直接继承Object)

```
public class Person /*extends Object*/ {
}
```

//Sutdent类代码(间接继承Object)

```
public class Sutdent extends Person{
}
```

//测试类代码

```
public class Test {
    public static void main(String[] args) {
        Person p = new Person();
        Student s = new Student();
        //Object中toStirng默认打印当前对象的地址值。
        System.out.println(p.toString());
        System.out.println(s.toString());
        System.out.println("---------------");
        //Object中equals默认比较两个对象的地址值
        System.out.println(p.equals(p));
        System.out.println(p.equals(s));
    }
}
```

小结:

```
```

知识点-- 重写toString方法

目标:

- 掌握toString方法的重写自定义

路径:

- toString方法的概述
- 演示toString方法的应用

讲解:

3.2.1toString方法的概述

- `public String toString()`：返回该对象的字符串表示。
- 应用:
  - 展示类中内容
  - 输出语句中展示的是该内容的字符串表现形式
    - `SYstem.out.println(对象)`等同于`SYstem.out.println(对象.toString())`
- 如果打印的的对象没有打印地址值，则说明该类已重写toString方法

3.2.2演示toString方法的应用

需求：通过学生类重写Object中的toString方法展示学生对象内容

//学生类代码

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

    public Student() {
    }

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        String str = "姓名:" + name + ",年龄:" + age;
        return str;
    }
}
```

//测试类代码

```
public class Test {
    public static void main(String[] args) {
        Student p = new Student("张三",18);
        System.out.println(p);//等同于下一行代码
        System.out.println(p.toString());
        System.out.println("-----------------");
        //  如果打印的的对象没有打印地址值，则说明该类已重写toString方法
        String s= new String("abc");
        System.out.println(s);
    }
}
```

> 在IntelliJ IDEA中，可以利用快捷方式生成toString方法
>
> > 点击`Code`菜单中的`Generate...`，点击`toString()`选项，选择需要包含的成员变量并确定。
> >
> > 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。

小结:

```

```

知识点-- 重写equals方法

目标:

- 掌握equals方法的重写自定义

路径:

- equals方法的概述
- 演示equals方法的应用

讲解:

3.3.1equals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”。
- 应用
  - equals方法用于比较两个对象地址值是否相同
  - 如果要改变比较规则，需要对该方法进行重写
  - 一般根据成员属性进行比较
- 自定义比较步骤

```
比较两个对象的地址值是否相同，如果相同，返回true
如果参数为空，或者类型不一致，返回false
将参数转换为当前类型
比较两个对象的内容是否相同，并返回比较结果
```

3.3.2演示equals方法的应用

需求：通过学生类重写Object中的equals方法演示比较两个学生

//学生类代码

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

  public Student() {
  }
  public Student(String name, int age) {
    this.name = name;
    this.age = age;
  }

  @Override
  public boolean equals(Object o) {
    //  步骤1：比较两个对象地址值，如果一样，则相同
    if (this == o) {
      return true;
    }
    //步骤2：如果参数为空，或者类型不一致，则不相同
    if (o == null || this.getClass() != o.getClass()) {
      return false;
    }
    //步骤3：将参数转换为本类
    Student s = (Student) o;
    //步骤4：判断两个对象的属性内容是否一致，完全一致，则相同
    // return this.age == s.age && this.name.equals(s.name);
    //解决空指针问题
    return this.age == s.age && Objects.equals(this.name,s.name);
  }

}
```

//测试类代码

```
public class Test {
    public static void main(String[] args) {
        Student s = new Student("张三", 18);
        Student s2 = new Student("李四", 28);
        System.out.println(s.equals(s2));
        Student s3 = new Student("张三", 28);
        System.out.println(s.equals(s3));
    }
}
```

> 在IntelliJ IDEA中，可以利用快捷方式生成equals方法
>
> > 点击`Code`菜单中`Generate...`，点击`equals() and hashCode()`，选择需要包含的成员变量并确定。
> >
> > 使用快捷键`alt+insert`，点击`equals() and hashCode()`选项，选择需要包含的成员变量并确定。

小结:

```

```

知识点-- Objects类

目标:

- 了解Objects类

路径:

- Objects类的概述
- 常见功能
- 演示Objects类中的equals方法

讲解:

3.4.1 Objects类的概述

- `java.util.Objects`：JDK7中添加，用来操作对象的工具类。
- 由nullsave（空指针安全的)或nulltolerant（容忍空指针的)两类方法组成
- 用于计算对象的hashCode值、返回对象的字符串表示形式、比较两个对象等。

3.4.2 常见功能

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

源码：

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

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

3.4.3演示Objects类中的equals方法

需求：演示Objects类与Object类中equals方法使用区别

//测试类代码

```
public class Test {
    public static void main(String[] args) {
        //Object中的equals方法
        String s1 = null;
        String s2 = "abc";
        s1.equals(s2);  //java.lang.NullPointerException
        //Objects中的equals方法
        System.out.println(Objects.equals(s1, s2));
    }
}
```

小结:

```

```

第四章 Date类

知识点--Date类

目标:

- 掌握Date类的基本使用

路径:

- Date类的概述
- 构造方法
- 常用方法
- 演示Date类的使用

讲解:

4.1Date类的概述

- java.util.Date类 表示特定的瞬间，精确到毫秒。
- 标准基准时间:【历元(epoch):1970年1月1日00:00:00 GMT)】,也称为时间原点。
- 表示距离时间原点以来的毫秒代表的时间。

4.2构造方法

- `public Date()`：为运行程序时到时间原点经历毫秒值，分配的Date对象,以表示该时刻。
- `public Date(long date)`：为到时间原点的指定毫秒值，分配的Date对象,以表示该时刻。

4.3常用方法

- `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

4.4演示Date类的使用

需求：演示Date类的构造方法与常用方法

//测试类代码

```
public class Test {
    public static void main(String[] args) {
        Date d = new Date();
        System.out.println(d);
        d.setTime(0);
        System.out.println(d);
        System.out.println("---------------");
        Date d2 = new Date(0);
        System.out.println(d2);
        System.out.println(d2.getTime());

    }
}
```

> 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。

小结:

第五章 DateFormat类

知识点--DateFormat类

目标:

- 掌握Date对象与String对象的转换操作

路径:

- DateFormat类的概述
- 构造方法
- 常用方法
- 演示日期对象的的格式化与解析

讲解:

5.1 DateFormat类的概述

- `java.text.DateFormat` 该类可以使得在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。

5.2 构造方法

- DateFormat为抽象类，需要使用其子类java.text.SimpleDateFormat创建对象。
- `public SimpleDateFormat(String pattern)`：用给定模式和默认语言环境日期格式符号构造参数
  - 参数pattern是一个字符串，代表日期时间的自定义格式。
  - 常见格式:yyyyMMdd HH:mm:ss 或者 yyyy年MM月dd日 HH:mm:ss .... 或者yyyyMMdd
- 构造格式规则

标识字母(区分大小写)	y	M	d	H	m
含义	年	月	日	时	分

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

5.3 常用方法

- `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。

5.4演示日期对象的的格式化与解析

需求：演示Date类与字符串类型的格式化和解析

//测试类代码

```
public class Test {
    public static void main(String[] args) throws ParseException {
        //格式化:Date-->String  format
        Date d = new Date(0);
        DateFormat df = new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss");
        String format = df.format(d);
        System.out.println(format);
        System.out.println("----------------");
        String strDate = "20201202 09:00:00";
        //由于转换类中记录的转换格式与字符串时间的格式不一致，导致报错
        //Date parse = df.parse(strDate);//java.text.ParseException
        DateFormat df2 = new SimpleDateFormat("yyyyMMdd HH:mm:ss");
        Date parse = df2.parse(strDate);
        System.out.println(parse);

    }
}
```

小结:

```

```

第六章 Calendar类

知识点--Calendar类

目标:

- 掌握Calendar的get/set/add方法的使用

路径:

- Calendar类的概述
- 获取对象
- 常用方法
- 演示Calendar类的使用

讲解:

6.1 Calendar类的概述

- java.util.Calendar类表示一个“日历类”，可以进行日期运算。
- 注意事项：
  - 日历对象中的星期是从1-7来表示，1表示星期天。
  - 日历对象中的月份是从0-11来表示，0表示一月份。

6.2获取对象

- Calendar类是一个抽象类，可以使用它的子类：java.util.GregorianCalendar类
- 通过Calendar的静态方法getInstance()方法获取GregorianCalendar对象
  - public static Calendar getInstance() 获取一个它的子类GregorianCalendar对象

6.3常用方法

- `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。
- `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
- 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：日
Calendar.HOUR : 时 | Calendar.MINUTE：分 | Calendar.SECOND：秒
Calendar.DAY_OF_WEEK：星期
```

6.4演示Calendar类的使用

需求：按照下述需求，演示Calendar类的构造方法与常用方法

```
1.获取当前日期对象，并展示当前日期详细时间。
2.设置当前日期对象为一个月以后的第一天，查看当时的详细时间。
3.将当前日期对象的月份增加100000分钟，查看当时的详细时间。
```

//测试类代码

```
public class Test {
    public static void main(String[] args) {
        //创建日历类的对象
        Calendar c1 = Calendar.getInstance();
        //展示日历类的内容
        //System.out.println(c1);
        showCalendar(c1);
        System.out.println("--------");
        //修改日历类的内容  public void set(int field,int value)
        Calendar c2 = Calendar.getInstance();
        c2.set(Calendar.YEAR, 2021);
        c2.set(Calendar.MONTH, 0);
        c2.set(Calendar.DAY_OF_MONTH, 1);
        c2.set(Calendar.HOUR_OF_DAY, 00);
        c2.set(Calendar.MINUTE, 00);
        c2.set(Calendar.SECOND, 00);
        showCalendar(c2);
/*        c2.set(Calendar.DAY_OF_WEEK,1);
        showCalendar(c2);*/
        System.out.println("--------");
        //修改日历的内容  public void add(int field,int amount)
        c2.add(Calendar.YEAR, 1);
        c2.add(Calendar.MONTH, 11);
        c2.add(Calendar.DAY_OF_MONTH, 30);
        c2.add(Calendar.HOUR_OF_DAY, 23);
        c2.add(Calendar.MINUTE, 59);
        c2.add(Calendar.SECOND, 59);
        showCalendar(c2);
        /*
        c2.add(Calendar.MONTH, 13);
        showCalendar(c2);
        */
    }

    public static void showCalendar(Calendar c) {
        //  public int get(int field)
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);
        //int hour = c.get(Calendar.HOUR);//12小时制，不好
        int hour = c.get(Calendar.HOUR_OF_DAY);//24小时制
        int minute = c.get(Calendar.MINUTE);
        int second = c.get(Calendar.SECOND);
        System.out.println(year + "年" + ++month + "月" + day + "日" + hour + ":"
+ minute + ":" + second + "秒");
        String week = getWeek(c);
        System.out.println("这一天是" + week);
```

```
    }

    public static String getWeek(Calendar c) {
        String[] weeks = {"星期天", "星期一", "星期二", "星期三", "星期四", "星期五",
"星期六"};
        int week = c.get(Calendar.DAY_OF_WEEK);
        String weekStr = weeks[--week];
        return weekStr;
    }
}
```

小结:

```
```

第七章 Math类

知识点-- Math类

目标:

- 掌握Math工具类的使用

路径:

- Math类的概述
- Math类常用方法
- 演示Math类的使用

讲解:

7.1 Math类的概述

- java.lang.Math: Math包含执行基本数字运算的方法的工具类。
- Math类构造方法被私有修饰，不能创建对象。
- 构造方法被私有修饰，不能创建对象，通过类名调用内部静态内容即可

7.2 Math类常用方法

```
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
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
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
```

7.3 演示Math类常用方法

需求：演示Math类中的常用方法

//测试类代码

```
public class Test {
```

```
    public static void main(String[] args) {
        System.out.println("5的绝对值" + Math.abs(5));
        System.out.println("负5的绝对值" + Math.abs(-5));
        System.out.println("------------------");
        System.out.println("1.5向上取整" + Math.ceil(1.5));
        System.out.println("1.5向下取整" + Math.floor(1.5));
        System.out.println("------------------");
        System.out.println("2的3次幂" + Math.pow(2, 3));
        System.out.println("------------------");
        System.out.println("2.3四舍五入" + Math.round(2.3));
        System.out.println("2.7四舍五入" + Math.round(2.7));
        System.out.println("------------------");
        System.out.println("10和20中的较大值"+Math.max(10, 20));
        System.out.println("10和20中的较小值"+Math.min(10, 20));
    }
}
```

小结:

```

```

第八章 System类

知识点--System类

目标:

- 学会System的基本用法

路径:

- System类的概述
- System类常用方法
- 演示System类常用方法
- 演示案例：计算运行时间

讲解:

8.1 System类的概述

- java.lang.System 可以获取与系统相关的信息或系统级操作的工具类
- System类构造方法被私有修饰，不能创建对象，通过类名调用内部静态内容即可。

8.2 System类常用方法

- `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(以毫秒为单位)

8.3 演示System类常用方法

需求：演示System类中的常用方法

//测试类代码

```
public class Test {
    public static void main(String[] args) {
        /*
        System.out.println("start");
        //0表示正常退出，非0表示异常退出
        //System.exit(0);
        //含有异常的代码
        int i = 1/0;
        System.out.println("end");
     */
        long time = System.currentTimeMillis();//1581596620194   1581596632048
        System.out.println(time);
    }
}
```

8.4演示计算运行时间案例

需求:在控制台输出100000次内容，计算这段代码执行了多少毫秒

//测试类代码

```
public class Test {
    public static void main(String[] args) {
        long startTime = System.currentTimeMillis();
        method();
        long endTime = System.currentTimeMillis();
        long time = endTime - startTime;
        System.out.println("此次运行一共用了:" + time + "毫秒");
    }

    public static void method() {
        System.out.println("开始运行");
        for (int i = 0; i < 1000000; i++) {
            System.out.println("count: " + i);
        }
        System.out.println("结束运行");
    }
}
```

小结:

```

```

第九章 BigDecimal类

知识点-- BigDecimal类

目标:

- 掌握BigDecimal的作用和使用

路径:

- 概述
- 构造方法
- 常用方法
- 演示BigDecimal的使用

讲解:

9.1概述

- java.math.BigDecimal 为浮点数提供精准计算的类
- 浮点数由指数和尾数组成，目的是增大数值范围，问题是容易丢失精确度，导致运算误差。

例如:

```
public static void main(String[] args) {
    System.out.println(0.09 + 0.01);
    System.out.println(1.0 - 0.32);
    System.out.println(1.015 * 100);
    System.out.println(1.301 / 100);
}
```

9.2 构造方法

- 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；

9.3 常用方法

- 加法运算
  public BigDecimal add(BigDecimal value)
- 减法运算
  public BigDecimal subtract(BigDecimal value)
- 乘法运算
  public BigDecimal multiply(BigDecimal value)
- 除法运算
  - public BigDecimal divide(BigDecimal value) 不推荐
  - public 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 枚举类型，示例： RoundingMode.HALF_UP 四舍五入
- > 对于divide方法来说，如果除不尽的话，就会出现java.lang.ArithmeticException异常，需要使用重载方法。BigDecimal divide(BigDecimal divisor, int scale, RoundingMode roundingMode)

9.4演示BigDecimal的使用

需求：演示BigDecimal类的构造方法与常用方法

//测试类代码

```
public class Test {
    public static void main(String[] args) {
        //在使用小数运算的时候，会造成一定的不准确性，而且数据越大，误差也会越严重。
        //System.out.println(0.09 + 0.01);
        //System.out.println(1.0 - 0.32);
        //System.out.println(1.015 * 100);
        //System.out.println(1.301 / 100);

        //传递double的方式不建议使用。
        //BigDecimal b1 =new BigDecimal(0.09);
        //BigDecimal b2 =new BigDecimal(0.01);
        //BigDecimal add = b1.add(b2);
        //System.out.println(add);

        BigDecimal b1 = new BigDecimal("0.09");
        BigDecimal b2 = new BigDecimal("0.01");
        BigDecimal add = b1.add(b2);
        System.out.println(add);
        System.out.println("------------------");
        BigDecimal b3 = new BigDecimal("1.0");
        BigDecimal b4 = new BigDecimal("0.32");
        BigDecimal subtract = b3.subtract(b4);
        System.out.println(subtract);
        System.out.println("------------------");
        BigDecimal b5 = new BigDecimal("1.015");
        BigDecimal b6 = new BigDecimal("100");
        BigDecimal multiply = b5.multiply(b6);
        System.out.println(multiply);
        System.out.println("------------------");
        BigDecimal b7 = new BigDecimal("10");
        BigDecimal b8 = new BigDecimal("3");
        //由于小数位数过多，导致数据进行转换处理的时候出现错误。
        //BigDecimal divide = b7.divide(b8);//java.lang.ArithmeticException
        BigDecimal divide = b7.divide(b8, 2, RoundingMode.HALF_UP);
        System.out.println(divide);
    }
}
```

小结:

```

```

第十章 包装类

知识点-- 概述

目标:

- 记住有哪些包装类

路径:

- 包装类概述
- 包装类类型
- Integer类构造方法
- Integer类常用方法

讲解:

10.1.1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，基本类型效率更高。

为了便于操作，java为在lang包下为基本类型创建了对应的引用类型，称为**包装类**

由于分类较多，接下来的讲解统一以Integer为例

10.1.2包装类类型

类型	byte	short	int	long	float	double	char	boolean
包装类	Byte	Short	Integer	Long	Float	Double	Character	Boolean

10.1.3Integer类构造方法

- `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)

10.1.4Integer类常用方法

- `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
- `public static int intValue()` 返回Integer对象的int形式

小结:

```

```

知识点-- 装箱与拆箱

目标:

- 理解什么是装箱和拆箱

路径:

- 装箱与拆箱概述
- 演示装箱与拆箱

讲解:

10.2.1装箱与拆箱概述

- **装箱**：从基本类型转换为对应的包装类对象(构造方法/valueOf)。
- **拆箱**：从包装类对象转换为对应的基本类型(intValue)。

10.2.2演示装箱与拆箱

需求：演示装箱与拆箱

//测试类代码

```
public class Test {
    public static void main(String[] args) {
        //装箱   基本数据类型转为包装类
        Integer ii1 = new Integer(10);
        Integer ii2 = new Integer("20");
        Integer ii3 = Integer.valueOf(30);
        Integer ii4 = Integer.valueOf("40");
        System.out.println(ii1);
        System.out.println(ii1);
        System.out.println(ii1);
        System.out.println(ii1);
        //拆箱    包装类类型转为基本数据
        int i1 = ii1.intValue();
        int i2 = ii2.intValue();
        int i3 = ii3.intValue();
        int i4 = ii4.intValue();
        System.out.println(i1);
        System.out.println(i2);
        System.out.println(i3);
        System.out.println(i4);
    }
}
```

小结:

知识点-- 自动装箱与自动拆箱

目标:

- 理解什么是自动装箱与自动拆箱

路径:

- 自动装箱与自动拆箱概述
- 演示自动装箱与自动拆箱

讲解:

10.3.1 自动装箱/拆箱概述

基本类型与包装类的转换较为常见，Java 5开始，装箱、拆箱动作可以自动完成。

自动装箱：基本类型传递给包装类型

自动拆箱：包装类型传递给基本类型

10.3.2 演示自动装箱与自动拆箱

需求：演示自动装箱与自动拆箱

//测试类代码

```
public class Test {
    public static void main(String[] args) {
        //自动装箱
        Integer i1 = 10;
        //自动拆箱
        int i = i1;
    }
}
```

小结:

```

```

知识点-- 基本类型与字符串之间的转换

目标:

- 掌握基本类型与字符串之间的转换

路径:

- 基本类型转换为String
- String转换成基本类型
- 演示基本类型与字符串之间的转换

讲解:

10.4.1基本类型转换为String

- 方式一：直接在数字后加一个空字符串 `数据+""`
- 方式二：String类静态方法valueOf(Xxx)

10.4.2String转换成基本类型

- 方式一：指定包装类的静态方法valueOf(String s)将字符串转为对应包装类
- 方式二：通过包装类的静态方法parseXxx(String s)将字符串转为对应包装类
- String转char类型只能使用String类中非静态方法char charAt(int index)
- 注意事项：数据要符合对应数据的类型格式

10.4.3演示基本类型与字符串之间的转换

需求：演示基本类型与字符串之间的转换

//测试类代码

```
public class Test {
    public static void main(String[] args) {
        //基本数据类型转字符串
        int i = 10;
        String num1 = i + "";
        int i2 = 20;
        String num2 = String.valueOf(i);
        System.out.println("-----------");
        //字符串转基本数据类型
        String num3 = "30";
        Integer i3 = Integer.valueOf(num3);
        String num4 = "40";
        int i4 = Integer.parseInt(num4);
        System.out.println("--------------");
        //java.lang.NumberFormatException String类型换基本类型，必须确保类型一致，否则
会出现异常
        //int i5 = Integer.parseInt("五十");
        //Integer i6 = Integer.valueOf("六十");
        System.out.println("--------------");
        //  因为Character类中没有对应的valueOf方法和parseInt方法，可以利用String类中的
charAt方法
        String s5 = "a";
        char c = s5.charAt(0);
        System.out.println(c);

    }
}
```

小结: