1.包装类

1.1基本类型包装类（记忆）

- 基本类型包装类的作用

  将基本数据类型封装成对象的好处在于可以在对象中定义更多的功能方法操作该数据

  常用的操作之一：用于基本数据类型与字符串之间的转换

- 基本类型对应的包装类

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

1.2Integer类（应用）

- 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

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)	返回一个保存指定值的 Integer 对象 String

- 示例代码

```
public class IntegerDemo {
    public static void main(String[] args) {
        //public Integer(int value)：根据 int 值创建 Integer 对象(过时)
```

```
        Integer i1 = new Integer(100);
        System.out.println(i1);

        //public Integer(String s)：根据 String 值创建 Integer 对象(过时)
        Integer i2 = new Integer("100");
//        Integer i2 = new Integer("abc"); //NumberFormatException
        System.out.println(i2);
        System.out.println("--------");

        //public static Integer valueOf(int i)：返回表示指定的 int 值的 Integer 实例
        Integer i3 = Integer.valueOf(100);
        System.out.println(i3);

        //public static Integer valueOf(String s)：返回一个保存指定值的Integer对象 String
        Integer i4 = Integer.valueOf("100");
        System.out.println(i4);
    }
}
```

1.3int和String类型的相互转换（记忆）

- int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) {
        //int --- String
        int number = 100;
        //方式1
        String s1 = number + "";
        System.out.println(s1);
        //方式2
        //public static String valueOf(int i)
        String s2 = String.valueOf(number);
        System.out.println(s2);
        System.out.println("--------");
    }
}
```

- String转换为int
  - 转换方式
    - 方式一：先将字符串数字转成Integer，再调用valueOf()方法
    - 方式二：通过Integer静态方法parseInt()进行转换
  - 示例代码

```java
public class IntegerDemo {
    public static void main(String[] args) {
        //String --- int
        String s = "100";
        //方式1：String --- Integer --- int
        Integer i = Integer.valueOf(s);
        //public int intValue()
        int x = i.intValue();
        System.out.println(x);
        //方式2
        //public static int parseInt(String s)
        int y = Integer.parseInt(s);
        System.out.println(y);
    }
}
```

1.4字符串数据排序案例（应用）

- 案例需求

  有一个字符串：“91 27 46 38 50”，请写程序实现最终输出结果是：“27 38 46 50 91”

- 代码实现

```java
public class IntegerTest {
    public static void main(String[] args) {
        //定义一个字符串
        String s = "91 27 46 38 50";

        //把字符串中的数字数据存储到一个int类型的数组中
        String[] strArray = s.split(" ");
//        for(int i=0; i<strArray.length; i++) {
//            System.out.println(strArray[i]);
//        }

        //定义一个int数组，把 String[] 数组中的每一个元素存储到 int 数组中
        int[] arr = new int[strArray.length];
        for(int i=0; i<arr.length; i++) {
            arr[i] = Integer.parseInt(strArray[i]);
        }

        //对 int 数组进行排序
        Arrays.sort(arr);

        //把排序后的int数组中的元素进行拼接得到一个字符串，这里拼接采用StringBuilder来实现
        StringBuilder sb = new StringBuilder();
        for(int i=0; i<arr.length; i++) {
            if(i == arr.length - 1) {
                sb.append(arr[i]);
            } else {
                sb.append(arr[i]).append(" ");
            }
        }
        String result = sb.toString();
```

```
31
32          //输出结果
33          System.out.println(result);
34      }
35  }
```

1.5自动拆箱和自动装箱（理解）

- 自动装箱

  把基本数据类型转换为对应的包装类类型

- 自动拆箱

  把包装类类型转换为对应的基本数据类型

- 示例代码

```
1  Integer i = 100;  // 自动装箱
2  i += 200;         // i = i + 200;  i + 200 自动拆箱；i = i + 200; 是自动装箱
```

2.时间日期类

2.1Date类（应用）

- Date类概述

  Date 代表了一个特定的时间，精确到毫秒

- Date类构造方法

方法名	说明
public Date()	分配一个 Date对象，并初始化，以便它代表它被分配的时间，精确到毫秒
public Date(long date)	分配一个 Date对象，并将其初始化为表示从标准基准时间起指定的毫秒数

- 示例代码

```
1  public class DateDemo01 {
2      public static void main(String[] args) {
3          //public Date()：分配一个 Date对象，并初始化，以便它代表它被分配的时间，精确到毫秒
4          Date d1 = new Date();
5          System.out.println(d1);
6
7          //public Date(long date)：分配一个 Date对象，并将其初始化为表示从标准基准时间起
   指定的毫秒数
8          long date = 1000*60*60;
9          Date d2 = new Date(date);
10         System.out.println(d2);
11     }
12 }
```

2.2Date类常用方法（应用）

- 常用方法

方法名	说明
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
public void setTime(long time)	设置时间，给的是毫秒值

- 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);
    }
}
```

2.3SimpleDateFormat类（应用）

- SimpleDateFormat类概述

  SimpleDateFormat是一个具体的类，用于以区域设置敏感的方式格式化和解析日期。

  我们重点学习日期格式化和解析

- SimpleDateFormat类构造方法

方法名	说明
public SimpleDateFormat()	构造一个SimpleDateFormat，使用默认模式和日期格式
public SimpleDateFormat(String pattern)	构造一个SimpleDateFormat使用给定的模式和默认的日期格式

- SimpleDateFormat类的常用方法
  - 格式化(从Date到String)
    - public final String format(Date date)：将日期格式化成日期/时间字符串
  - 解析(从String到Date)

    - public Date parse(String source)：从给定字符串的开始解析文本以生成日期
- 示例代码

```
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();
        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);
    }
}
```

2.4日期工具类案例（应用）

- 案例需求

  定义一个日期工具类(DateUtils)，包含两个方法：把日期转换为指定格式的字符串；把字符串解析为指定格式的日期，然后定义一个测试类(DateDemo)，测试日期工具类的方法

- 代码实现
    - 工具类

```
public class DateUtils {
    private DateUtils() {}

    /*
        把日期转为指定格式的字符串
        返回值类型：String
        参数：Date date, String format
     */
    public static String dateToString(Date date, String format) {
        SimpleDateFormat sdf = new SimpleDateFormat(format);
        String s = sdf.format(date);
        return s;
    }


    /*
        把字符串解析为指定格式的日期
        返回值类型：Date
        参数：String s, String format
     */
```

```
    public static Date stringToDate(String s, String format) throws
ParseException {
        SimpleDateFormat sdf = new SimpleDateFormat(format);
        Date d = sdf.parse(s);
        return d;
    }

}
```

- 测试类

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

        String s1 = DateUtils.dateToString(d, "yyyy年MM月dd日 HH:mm:ss");
        System.out.println(s1);

        String s2 = DateUtils.dateToString(d, "yyyy年MM月dd日");
        System.out.println(s2);

        String s3 = DateUtils.dateToString(d, "HH:mm:ss");
        System.out.println(s3);
        System.out.println("--------");

        String s = "2048-08-09 12:12:12";
        Date dd = DateUtils.stringToDate(s, "yyyy-MM-dd HH:mm:ss");
        System.out.println(dd);
    }
}
```

2.5Calendar类（应用）

- Calendar类概述

  Calendar 为特定瞬间与一组日历字段之间的转换提供了一些方法，并为操作日历字段提供了一些方法

  Calendar 提供了一个类方法 getInstance 用于获取这种类型的一般有用的对象。

  该方法返回一个Calendar 对象。

  其日历字段已使用当前日期和时间初始化：Calendar rightNow = Calendar.getInstance();

- Calendar类常用方法

方法名	说明
public int get(int field)	返回给定日历字段的值
public abstract void add(int field, int amount)	根据日历的规则，将指定的时间量添加或减去给定的日历字段
public final void set(int year,int month,int date)	设置当前日历的年月日

- 示例代码

```
public class CalendarDemo {
    public static void main(String[] args) {
        //获取日历类对象
        Calendar c = Calendar.getInstance();

        //public int get(int field):返回给定日历字段的值
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH) + 1;
        int date = c.get(Calendar.DATE);
        System.out.println(year + "年" + month + "月" + date + "日");

        //public abstract void add(int field, int amount):根据日历的规则，将指定的时间量添加或减去给定的日历字段
        //需求1:3年前的今天
//        c.add(Calendar.YEAR,-3);
//        year = c.get(Calendar.YEAR);
//        month = c.get(Calendar.MONTH) + 1;
//        date = c.get(Calendar.DATE);
//        System.out.println(year + "年" + month + "月" + date + "日");

        //需求2:10年后的10天前
//        c.add(Calendar.YEAR,10);
//        c.add(Calendar.DATE,-10);
//        year = c.get(Calendar.YEAR);
//        month = c.get(Calendar.MONTH) + 1;
//        date = c.get(Calendar.DATE);
//        System.out.println(year + "年" + month + "月" + date + "日");

        //public final void set(int year,int month,int date):设置当前日历的年月日
        c.set(2050,10,10);
        year = c.get(Calendar.YEAR);
        month = c.get(Calendar.MONTH) + 1;
        date = c.get(Calendar.DATE);
        System.out.println(year + "年" + month + "月" + date + "日");

    }
}
```

2.6二月天案例（应用）

- 案例需求

  获取任意一年的二月有多少天

- 代码实现

```
public class CalendarTest {
    public static void main(String[] args) {
        //键盘录入任意的年份
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入年：");
        int year = sc.nextInt();

        //设置日历对象的年、月、日
        Calendar c = Calendar.getInstance();
        c.set(year, 2, 1);

        //3月1日往前推一天，就是2月的最后一天
        c.add(Calendar.DATE, -1);

        //获取这一天输出即可
        int date = c.get(Calendar.DATE);
        System.out.println(year + "年的2月份有" + date + "天");
    }
}
```

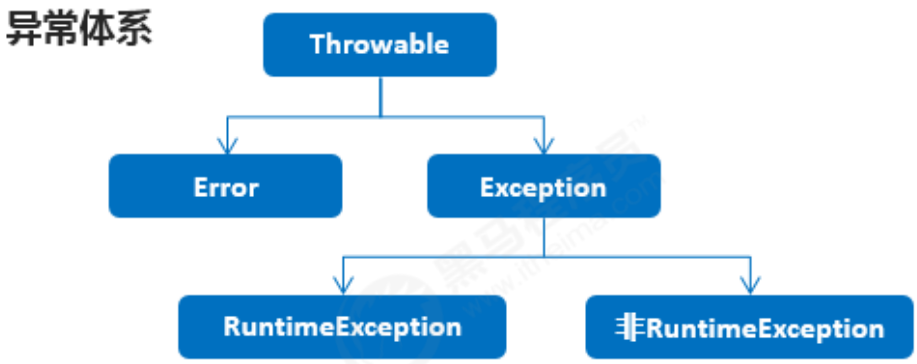
3.异常

3.1异常（记忆）

- 异常的概述

  异常就是程序出现了不正常的情况

- 异常的体系结构



Error：严重问题，不需要处理

Exception：称为异常类，它表示程序本身可以处理的问题

- RuntimeException：在编译期是不检查的，出现问题后，需要我们回来修改代码
- 非 RuntimeException：编译期就必须处理的，否则程序不能通过编译，就更不能正常运行了

3.2JVM默认处理异常的方式（理解）

- 如果程序出现了问题，我们没有做任何处理，最终JVM 会做默认的处理，处理方式有如下两个步骤：
- 把异常的名称，错误原因及异常出现的位置等信息输出在了控制台
- 程序停止执行

3.3try-catch方式处理异常（应用）

- 定义格式

```
try {
    可能出现异常的代码;
} catch(异常类名 变量名) {
    异常的处理代码;
}
```

- 执行流程
  - 程序从 try 里面的代码开始执行
  - 出现异常，就会跳转到对应的 catch 里面去执行
  - 执行完毕之后，程序还可以继续往下执行
- 示例代码

```
public class ExceptionDemo01 {
    public static void main(String[] args) {
        System.out.println("开始");
        method();
        System.out.println("结束");
    }

    public static void method() {
        try {
            int[] arr = {1, 2, 3};
            System.out.println(arr[3]);
            System.out.println("这里能够访问到吗");
        } catch (ArrayIndexOutOfBoundsException e) {
//            System.out.println("你访问的数组索引不存在，请回去修改为正确的索引");
            e.printStackTrace();
        }
    }
}
```

3.4Throwable成员方法（应用）

- 常用方法

方法名	说明
public String getMessage()	返回此 throwable 的详细消息字符串
public String toString()	返回此可抛出的简短描述
public void printStackTrace()	把异常的错误信息输出在控制台

- 示例代码

```
public class ExceptionDemo02 {
    public static void main(String[] args) {
        System.out.println("开始");
        method();
        System.out.println("结束");
    }

    public static void method() {
        try {
            int[] arr = {1, 2, 3};
            System.out.println(arr[3]); //new ArrayIndexOutOfBoundsException();
            System.out.println("这里能够访问到吗");
        } catch (ArrayIndexOutOfBoundsException e) { //new
ArrayIndexOutOfBoundsException();
//            e.printStackTrace();

            //public String getMessage():返回此 throwable 的详细消息字符串
//            System.out.println(e.getMessage());
            //Index 3 out of bounds for length 3

            //public String toString():返回此可抛出的简短描述
//            System.out.println(e.toString());
            //java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds
for length 3

            //public void printStackTrace():把异常的错误信息输出在控制台
            e.printStackTrace();
//            java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds
for length 3
//            at com.itheima_02.ExceptionDemo02.method(ExceptionDemo02.java:18)
//            at com.itheima_02.ExceptionDemo02.main(ExceptionDemo02.java:11)

        }
    }
}
```

3.5编译时异常和运行时异常的区别（记忆）

- 编译时异常
  - 都是Exception类及其子类
  - 必须显示处理，否则程序就会发生错误，无法通过编译
- 运行时异常
  - 都是RuntimeException类及其子类
  - 无需显示处理，也可以和编译时异常一样处理

3.6throws方式处理异常（应用）

- 定义格式

```
public void 方法() throws 异常类名 {

}
```

- 示例代码

```
public class ExceptionDemo {
    public static void main(String[] args) {
        System.out.println("开始");
//        method();
        try {
            method2();
        }catch (ParseException e) {
            e.printStackTrace();
        }
        System.out.println("结束");
    }

    //编译时异常
    public static void method2() throws ParseException {
        String s = "2048-08-09";
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        Date d = sdf.parse(s);
        System.out.println(d);
    }

    //运行时异常
    public static void method() throws ArrayIndexOutOfBoundsException {
        int[] arr = {1, 2, 3};
        System.out.println(arr[3]);
    }
}
```

- 注意事项
  - 这个throws格式是跟在方法的括号后面的
  - 编译时异常必须要进行处理，两种处理方案：try...catch ...或者 throws，如果采用 throws 这种方案，将来谁调用谁处理
  - 运行时异常可以不处理，出现问题后，需要我们回来修改代码

3.7throws和throw的区别（记忆）

throws

- 用在方法声明后面，跟的是异常类名
- 表示抛出异常，由该方法的调用者来处理
- 表示出现异常的一种可能性，并不一定会发生这些异常

throw

- 用在方法体内，跟的是异常对象名
- 表示抛出异常，由方法体内的语句处理
- 执行 throw 一定抛出了某种异常

3.8自定义异常（应用）

- 自定义异常类

```java
public class ScoreException extends Exception {

    public ScoreException() {}

    public ScoreException(String message) {
        super(message);
    }

}
```

- 老师类

```java
public class Teacher {
    public void checkScore(int score) throws ScoreException {
        if(score<0 || score>100) {
//            throw new ScoreException();
            throw new ScoreException("你给的分数有误，分数应该在0-100之间");
        } else {
            System.out.println("成绩正常");
        }
    }
}
```

- 测试类

```java
public class Demo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入分数：");

        int score = sc.nextInt();

        Teacher t = new Teacher();
        try {
            t.checkScore(score);
        } catch (ScoreException e) {
            e.printStackTrace();
        }
    }
}
```