JAVA SE 知识点总结

2020年12月9日 19:01

01 Java语言基础：

（好像没啥好说的，基本语法与c++相同）

- java的原生数据类型：byte，int，short，long，float，double，boolean，char
  - 对于这些数据类型，传递参数和用=赋值的时候，直接拷贝变量的值
  - 而对于其他任何类型，不论是java自带的还是用户创建的，都视为引用类型（reference type），其创建的是引用变量而不是原生变量，传递对象变量的时候只传递引用（直接打印对象变量名你会发现结果是对象的地址）
- java定义数组（数组是一个对象Object，数组名是一个变量引用Variable）：
  - 一维数组：
    - int[] 数组名 = new int[数组大小] // 只能用new来创建数组，否则你只能得到一个空的变量引用
    - 【java的数组都存储在堆中】
    - 【数组的属性和方法：
      .length 返回数组的长度
      .clone() 返回一个相同的数组拷贝】
    - 【java的函数返回值类型可以是数组（如int[]）】
  - 二维数组：
    - int[][] 数组名 -> 相当于数组嵌套数组，因此调用数组名.length你只能得到二维数组有几行
  - 对象引用数组：
    - 当声明数组的类型是引用类型时，数组中存储的所有元素都是对象引用，而非对象本事（切记！）
    - 例：创建int[] list = new int[10]时，每个list[i]都有一个值0，
      但创建Object[] list_obj = new Object[10]时，每个list_obj[i]都是空引用null,需要对每个list_obj[i]再进行一次new对象的操作

Java数据域中各种类型的值范围：

	最小值	最大值	默认值
byte（8位字节）	-128	127	0
short（16位整数）	-2^15	2^15-1	0
int（32位整数）	-2^31	2^31-1	0
long（64位整数）	-2^63	2^63-1	0L
float(32位单精度浮点)			0.0f
double(64位双精度浮点)			0.0d
boolean(1位布尔值)	false	true	false
char(16位Unicode字符)	'\u0000'	'\uffff'	'\u0000'
reference type(引用类型)			null

注意，默认值只对类的全局变量和对象中的state数据有效，任何函数中的局部变量，java都不会为其赋默认值，所以所有的局部变量一定要手动赋初值。

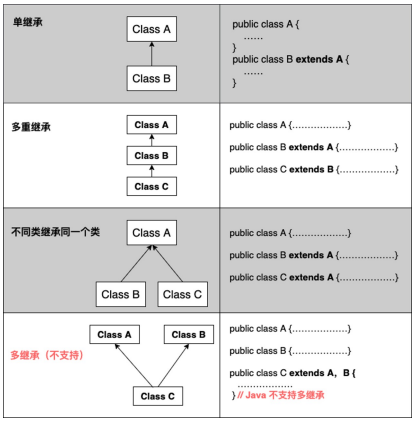
---

02 Java面向对象知识点：

- java中，一个文件中只能有一个public的类（不计类中类），文件名与该类名相同。且java的每个文件都必须要有一个与该文件名相同的类。
- java类的访问控制修饰字（可以直接修饰类，也可以单独修饰类的成员）：
  - public：可被所有域访问（类内部，本包，子类和外部包）
  - private：只能在类内部被访问
  - protected：不能被外部包访问，可以被本包、类内部和子类访问
  - default(不加访问关键字)：只能被本包和类内部访问，即必须是在同一个包下的类才可以访问
- java中的每个对象有两个属性：
  - 状态 state：表示该对象中存储的一些数据，又称数据域
  - 行为 behavior：表示该对象可以调用的函数，又称方法
- java只能在类中定义方法
- java类的构造函数可以重载，ctor只在对象使用new关键字创建的时候调用
- java类也有默认构造函数，原理与c++相同
  - 如果一个类手动定义了有参构造函数，编译器将不会为其构造默认构造函数
    - 这可能引起子类在隐式调用父类默认ctor时发生异常，所以建议在定义有参构造函数时也要手动定义无参构造函数。
- java采用自动垃圾回收机制，没有析构函数的说法，但可以重载每个对象的.finalize()的函数来实现回收内存时执行某些操作
- 静态类的实现 - 静态修饰字static
  - 静态属性：
    - 类中声明的变量前加上static关键字：静态属性在类本身加载的时候就创建，不受任何类对象的限制，可以直接通过 类名.静态属性名 访问，静态对象只有一份拷贝，任何对象修改静态对象时，就是在修改整个类的静态对象
  - 静态方法：
    - 方法前加上static修饰字：静态方法在类本身加载的时候就创建，类对象都共用静态方法，可以通过 类名.静态方法名 访问
    - 【注意：类的静态方法中不允许调用任何类的非静态方法、使用和修改非静态成员，以及使用this和super关键字】
- 静态内部类：在一个类中嵌套一个类，并且使用static设置为静态类，则使用外部类的类名可以直接访问静态内部类中的所有内容（可以类比c的结构体理解）
- 抽象类的实现 - 抽象修饰字abstract
  - 抽象类：声明类前加入修饰字abstract，抽象类不能实例化出对象，但仍然可以定义数据域和方法，并且可以被继承
  - 抽象方法：声明方法前加入修饰字abstract，抽象方法的声明直接以;结尾，不实现任何函数体内容。
    【拥有抽象方法的类必须声明为抽象类】
    【如果一个子类从含有抽象方法的类中继承，则其必须实现父类的所有抽象方法，否则该子类仍

必须被定义为抽象类】

- 常量:
  - 使用final修饰字修饰变量，则该变量不能再被域中的其他语句修改
  - 在java类中，使用static final 修饰的变量可以被视为该类中不可修改和拷贝的常量
  - final可以修饰类或方法，被final修饰的类不能被继承，被final修饰的方法不能被子类覆写（与c++类似）
- 继承语法:
  - 使用extends关键字，子类只会继承父类的非private方法
  - Java只有单继承
  - java的所有类都继承Object根类
  - 子类可以覆写父类的方法，前提是 函数的名称、所有参数名称和返回值类型都必须与父类相同，且覆写的方法是子类可以访问的方法。
    - 也就是说，private方法不能被覆写，static和final方法可以被继承但不能被覆写
- this关键词:
  - 在遇到this时，会立刻用当前的类对象代替this，即this代表当前对象
  - 可以用this.xxx引用当前对象的属性
  - 可以用this(xxx)调用当前对象所在类的重载构造函数（要先重载对应参数的ctor哦）
- super关键字:
  - 只能在子类中调用，super代表当前对象的一个父类对象
  - 可以用super调用父类的构造函数或方法
  - 如果在子类的构造函数中没有显式地调用super(xxx),编译器会自动在构造函数的首局加上super()调用父类的默认构造函数
- java的接口 interface:
  - 声明: [访问控制符] interface 接口名 （extends [其他的接口]）
  - 接口与类相似，可以声明数据域变量和方法，但变量必须是常量（static final类型），方法必须是抽象方法。
    - （Java8以后允许接口中定义默认(不加修饰词的default)方法和静态方法，静态方法只能通过接口名.方法名 调用，而默认方法只能通过实例化实现了接口的类的对象来调用）
  - 接口不能实例化对象，也没有构造函数
  - 类可以实现接口，在类的声明后加入implements关键字即可，实现接口的类必须实现接口中的所有抽象方法
  - 接口支持“多继承”，即一个接口可以被多个类实现
- 多态
  - 多态存在的条件:
    - 有继承链
    - 子类覆写了父类的方法
    - 存在父类引用指向子类的对象（或方法）
  - 实现方法：动态联编 dynamic Binding
    - 如果一个对象o是类C1的对象，同时C1的父类是C2,C2的父类是C3……直到根类Cn
    - 那么如果要调用o的一个函数func，JVM会按顺序在C1、C2…Cn中查找对应函数func，如果存在该函数，最先找到的函数将被调用
    - 因此，如果在不知道传入的参数是那个子类的引用时，可以直接使用根类对象引用作为参数，只要继承链上都继承或覆写了函数即可，这就是java实现的狭义多态。
  - 具体实现上，可以使用直接覆写同名函数、使用接口和使用抽象类的方式实现多态
- Java类型转换:
  - 只要强制转换 类型B b = (类型B)类型A的对象a
  - 可以上转下也可以下转上，但下转上可能丢失数据



---

03 JavaFX基本框架使用:

- 基本布局:
  - Application - Stage - Sence - Pane - [Elements]
  - 应用程序 - 舞台 - 场景 - 面板(可嵌套) - 元件
- javafx使用Application类来创建GUI程序，所写程序的主类必须继承Application类，并且覆写该类中的start函数，然后再main函数中调用`launch(args)`函数启动程序
- start函数需要传入一个舞台（Stage）类对象引用作为参数，舞台可以视为一个窗口，默认情况下使用javafx定义好的`primaryStage`作为参数创建一个窗口即可。
- 舞台需要设置场景（Sence），场景可以被视为一个GUI程序窗口在一个特定状态下的所有组件的集合，在start函数中需要调用`primaryStage.setScene(scene)`将创建好的场景载入舞台，然后调用`primaryStage.show()`函数显示舞台。
- 在start函数中你需要创建一个场景类对象，基本语法如下:
  - Sence mysence = new Scene(面板或元素对象引用 , [场景宽, 场景高])
  - 在javafx的场景中，坐标轴的原点位于左上角，向右x增大，向下y增大
- 场景中一般以一个父节点Parent为根，父节点一般包括控制类元素(Control)和面板(Pane)，面板为GUI程序提供了排版和布局的框架，javafx的提供如下几种主要面板，其各自代表不同的布局方法:
  Pane布局方法:
    - Pane：默认的面板，不通过任何布局

```
public class [你的主类名称] extends Application {
  @Override
  public void start(Stage primaryStage) {

      /** DO SOMETHING */

    //设置舞台（窗口）标题
    primaryStage.setTitle("Hello, JavaFX!");
    // 将场景放到舞台上(setScene)
    primaryStage.setScene(scene);
    // 显示舞台(show)
    primaryStage.show();
  }
 public static void main(String[] args) {
    launch(args);
  }
}
```

- □ 以下布局面板都是Pane的子类，也可以自行继承Pane编写你需要的布局
- □ Pane类的所有子节点构成一个列表类，可以使用getChildren()获取到该列表，对列表类进行处理可以添加或删除面板里的元素。

▪ HBox：水平布局 横向排列所有节点

▪ Vbox：垂直布局 垂直排列所有节点

Hbox和Vbox的数据域：alignment 决定排列节点的对齐，用Pos枚举类表示，默认靠左
fillHeight 布尔类型，是否将可变大小的节点元素拉伸以铺满布局，默认是
spaceing 浮点类型，决定两个节点之间的间隔，默认0，可在构造函数中指定

- □ 向Hbox和Vbox里增加元素，使用.getChildren().add(Node) 方法

Pos枚举：

Pos.TOP_LEFT/TOP_RIGHT/TOP_CENTER	顶部靠左/靠右/中心
Pos.BOTTOM_LEFT/BOTTOM_RIGHT/BOTTOM_CENTER	底部靠左/靠右/中心
Pos.CENTER/CENTER_LEFT/CENTER_RIGHT	正中心/中心靠左/中心靠右

▪ BorderPane：四周布局，可以塞入五个节点，按照上、下、左、右、中心排列

主要数据域：top、right、bottom、left、center
均为节点引用类型，表示面板四周布置的节点，默认值为null

- □ BorderPane里的每一个节点有自己的对齐方式，使用setAlignment(childNode,Pos)设置
- □ 向BorderPane里增加元素，使用.setTop/Bottom/Left/Right/Center(Node) 方法

▪ StackPane：堆叠布局 新的节点覆盖在旧的节点上方

- □ 向StackPane里增加元素，使用.getChildren().add(Node) 方法

▪ GridPane：网格布局 按照网格坐标确定的相对位置排列节点

主要数据域：alignment 节点排列对齐
hgap、vgap 浮点类型，决定行与行或列与列之间的间隔，默认0，可在构造函数中指定
gridLineVisible 布尔类型，决定网格线是否可见，默认false

- □ 向GridPane里增加元素，可以使用：
  - ◆ .add(Node,A,B)：向网格坐标(A,B)插入节点node
  - ◆ .addColumn(X,Nodes....)：向网格第x列加入若干节点
  - ◆ .addRow(X,Nodes....)：向网格第x行加入若干节点
- □ 可以使用getRowIndex()或getColunmnIndex()来获得某个节点在网格中的位置

▪ FlowPane：流布局 向一个方向排列元素，直到场景的边缘后换行继续排列

主要数据域：alignment 节点排列对齐
hgap、vgap 浮点类型，决定行与行或列与列之间的间隔，默认0，可在构造函数中指定
orientation Orientation类型，决定节点的生长方向，值为水平(Orientation.HORZONTAL)或垂直(Orientation.VERTICAL)，默认水平，可在构造函数中指定

- □ 向FlowPane中添加元素，使用.getChildren().add(Node) 方法，添加多个元素可以使用.getChildren().addAll(Node)方法

- 元素：在javafx的底层，即面板或父节点里需要放入的东西，其可以是文本、图片、可交互对象等，因为所有元素类都继承节点(Node)类，所以也称元素为节点。
  【所有的节点都可以调用 setStyle(str)函数，其参数为一段CSS格式的字符串，可以方便地设置节点的外观属性，并且可以调用setRotate(degree)函数直接设置旋转角度】
  总结javafx提供的常用节点类：

【形状类：都继承Shape父类】

○ 文字展示框类：Text

▪ 主要数据域：text String类型，要展示的文本
x、y 浮点类型，文本的位置
underline、strikethrough 布尔类型，决定文本是否有下划线删除线
font Font对象引用类型，决定文本的字体

○ 图形类：通过CG绘制图形

子类	主要数据域
Line	startX、startY、endX、endY，表示线起终点的坐标
Circle	centerX、centerY 表示圆心坐标 radius 表示半径
Rectangle	x、y 表示左上角顶点的坐标 width、height 表示矩形的宽和高 arcWidth、arcHeight表示矩形宽和高圆角边的半径
Ellipse	centerX、centerY 表示椭圆中心坐标 radiusX、radiusY 表示椭圆的两个半径
Arc	centerX、centerY 表示弧所在椭圆中心坐标 radiusX、radiusY 表示弧所在椭圆的两个半径 startAngle 弧开始点在椭圆上的角度 length 弧的长度 type 弧的类型：ArcType.OPEN 开放弧 / ArcType.ROUND 扇形 / ArcType.CHORD 弓形
Polygon	x[] y[] 记录每个顶点的坐标

【图像类】

○ 图片类：Image

▪ 使用 Image(图像的URL网址或文件地址) 有参构造函数创建，只能加载图片，不能显示图片

○ 图片展示框类：ImageView 用来显示图片

▪ 可以使用一个Image对象创建，也可以直接使用URL或文件地址创建
▪ 主要数据域：
- □ x、y 图片展示框的大小
- □ fixHeight、fixWidth 图片展示框中图片的大小，如果图片大小于此不符，强制拉伸图片
- □ image 要展示的图片对象

【控制类】（控制类也可以直接放入父节点）

- 按钮类：Botton
  - 有参构造函数传入一个字符串，指定按钮上要显示的文字
- 文本标签类： Lable 显示单行文本
  - 有参构造函数传入一个字符串，指定标签上要显示的文字
- 文本输入框类：TextField 单行文字输入框 / TextArea 多行文字输入框
  - 有参构造函数可以传入一个字符串，指定输入框为空时的提示文字
- 单项选择框类：RadioButton / 复选框类：CheckBox
  - 有参构造函数可以传入一个字符串，指定选择框的提示文字
- 滚动条类 ScrollBar / 滑条类 Slider
  - 用.setOrientation()确定条的水平和垂直方向
  - 滚动条要使用.valuePropery监听和绑定滚动查看的其他元素
  - 滑条可以直接使用.setValue(int)设置值，并可以使用.setShowTickLabels()显示当前的值

- 事件处理：

  节点对象在用户于其互动操作时会触发相应事件类的相应，程序员通过编写实现EventHandler事件处理接口的类，来处理事件。

  - javafx里的事件类型主要有ActionEvent（控制类UI组件收到激活的事件）、MouseEvent、KeyEvent（鼠标键盘输入事件）等
  - 事件处理函数：用 节点对象.setOnAction(事件处理类对象) 等事件注册函数（这里以ActionEven举例）来调用
  - 编写内部类和匿名内部类：处理一个事件需要写一个类，而一般这个类在该事件之外就不会再使用，所以为了方便，可以使用匿名内部类，直接在事件处理函数中编写类，语法：

    ```
    [事件处理函数](new EventHandler<[事件类型]> {
        @Override    //一定要覆写hanle函数
        public void handle([事件类型] e) {
          /** HANDLE THE EVENT e */;
        }
    });
    ```

  - lambda表达式：跳过类和函数语句，直接对变量进行语句执行的一种语法
    - 格式 (type1 param1, type2 param2, ...) -> { statements; } (单个语句时可以不打括号和分号)
      - 表达式中的type必须是明确的类型，当只有一个变量时，其类型可以省略，由编译器自行推导
    - 可以再一次简化匿名内部类，语法如下：

      ```
      [事件处理函数](e -> {  //e的类型自动推导了
          /** HANDLE THE EVENT e */;
      });
      ```

- 事件监听：
  - javafx里的为常见的数据类型（int/double）编写了以Property结尾的属性对象（如IntProperty、DoubleProperty），可以通过getXXXPropery()获得，其与元素的数据域相对应，属性对象可以用于元素之间绑定(使用.bind()函数绑定两个属性对象，使得一个元素的属性变化时，另一个元素的属性也变化)
  - 属性对象可以用于事件的监听，即程序员编写监听类Listener，使得当元素的对应属性变化时，立刻对事件进行处理
  - 实现事件监听需要让 属性对象 调用 .addListener(监听类对象) 函数，监听类必须实现InvalidationListener接口，且覆写invalidated(Observable ov)函数，函数传入的对象时一个多态的Observable对象
  - 同样可以使用lambda表达式简化监听函数的实现，语法如下：

    ```
    .addListener(ov -> {  //ov的类型自动推导了
        /** HANDLE THE OBSERVABLE OBJECT ov */;
    });
    ```

- 动画实现：
  - Animation类用来在javafx中实现动画
  - Timeline类是Animation类的重要子类，用来实现动画的核心——打关键帧
    - 创建Timeline类： Timeline [你的时间轴名称] = new Timeline([关键帧集合])
    - 其中关键帧集合用KeyFrame类实现，新建一个KeyFrame类对象使用：
      - new KeyFrame([一帧的时间]，[每一帧的事件处理对象]);
      - 一帧的时间可以使用Duration.mills(int)赋值，单位为毫秒
      - 时间处理对象即使用ActionEvent事件类型创建，可以用lambda表达式
    - 使用 [时间轴名称].setCycleCount() 设置动画的循环次数，参数传入Timeline.INDEFINITE即可无尽循环播放
    - 最后使用 [时间轴名称].play()即可播放你的动画
    - 其他控制播放的函数：
      - .pause() 暂停播放，再次调用.play()可以继续
      - .stop() 停止播放
      - .setDelay(Duration类值) 设置延迟时间
      - .setRate(double值)设置播放速率

```
EventHandler<ActionEvent> eventHandler = e -> {
     /** MAKE ANIMATION MOVE */
};
// Create an animation
Timeline animation = new Timeline(
      new KeyFrame(Duration.millis(500), eventHandler));
animation.setCycleCount(Timeline.INDEFINITE);
animation.play(); // Start animation
```

---

04 Java异常处理框架：

- java异常类：
  - 异常全部继承于Exception父类，可以手动继承创建你自己的异常类
  - 错误异常全部继承于Error父类，Error和Exception都继承于Throwable父类，可以被catch（但不建议catch由内存错误或虚拟机错误引起的Error）
- 可查异常：在程序编写时就可以得知是否可能引起的异常，否则称为不可查异常
- 不可查异常包括RuntimeError（运行时异常）和错误Error，其他异常都是可查异常，不可查异常可能出现在程序的任何地方。
- java会强制要求程序员处理可查异常，但不会强制处理不可查异常
- 可以在函数前加入throw XXXException的声明，则函数中扔出的异常可以不使用try-catch处理，而是交由调用该函数的上一层代码处理。
- try-catch-finally框架:

```
try{
    //代码
    //可能包含throw语句，throw语句会扔出异常
}
catch(某一个异常类对象e){
    //如果捕捉到与e相同类型的异常，终止try块中的代码，执行catch块中的代码
    //一个try可以对应多种catch，如果try中可能出现若干可查异常，则这些异常都必须被catch
    //或者直接使用Exception e 接住所有可能产生的异常
}
finally{
    //无论有没有发生异常，都会执行该语句块的代码
}
```

- try with resource:
  - java9新特性
  - 在try后面加入([声明try块中使用的变量资源])，声明一些仅在该try块中使用的局部变量

---

05 Java I/O类和文件类：

操作文件：文件类 File

用文件名或路径（String类型）创建文件对象，可以获取其长度、可见性、可读性等属性

文件类对象只用来对文件进行操作（换句话说，对FCB操作），不能读写文件内容

读写文件：Java的I/O操作类简单来说可以分为三种：

- 字符文本：直接使用Unicode字符进行输入输出，一般命名为XXXReader/XXXWriter
  - FileReader和FileWriter：
    - 读文件：new FileReader([String类型的文件名 或 File对象])
      - 文件名必须存在，否则会扔出文件未找到异常
      - 读文件方法：
        - .read()方法读取一个Unicode字符，注意返回值是对应的int（0-65536）
        - .read(char[])的形式从输入流读取字符到数组
        - .read(char[],pos,len)读取字符到数组中的[pos….pos+len-1]的位置
    - 写文件：new FileWriter([String文件名或File对象] , [{布尔类型)是否为追加模式])
      - 如果文件名不存在，会新建一个文件
      - 如果追加模式的参数不提供或者传入的为false，writer会覆写打开的文件
      - 写文件方法：
        - .write(int c)方法写一个Unicode字符
        - .write(char[]) 和.write(char[],pos,len) 将字符数组写入输出流
        - .write(String) 和.write(String,pos,len) 将字符串写入输出流
        - .close()关闭文件【切记不要忘了】
  - FileReader和FileWriter的父类是InputStreamReader和OutputStreamWriter实质上是将字符转化成字节bytes的形式。
  - 格式化输出：使用PrintWriter和PrintStream中的print方法，类似于System.out.print
- 二进制流：使用字节码的形式进行数据传输
  - 1、直接读写字节类型：FileInputStream和FileOutputStream：
    - 读文件：new FileInputStream([String类型的文件名 或 File对象])
      - 文件名必须存在，否则会扔出文件未找到异常
      - 读文件方法：
        - .read()方法读取一个字节bytes，注意返回值是对应的int（0-255）
          - 如果读取到文件尾没有可用的字节时，会返回-1
        - .read(byte[]) 从输入流读取字节到字节数组类型
        - .read(byte[],pos,len) 读取字节到数组中的[pos….pos+len-1]的位置
    - 写文件：new FileOutputStream([String文件名或File对象] , [是否追加模式])
      - 如果文件名不存在，会新建一个文件
      - 如果追加模式的参数不提供或者传入的为false，writer会覆写打开的文件
      - 写文件方法：
        - .write(byte b)方法写一个字节
        - .write(byte[]) 和.write(byte[],pos,len) 将字节数组写入输出流
        - .close()关闭文件【切记不要忘了】

- 2、从buffer中读写数据：DataInputStream和DataOutputStream：
  - 对象都不能直接构造，必须从另一个Stream对象中构造
  - 因此要实现从文件和缓冲区Buffer读写数据，数据流读写对象应声明为：

    数据写出流：
    ```
    DataOutputStream output = new DataOutputStream(
       new BufferedOutputStream(new FileOutputStream([文件名])));
    ```
    - ◇ 然后使用output中的各种封装好的函数即可写入多种数据类型
    - ◇ 使用后记得close()

    数据读入流：
    ```
    DataInputStream input = new DataInputStream(
       newBufferedInputStream(new FileInputStream([文件名])));
    ```
    - ◇ 然后使用input中的各种封装好的函数即可写入多种数据类型
- 3、读写对象数据流类型：ObjectInputStream和ObjectOutputStream：
  - Data流不能读写对象，要实现对象的二进制流读写必须使用Object

    对象数据写出流：
    ```
    ObjectOutputStream output = new ObjectOutputStream(
       new FileOutputStream([文件名]));
    ```
    - ◇ 使用output.writeObjcet([对象引用])即可写入对象,并且同样支持Data流的所有写入函数
    - ◇ 使用后记得close()

    对象数据读入流：
    ```
    ObjectInputStream input = new ObjectInputStream(
       new FileInputStream([文件名]));
    ```
    - ◇ 然后使用input.readObject()即可读出对象，读出的对象要进行强制类型转换才能赋值

封装好的读写函数诸如:
.writeDouble/.readDouble 读写浮点类型
.writeUTF/.readUTF 读写UTF字符串类型

- 随机访问文件：使用RandomAccessFile类
  - 创建随机访问文件对象：
    - RandomAccessFile inout = new RandomAccessFile([文件名], [打开模式]);
      - ◇ 打开模式："r"只读 "w"只写 "rw"可读可写
  - 使用inout.seek(pos)函数寻找文件指针的位置，随机访问文件中的文件指针以字节为单位移动
  - 使用inout中对应的read和write函数（与Data流的类似），从当前文件指针的位置开始读写文件。

---

06 Java多线程编程

- java中每个线程都是一个线程类的对象
- 创建线程的办法：
  - 1、写一个类继承Thread类，并在其中覆写线程启动的run()函数
    run函数里的内容即为你的线程工作时执行的代码
  - 2、写一个类实现Runnable接口，同样要覆写其中的run()函数
  - 在主函数里创建线程：新建一个你的线程类的对象：
    - Thread myt = new [你的线程类名]();　　或者：
    - Thread myt = new Thread(new [你的实现接口的类名]);
  - 启动线程：myt.start();
    - 【线程会进入等待队列，CPU有空闲后就开始工作，直到所有代码执行完毕后自动退出】
    - 【直接调用线程的run函数会让主函数的线程执行代码，不能使新的线程开始工作！】
  - 其他线程类的方法：
    - isAlive() 判断线程是否在工作
    - isInterrupted() 判断线程是否在阻塞状态
    - interrupt() 立刻中断线程，如果线程已经处于阻塞，将其放入等待（ready）队列
    - setPriority() 设置线程优先级（越小越优先
    - sleep(long) 线程睡眠一段时间，单位是毫秒
    - yield() 阻塞线程一小段时间（由计算机系统决定，一般为足够一个其他线程运行一段时间）
  - 3、使用线程池（好像没说要考）：
    - 创建新的ExecutorService线程池对象：
      `ExecutorService executor = Executors.newCachedThreadPool();`
    - 然后使用`executor.execute(new [你的线程类或实现接口的类]());`即可创建一个线程
    - 使用线程池创建的线程会立刻开始工作，结束后自动退出
    - 使用 `executor.shutdown()` 让所有线程池创建的线程结束后关闭线程池，以免浪费资源
- 线程同步方法：防止两个或多个线程同时进入临界区

  【调用线程相关函数时需要使用try-catch捕捉异常！】

  1、使用join()函数：
  一旦一个线程运行到threadX.join()【其中threadX是另外一个线程对象】时，其会强制中断，直到threadX线程运行完毕并退出，才会恢复运行。

  2、使用wait()和notify()/notifyAll()函数：
  - 这三个函数是写在根类Object里的（说是刻在java DNA里的应该不过分吧~）
  - 因此任何对象都可用调用，在wait()前什么都不写调用函数时候，默认使用this.调用函数（即

当前类的对象）
    - 调用wait()时，当前线程在该对象的管程(monitor)上等待(中断)，只有对应对象的notify()或notifyAll()函数被其他线程激活时，才会继续执行当前线程
    - 如果一个对象的管程上有多个线程正在等待，notify() 由系统选择一个唤醒，notifyAll()唤醒全部。
    - 因此，只有在同一个类中进行wait()和notify()的操作时才能成功互相唤醒。如果需要同步的代码范围超过了一个类，则需要在同步的类上都添加一个Object类对象，实例化对象时需要传入相同的Object对象，并记住调用该对象的.wait()和.notify()函数进行同步。
    - 【注意：直接使用这三个函数无法对临界区上锁，因此必须进一步同步！】
- 使用synchronize进行线程同步：相当于上互斥锁
    - 对所有使用了wait()和notify()/notifyAll()进行同步的函数，都要在函数前加入 synchronized 修饰字，以标识该函数中存在wait()和notify()调用，需要上锁。
    - 或者，在函数中可用使用 synchronized（obj）{ } 规定一部分语句块为临界区，其中obj为进行同步的管程对象
        - 因此在同一个类中同步时，obj指定为this即可
        - 在不同的类之间同步时，按照上面的方法传入一个相同的Object类对象

---

07 Java网络编程

网路传输的五个主要的层：

应用层（http协议等）- 传输层(TCP协议等) - 网络层(IP地址连接) - 链接层(硬件设备链接) - 物理层(传输1和0)

客户端 Cilent，服务端 Server

【进行网络传输的部分代码（从创建socket开始），需要用try-catch捕捉IO异常！】

实现TCP协议的网络传输：

TCP需要服务端对每一个客户端都进行链接（三次握手），然后才可用传输数据，因此TCP要求服务端使用多线程同时对付多个客户端

TCP的简要步骤：

服务端	客户端
使用Server socket监听	向服务端发送请求
接受客户请求 创建一个新的流sokect连接客户端 创建一个新的线程使用该sokect创建Data流传输与客户端交流	
继续监听其他客户端	接收到服务端的sokect，可以进行流数据传输
	其他客户端继续向服务端发送请求
.....	.....

TCP具体代码框架：

服务端

```
try{
   ServerSocket ss = new ServerSocket(port);
       // 创建一个 server socket , port是监听的端口号
      System.out.println("server listening on port " + port);
   while( /**判断服务器工作的条件*/) {
         // 监听并等待接受连接：使用ServerSocket对象的accept()
          Socket socket = ss.accept();
      // 获得客户端地址
          InetAddress in_adress = socket.getInetAddress();
          System.out.println("conneted from "
      +in_adress.getHostAddress());
          // 在控制台输出客户端连接信息

          //创建线程worker，worker处理与客户端的通信：
         new Worker(socket).start();
      }
      //关闭线程
      ss.close();
   }
   catch(Exception e) {
      e.printStackTrace();
   }
===============
Worker线程的run函数部分：
try{
   // 传入socket，通过socket的输入输出流创建Data数据流
   DataInputStream inputFromClient = new DataInputStream(
               socket.getInputStream());
   DataOutputStream outputToClient = new DataOutputStream(
               socket.getOutputStream());
   // 读取客户端传来的数据，没有数据传输则等待
   XXX = inputFromClient.readXXX();
```

客户端(省略了try-catch)

```
    // 创建网络地址，传入的字符串为域名或ip地址，"localhost"表示本地
   InetAddress address = InetAddress.getByName("localhost");

   // 创建与服务器通信的socket
   Socket socket = new Socket(address,port);
      // 其他的socket创建方法：
      // Socket socket = new Socket("www.bupt.edu.cn", 80);
      // Socket socket = new Socket("10.3.9.161", 80);

   //使用该socket 创建Data数据流
   DataInputStream dis = new DataInputStream(socket.getInputStream());
   DataOutputStream dos = new
   DataOutputStream(socket.getOutputStream());

   while( //** 判断与服务器继续通信的条件 */ ){
      /** 处理数据 */
      dos.writeXXXX(XXXX); // 向服务器发送数据
      XXX = dis.readXXX(); // 从服务器接受返回的数据
   }
   socket.close();
```

```
    /** 处理数据 */
    outputToClient.writeXXXX(XXXX);// 向客户端发回数据
    ……
}
catch(Exception e) {
    e.printStackTrace();
}
```

实现UDP协议的网络传输：

UDP使用 DatagramPacket数据包 和 DatagramSocket进行传输，服务端不和客户端进行连接，双方的地址等数据全部打包到DatagramPacket传输，服务端一次只处理一个包，不需要多线程

UDP的简要步骤：

服务端	客户端
确定对应端口的DatagramSocket 创建接受包、发送返回包	创建接受返回包、发送包
等待从客户端收到包，如果同时有很多包传过来，选择一个接受	向服务端发送包
从包内读取数据进行处理， 将读到的客户端的地址和要返回的数据写入发送返回包 向对应的客户端发送返回包	等待接受返回包
继续等待接受下一个包	接收到返回包 继续向服务端发送包
……	……

UDP具体代码框架：

服务端

```
try{
      // 创建服务器对应端口的DatagramSocke
      DatagramSocket socket = new DatagramSocket(8000);
      System.out.println("Server started at " + new Date() + '\n');

      // 创建接受包和返回的发送包
      DatagramPacket receivePacket = new DatagramPacket(buf, buf.length);
      DatagramPacket sendPacket = new DatagramPacket(buf, buf.length);

      while (/**判断服务器工作的条件*/) {
        // 初始化处理包的字节数组buf的值 （buf是Bytes[]类型的数组，大小为256字节）
        Arrays.fill(buf, (byte) 0);

        // 从客户端收到包
        socket.receive(receivePacket);
        【receivePacket.getAddress().getHostName() 可以获取客户端的地址和名称】
        【receivePacket.getPort() 可以获得客户端发送数据包的端口】
        【客户端数据包的内容已经被存入buf中，使用new String(buf).trim()可以讲将字节数据转化为一个字符串对象】

        /** 处理数据 * /

        // 打包要返回的数据，并发送返回包
        sendPacket.setAddress(receivePacket.getAddress()); //打包地址
        sendPacket.setPort(receivePacket.getPort()); // 打包端口
        sendPacket.setData(XXXX.toString().getBytes()); // 打包发送的数据(字节类型)
        socket.send(sendPacket);
      }
    } catch (IOException ex) {
      ex.printStackTrace();
    }
  }
```

客户端

```
try {
      // 创建DatagramSocket
      DatagramSocket socket = new DatagramSocket();

      // 创建网络地址
      InetAddress address = InetAddress.getByName("localhost");

      // 创建发送数据的包，注意端口写在包的构造函数里
      DatagramPacket sendPacket
           = new DatagramPacket(buf, buf.length, address, 8000);

      // 创建接受返回数据的包
      DatagramPacket receivePacket
          = new DatagramPacket(buf, buf.length);

      // 初始化处理包的字节数组buf的值
      Arrays.fill(buf, (byte) 0);

      /** 处理要发送的数据 */

      // 打包并发送数据
      sendPacket.setData(Double.valueOf(r).toString().getBytes());
      socket.send(sendPacket);

      // 从服务器收到返回包
      socket.receive(receivePacket);

      /** 处理返回回来的数据 */

    } catch (IOException ex) {
      ex.printStackTrace();
    }
```

---

08 Java数据库编程：

- SQL中一个数据库由多个数据表（Table）组成，每个数据表包含列（又称 attribute属性），行（又称 tuple 记录）
- SQL语句是大小写不敏感的，可以用大写单词，也可以用小写

- SQL数据库基本语句：
  - 选择数据库

    ```
    use [数据库名称];
    ```
  - 创建数据库

    ```
    create database [数据库名称];
    ```
  - 建立数据表

    ```
    create table [数据表名称](
       [列名称] [数据类型]([最大大小]),
       …
       …
       primary key ([某一列的名称])   // 用来设置关键字，可选
    );
    ```
  - 查找数据表

    ```
    select [列名称，多个用逗号隔开] from [数据表名] where [过滤表达式];
    ```
    - select * from [数据表名]; 可用直接获得整个表格
    - 过滤表达式的格式是： [列名称] [比较运算符] [记录值]
      - 列名称不需要打引号，记录值根据记录的数据类型判断要不要打引号
      - 多个比较表达式可以用 and 或 or 连接
      - 比较运算符有 > 、<、 <>、==、>=、<=
        - 额外的比较语法有：
          - [列名称] is null --> 判断是否为空
          - [列名称] between A and B --> 判断值是否在[A,B]范围内
          - [列名称] in (A,B,C,...) --> 判断值是否在(A,B,C,...)这个列表中
  - 删除数据表

    ```
    drop table [数据表名称];
    ```
  - 增加记录

    ```
    insert into [数据表名称] ([列名称，多个用逗号隔开]) values ([记录值，多个用逗号隔开]);
    ```
  - 删除记录

    ```
    delete from [数据表名称] where [过滤表达式];
    ```
  - 修改记录

    ```
    update [数据表名称] set [列名称] = [新的记录值] where [过滤表达式];
    ```
- 使用JDBC操作数据库：

  【注意：执行数据库语句时，可能产生SQL异常，要加入try-actch语句】

  ```
  [函数名] throws SQLException, ClassNotFoundException {
      // 加载 JDBC driver (驱动类) 到程序中
      // Class.forName("com.mysql.jdbc.Driver"); 【mysql 5.x的加载语句】
      Class.forName("com.mysql.cj.jdbc.Driver");  【mysql 8.x的加载语句】

      // 建立与数据库的连接
      // String url5 = "jdbc:mysql://localhost/[数据库名]"; 【5.x】
      String url8 = "jdbc:mysql://localhost/[数据库名]?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC";【8.x】
      Connection connection = DriverManager.getConnection(url8, [登录数据库的用户名], [密码]);

      // 创建一个表达式对象，其可用用来执行SQL语句
      Statement statement = connection.createStatement();
  ```

  【表达式对象可用调用的方法：

statement.executeUpdate(字符串类型的SQL语句)	用来执行一条update、delete或insert语句
statement.executeQuery(字符串类型的SQL语句)	用来执行一条select语句

  】

  ```
  //ResultSet 类型的对象用来存储选择语句的返回值，其中存储了一个表格选择出的信息
  ResultSet resultSet = statement.executeQuery("XXXXX");
  // 默认resultSet的指针指向第一行，使用.next()可用将指针移到下一行
  // 使用.getString(n)可以以字符串的形式获得当前指针行第n列的数据(n从1开始计数)
  ```