《Android 游戏案例开发与关键技术》

作者：华清远见

第 3 章 游戏中的多线程

本章简介

在游戏场景中，我们可以看到各个游戏元素都在发生一些变化，比如主角的移动、背景的变化、道具的运动等，这些变化有些相互影响，有些则不然。在捕鱼游戏中，我们发射一枚炮弹，这枚炮弹会沿着发射轨迹移动，鱼群也会以不同的方向自由地移动，这些操作都是通过多线程来实现的。多线程游戏模型的算法思想是：将各游戏元素的动作用独立的线程进行处理，特定功能模块用指定的线程进行处理。我们在程序中实现的都是模拟的多线程，用以处理不同的事物，但实际上在计算机处理的物理层，仍然以单线程的模式工作，或者同时运行计算机内核数量的线程。Java 当中已经封装好了多线程的机制，即 Android 当中多线程不是依赖于虚拟机的，而是依赖于底层 Linux 的。

3.1 多线程的使用

3.1.1 游戏逻辑的实现

在 Android 游戏开发中，逻辑部分通常由线程实现，比如小鱼的游动、小鱼的状态处理、子弹的路径、碰撞检测等。

这里我们以小鱼游动为例，介绍游戏中的线程的应用：我们让线程以一个周期的方式循环变化鱼在屏幕上的坐标，这样，每次绘制周期中，绘制的鱼就在不断变换，看起来就像是鱼儿在游动一样。在程序中使用 run()方法来控制每条鱼的坐标，代码清单 3-1 为 FishRunThread.java 的实现。

代码清单 3-1 控制鱼儿游动的线程

```
/**
 * 控制鱼儿游动的线程
 * @author Xiloerfan
 *
 */
public class FishRunThread extends Thread{
 private Fish fish;
 private boolean isRun;
 /**
  * 屏幕宽度
  */
 private int screenWidth;
 public FishRunThread(Fish fish,int screeWidth){
      this.fish = fish;
      this.screenWidth = screeWidth;
 }
 public void stopFishRun(){
      isRun = false;
 }
 @Override
 public void run() {
      isRun = true;
      // TODO Auto-generated method stub
      int fishX = 0;
      while(isRun){
           /**
            * 累加 X，让鱼一直向右移动
            */
           fish.getPicMatrix().setTranslate(fishX,0);
           fishX++;
           if(fishX>screenWidth){
                /**
                 * 减去宽度是为了让鱼的起始位置位于屏幕左边没有出现的位置
                 */
                fishX = 0-fish.getCurrentPic().getWidth();
           }
           try {
                Thread.sleep(50);
           } catch (Exception e) {
                // TODO: handle exception
           }
      }
 }
}
```

3.1.2 创建多个线程

掌握了线程在游戏中实现功能的方法后，我们就可以创建多个线程来完成不同的任务。以“小鱼快跑”

为例，我们可以开启不同的线程来控制鱼群的移动、子弹的发射等。

3.2 多线程的注意事项

3.2.1 同步问题

在以上代码中，我们用一个 while 循环对鱼儿的位置进行处理，以 Boolean 型变量 isRun 来作为判断的条件。isRun 为 true 代表游戏正在进行，当游戏结束时在 stopFishRun()中将 isRun 设为 false，退出循环。这样做对一个单独的线程没有影响，但是在整个游戏当中若存在两个或两个以上的线程，每个线程都有各自的 isRun 属性，当其中一个线程退出时，游戏就应该结束，游戏中所有的线程都应该结束。但是这时只有引起游戏结束的线程的 isRun 属性被设置为 false，而其他线程仍在运行，这样将会导致游戏不能完全退出和一些其他的问题。

为了解决以上问题，我们用一个 Boolean 值变量 gameInfo 来表示整个系统的运行状态，并将其存储于一个公开类 GamingInfo.java 中，使用 get 和 set 方法来获取和修改它的值。在程序运行时，每一个线程只需要调用这一个属性，就可以获知游戏是否结束。我们将游戏中其他的公共属性都存储于 GamingInfo.java 中，用相应的 get 和 set 方法来使用。代码清单 3-2 为 GamingInfo.java 的实现，代码清单 3-3 为 FishRunTread.java 中改进之后的 run()方法。

代码清单 3-2 GamingInfo.java 的实现

```
// 游戏进行中一些需要共用的变量
public class GamingInfo {
  private int screenWidth;
  private int screenHeight;
  private static GamingInfo gameInfo;        // 单例模式需要
  private boolean isGaming;                  // 是否处于游戏状态
  private boolean isPause;                   // 是否处于暂停状态
  private MainSurface surface;               // 主屏幕
  private Activity activity;
  private ArrayList<Fish> fish = new ArrayList<Fish>(); // 所有的鱼
  private ShoalManager shoalManager;         // 鱼群管理器
  private SoundManager soundManager;         // 音效管理器
  private float cannonLayoutX;               // 大炮旋转 X 坐标
  private float cannonLayoutY;               // 大炮旋转 Y 坐标
  private int score = 100;                   // 当前的分数值


  public int getScore() {
      return score;
  }

  public void setScore(int score) {
      this.score = score;
  }

  /**
   * 清除 GamingInfo 实例
   */
  public static void clearGameInfo() {
      gameInfo = null;
  }

  private GamingInfo() {
  }

  public static GamingInfo getGamingInfo() {
      if (gameInfo == null) {
```

```
            gameInfo = new GamingInfo();
        }
        return gameInfo;
    }

    public boolean isGaming() {
        return isGaming;
    }

    public void setGaming(boolean isGaming) {
        this.isGaming = isGaming;
    }

    public ArrayList<Fish> getFish() {
        return fish;
    }

    public void setFish(ArrayList<Fish> fish) {
        this.fish = fish;
    }

    public MainSurface getSurface() {
        return surface;
    }

    public void setSurface(MainSurface surface) {
        this.surface = surface;
    }

    public ShoalManager getShoalManager() {
        return shoalManager;
    }

    public void setShoalManager(ShoalManager shoalManager) {
        this.shoalManager = shoalManager;
    }

    public int getScreenWidth() {
        return screenWidth;
    }

    public void setScreenWidth(int screenWidth) {
        this.screenWidth = screenWidth;
    }

    public int getScreenHeight() {
        return screenHeight;
    }

    public void setScreenHeight(int screenHeight) {
        this.screenHeight = screenHeight;
    }

    public Activity getActivity() {
        return activity;
    }

    public void setActivity(Activity activity) {
        this.activity = activity;
    }

    public SoundManager getSoundManager() {
        return soundManager;
    }

    public void setSoundManager(SoundManager soundManager) {
```

```
        this.soundManager = soundManager;
    }

    public float getCannonLayoutX() {
        return cannonLayoutX;
    }

    public void setCannonLayoutX(float cannonLayoutX) {
        this.cannonLayoutX = cannonLayoutX;
    }

    public float getCannonLayoutY() {
        return cannonLayoutY;
    }

    public void setCannonLayoutY(float cannonLayoutY) {
        this.cannonLayoutY = cannonLayoutY;
    }

    public boolean isPause() {
        return isPause;
    }

    public void setPause(boolean isPause) {
        this.isPause = isPause;
    }
}
```

代码清单 3-3　线程中的 run()方法

```
public void run() {
        int[][] path = PathManager.getDefaultPath(fish);
        while (GamingInfo.getGamingInfo().isGaming()) {
            while(isRun &&!GamingInfo.getGamingInfo().isPause()){
                for (int[] pathMode : path) {
                    if(fishIsOut||!isRun){
                        break;
                    }
                    // 如果路径为旋转模式
                    if (pathMode[0] == PathManager.PATH_MODE_ROTATE) {
                        /**
                         * 这里做了一个处理，就是分析鱼头朝向和所在位置让鱼儿进行不同的旋转路线
                         */
                        // 如果鱼儿处于第 1 或第 2 象限
                    if (fish.getFish_X() <= GamingInfo.getGamingInfo().getScreenWidth() / 2
                    && fish.getFish_Y() <= GamingInfo.getGamingInfo().getScreenHeight() / 2
                    || fish.getFish_X() > GamingInfo.getGamingInfo().getScreenWidth() / 2
                    && fish.getFish_Y() <= GamingInfo.getGamingInfo().getScreenHeight() / 2) {
                            // 如果鱼头是朝向 x 的负坐标方向
                            if (fish.getCurrentRotate() >= 90
                                    && fish.getCurrentRotate() <= 270
                                    || fish.getCurrentRotate() <= -90
                                    && fish.getCurrentRotate() >= -270) {
                                rotateLeftFish(pathMode[1]);
                            } else {
                                rotateRightFish(pathMode[1]);
                            }
                            // 如果鱼儿处于第 3 或第 4 象限
                        } else {
                            // 如果鱼头是朝向 x 的负坐标方向
                            if (fish.getCurrentRotate() >= 90
                                    && fish.getCurrentRotate() <= 270
                                    || fish.getCurrentRotate() <= -90
                                    && fish.getCurrentRotate() >= -270) {
                                rotateRightFish(pathMode[1]);
                            } else {
```

```
                                rotateLeftFish(pathMode[1]);
                            }
                        }
                        // 如果路径为直行模式
                    } else {
                        goStraight(pathMode[1]);
                    }
                }
                if(!fishIsOut){
                // 重新获取新路径
                path = PathManager.getDefaultPath(fish);
                }else{
                    while(isRun && GamingInfo.getGamingInfo().isGaming()){
                        // 如果超出屏幕，走一个直线
                        goStraight(100);
                    }
                }
            }
            break;
        }
    }
```

3.2.2 数据安全问题——线程锁

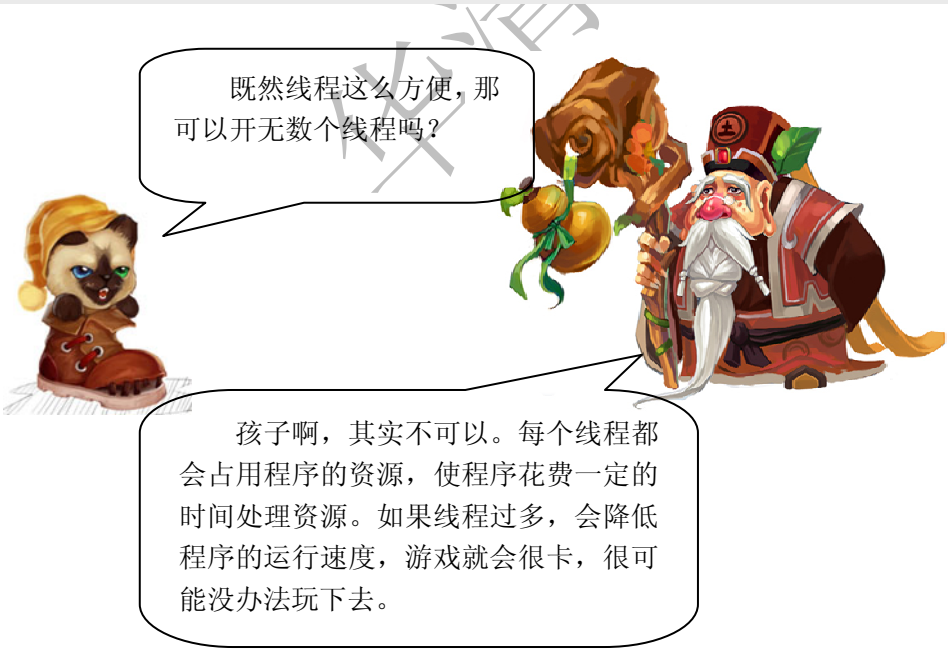
游戏中经常会使用多线程来访问同一个数据，这样可能会带来数据的不安全问题，比如两个线程 T1 和 T2 都访问同一数据 D，当 T1 开始改变 D 的值时，理论上 D 应该是新的值，这时若 T2 恰好也要访问 D，而 D 并没有被完全改变，那么 T2 可能会得到 D 原来的值，即得到错误的信息。为了避免这种情况的发生，我们使用线程锁来解决，即当 D 的值正在改变时，给 D 加一个线程锁，使任何其他线程都无法访问，直到完成了 D 值的改变。我们在更新图层的时候用到了线程锁，代码清单 3-4 显示了更新图层的过程。

代码清单 3-4 更新图层

```
/**
 * 更新图层，这里分为三种操作方式，分别是更新临时图层中的内容到绘制图层中，删除绘制图层中的元素，添加绘制图层中的元素
 * 这里加了个线程锁，保证多线程下操作图层的安全性
 * @param mode 绘制图层的操作类型，对应当前类的 CHANGE_MODE 常量
 * @param layerId  操作的图层 ID
 * @param draw     操作的图层元素
 */
private synchronized void updatePicLayer(int mode,int layerId,Drawable draw){
    switch(mode){
    // 将临时图层中的内容更新至绘制图层中
    case CHANGE_MODE_UPDATE:
        // 如果有修改
        if(changeLayer){
            // 向图层添加新的元素
            for(Integer id:addPicLayer.keySet()){
                for(Drawable d:addPicLayer.get(id)){
                    // 如果要添加的元素所处图层不存在，则创建这个图层，并更新图层 ID 数组
                    if(this.picLayer.get(id)==null){
                        this.picLayer.put(id, new ArrayList<Drawable>());
                        updateLayerIds(id);
                    }
                    this.picLayer.get(id).add(d);
                }
            }
            addPicLayer.clear();
            // 删除图层中的元素
            for(Integer id:removePicLayer.keySet()){
                for(Drawable d:removePicLayer.get(id)){
                    try {
                        this.picLayer.get(id).remove(d);
                    } catch (Exception e) {
                        System.out.println("图层内容不存在:"+id);
```

```
                        }

                    }
                }
                removePicLayer.clear();
                changeLayer = false;
            }
            break;
        /**
         * 无论是向绘图图层中添加还是删除元素，都不是直接操作绘制图层，都是存放在对应的临时图层中，等待绘制方法绘制周期中将变化的内容更新到绘制图层中
         * 保证多线程操作情况下的安全性
         */
        // 添加一个元素
        case CHANGE_MODE_ADD:
            ArrayList<Drawable> al = addPicLayer.get(layerId);
            if(al==null){
                al = new ArrayList<Drawable>();
                addPicLayer.put(layerId, al);
            }
            al.add(draw);
            changeLayer = true;
            break;
        // 删除一个元素
        case CHANGE_MODE_REMOVE:
            ArrayList<Drawable> al1 = removePicLayer.get(layerId);
            if(al1==null){
                al1 = new ArrayList<Drawable>();
                removePicLayer.put(layerId, al1);
            }
            al1.add(draw);
            changeLayer = true;
            break;
        }
    }
```



3.3 本章小结

本章主要介绍了在游戏中使用线程处理逻辑和使用多线程的方法，并提到了使用多线程可能会遇到的问题

及常用解决方案。

联系方式

集团官网：www.hqyj.com 嵌入式学院：www.embedu.org 移动互联网学院：www.3g-edu.org

企业学院：www.farsight.com.cn 物联网学院：www.topsight.cn 研发中心：dev.hqyj.com

集团总部地址：北京市海淀区西三旗悦秀路北京明园大学校内 华清远见教育集团

北京地址：北京市海淀区西三旗悦秀路北京明园大学校区，电话：010-82600386/5

上海地址：上海市徐汇区漕溪路 250 号银海大厦 11 层 B 区，电话：021-54485127

深圳地址：深圳市龙华新区人民北路美丽 AAA 大厦 15 层，电话：0755-25590506

成都地址：成都市武侯区科华北路 99 号科华大厦 6 层，电话：028-85405115

南京地址：南京市白下区汉中路 185 号鸿运大厦 10 层，电话：025-86551900

武汉地址：武汉市工程大学卓刀泉校区科技孵化器大楼 8 层，电话：027-87804688

西安地址：西安市高新区高新一路 12 号创业大厦 D3 楼 5 层，电话：029-68785218

广州地址：广州市天河区中山大道 268 号天河广场 3 层，电话：020-28916067