**Launcher学习**

**编 写 人：桌面品牌产品线**

版本历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 作者 | 操作 | 日期 | 说明 |
| V1.0 | 叶振南 | 创建 | 2013/08/05 | Launcher大致框架讲解 |
| V1.1 | 钟其鸿 | 添加内容 | 2013/08/21 | Widget3D的加载 |
| V1.2 | 李兴旺 | 添加内容、图表重绘 | 2014/10/18 | Launcher事件分发,图表的重绘 |
|  |  |  |  |  |

目录

[1 Launcher总体框架介绍 4](#_Toc401409084)

[2 Launcher代码环境的搭建 7](#_Toc401409085)

[3 Launcher代码与Libgdx引擎是怎样关联的 8](#_Toc401409086)

[4 Launcher事件分发 11](#_Toc401409087)

[5 Widget3D的介绍 21](#_Toc401409088)

[6 Widget3D项目的建立 22](#_Toc401409089)

[7 Widget3D的配置 27](#_Toc401409090)

[8 Widget3D细节之一 32](#_Toc401409091)

### 1 Launcher总体框架介绍

我们的Launcher桌面产品是基于Libgdx引擎进行开发的，因此在介绍Launcher的总体框架之前，有必要介绍下Libgdx引擎。

#### 1.1 Libgdx引擎介绍

Libgdx是一个开源的跨平台的2D/3D的游戏开发框架，它由Java/C/C++语言编写而成。主体开发语言是java，C/C++语言则是通过JNI代码开发底层库。引擎的底层图形渲染则是采用的OpenGL，正是因为有了OpenGL的底层支持，Libgdx才能够很好的支持3D渲染。

Application

Files

Audio

Input

Graphics

图 1.1

上图1.1是Libgdx的主要构成，其由audio、files、graphics、math、physics、scenes、utils这些主要模块所组成，它们分别对应了Libgdx中的音频操作，文件读取，2D/3D渲染，Libgdx绘图相关运算，Box2D封装，2D/3D游戏组件，以及Libgdx内置工具类。

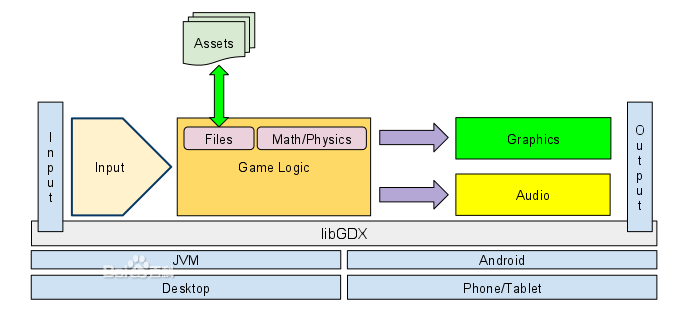


图1.2

图1.2则是Libgdx引擎的总体架构设计图。

紧接着再来大致了解一下Libgdx引擎中各组件的架构组成关系。

**Application**

**Application**

**Screen**

**Screen**

**Screen**

**Stage**

**Stage**

**Stage**

**Stage**

**Stage**

**Stage**

**Actor**

**Actor**

**Actor**

**Actor**

**Actor**

**Actor**

图1.3

图1.3为Libgdx引擎中各组件的架构组成关系

#### 1.2 Launcher总体架构

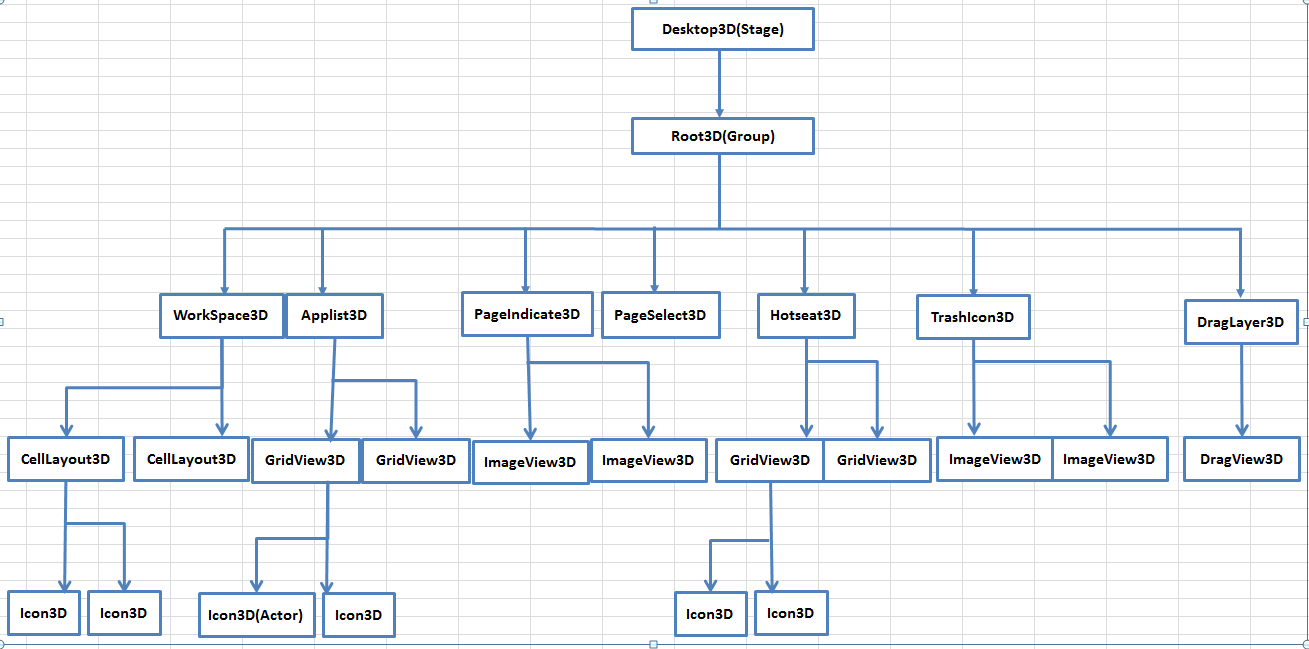
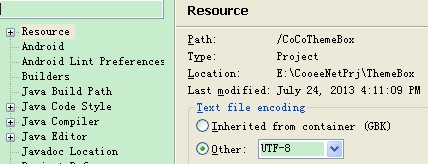


图1.4

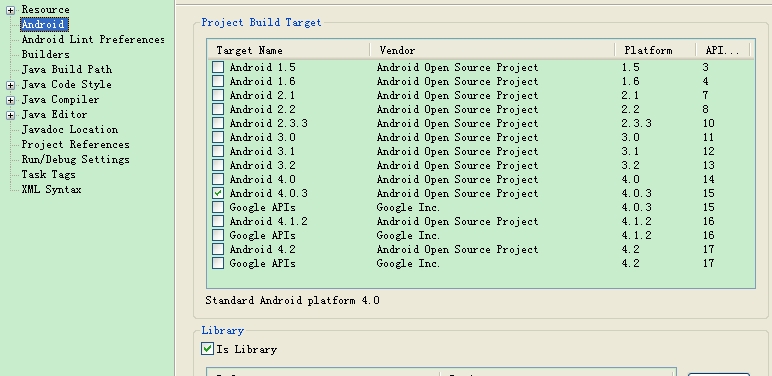
Desktop3D对应的Libgdx中的Stage类，意为舞台，而最终的子节点Icon3D以及ImageView3D都对应于Libgdx中的Actor，意为演员。整个的场景都由演员来展现。

### 2 Launcher代码环境的搭建

#### 2.1 ThemeBox



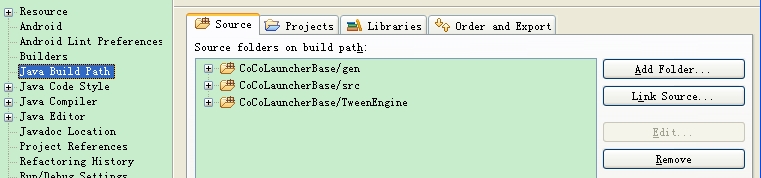
Resource选项，编码格式请配置为UTF-8格式。LauncherBase以及CoCoLauncher 项目请均配置为UTF-8格式。



Android选项中，Project Build Target请配置为2.3.3以上版本，因为有些接口低版本可能不支持。Is Library选项请勾选。LauncherBase 配置同此。

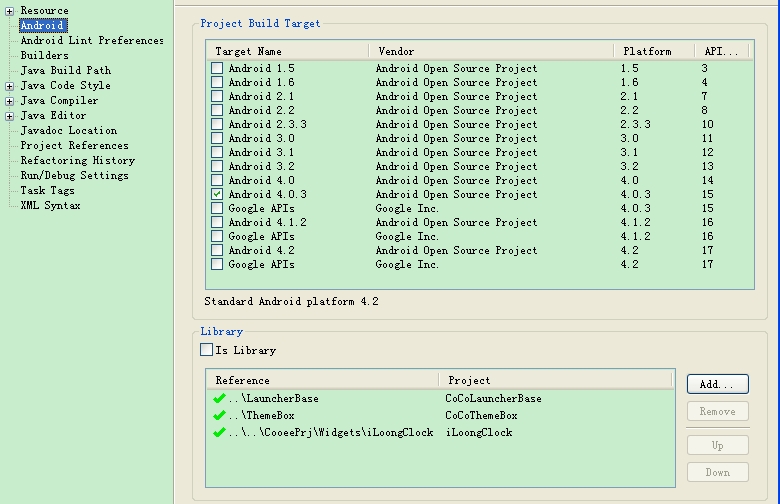
#### 2.2 LauncherBase

Resource选项和Android选项同“ThemeBox”。



Java Build Path中“Sorce->Add Folder”将TweenEngine源码导入。

#### 2.3 CoCoLauncher



Project Build Target请配置为2.3.3以上版本，因为有些接口低版本可能不支持。Library中“Add”，将LauncherBase和ThemeBox作为库加入。

### 3 Launcher代码与Libgdx引擎是怎样关联的

#### 3.1 Launcher代码的组织构成

Launcher代码包括3部分，LauncherBase、ThemeBox和CoCoLauncher，其中LauncherBase和ThemeBox均作为Library库的形式供CoCoLauncher主体代码使用，LauncherBase中的代码是被抽象出来的基础类代码，并且这部分代码借鉴了Libgdx引擎的组件架构思想。ThemeBox则是主题内容相关的代码部分，其提供给Launcher各种各样的主题内容。

#### 3.2 Libgdx引擎如何被引用

Libgdx是以jar包的形式被引用的，gdx.jar和gdx\_backends\_android.jar两个主体jar存在于LauncherBase中的libs目录下。

iLoongLauncher类为Launcher桌面的入口。由图3.1中AndroidManifest.xml文件的配置情况中也能看出这一点。

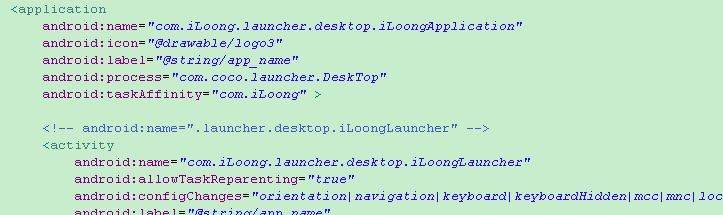


图3.1

那么，这个类是如何与Libgdx中的相关类联系的呢，我们来看看iLoongLauncher的继承关系，便会一目了然了。



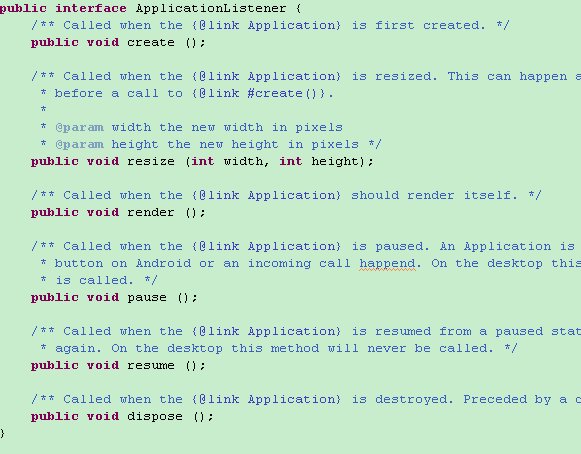
iLoongLauncher的父类AndroidApplication就是Libgdx中的类，而它又是Android原生类Activity的一个子类，这时大家心里应该都很清楚了，我们的入口iLoongLauncher实际就是一个Activity窗口，通过它来向用户展示所有的内容。

到这里，大家可能还有一个疑问？那么Launcher的刷新又是通过怎样的关联最终由Libgdx的OpenGL来渲染的呢？

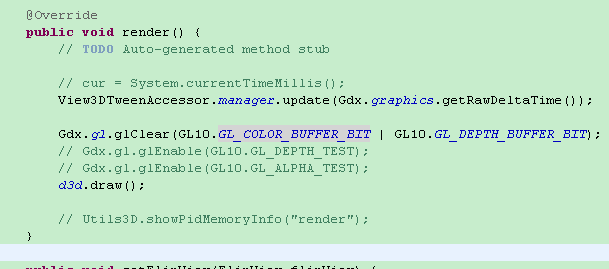
那就要看Launcher中的Desktop3DListener这个类了。



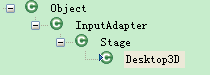
它实现了ApplicationListener这样一个接口，而这个接口正是Libgdx中的一个接口类。



这个接口类实际也很简单，里面申明了一些生命周期的函数接口，其中render函数接口就是渲染刷新的接口。我们来大致看看Desktop3DListener对这一函数接口的实现情况。



其中的d3d是Desktop3D类型。



其继承于Libgdx中的Stage类。

### 4 Launcher事件分发

#### 4.1介绍

Launcher是使用libgdx开发的3D应用，事件的分发处理与一般的应用不同。

通过调用 Gdx.input.setInputProcessor(InputProcessor processor)函数，libgdx将事件全部转移到 InputProcessor 进行处理。

InputProcessor 定义如下：

public interface InputProcessor {

public boolean keyDown (int keycode);

public boolean keyUp (int keycode);

public boolean keyTyped (char character);

public boolean keyTyped (String characters);

public boolean touchDown (int x, int y, int pointer, int button);

public boolean touchUp (int x, int y, int pointer, int button);

public boolean touchDragged (int x, int y, int pointer);

public boolean touchMoved (int x, int y);

public boolean scrolled (int amount);

}

我们重点查看 touchdown, touchup 和 touchDragged 这三部分，这分别对应触笔按下，触笔抬起和触笔滑动的操作，其他的双击，多指滑动都是通过这些事件扩展出来的。

#### 4.2具体事件函数

##### 4.2.1 touchDown事件

touchDown 对应触笔按下的操作，在 Launcher 中，InputProcess 是在 Desktop3D 中实现的，我们直接看一下 Desktop3D.touchDown 的实现。

###### 4.2.1.1 Desktop3D.touchDown

Desktop3D.touchDown 的核心代码如下：

//取得当前的时间

long timeStamp = Gdx.input.getCurrentEventTime();

if(timeStamp > UtilsBase.resumeTime && timeStamp < UtilsBase.pauseTime){

} else{

//如果不是正常运行状态，不处理，直接返回

Log.i("click", "down dismiss");

return true;

}

//标记事件非成对状态，只处理了down事件

isAPairEvent = false;

if(!waitDouble){

//等待单击状态，直接返回

return true;

}

//转到gesture进行处理down事件

gesture.touchDown(x, y, pointer, newParam);

//true表示已经处理了事件，通知android框架不要分发给其他控件

return true;

从上面的流程中，可以看到逻辑很简单，判断了一些变量，然后转到gesture（GestureDectector3D）进行处理。

GestureListener3D

GestureListener3D的定义如下：

public static interface GestureListener3D {

//单击

public boolean tap (int x, int y, int count);

//长按

public boolean longPress (int x, int y);

//滑动

public boolean fling (float velocityX, float velocityY);

//pan

public boolean pan (int x, int y, int deltaX, int deltaY);

//双指缩放

public boolean zoom (float originalDistance, float currentDistance);

//双指移动

public boolean pinch (Vector2 initialFirstPointer, Vector2 initialSecondPointer, Vector2 firstPointer, Vector2 secondPointer);

//触笔按下

public boolean onTouchDown (int x, int y, int pointer);

//触笔滑动

public boolean onTouchDragged (int x, int y, int pointer);

//触笔弹起

public boolean onTouchUp (int x, int y, int pointer);

}

GestureListener3D是一个接口，而Desktop3D实现了这个接口，这个接口的每一个函数表明了一个事件的被触发，而 GestureDetector3D 类是用来转换事件的，将最基本的touchDown，touchDragged，touchUp转换为上面的高级事件。

具体高级事件的用法后面再补充。

###### 4.2.1.2 GestureDectector3D.touchDown

细节略，GestureDectector3D.touchDown 最后会调用 GestureListener3D.onTouchDown，既Desktop3D.onTouchDown。

###### 4.2.1.3 Desktop3D.onTouchDown

这里才是真正处理触笔按下事件的地方，我们来看一下实现。

核心代码如下：

public boolean onTouchDown(int x, int y, int pointer) {

//设置状态为已经按下

hasDown = true;

//转换坐标，左下改为(0,0)

this.toStageCoordinates(x, y, point);

if (this.foucusView != null) {

//如果focusView不为空，则转换坐标，

//并将事件转到focusView处理

foucusView.toLocalCoordinates(point);

foucusView.onTouchDown(point.x, point.y, pointer);

return true;

}

//对子控件进行轮询

int len = this.getChildCount() - 1;

for (int i = len; i >= 0; i--) {

View3D child = this.getViewAt(i);

//如果子控件不可触摸或者不可见则跳过

if (!child.touchable || !child.visible)

continue;

coords.x = point.x;

coords.y = point.y;

if (child.pointerInParent(coords.x, coords.y)) {

//如果子控件在包含按下的点，则将点的坐标转换为控件坐标；

//尝试onTouchDown处理，如果成功则直接返回

//注意这里的坐标转换全部是针对于2D的，Z=0

child.toLocalCoordinates(coords);

if (child.onTouchDown(coords.x, coords.y, pointer)) {

return true;

}

}

}

return false;

}

在上面的代码中，我们看到，如果focusView不为空，则直接处理onTouchDown事件。

如果focusView为空，则轮询子控件，尝试处理onTouchDown事件。

由于在Launcher中，Desktop3D的子控件都是ViewGroup3D和View3D，接下来看一下ViewGroup3D的实现方式。

###### 4.2.1.4 ViewGroup3D.onTouchDown

核心代码如下：

public boolean onTouchDown(float x, float y, int pointer){

//如果控件不可触摸或者不可见，直接返回false

if (!touchable || !visible) return false;

//轮询子控件

int len = children.size() - 1;

for (int i = len; i >= 0; i--) {

View3D child = children.get(i);

//如果子控件不可触摸或者不可见，则跳过

if (!child.touchable || !child.visible) continue;

point.x = x;

point.y = y;

if (child.pointerInParent(point.x, point.y)){

toLocalCoordinates(child,point);

//如果子控件在包含按下的点，则将点的坐标转换为控件坐标；

//尝试onTouchDown处理，如果成功则直接返回

//注意这里的坐标转换全部是针对于2D的，Z=0

if (child.onTouchDown(point.x, point.y, pointer)) {

//如果子控件已经处理了onTouchDown，则设置lastTouchedChild，

//注意ViewGroup3D和其他的控件处理lastTouchedChild的方式不同

if (child instanceof ViewGroup3D)

lastTouchedChild = ((ViewGroup3D)child).lastTouchedChild;

else

lastTouchedChild = child;

return true;

}

}

}

return false;

}

最终，onTouchDown到具体的View3D中进行执行。

###### 4.2.1.5 View3D.onTouchDown

代码如下，非常简单：

public boolean onTouchDown(float x, float y, int pointer) {

return false;

}

View3D默认没有进行任何操作，返回false，表示此控件没有处理onTouchDown消息，让父控件传递给其他控件尝试处理。

###### 4.2.1.6 touchDown流程图

具体流程图如下



##### 4.2.2 touchDragged事件

touchDragged对应触笔按下后拖动的操作，看一下Desktop3D.touchDragged的实现。

###### 4.2.2.1 Desktop3D.touchDragged

核心代码如下：

public boolean touchDragged(int x, int y, int pointer) {

long timeStamp = Gdx.input.getCurrentEventTime();

if(timeStamp > UtilsBase.resumeTime && timeStamp < UtilsBase.pauseTime){

} else{

//如果是暂停状态，直接返回

return true;

}

//等待单击时 不发第二个消息

if(!waitDouble) return true;

//转到gesture进行处理down事件

gesture.touchDragged(x, y, pointer);

return true;

}

从上面的流程中，可以看到逻辑很简单，判断了一些变量，然后转到gesture（GestureDectector3D）进行处理。

###### 4.2.2.2 GestureDectector3D.touchDragged

细节略。

值得注意的是在这个函数中会调用GestureListener3D.pinch， GestureListener3D.zoom，GestureListener3D.pan， GestureListener.ouTouchDragged。分别对应 Desktop3D 中的同名函数。

##### 4.2.3 Desktop3D.pinch

Pinch表示双指移动时的事件，核心代码如下：

public boolean pinch(Vector2 initialFirstPointer,

Vector2 initialSecondPointer, Vector2 firstPointer,

Vector2 secondPointer) {

//转换坐标系

toStageCoordinates((int) initialFirstPointer.x, (int) initialFirstPointer.y, point);

toStageCoordinates((int) initialSecondPointer.x, (int) initialSecondPointer.y, coords);

toStageCoordinates((int) firstPointer.x, (int) firstPointer.y, point1);

toStageCoordinates((int) secondPointer.x, (int) secondPointer.y, point2);

if (this.foucusView != null) {

//如果focusView不为空，将事件转到focusView处理

//注意这里没有将坐标系转换为focusView的本地坐标系

foucusView.multiTouch2(point, coords, point1, point2);

return true;

}

//轮询子控件

int len = this.getChildCount() - 1;

for (int i = len; i >= 0; i--) {

View3D child = this.getViewAt(i);

//如果不可触摸，或不可见，则跳过

if (!child.touchable || !child.visible)

continue;

//调用子控件multiTouch2成功，则返回

if (child.multiTouch2(point, coords, point1, point2)) {

return true;

}

}

return false;

}

在上面的代码中，我们看到，如果focusView不为空，则直接处理multiTouch2事件。

如果focusView为空，则轮询子控件，尝试处理multiTouch2事件。

由于在Launcher中，Desktop3D的子控件都是ViewGroup3D和View3D，接下来看一下ViewGroup3D的实现方式。

##### 4.2.4 multiTouch2事件

###### 4.2.4.1 ViewGroup3D.multiTouch2

代码如下：

public boolean multiTouch2(Vector2 initialFirstPointer,

Vector2 initialSecondPointer, Vector2 firstPointer,

Vector2 secondPointer) {

//如果不可触摸，或不可见，则返回

if (!touchable || !visible) return false;

//轮询子控件

int len = this.getChildCount() - 1;

for (int i = len; i >= 0; i--) {

View3D child = this.getChildAt(i);

//如果不可触摸，或不可见，则跳过

if (!child.touchable || !child.visible) continue;

//调用子控件multiTouch2成功，则返回

if(child.multiTouch2(initialFirstPointer,initialSecondPointer, firstPointer,secondPointer)){

return true;

}

}

return false;

}

逻辑很简单，直接转到子控件处理multiTouch2事件。

注意坐标系都是stage的坐标系，没有对控件做转换。

另外可以看到子控件multiTouch2的调用跟onTouchDown中的lastTouchedChild没有任何关系。

###### 4.2.4.2 View3D.multiTouch2

代码如下，非常简单：

public boolean multiTouch2(Vector2 initialFirstPointer,

Vector2 initialSecondPointer, Vector2 firstPointer,

Vector2 secondPointer) {

return false;

}

View3D默认不做处理，返回false，让父控件传递给其他控件尝试处理。

##### 4.2.5 Zoom事件

###### 4.2.5.1 Desktop3D.zoom

Zoom表示双指缩放的事件，核心代码如下：

public boolean zoom(float originalDistance, float currentDistance) {

if (this.foucusView != null) {

//如果focusView不为空，将事件转到focusView处理

//注意这里没有将坐标系转换为focusView的本地坐标系

foucusView.zoom(originalDistance, currentDistance);

return true;

}

//轮询子控件

int len = this.getChildCount() - 1;

for (int i = len; i >= 0; i--) {

View3D child = this.getViewAt(i);

//如果不可触摸，或不可见，则跳过

if (!child.touchable || !child.visible)

continue;

//调用子控件zoom成功，则返回

if (child.zoom(originalDistance, currentDistance)) {

return true;

}

}

return false;

}

在上面的代码中，我们看到，如果focusView不为空，则直接处理zoom事件。

如果focusView为空，则轮询子控件，尝试处理zoom事件。

由于在Launcher中，Desktop3D的子控件都是ViewGroup3D和View3D，接下来看一下ViewGroup3D的实现方式。

###### 4.2.5.2 ViewGroup3D.zoom

核心代码如下：

public boolean zoom(float originalDistance, float currentDistance) {

//如果不可触摸，或不可见，则返回

if (!touchable || !visible) return false;

//轮询子控件

int len = this.getChildCount() - 1;

for (int i = len; i >= 0; i--) {

View3D child = this.getChildAt(i);

//如果不可触摸，或不可见，则跳过

if (!child.touchable || !child.visible) continue;

//调用子控件zoom成功，则返回

if(child.zoom(originalDistance,currentDistance)) {

return true;

}

}

return false;

}

逻辑很简单，直接转到子控件处理zoom事件。

注意整个过程中，originalDistance和currentDistance的值是保持不变的。

另外可以看到子控件multiTouch2的调用跟onTouchDown中的lastTouchedChild没有任何关系。

###### 4.2.5.3 View3D.zoom

代码如下：

public boolean zoom(float originalDistance, float currentDistance) {

return false;

}

View3D默认不做处理，返回false，让父控件传递给其他控件尝试处理。

##### 4.2.6 Desktop3D.pan

### 5 Widget3D的介绍

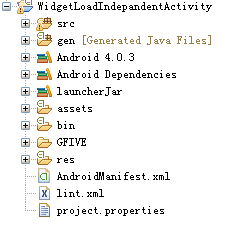
直观上来说，Widget3D 是桌面上的一些具有3D效果的widget。比如3D时钟，机器人等。

Widget3D有两种形式存在，一种是安装版本，另一种是内置版本。本课题只涉及内置版本的内容。在原理上，安装版本和内置版本是一样的，两者比较大的区别是加载的方式和资源的读取方式。

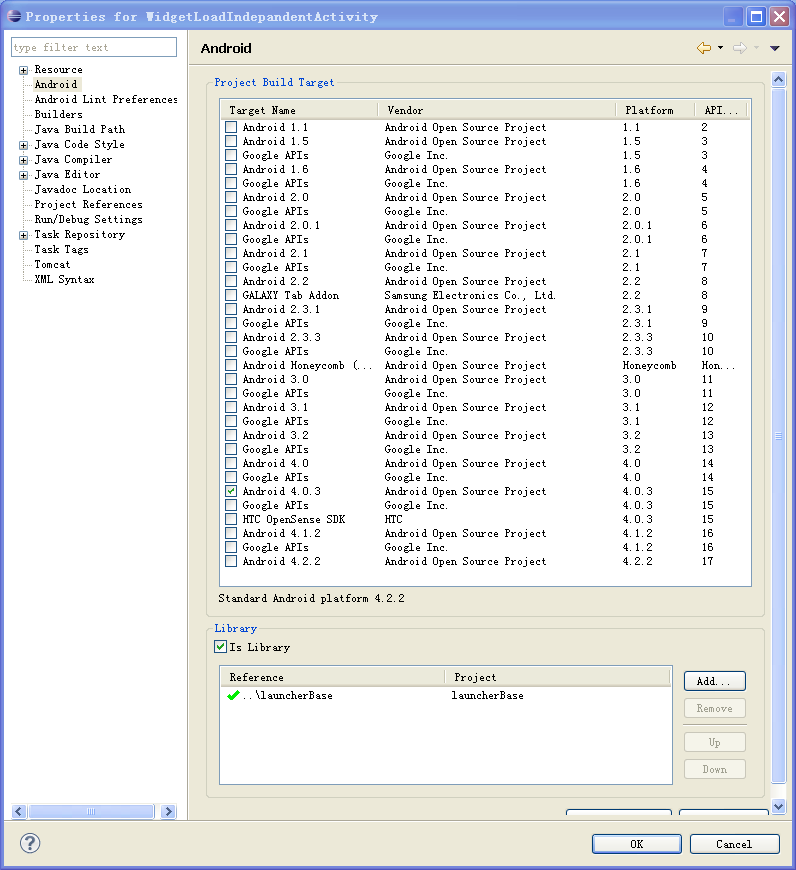
### 6 Widget3D项目的建立

开发Widget3D项目是一个独立的apk开发过程，Widget3D项目中所有的基类都继承于launcher项目，需要把launcher打包成一个jar包，然后被Widget3D项目以内部库的形式引入。同时，由于Widget3D是服务于launcher，最终的表现形式需要显示在launcher桌面上，所以Widget3D也是一个库的形式存在，并且被launcher引用。下面以图文的形式具体的说明Widget3D项目的建立流程。

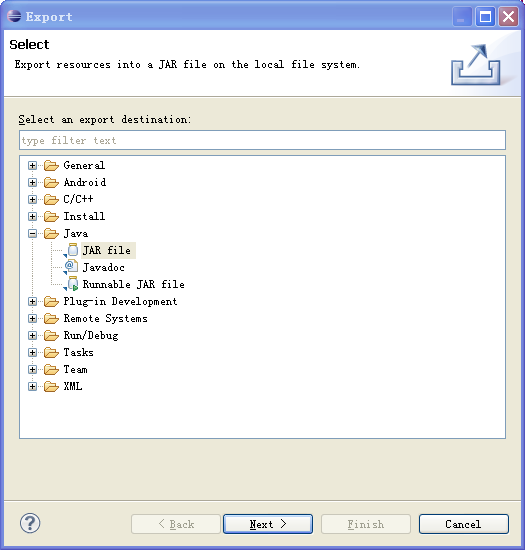
#### 6.1第一步：建立Widget3D工程项目。



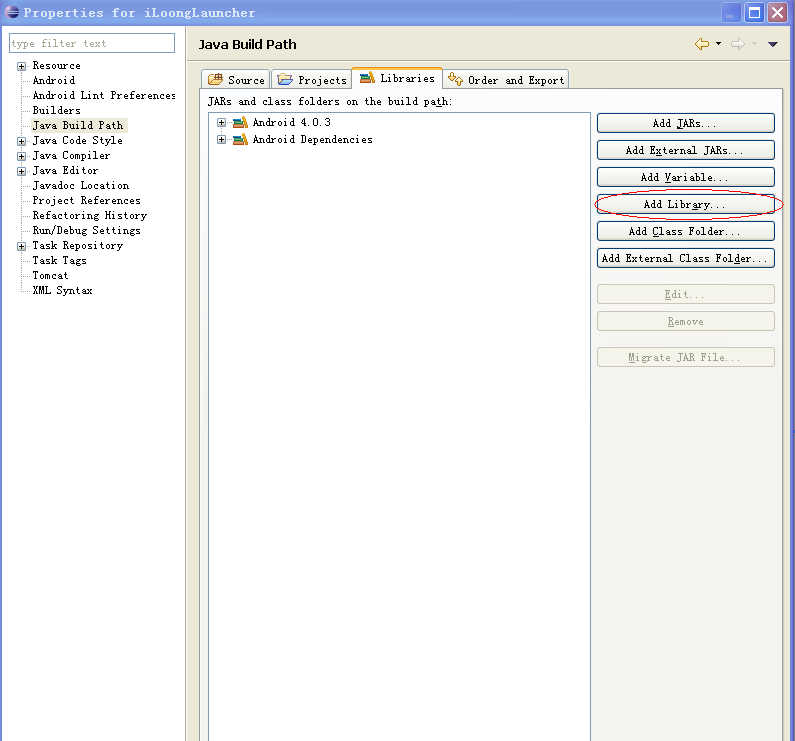
设置项目属性，引入LauncherBase库，同时把自己设置成Is Library。



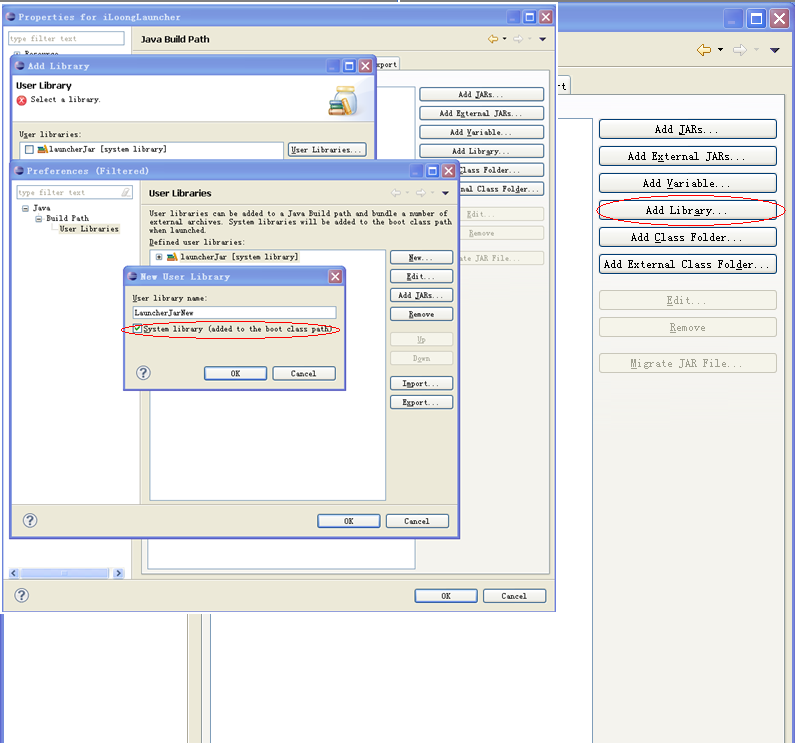
#### 6.2第二步：将launcher项目Export成一个jar包，保存到目录下面。



#### 6.3第三步：在Launcher中把Widget3D的项目以库的形式加入到工程中。



注意：在创建lib库文件夹的时候必须选择System library。



到此为止，Widget3D项目已经建立起来了。后面部分将介绍，Launcher是如何加载Widget3D到桌面的。

### 7 Widget3D的配置

在文件default\_layout.xml文件中详细配置Widget3D的信息。



参数的详细解释：

Id : 第几页显示。

name: 中英文显示的名字。

calssname: 类名

pkgname: 包名

cellX: x方向第几个格子开始显示，注意不是坐标。

CellY: y方向第几个格子开始显示。

…..

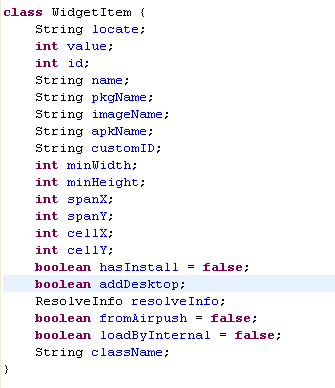
loadByInternal:是否内置

Spanx: x方向占用了几个格子

Spany: y方向占用了几个格子

#### 7.1 配置文件的读取和保存

default\_layout.xml中所有的配置项几乎都是在DefaultLayout.java文件中读取。对于Widget3D，defaultLayout.java文件中专门定义了一个WidgetItem类用于存放上面配置的信息，每个属性都是一一对应的。解析完配置文件中的所有配置项后，这些类的对象会暂时保存在allWidgetFinal列表中，然后被送入数据库。注意default\_layout.xml只在launcher安装后的第一次启动的时候会被读取，以后的操作只跟数据库打交道。



#### 7.2 桌面应用的描述

Launcher在启动的时候会调用两个很重要的函数loadAndBindWorkspace和loadAndBindAllApps。后者是初始化主菜单中所有的应用，前者是初始化桌面的快捷键和widget等，Widget3D就是在这个函数中被初始化后加入到桌面的ViewGroup中的。

前面说过，Widget3D的配置信息在default\_layout.xml文件中读取后会被送入数据库。Launcher在启动的过程中会调用loadWorkspace这个函数去读取数据库中的信息。

**private** **void** loadWorkspace() {

**int** itemType = c.getInt(itemTypeIndex);

**switch** (itemType) {

//读取其他类型的数据

**case** LauncherSettings.Favorites.*ITEM\_TYPE\_APPWIDGET*:

//获取Widget3D的配置信息

mWidget3Ds.add(widget3DInfo);

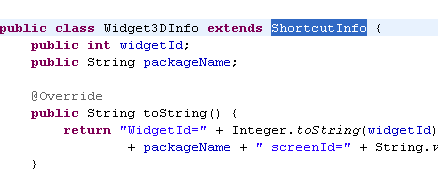
}

}

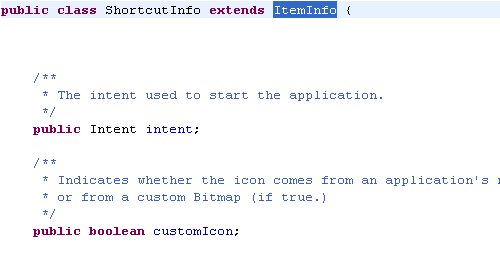
mWidget3Ds是一个Widget3DInfo对象的数组，其定义如下：



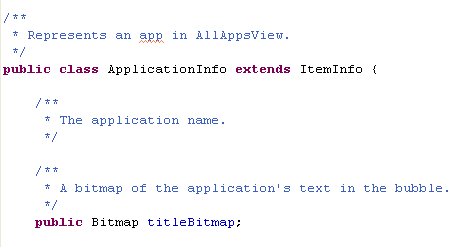
Widget3Dinfo的继承关系是：



说明Widget3Dinfo是一个快捷键，也就是定义在桌面的应用。



能够在显示在桌面的所有应用都必须继承于ItemInfo这个类。由于主菜单的应用也可以被拖拽到桌面，所以所有的应用页必须继承于这个类。下面是主菜单应用类的继承关系。

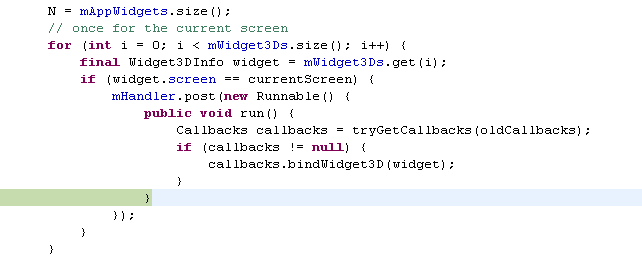


#### 7.3 绑定Widget3D

绑定的过程就是将配置的Widget3D信息，变成一个真正可以在桌面显示的View3D的过程。

//将获取的数据绑定到桌面，也就是将这//Widget3D(ViewGroup3D)add到//Cellayout3D

**private** **void** bindWorkspace() {



}

通过配置的类名，和各项配置信息构建成一个真正的Widget3D，并且加入到workspace的子节点中。

**public** **void** bindWidget3D(**final** Widget3DInfo item) {

**if** (find && widgetItem.loadByInternal) {

//得到widget3D

widget3D = Widget3DManager.*getInstance*().getWidget3D(

widgetItem.pkgName, widgetItem.className);

widget3D.setItemInfo(item);

} **else** {

widget3D = mWidget3DManagerInstance.getWidget3D(item);

}

//这一步很关键，把这个widget3D加入到桌面

**if** (widget3D != **null**) {

workspace.addInScreen(widget3D, item.screen, item.x,item.y, **false**);

}

}

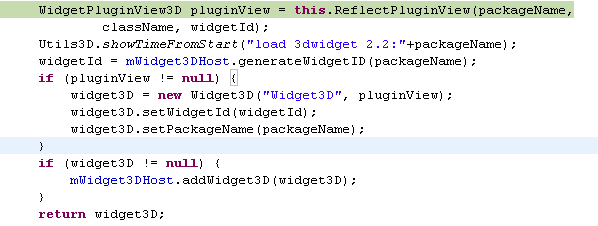
### 8 Widget3D细节之一

如何通过类名搜索一个已经安装的应用，并且构造一个实例。

//通过类名去构造一个Widget3D

**public** Widget3D getWidget3D(String packageName, **final** String className) {

//通过类动态加载方式找到内置的Widget3D



}