Android中的拖拽

版本:2018/3/30-(11:05)

这样一个效果该如何实现?
“我想要拖拽一个控件到另一个布局中，手一松就添加到了该布局中”。

主要的方法有通过View的滑动(如改变控件的布局参数等方法)，滑动到目标区域即可。可以参考教程：https://blog.csdn.net/feather_wch/article/details/78679327

本文的方法是通过给控件设置 `OnDragListener(拖拽监听器)` 直接完成该效果，更为简单。

简单实例：将一个控件拖拽到一个布局中

> 主要思路:
>
> 1. 给被拖拽控件添加点击事件监听器，并在内部转交给 `View.startDrag` 方法去实现拖拽。
> 2. 给接收方添加 `OnDrageListener` ， 会监听到发生在自身区域内的 `拖拽事件` ， 在拖拽完成的回调中将 `被拖拽控件` 添加进来

被拖拽控件:

```
mAddButton.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View view, MotionEvent event) {
        if(event.getAction() == MotionEvent.ACTION_DOWN){
            //1. 剪切板可以保存数据
            ClipData data = ClipData.newPlainText("", "");
            //2. 影子
          E/Parcel: Class not found when unmarshallingE/Parcel: Class not found when unmars
            //3. 震动反馈，不需要震动权限
            view.performHapticFeedback(HapticFeedbackConstants.LONG_PRESS, HapticFeedba
            //4. 拖拽
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
                view.startDragAndDrop(data, shadowBuilder, view, 0);
            }else{
                view.startDrag(data, shadowBuilder, view, 0);
            }
            return true;
        }else{
            return false;
        }
    }
});
```

目标布局接收拖拽的控件：

```
mRelativeLayout.setOnDragListener(new View.OnDragListener() {
    @Override
    public boolean onDrag(View v, DragEvent event) {
        switch (event.getAction()){
            case DragEvent.ACTION_DROP:
                //1. 响应拖拽，将控件安置到该位置
                View otherView = (View) event.getLocalState();
                ViewGroup owner = (ViewGroup) otherView.getParent();
                owner.removeView(otherView);
                RelativeLayout relativeLayout = (RelativeLayout) v;
                relativeLayout.addView(otherView);
                otherView.setVisibility(View.VISIBLE);
                break;
        }
        return true;
    }
});
```

实例：从RecyclerViewA中将Item拖拽到另一个RecyclerViewB中

> 主要思路也就是和上面的一样。
> 区别在于：接收方和被拖拽方都是 `RecyclerView` ， 需要通过特定的方法去完成 拖拽和添加`Item` 的

> 功能。

1、RVA中需要实现onDrag拖拽和onItemLongclick长按开始拖拽的两种监听器

```
public interface RecyclerViewOnItemLongClickListenr{
    public boolean onLongClick(View view, ItemDataBean data);
}
public class RecyclerViewOnTouchAndDragListener implements View.OnDragListener, RecyclerViewOnI
    int fromPosition = -1;
    int toPosition = 0;
    RecyclerView mRecyclerView;
    RVAdapter mAdapter;

    public int getFromPosition() {
        return fromPosition;
    }

    public void setFromPosition(int fromPosition) {
        this.fromPosition = fromPosition;
    }

    public int getToPosition() {
        return toPosition;
    }

    public RecyclerViewOnTouchAndDragListener(){
    }
    public RecyclerViewOnTouchAndDragListener(RecyclerView recyclerView, RVAdapter adapter) {
        mRecyclerView = recyclerView;
        mAdapter = adapter;
    }
    @Override
    public boolean onDrag(View v, DragEvent event) {
        if (mRecyclerView == null || mAdapter == null) {
            return false;
        }
        final int action = event.getAction();
        switch (action) {
            /**=========================================================
             * 1. 开始拖拽需要确定当前点击的ItemView
             * 2. 如果点击在界外，则fromPosition= -1, 需要在其他阶段确定点击的Item
             *=========================================================*/
            case DragEvent.ACTION_DRAG_STARTED:
                View itemView = mRecyclerView.findChildViewUnder(event.getX(), event.getY());
                if (itemView != null) {
                    RecyclerView.ViewHolder startViewHolder = mRecyclerView.getChildViewHolder(
                    fromPosition = startViewHolder.getAdapterPosition();
                    //点击的ItemView不可见，如同已经被拖走
                    startViewHolder.itemView.setVisibility(View.INVISIBLE);
                }
                return true;
            /**=========================================================
             * 2. 拖拽进入了当前控件的区域，如果第一阶段没有确定ItemView，该阶段补充确定
             *=========================================================*/
            case DragEvent.ACTION_DRAG_ENTERED:
                if (fromPosition == -1) {
                    itemView = mRecyclerView.findChildViewUnder(event.getX(), event.getY());
                    if (itemView != null) {
```

```
                    RecyclerView.ViewHolder startViewHolder = mRecyclerView.getChildViewHol
                    fromPosition = startViewHolder.getAdapterPosition();
                    //点击的ItemView不可见
                    startViewHolder.itemView.setVisibility(View.INVISIBLE);
                }
            }
            return true;
        /**==========================================================
         * 3. 拖拽过程中对RecyclerView里面的ItemView顺序进行切换
         *==========================================================*/
        case DragEvent.ACTION_DRAG_LOCATION:
            //1. 获取目标View的VeiwHolder
            itemView = mRecyclerView.findChildViewUnder(event.getX(), event.getY());
            if (itemView != null) {
                RecyclerView.ViewHolder toViewHolder = mRecyclerView.getChildViewHolder(ite
                //2. 获得目标View的Position
                toPosition = toViewHolder.getAdapterPosition();
                //3. 可能开始拖拽的位置不在RecylcerView中
                if (fromPosition == -1) {
                    fromPosition = toPosition;
                }
                //4. from 和 to 之间进行位置变换
                if (fromPosition < toPosition) {
                    for (int i = fromPosition; i < toPosition; i++) {
                        Collections.swap(mAdapter.getCitys(), i, i + 1);
                    }
                } else {
                    for (int i = fromPosition; i > toPosition; i--) {
                        Collections.swap(mAdapter.getCitys(), i, i - 1);
                    }
                }
                mAdapter.notifyItemMoved(fromPosition, toPosition);
                //5. 已经完成了位置交换
                fromPosition = toPosition;
            }
            return true;
        case DragEvent.ACTION_DRAG_EXITED:
            v.invalidate();
            return true;
        /**==========================================================
         * 4. 拖拽行为停止，将之前不可见的子Item的View重新可见
         *==========================================================*/
        case DragEvent.ACTION_DROP:
            mRecyclerView.findViewHolderForAdapterPosition(fromPosition).itemView.setVisibi
            return true;
        case DragEvent.ACTION_DRAG_ENDED:
            return true;
        default:
            break;
    }
    return false;
}

/**==========================================================
 * 长按进入拖拽状态，并将控件的数据保存到“剪切板”中
```

```
     *=========================================================*/
    @Override
    public boolean onLongClick(View view, ItemDataBean data) {
        //1. 将数据序列化-通过剪切板传输
        Bundle bundle = new Bundle();
        bundle.putParcelable("data", data);
        Intent intent = new Intent();
        intent.putExtra("bundle", bundle);
        ClipData clipData = ClipData.newIntent("intent", intent);
        //2. 影子
        View.DragShadowBuilder shadowBuilder = new View.DragShadowBuilder(view);
        //3. 震动反馈，不需要震动权限
        view.performHapticFeedback(HapticFeedbackConstants.LONG_PRESS, HapticFeedbackConstants.
        //4. 拖拽
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
            view.startDragAndDrop(clipData, shadowBuilder, view, 0);
        } else {
            view.startDrag(clipData, shadowBuilder, view, 0);
        }
        return true;
    }
}
```

2、RecyclerView的Adapter，将ItemView的长按监听的具体实现转移到我们的点击接口中(只有核心代码)

```
private RecyclerViewOnItemLongClickListenr mOnItemLongClickListenr;
public void setOnItemLongClickListener(RecyclerViewOnItemLongClickListenr onItemLongClickLister
    mOnItemLongClickListenr = onItemLongClickListener;
}
@Override
public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {
    final ViewHolder viewHolder = (ViewHolder) holder;
    viewHolder.mTextView.setText(citys.get(position));
    /**
     * 借助OnLongClickListenr将点击事件转移到自定义的Item长按事件接口
     **/
    if(mOnItemLongClickListenr != null){
        viewHolder.itemView.setOnLongClickListener(new View.OnLongClickListener() {
            @Override
            public boolean onLongClick(View v) {
                ItemDataBean itemData = new ItemDataBean();
                itemData.text =  viewHolder.mTextView.getText().toString();
                return mOnItemLongClickListenr.onLongClick(v, itemData);
            }
        });
    }
}
```

3、需要通过序列化传给目标RV的数据实体类

```
public class ItemDataBean implements Parcelable {
    public String text;

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(this.text);
    }

    public ItemDataBean() {
    }

    protected ItemDataBean(Parcel in) {
        this.text = in.readString();
    }

    public static final Parcelable.Creator<ItemDataBean> CREATOR = new Parcelable.Creator<ItemD
        @Override
        public ItemDataBean createFromParcel(Parcel source) {
            return new ItemDataBean(source);
        }

        @Override
        public ItemDataBean[] newArray(int size) {
            return new ItemDataBean[size];
        }
    };
}
```

4、目标RVB接收拖拽控件的代码

```
mRecyclerView2.setOnDragListener(new View.OnDragListener() {
    @Override
    public boolean onDrag(View v, DragEvent event) {
        switch (event.getAction()) {
            /**============================
             *  接收其他的控件放置到该RV内部
             *============================*/
            case DragEvent.ACTION_DROP:
                //1. 层层获取到传递来的数据(通过Parcelable序列化)
                ClipData clipData = event.getClipData();
                ClipData.Item clipDataItem= clipData.getItemAt(0);
                Intent intent = clipDataItem.getIntent();
                Bundle bundle = intent.getBundleExtra("bundle");
                bundle.setClassLoader(getClass().getClassLoader());
                ItemDataBean itemData = bundle.getParcelable("data");

                //2. 获取到想要加入的位置
                View childView = mRecyclerView2.findChildViewUnder(event.getX(), event.getY());
                if(childView != null){
                    RecyclerView.ViewHolder toViewHolder = mRecyclerView2.getChildViewHolder(ch
                    int targetPosition = toViewHolder.getAdapterPosition();

                    //3. 将控件添加到本控件内部
                    mRVAdapter2.getCitys().add(targetPosition, itemData.text);
                    mRVAdapter2.notifyItemInserted(targetPosition);
                    mRecyclerView2.scrollToPosition(targetPosition);

                    //4. 让第一个RV删除掉移动过来的Item
                    int fromPosition = mViewOnTouchAndDragListener.getFromPosition();
                    if(fromPosition != -1){
                        mRVAdapter.getCitys().remove(fromPosition);
                        mRVAdapter.notifyItemRemoved(fromPosition);
                        mViewOnTouchAndDragListener.setFromPosition(-1); //归零防止出问题
                    }
                }
                break;
            default:
                break;
        }
        return true;
    }
});
```

5、给RVA设置两种监听器

```
//1. 相应拖拽：自动排序效果
mRecyclerView.setOnDragListener(mViewOnTouchAndDragListener = new RecyclerViewOnTouchAndDragLis
//2. 长按开启拖拽
mRVAdapter.setOnItemLongClickListener(new RecyclerViewOnTouchAndDragListener());
```

参考和学习资料

1. 玩玩Andoid的拖拽——实现一款万能遥控器
2. Android开发之Drag&Drop框架实现拖放手势
3. 剪贴板框架：ClipData解析