自定义RadioGroup

> 在Android系统中，自带的RadioGroup只能指定横向和纵向两种布局，所以有的时候我们需要自定义RadioGroup。

首先分析一下，就是在系统自带的RadioGroup中，如果我们嵌套了，LinearLayout的话，就会失效，因为系统的RadioGroup没有考虑到这种情况，所以我们需要自定义一个Group，初步的打算是继承自LinearLayout。

具体代码如下：

```
package linsir.fuyizhulao.com.love_map;


import android.content.Context;
import android.content.res.TypedArray;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewGroup;
import android.view.accessibility.AccessibilityEvent;
import android.view.accessibility.AccessibilityNodeInfo;
import android.widget.CompoundButton;
import android.widget.LinearLayout;
import android.widget.RadioButton;


/**
 * <p>This class is used to create a multiple-exclusion scope for a set of radio
 * buttons. Checking one radio button that belongs to a radio group unchecks
 * any previously checked radio button within the same group.</p>
 *
 * <p>Intially, all of the radio buttons are unchecked. While it is not possible
 * to uncheck a particular radio button, the radio group can be cleared to
 * remove the checked state.</p>
 *
 * <p>The selection is identified by the unique id of the radio button as defined
 * in the XML layout file.</p>
 *
 * <p><strong>XML Attributes</strong></p>
 * <p>See {@link android.R.styleable#RadioGroup RadioGroup Attributes},
 * {@link android.R.styleable#LinearLayout LinearLayout Attributes},
 * {@link android.R.styleable#ViewGroup ViewGroup Attributes},
 * {@link android.R.styleable#View View Attributes}</p>
 * <p>Also see
 * {@link android.widget.LinearLayout.LayoutParams LinearLayout.LayoutParams}
 * for layout attributes.</p>
 *
 * @see RadioButton
 *
 */
public class RadioGroup extends LinearLayout {
    // holds the checked id; the selection is empty by default
    private int mCheckedId = -1;
    // tracks children radio buttons checked state
    private CompoundButton.OnCheckedChangeListener mChildOnCheckedChangeListener;
    // when true, mOnCheckedChangeListener discards events
    private boolean mProtectFromCheckedChange = false;
        private OnCheckedChangeListener mOnCheckedChangeListener;
        private PassThroughHierarchyChangeListener mPassThroughListener;

        /**
         * {@inheritDoc}
         */
    public RadioGroup(Context context) {
            super(context);
            setOrientation(VERTICAL);
            init();
    }

    /**
     * {@inheritDoc}
     */
    public RadioGroup(Context context, AttributeSet attrs) {
        super(context, attrs);
        mCheckedId = View.NO_ID;

        final int index = VERTICAL;
        setOrientation(index);

        init();
    }

    private void init() {
        mChildOnCheckedChangeListener = new CheckedStateTracker();
        mPassThroughListener = new PassThroughHierarchyChangeListener();
        super.setOnHierarchyChangeListener(mPassThroughListener);
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public void setOnHierarchyChangeListener(OnHierarchyChangeListener listener) {
        // the user listener is delegated to our pass-through listener
        mPassThroughListener.mOnHierarchyChangeListener = listener;
    }

    /**
     * {@inheritDoc}
     */
    @Override
    protected void onFinishInflate() {
        super.onFinishInflate();

        // checks the appropriate radio button as requested in the XML file
        if (mCheckedId != -1) {
            mProtectFromCheckedChange = true;
            setCheckedStateForView(mCheckedId, true);
            mProtectFromCheckedChange = false;
            setCheckedId(mCheckedId);
        }
    }

    @Override
    public void addView(final View child, int index, ViewGroup.LayoutParams params) {
        if (child instanceof RadioButton) {

            ((RadioButton) child).setOnTouchListener(new OnTouchListener() {

                @Override
                public boolean onTouch(View v, MotionEvent event) {
                    ((RadioButton) child).setChecked(true);
                    checkRadioButton((RadioButton) child);
                    if(mOnCheckedChangeListener != null){
                        mOnCheckedChangeListener.onCheckedChanged(RadioGroup.this, child.getId());
                    }
                    return true;
                }
            });

        } else if(child instanceof LinearLayout){
            int childCount = ((LinearLayout) child).getChildCount();
            for(int i = 0; i < childCount; i++){
                View view = ((LinearLayout) child).getChildAt(i);
                if (view instanceof RadioButton) {
                    final RadioButton button = (RadioButton) view;


                    ((RadioButton) button).setOnTouchListener(new OnTouchListener() {

                        @Override
                        public boolean onTouch(View v, MotionEvent event) {
                            ((RadioButton) button).setChecked(true);
                            checkRadioButton((RadioButton) button);
                            if(mOnCheckedChangeListener != null){
                                mOnCheckedChangeListener.onCheckedChanged(RadioGroup.this, button.get
                            }
                            return true;
                        }
                    });

                }
            }
        }

        super.addView(child, index, params);
    }

    private void checkRadioButton(RadioButton radioButton){
```

```java
        View child;
        int radioCount = getChildCount();
        for(int i = 0; i < radioCount; i++){
            child = getChildAt(i);
            if (child instanceof RadioButton) {
                if(child == radioButton){
                    // do nothing
                } else {
                    ((RadioButton) child).setChecked(false);
                }
            } else if(child instanceof LinearLayout){
                int childCount = ((LinearLayout) child).getChildCount();
                for(int j = 0; j < childCount; j++){
                    View view = ((LinearLayout) child).getChildAt(j);
                    if (view instanceof RadioButton) {
                        final RadioButton button = (RadioButton) view;
                        if(button == radioButton){
                            // do nothing
                        } else {
                            ((RadioButton) button).setChecked(false);
                        }
                    }
                }
            }
        }
    }

    /**
     * <p>Sets the selection to the radio button whose identifier is passed in
     * parameter. Using -1 as the selection identifier clears the selection;
     * such an operation is equivalent to invoking {@link #clearCheck()}.</p>
     *
     * @param id the unique id of the radio button to select in this group
     *
     * @see #getCheckedRadioButtonId()
     * @see #clearCheck()
     */
    public void check(int id) {
        // don't even bother
        if (id != -1 && (id == mCheckedId)) {
            return;
        }

        if (mCheckedId != -1) {
            setCheckedStateForView(mCheckedId, false);
        }

        if (id != -1) {
            setCheckedStateForView(id, true);
        }

        setCheckedId(id);
    }

    private void setCheckedId(int id) {
        mCheckedId = id;
    }

    private void setCheckedStateForView(int viewId, boolean checked) {
        View checkedView = findViewById(viewId);
        if (checkedView != null && checkedView instanceof RadioButton) {
            ((RadioButton) checkedView).setChecked(checked);
        }
    }

    /**
     * <p>Returns the identifier of the selected radio button in this group.
     * Upon empty selection, the returned value is -1.</p>
     *
     * @return the unique id of the selected radio button in this group
     *
     * @see #check(int)
     * @see #clearCheck()
     *
     * @attr ref android.R.styleable#RadioGroup_checkedButton
     */
    public int getCheckedRadioButtonId() {
        return mCheckedId;
    }

    /**
```

```
     * <p>Clears the selection. When the selection is cleared, no radio button
     * in this group is selected and {@link #getCheckedRadioButtonId()} returns
     * null.</p>
     *
     * @see #check(int)
     * @see #getCheckedRadioButtonId()
     */
    public void clearCheck() {
        check(-1);
    }

    /**
     * <p>Register a callback to be invoked when the checked radio button
     * changes in this group.</p>
     *
     * @param listener the callback to call on checked state change
     */
    public void setOnCheckedChangeListener(OnCheckedChangeListener listener) {
        mOnCheckedChangeListener = listener;
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public LayoutParams generateLayoutParams(AttributeSet attrs) {
        return new RadioGroup.LayoutParams(getContext(), attrs);
    }

    /**
     * {@inheritDoc}
     */
    @Override
    protected boolean checkLayoutParams(ViewGroup.LayoutParams p) {
        return p instanceof RadioGroup.LayoutParams;
    }

    @Override
    protected LinearLayout.LayoutParams generateDefaultLayoutParams() {
        return new LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
    }

    @Override
    public void onInitializeAccessibilityEvent(AccessibilityEvent event) {
        super.onInitializeAccessibilityEvent(event);
        event.setClassName(RadioGroup.class.getName());
    }

    @Override
    public void onInitializeAccessibilityNodeInfo(AccessibilityNodeInfo info) {
        super.onInitializeAccessibilityNodeInfo(info);
        info.setClassName(RadioGroup.class.getName());
    }

    /**
     * <p>This set of layout parameters defaults the width and the height of
     * the children to {@link #WRAP_CONTENT} when they are not specified in the
     * XML file. Otherwise, this class ussed the value read from the XML file.</p>
     *
     * <p>See
     * {@link android.R.styleable#LinearLayout_Layout LinearLayout Attributes}
     * for a list of all child view attributes that this class supports.</p>
     *
     */
    public static class LayoutParams extends LinearLayout.LayoutParams {
        /**
         * {@inheritDoc}
         */
        public LayoutParams(Context c, AttributeSet attrs) {
            super(c, attrs);
        }

        /**
         * {@inheritDoc}
         */
        public LayoutParams(int w, int h) {
            super(w, h);
        }

        /**
         * {@inheritDoc}
         */
```

```java
        public LayoutParams(int w, int h, float initWeight) {
            super(w, h, initWeight);
        }

        /**
         * {@inheritDoc}
         */
        public LayoutParams(ViewGroup.LayoutParams p) {
            super(p);
        }

        /**
         * {@inheritDoc}
         */
        public LayoutParams(MarginLayoutParams source) {
            super(source);
        }

        /**
         * <p>Fixes the child's width to
         * {@link android.view.ViewGroup.LayoutParams#WRAP_CONTENT} and the child's
         * height to  {@link android.view.ViewGroup.LayoutParams#WRAP_CONTENT}
         * when not specified in the XML file.</p>
         *
         * @param a the styled attributes set
         * @param widthAttr the width attribute to fetch
         * @param heightAttr the height attribute to fetch
         */
        @Override
        protected void setBaseAttributes(TypedArray a,
                                         int widthAttr, int heightAttr) {

            if (a.hasValue(widthAttr)) {
                width = a.getLayoutDimension(widthAttr, "layout_width");
            } else {
                width = WRAP_CONTENT;
            }

            if (a.hasValue(heightAttr)) {
                height = a.getLayoutDimension(heightAttr, "layout_height");
            } else {
                height = WRAP_CONTENT;
            }
        }
    }

    /**
     * <p>Interface definition for a callback to be invoked when the checked
     * radio button changed in this group.</p>
     */
    public interface OnCheckedChangeListener {
        /**
         * <p>Called when the checked radio button has changed. When the
         * selection is cleared, checkedId is -1.</p>
         *
         * @param group the group in which the checked radio button has changed
         * @param checkedId the unique identifier of the newly checked radio button
         */
        public void onCheckedChanged(RadioGroup group, int checkedId);
    }

    private class CheckedStateTracker implements CompoundButton.OnCheckedChangeListener {
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
            // prevents from infinite recursion
            if (mProtectFromCheckedChange) {
                return;
            }

            mProtectFromCheckedChange = true;
            if (mCheckedId != -1) {
                setCheckedStateForView(mCheckedId, false);
            }
            mProtectFromCheckedChange = false;

            int id = buttonView.getId();
            setCheckedId(id);
        }
    }

    /**
     * <p>A pass-through listener acts upon the events and dispatches them
     * to another listener. This allows the table layout to set its own internal
     * to another listener. This allows the table layout to set its own internal
     * hierarchy change listener without preventing the user to setup his.</p>
     */
    private class PassThroughHierarchyChangeListener implements
            ViewGroup.OnHierarchyChangeListener {
        private ViewGroup.OnHierarchyChangeListener mOnHierarchyChangeListener;

        /**
         * {@inheritDoc}
         */
        public void onChildViewAdded(View parent, View child) {
            if (parent == RadioGroup.this && child instanceof RadioButton) {
                int id = child.getId();
                // generates an id if it's missing
                if (id == View.NO_ID) {
                    id = child.hashCode();
                    child.setId(id);
                }
                ((RadioButton) child).setOnCheckedChangeListener(
                        mChildOnCheckedChangeListener);
            }

            if (mOnHierarchyChangeListener != null) {
                mOnHierarchyChangeListener.onChildViewAdded(parent, child);
            }
        }

        /**
         * {@inheritDoc}
         */
        public void onChildViewRemoved(View parent, View child) {
            if (parent == RadioGroup.this && child instanceof RadioButton) {
                ((RadioButton) child).setOnCheckedChangeListener(null);
            }

            if (mOnHierarchyChangeListener != null) {
                mOnHierarchyChangeListener.onChildViewRemoved(parent, child);
            }
        }
    }
}
```

这样我们的RadioGroup下面就可以使用布局了，不过目前仅对LinearLayout做了兼容，一般来说这样，就已经可以满足我们的需求了，当然如果我们喜欢的话，也可以对其他的布局进行兼容。