菜单

https://developer.android.google.cn/develop/ui/views/components/menus?hl=zh-cn#xml
Android菜单有三种类型：选项菜单和应用栏、上下文菜单和关联操作模式、弹出式菜单。

使用XML定义菜单

所有菜单类型都可以通过XML定义。要定义菜单，要在项目的`res/menu/`目录下创建一个XML资源文件。

- <menu>菜单项的容器，<menu>必须是该文件的根节点，可以包含一个或多个<item>和<group>元素。
- <item>创建MenuItem对象，表示菜单中的一项，可以嵌套，使用<item>创建子菜单。
- <group>创建MenuGroup对象，表示菜单项的分组，组内共享属性。
- 名为game_menu.xml的菜单示例：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game"
          app:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

- <item>元素支持多个属性：
  - android:id：菜单项的唯一ID，必须是@+id/开头的资源ID。
  - android:icon：菜单项的图标。
  - android:title：菜单项的标题。
  - android:showAsAction：菜单项的显示方式。
    - ifRoom：如果有空间，菜单项将显示在应用栏中。
    - never：菜单项将永远不会显示在应用栏中。
    - withText：如果有空间，菜单项将显示在应用栏中，并且菜单项的标题将显示在应用栏中。
    - always：菜单项将始终显示在应用栏中。

创建选项菜单

开启应用栏`ActionBar`,修改主题`res/values/themes/themes.xml`，去掉`NoActionBar`主题。
重构`MainActivity`类，添加`onCreateOptionsMenu()`方法，加载选项菜单。

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

- 可以使用`add()`和`findItem()`修改属性。

处理点击事件

重构`MainActivity`类，添加`onOptionsItemSelected()`方法，处理点击事件。

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection.
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

- 成功处理返回值`true`，不处理菜单项调用父类的`onOptionsItemSelected()`方法，返回`false`。

运行时更改菜单项

重构`MainActivity`类，添加`onPrepareOptionsMenu()`方法，运行时更改菜单项。

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    // If the game is running, disable the menu options.
    if (mGameRunning) {
        MenuItem item = menu.findItem(R.id.new_game);
        item.setEnabled(false);
    }
    return true;
}
```

创建上下文菜单

1. 在`OnCreate()`中调用`registerForContextMenu()`方法向其传递上下文菜单ID注册上下文菜单。
2. 在`OnCreateContextMenu()`中为指定的上下文菜单ID创建上下文菜单。

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

3. 在`OnContextItemSelected()`中处理点击事件。

```
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
        switch (item.getItemId()) {
            case R.id.edit:
                editNote(info.id);
                return true;
            case R.id.delete:
                deleteNote(info.id);
                return true;
            default:
                return super.onContextItemSelected(item);
        }
    }
```

创建菜单组

- 使用`<group>`元素内部嵌套`<item>`元素创建菜单组。在`<group>`元素中使用`<android:checkableBehavior>`属性指定菜单项的行为。
    - `single`：菜单项将成为单选按钮组。
    - `all`：菜单项将成为复选框组。
    - `none`：菜单项将成为普通菜单项。

```
// 使用单选按钮的菜单组
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item android:id="@+id/red"
              android:title="@string/red" />
        <item android:id="@+id/blue"
              android:title="@string/blue" />
    </group>
</menu>
```

- 在`onOptionsItemSelected()`方法中处理点击事件，使用`isChecked()`方法判断菜单项是否被选中，使用`setChecked()`方法设置菜单项的选中状态。

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection.
    switch (item.getItemId()) {
        case R.id.red:
            if (item.isChecked()) {
                item.setChecked(false);
            } else {
                item.setChecked(true);
            }
            return true;
        case R.id.blue:
            if (item.isChecked()) {
                item.setChecked(false);
            } else {
                item.setChecked(true);
            }
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

对话框

https://developer.android.google.cn/develop/ui/views/components/dialogs?hl=zh-cn

- Dialog类是对话框的基类，但是不能实例化。而是使用子类：
  - AlertDialog：可显示标题、最多三个按钮、按钮或自定义布局。
  - ProgressDialog：显示进度条。
  - DatePickerDialog：显示日期选择器。
  - TimePickerDialog：显示时间选择器。
  - NumberPickerDialog：显示数字选择器。

创建对话框

- 使用AlertDialog.Builder创建对话框。
- 使用setTitle()设置对话框的标题。
- 使用setMessage()设置对话框的消息。
- 使用setPositiveButton()设置对话框的确定按钮。
- 使用setNegativeButton()设置对话框的取消按钮。
- 使用setNeutralButton()设置对话框的中立按钮。
- 使用setView()设置对话框的自定义布局。
- 使用setCancelable()设置对话框是否点击空白处可取消。
- 使用setOnCancelListener()设置对话框的取消回调。
- 使用setItems()设置对话框的列表。
- 使用setSingleChoiceItems()设置对话框的单选按钮组。
- 使用setMultiChoiceItems()设置对话框的复选框组。
- 使用create()创建对话框。
- 使用show()显示对话框。

```
// 创建对话框
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Title");
builder.setMessage("Message");
builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        // 处理点击事件
    }
});
builder.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        // 处理点击事件
    }
});
builder.setNeutralButton("Neutral", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        // 处理点击事件
    }
});
// 创建对话框
AlertDialog dialog = builder.create();
// 显示对话框
dialog.show();
```

列表视图（ListView）

https://developer.android.google.cn/reference/android/widget/ListView
显示可垂直滚动的视图集合。

```
// 使用`<ListView>`创建列表视图。
<ListView
      android:id="@+id/list_view"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />
```

- 列表视图会根据ListAdapter的内容自动调整大小，使用setAdapter(android.widget.ListAdapter)方法将ListAdapter设置到列表视图。
- 这里使用ArrayAdapter创建ListAdapter。
- 重写ArrayAdapter的getView()方法，自定义列表视图的布局。

```
// 子类化`ArrayAdapter`，重写`getView()`方法。
public class MyAdapter extends ArrayAdapter<String> {
    public MyAdapter(Context context, ArrayList<String> words) {
        super(context, 0, words);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // 获取当前项的单词。
        String word = getItem(position);
        // 如果`convertView`为空，使用`LayoutInflater`加载布局。
        if (convertView == null) {
            convertView = LayoutInflater.from(getContext()).inflate(R.layout.item_word, pa
        }
        // 查找布局中的`TextView`。
        TextView tvWord = (TextView) convertView.findViewById(R.id.tv_word);
        // 设置`TextView`的文本。
        tvWord.setText(word);
        // 返回布局。
        return convertView;
    }
}

// 使用布局文件实例化一个视图
View view = LayoutInflater.from(getContext()).inflate(R.layout.item_word, parent, false);
```

- 使用AdapterView.OnItemClickListener处理列表视图的点击事件。

```
// 处理列表视图的点击事件。
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        // 处理点击事件
    }
});
```

使用 RecyclerView 创建动态列表

https://developer.android.google.cn/develop/ui/views/layout/recyclerview?hl=zh-cn#java

规划布局

RecyclerView 中的列表项由 LayoutManager 类负责排列，三种布局管理器：

- LinearLayoutManager：线性布局管理器。
- GridLayoutManager：网格布局管理器。
- StaggeredGridLayoutManager：瀑布流布局管理器。

实现适配器和 ViewHolder

适配器继承自 RecyclerView.Adapter 类，实现 onCreateViewHolder() 、 onBindViewHolder() 和 getItemCount() 方法。

```
// 实现适配器和 ViewHolder。
public class WordListAdapter extends RecyclerView.Adapter<WordListAdapter.WordViewHolder>
    private final LayoutInflater mInflater;
    private List<Word> mWords; // Cached copy of words

    WordListAdapter(Context context) {
        mInflater = LayoutInflater.from(context);
    }

    @Override
    public WordViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View itemView = mInflater.inflate(R.layout.recyclerview_item, parent, false);
        return new WordViewHolder(itemView);
    }

    @Override
    public void onBindViewHolder(WordViewHolder holder, int position) {
        if (mWords!= null) {
            Word current = mWords.get(position);
            holder.wordItemView.setText(current.getWord());
        } else {
            // Covers the case of data not being ready yet.
            holder.wordItemView.setText("No Word");
        }
    }

    void setWords(List<Word> words){
        mWords = words;
        notifyDataSetChanged();
    }

    // getItemCount() is called many times, and when it is first called,
    // mWords has not been updated (means initially, it's null, and we can't return null).
    @Override
    public int getItemCount() {
        if (mWords!= null)
            return mWords.size();
        else return 0;
    }

    class WordViewHolder extends RecyclerView.ViewHolder {
        private final TextView wordItemView;

        private WordViewHolder(View itemView) {
            super(itemView);
            wordItemView = itemView.findViewById(R.id.textView);
        }
    }
}
```

关联适配器和布局管理器

```
// 关联适配器和布局管理器。
mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
mAdapter = new WordListAdapter(this);
mRecyclerView.setAdapter(mAdapter);
```