应用 Twire 详细报告

1. 应用简介

应用名称：Twire
应用类型：Multi-Media
应用描述：Twire 是一款适用于 Android 的开源、无广告的 Twitch 流媒体播放器。

2. 功能场景与详细设计

场景序号	场景名称	包含组件数
1	热门流媒体显示场景	2
2	热门游戏显示场景	2
3	搜索场景	2
4	设置场景	5

2.1 热门流媒体显示场景

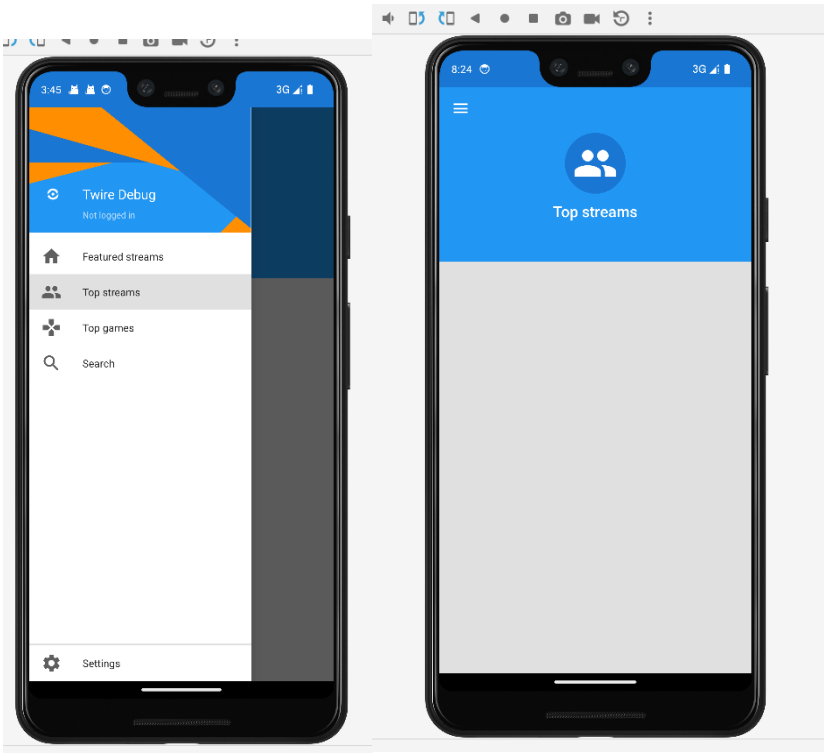
主要功能职责：打开侧边栏点击 "TopStreams" 按钮后会在页面显示当前热门的流媒体
与其他功能场景的切换：

- 打开侧边栏，点击 "Settings" 按钮，可切换至设置场景；
- 打开侧边栏，点击 "Search" 按钮，可切换至搜索场景；
- 打开侧边栏，点击 "Top games" 按钮，可切换至热门游戏显示场景；

包含组件： MainActivity, TopStreamsActivity

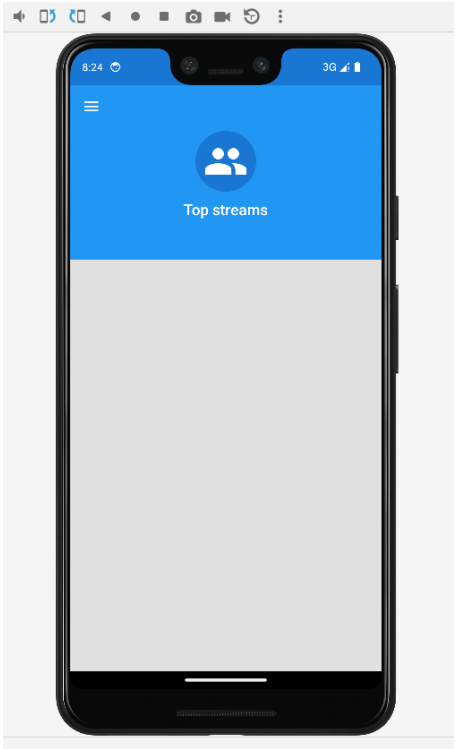
MainActivity:

- **功能职责:** 作为应用打开的主界面，默认情况下会显示当前热门的流媒体，并且会通过 customizeActivity()进行定制化的界面显示
- **截图**



TopStreamsActivity:

- **功能职责:** 显示热门的流媒体内容并在主界面显示
- **截图**



场景内组件间跳转关系（图像可选）:

- MainActivity 在应用打开后默认显示 TopStreamsActivity 的内容。

2.2 热门游戏显示场景

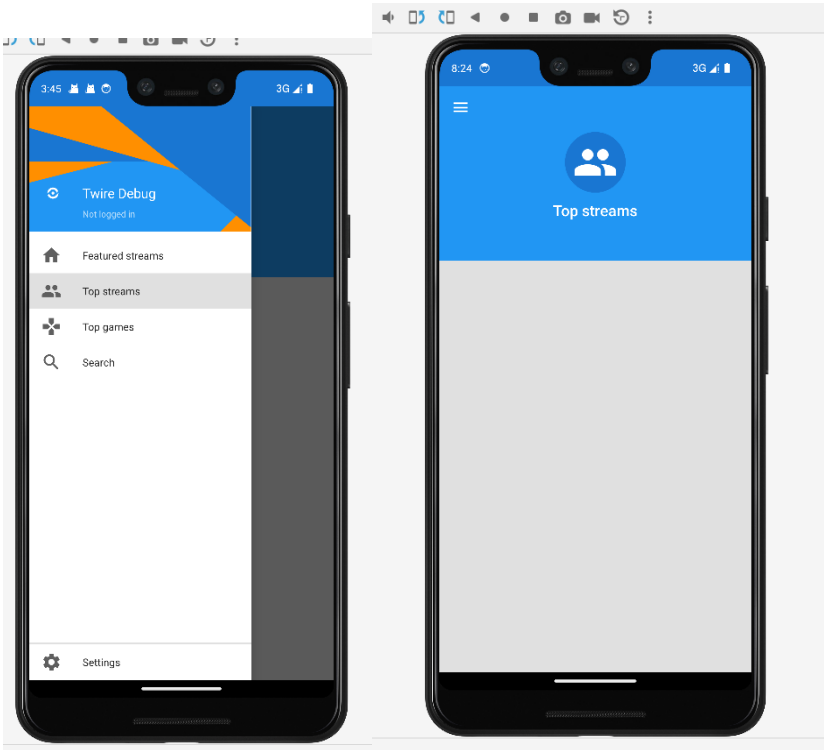
主要功能职责：打开侧边栏点击 "Top games" 按钮后会在页面显示当前热门的流媒体
与其他功能场景的切换：

- 打开侧边栏，点击 "Settings" 按钮，可切换至设置场景；
- 打开侧边栏，点击 "Search" 按钮，可切换至搜索场景；
- 打开侧边栏，点击 "Top streams" 按钮，可切换至热门流媒体显示场景；

包含组件： MainActivity, TopGamesActivity

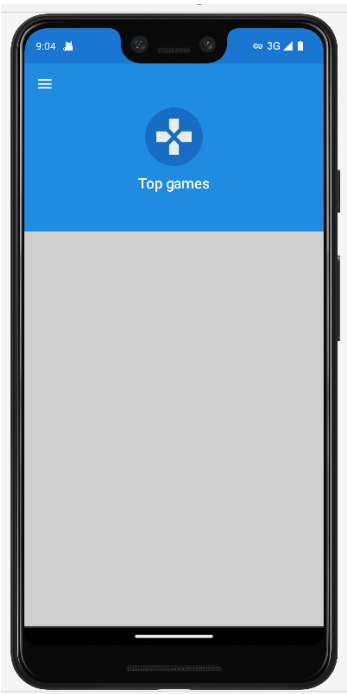
MainActivity:

- **功能职责:** 作为应用打开的主界面，默认情况下会显示当前热门的流媒体，并且会通过 customizeActivity()进行定制化的界面显示
- **截图**



TopGamesActivity:

- **功能职责:** 显示热门的游戏内容并在主界面显示
- **截图**



场景内组件间跳转关系（图像可选）：

- 通过定制化操作可以让 MainActivity 在主页显示 TopGamesActivity 的内容。

2.3 设置显示场景

主要功能职责：打开侧边栏点击 "Settings" 按钮后会在页面显示当前热门的流媒体

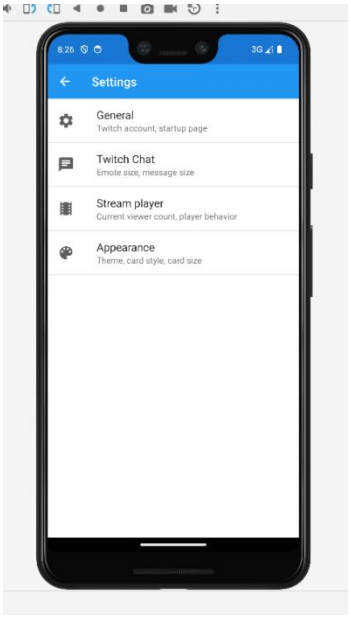
与其他功能场景的切换：

- 打开侧边栏，点击 "Top Games" 按钮，可切换至游戏直播显示场景；
- 打开侧边栏，点击 "Search" 按钮，可切换至搜索场景；
- 打开侧边栏，点击 "Top streams" 按钮，可切换至热门流媒体显示场景。

包含组件： SettingsActivity 、 ThemeActivity 、 SettingsGeneralActivity 、 SettingsTwitchChatActivity、SettingsStreamPlayerActivity、SettingsAppearanceActivity

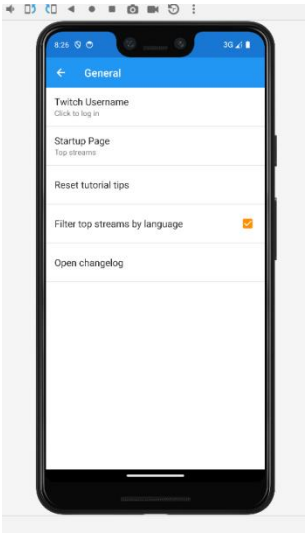
SettingsActivity:

- **功能职责:** 作为应用打开设置的主界面，默认情况下会显示设置选项
- **截图**



SettingsGeneralActivity:

- **功能职责:** 作为应用打开 General 的界面，默认情况下会显示 General 设置选项
- **截图**

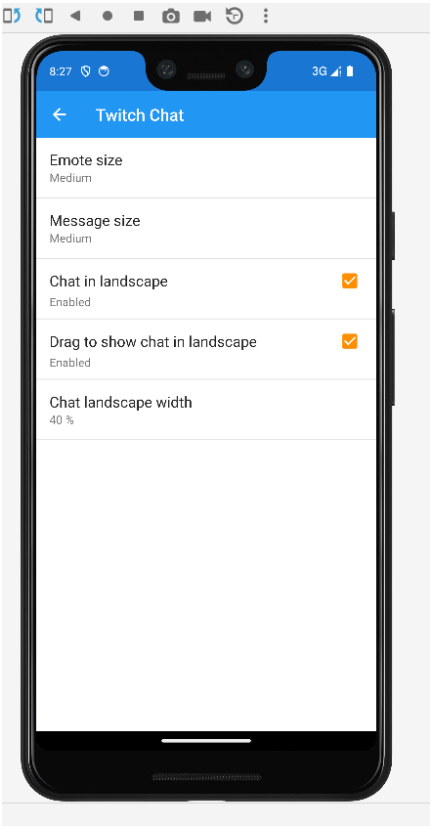


场景内组件间跳转关系：

- 通过点击操作可以让 SettingsActivity 在主页显示 SettingsGeneralActivity 的内容。

SettingsTwitchChatActivity:

- **功能职责:** 作为应用打开 TwitchChat 的界面，默认情况下会显示 TwitchChat 设置选项，设置对话的字体、颜色等
- **截图**

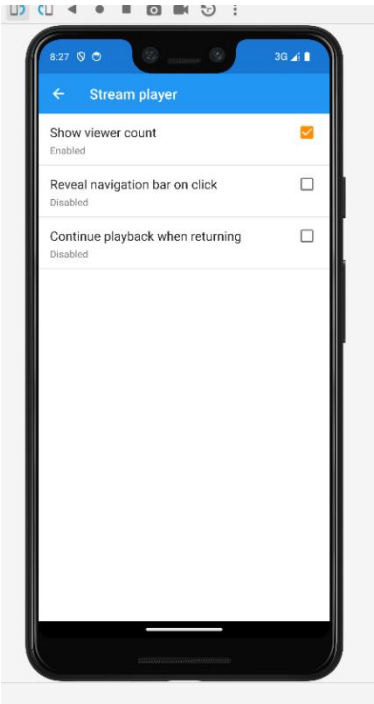


场景内组件间跳转关系：

- 通过点击操作可以让 SettingsActivity 在主页显示 SettingsTwitchChatActivity 的内容。

SettingsStreamPlayerActivity:

- **功能职责:** 作为应用打开 StreamPlayer 的界面，默认情况下会显示 StreamPlayer 设置选项
- **截图**

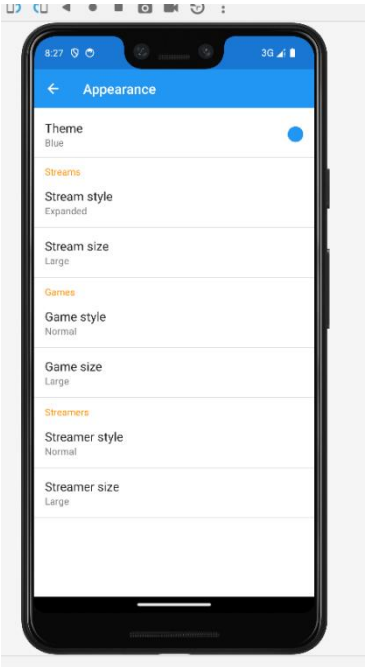


场景内组件间跳转关系：

- 通过点击操作可以让 SettingsActivity 在主页显示 SettingsStreamPlayerActivity 的内容。

SettingsAppearanceActivity:

- **功能职责:** 作为应用打开 Appearance 的界面，默认情况下会显示 Appearance 设置选项，设置主题、媒体风格、游戏风格等
- **截图**



场景内组件间跳转关系：

- 通过点击操作可以让 SettingsActivity 在主页显示 SettingsAppearanceActivity 的内容。

2.4 搜索显示场景

主要功能职责：打开侧边栏点击 "Search" 按钮后会在页面显示当前热门的流媒体
与其他功能场景的切换：

- 搜索栏下面有三个按钮，点击 "Streams" 按钮，可切换至流媒体直播显示场景；
- 搜索栏下面有三个按钮，点击 "Games" 按钮，可切换至游戏直播显示场景；
- 搜索栏下面有三个按钮，点击 "Streamers" 按钮，可切换至设置场景。

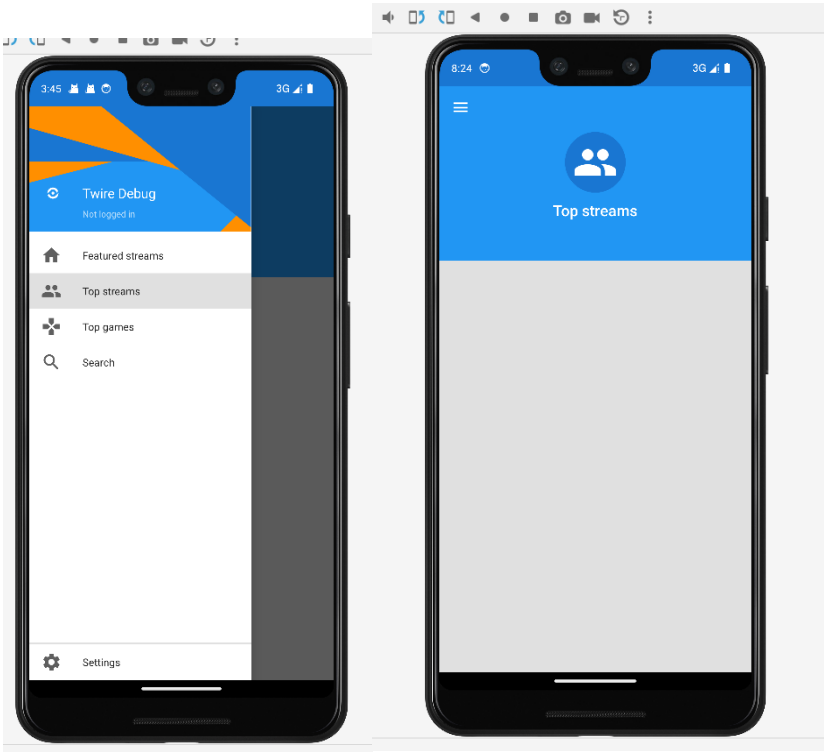
包含组件：SearchActivity，MainActivity

SearchActivity:

- **功能职责：**作为应用打开搜索的主界面，默认情况下会显示设置选项，点击 "STREAMS" 按钮，可切换至直播场景；点击 "STREAMERS" 按钮，可切换至主播场景；点击 "GAMES" 按钮，可切换至游戏场景。
- **截图**

MainActivity:

- **功能职责:** 作为应用打开的主界面，默认情况下会显示当前热门的流媒体，并且会通过 customizeActivity()进行定制化的界面显示
- **截图**



场景内组件间跳转关系：

- 通过点击返回键会回到主页即 MainActivity

3. 场景内详细设计

3.1 适配器模式

功能场景位置：热门流媒体显示场景
组件位置：TopStreamActivity
涉及类：TopStreamActivity，StreamAdapter，MainActivityAdapter，RecyclerView.Adapter，StreamInfo, StreamViewHolder
描述（图像可选）：
这是一种类适配器的模式。RecyclerView.Adapter 是目标接口，它规定了适配器应该实现的方法，例如 onCreateViewHolder () 或 onBindViewHolder()，用于在界面上显示流媒体 list 的数据。MainActivityAdapter 是一个通用的抽象适配器类，它继承 RecyclerView 并且对方法进行重写。StreamAdapter 是适配器同时也是被适配者，它继承 MainActivityAdapter 获取重写的方法，并且用泛型<StreamInfo, StreamViewHolder>规定了 list 存放的是流媒体相关信息，同时负责将其中包含的 List< StreamInfo >数据适配至 StreamAdapter 中进行列表显示。

```
// It tells the adapter how we want to the layout of the data for each row should be fo
public T onCreateViewHolder(ViewGroup viewGroup, int i) {
    View itemView = LayoutInflater.
            from(viewGroup.getContext()).
            inflate(getLayoutResource(), viewGroup, attachToRoot: false);

    itemView.setOnClickListener(mOnClickListener);
    itemView.setOnLongClickListener(mOnLongClickListener);
    return getElementsViewHolder(itemView);
}
```

```
        update.
public void onBindViewHolder(@NonNull VH holder, int position,
        @NonNull List<Object> payloads) {
    onBindViewHolder(holder, position);
}
```

3.2 模板方法模式

功能场景位置：热门游戏显示场景
组件位置：TopGameActivity
涉及类：TopGameActivity, LazyMainActivity, MainActivity
描述（图像可选）：
TopGameActivity 出现了模板方法模式的使用。在 MainActivity 中，定义了一系列抽象方法，例如 constructAdapter、customizeActivity、getActivityIconRes 等， GameActivity 继承 LazyMainActivity，LazyMainActivity 继承 MainActivity， GameActivity 重写了 MainActivity 定义的这些抽象方法来完成具体的操作。

```
 */
3 usages  3 overrides  Sebastian Rask Jepsen
protected void customizeActivity() {
}
```

```
3 usages  Sebastian Rask Jepsen
@Override
protected void customizeActivity() {
    super.customizeActivity();
    setLimit(20);
}
```

3.3 工厂方法模式

功能场景位置：setup 场景内
组件位置：ConfirmSetupActivity
涉及类：ConfirmSetupActivity、SupportAnimator

描述：
工厂模式:类的创建依赖工厂类，定义一个创建对象的抽象方法，这样一旦需要增加新的功能，工厂类是 ConfirmSetupActivity，用它创建 SupportAnimator。

```
private SupportAnimator showTransitionAnimation() {
    // Get the center for the FAB
    int cx = (int) mContinueFABContainer.getX() + mContinueFABContainer.getMeasuredHeight() / 2;
    int cy = (int) mContinueFABContainer.getY() + mContinueFABContainer.getMeasuredWidth() / 2;

    // get the final radius for the clipping circle
    int dx = Math.max(cx, mTransitionViewWhite.getWidth() - cx);
    int dy = Math.max(cy, mTransitionViewWhite.getHeight() - cy);
    float finalRadius = (float) Math.hypot(dx, dy);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {...}

    final SupportAnimator blueTransitionAnimation =
            ViewAnimationUtils.createCircularReveal(mTransitionViewWhite, cx, cy, 0, finalRadius);
    blueTransitionAnimation.setInterpolator(new AccelerateDecelerateInterpolator());
    blueTransitionAnimation.setDuration(REVEAL_ANIMATION_DURATION);
    blueTransitionAnimation.addListener(new SupportAnimator.AnimatorListener() {...});

    new Handler().postDelayed(blueTransitionAnimation::start, REVEAL_ANIMATION_DELAY);

    return blueTransitionAnimation;
}
```

3.4 备忘录模式

功能场景位置：setting 场景内
组件位置：SettingsActivity
涉及类：SettingsActivity、Bundle、ThemeActivity
描述：
发起人角色：进入设置页面时候记录当前时刻的内部状态信息，提供创建备忘录和恢复备忘录数据的功能，实现其他业务功能，它可以访问备忘录里的所有信息。
备忘录角色：Bundle 负责存储发起人的内部状态，在需要的时候提供这些内部状态给发起人。
管理者角色：SettingsActivity 对备忘录进行管理，提供保存与获取备忘录的功能，但其不能对备忘录的内容进行访问与修改

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_settings);
    ButterKnife.bind(this);

    setSupportActionBar(mToolbar);
    if (getSupportActionBar() != null) {
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    }
    SettingsCategoryAdapter mAdapter = new SettingsCategoryAdapter();
    mAdapter.setItemCallback(this);
    mAdapter.addItems(constructSettingsCategories());

    mCategoryList.setAdapter(mAdapter);
    mCategoryList.setLayoutManager(new LinearLayoutManager(getBaseContext()));
    mCategoryList.setItemAnimator(new DefaultItemAnimator());
}
```

3.5 迭代器模式

功能场景位置：setting 场景内
组件位置：SettingsActivity
涉及类：SettingsActivity、SettingsCategory
描述：
SettingsActivity 使用列表保存 SettingsCategory 的实例

```
private List<SettingsCategory> constructSettingsCategories() {
    return new ArrayList<>(Arrays.asList(
            new SettingsCategory(
                    R.string.settings_general_name,
                    R.string.settings_general_name_summary,
                    R.drawable.ic_settings,
                    constructCategoryIntent(SettingsGeneralActivity.class)
            ),
            new SettingsCategory(
                    R.string.settings_chat_name,
                    R.string.settings_chat_name_summary,
                    R.drawable.ic_chat,
                    constructCategoryIntent(SettingsTwitchChatActivity.class)
            ),
            new SettingsCategory(
                    R.string.settings_stream_player_name,
                    R.string.settings_stream_player_summary,
                    R.drawable.ic_theaters,
                    constructCategoryIntent(SettingsStreamPlayerActivity.class)
            ),
            new SettingsCategory(
                    R.string.settings_appearance_name,
                    R.string.settings_appearance_summary,
                    R.drawable.ic_palette,
                    constructCategoryIntent(SettingsAppearanceActivity.class)
```

3.6 生成器模式

功能场景位置：setup 场景内
组件位置：LoginActivity
涉及类：SupportAnimator、LoginActivity、SetupBaseActivity
描述：
生成器模式将一个对象分解为各个组件、将对象组件的构造封装起来、可以控制整个对象的生成过程。产品：具体生产器要构造的复杂对象 AnimatorListener；
具体生产器：SupportAnimator 实现 SetupBaseActivity 接口的类，具体生成器将实现 Builder 接口所定义的方法；指挥者：指挥者是 LoginActivity 类，该类需要含有 SupportAnimator 接口声明的变量。指挥者的职责是负责向用户提供具体生成器，即指挥者将请求具体生成器类来构造用户所需要的 AnimatorListener 对象，如果所请求的具体生成器成功地构造出 AnimatorListener 象，指挥者就可以让该具体生产器返回所构造的 AnimatorListener 对象。

3.7 观察者模式

功能场景位置：search 场景内
组件位置：SearchActivity
涉及类：SearchActivity、TextWatcher, EditText
描述：
TextWatcher 接口的使用是一个观察者模式的示例。TextWatcher 定义了一组回调方法，当 EditText 文本内容发生变化时，通知监听器执行相应的操作。其中，mSearchText 是被观察

者，这个匿名内部类是一个观察者。当搜索框中的文本发生变化时，被观察者 mSearchText 会触发 TextWatcher 接口中的 onTextChanged()方法对搜索框进行改变。

```
    Sebastian Rask Jepsen +1
    mSearchText.addTextChangedListener(new TextWatcher() {

        Sebastian Rask Jepsen
        @Override
        public void onTextChanged(CharSequence s, int start, int before, int count) {
            Log.d(LOG_TAG,  msg: "Text changed. Resetting fragments");
            String newQuery = s.toString().replace( target: " ",  replacement: "%20");
            mChannelsFragment.reset(newQuery);
            mStreamsFragment.reset(newQuery);
            mGamesFragment.reset(newQuery);
            query = newQuery;
        }

        TacoTheDank +1
        @Override
        public void afterTextChanged(Editable s) {
        }

        TacoTheDank +1
        @Override
        public void beforeTextChanged(CharSequence s, int start, int count, int after) {
        }
    });
    mBackIcon.setOnClickListener(v -> onBackPressed());
}
```

4. 其他详细设计

4.1 单例模式

位置：TLSSocketFactoryCompat.java
涉及类：TLSSocketFactoryCompat, SSLSocketFactory
描述（图像可选）：
TLSSocketFactoryCompat 出现了懒汉式单例模式的使用。TLSSocketFactoryCompat 继承 SSLSocketFactory 类。TLSSocketFactoryCompat 中，设立私有变量 instance 为 null，提供 getInstance 保证这个变量只能从这个方法访问。在 getInstance()方法中，通过检查 instance 变量是否为 null，来确保只有一个 TLSSocketFactoryCompat 实例被创建并且在后续调用时被返回，这是典型的懒汉式单例模式。

```
public class TLSSocketFactoryCompat extends SSLSocketFactory {


    4 usages
    private static TLSSocketFactoryCompat instance = null;

    9 usages
    private final SSLSocketFactory internalSSLSocketFactory;
```

```
1 usage   tossj
public static TLSSocketFactoryCompat getInstance()
        throws NoSuchAlgorithmException, KeyManagementException {
    if (instance != null) {
        return instance;
    }
    instance = new TLSSocketFactoryCompat();
    return instance;
}
```

4.2 迭代器模式

位置：ChatEmoteManager.java
涉及类：ChatEmoteManager
描述（图像可选）：
ChatEmoteManager 出现了迭代器模式的使用。loadCustomEmotes 方法中通过获取迭代器对 channelsets 进行遍历访问与处理。通过 hasNext 查看 set 中是否还有下一个，通过 next 获取下一个元素，是经典的迭代器模式的使用。

```
        String ffzResponse = Service.urlToJSONString(FFZ_CHANNEL_URL);
        if (!ffzResponse.isEmpty()) {
            JSONObject channelTopObject = new JSONObject(ffzResponse);
            JSONObject channelSets = channelTopObject.getJSONObject(SETS);
            for (Iterator<String> iterator = channelSets.keys(); iterator.hasNext(); ) {
                JSONArray emoticons = channelSets.getJSONObject(iterator.next()).getJSONArray(EMOTICONS);
                for (int emoteIndex = 0; emoteIndex < emoticons.length(); emoteIndex++) {
                    Emote emote = ToFFZ(emoticons.getJSONObject(emoteIndex));
                    emote.setCustomChannelEmote(true);
                    customChannel.add(emote);
                    result.put(emote.getKeyword(), emote);
                }
            }
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
```