01. 需求介绍

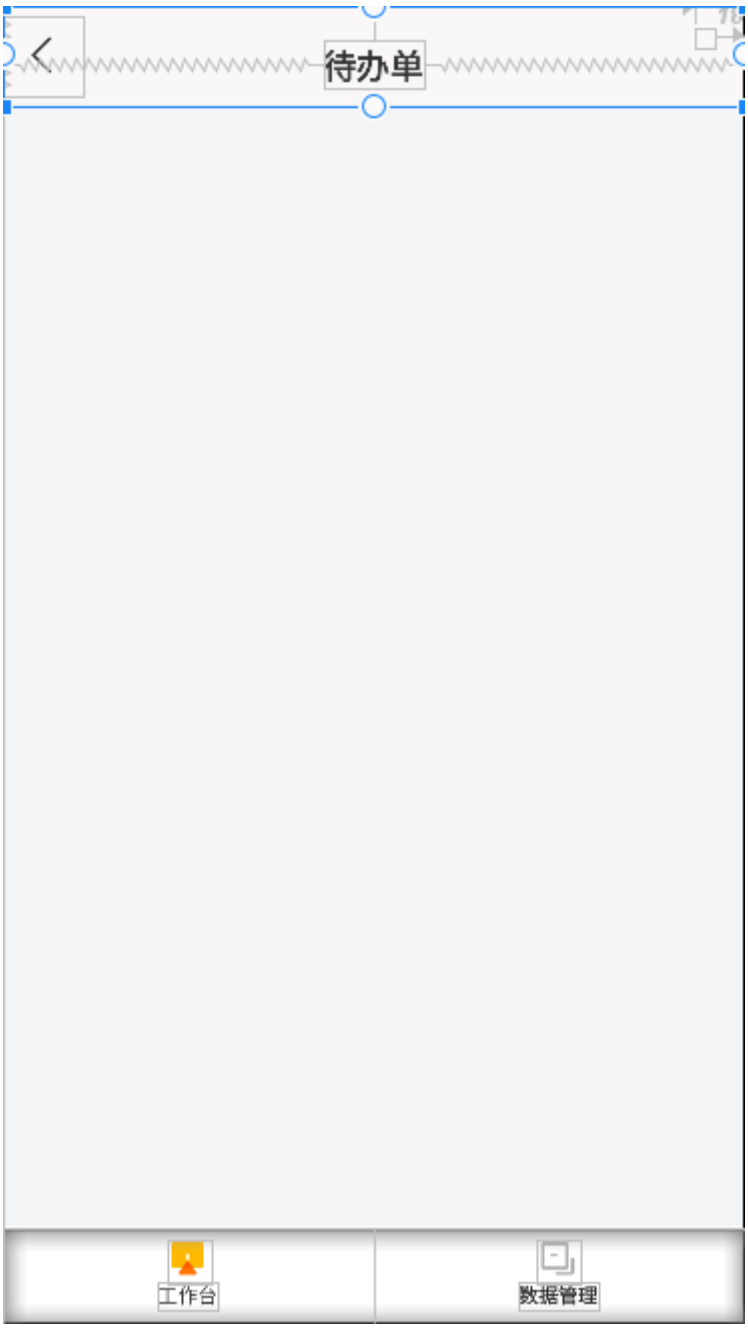
- https://codesign.qq.com/app/s/512680383255699
- 

- 数据明细页面UI更新。
- 筛选框添加保存当前筛选按钮和逻辑
- 保存筛选要有弹窗。

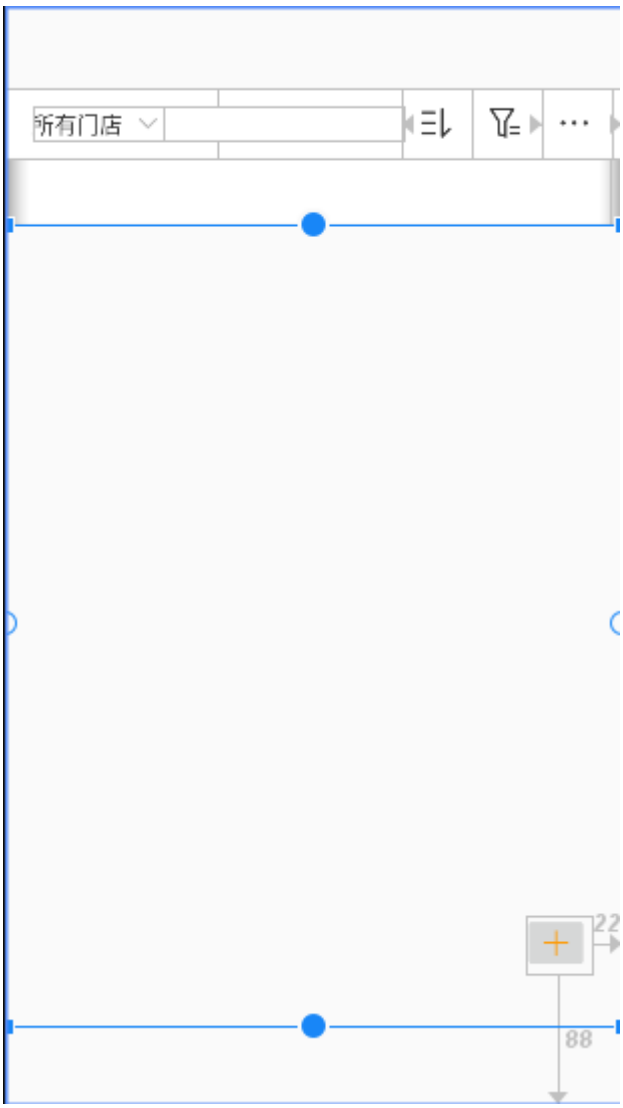
02.相关代码逻辑分析

2.1 数据明细页面UI相关代码逻辑

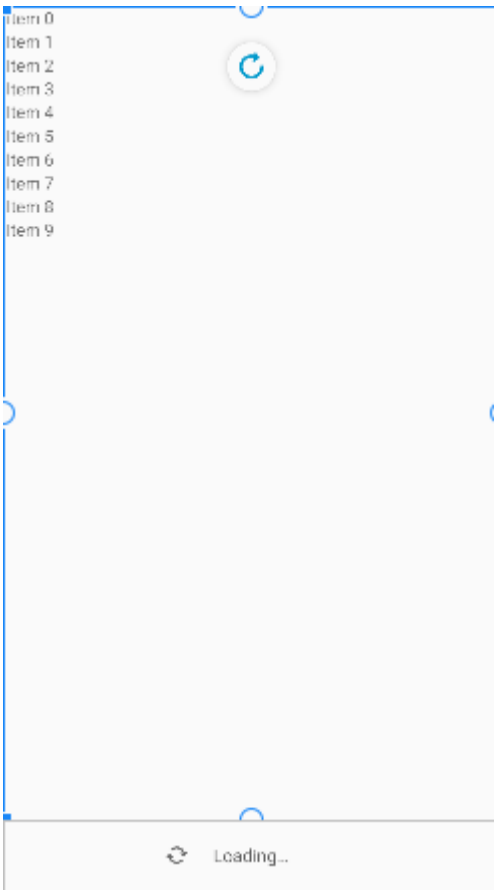
- 首先是代办单Activity：ProblemChangeActivity



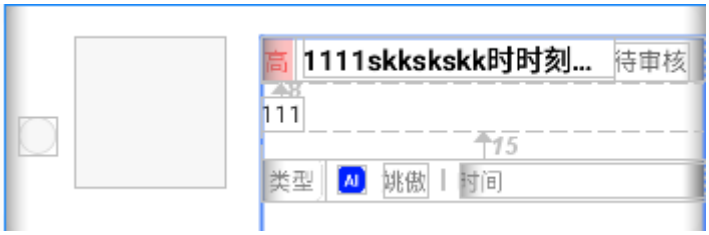
    - 内部包含两个Fragment，分别是：workBenchFragment和dataManageFragment。他们通过FragmentManager进行管理。
- 看看工作台Fragment：workBenchFragment

    - 

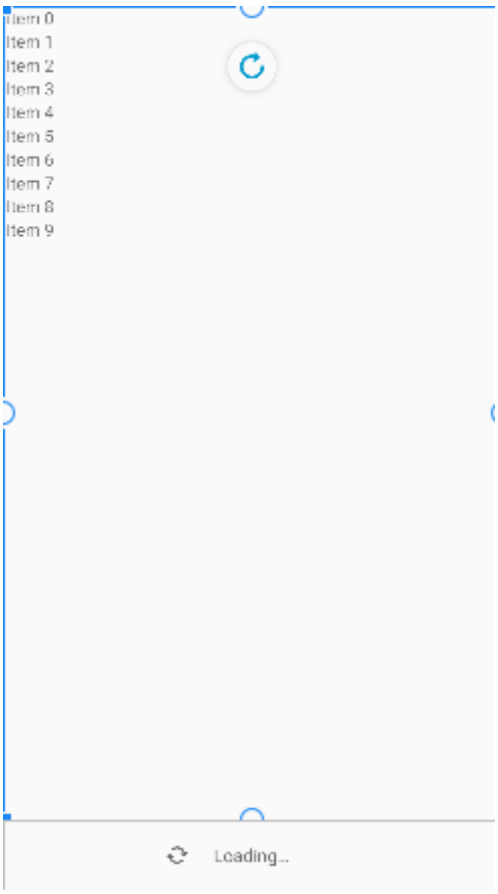
    - 内部包含一个ViewPager，其Fragment数据集为 `private val fragments = ArrayList<ProblemChangeFragment>()`
- 看看ProblemChangeFragment类型Fragment，也就是ViewPager填充的部分



    - 内部是一个RecyclerView去包含子条目
- 我们来看看子条目视图逻辑
    - 首先是RecyclerView的adapter，`lateinit var adapter: ProblemMultiModeAdapter`
    - 其在onCreateViewHolder阶段，根据ViewType填充不同的子条目
        - ```
          override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
          RecyclerView.ViewHolder {
                  if (viewType == MODE_LIST) {
                      val binding = ItemProblemChangeOBinding.inflate(
                          LayoutInflater.from(parent.context),
                          parent,
                          false
                      )
                      return ProblemListViewHolder(binding)
                  } else if (viewType == MODE_STORE) {
                      val binding =
                          ItemProblemStatisticsBinding.inflate(
                              LayoutInflater.from(parent.context),
                              parent,
                              false
                          )
                      return ProblemStoreViewHolder(binding)
                  }
                  val binding = ItemProblemChangeOBinding.inflate(
                      LayoutInflater.from(parent.context),
                      parent,
                      false
                  )
                  return ProblemListViewHolder(binding)
              }
          ```
    - 首先是ItemProblemChangeOBinding
          ```

    - 

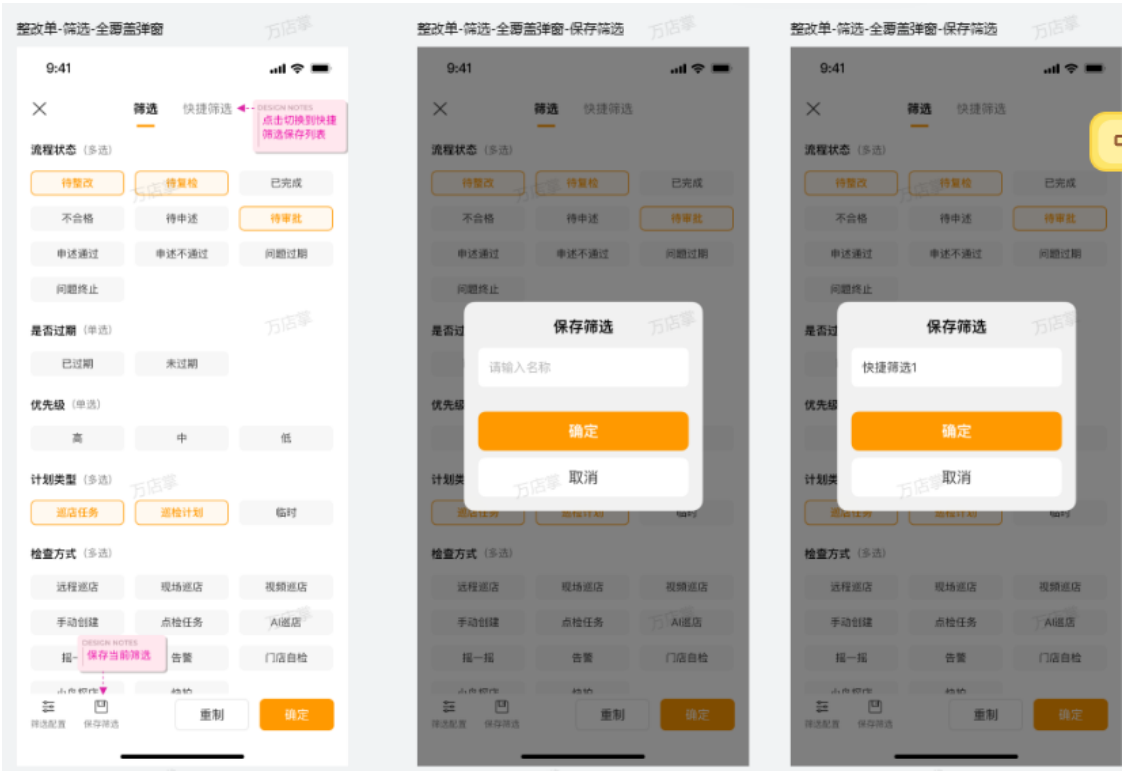
    - 然后是ItemProblemStatisticsBinding
    - 

- 好，我们现在回到DataManagerFragment
    - 

    - 视图逻辑和ProblemChangeFragment，也就是工作台Fragment内部ViewPager填充部分一样。

- 看看Adapter：`lateinit var adapter: ProblemDataManageAdapter`
- ProblemDataManageAdapter内部子条目条目视图ItemDataManageBinding的逻辑：
  - 

2.2 筛选框新增按钮相关逻辑

- 

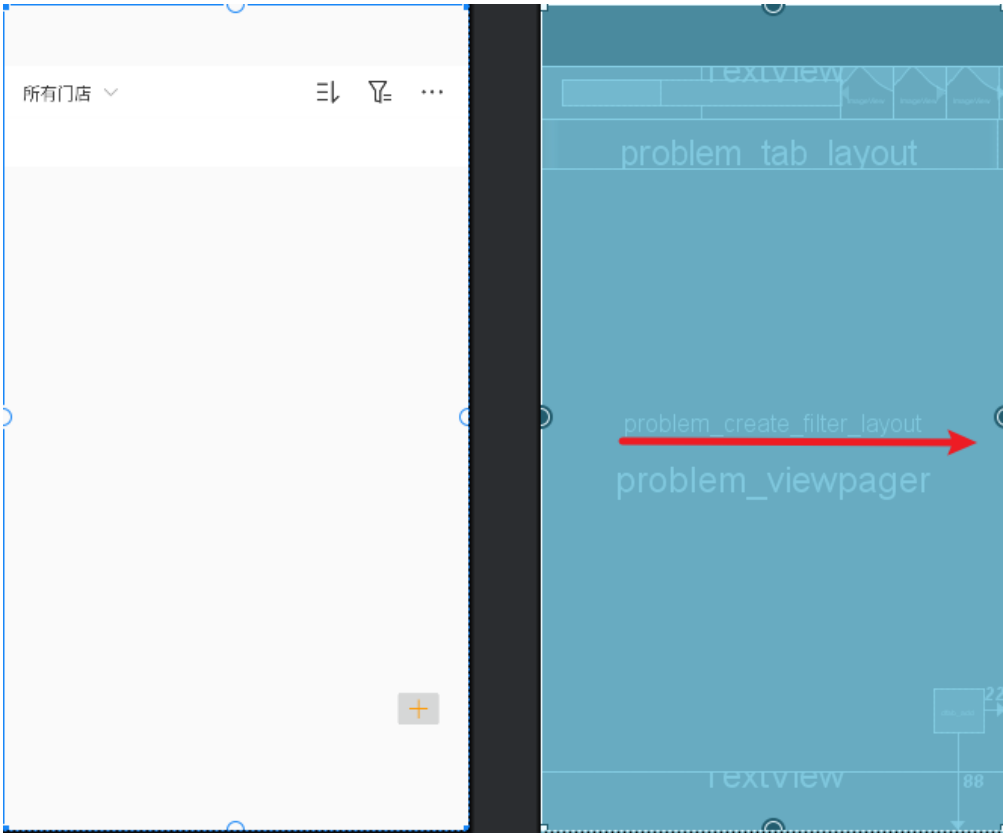
- 我们点击工作台页面中的select_btn，也就是筛选按钮，可以进入到这个Activity中
  - ```
    binding.problemSelectBtn.setOnClickListener {
                isShopRefresh = false
                //弹出筛选
                animateReveal(binding.problemCreateFilterLayout, true)
                WDZBuriedPointManager.instance.addPoint(
                    requireActivity(),
                    StatisticModuleNameId.BP_TODO_FILTER.pageId,
                    StatisticModuleNameId.BP_TODO_FILTER.pageName,
                    BuriedPointEventType.Click
                )
            }
    ```
  - **作用**：通过 **CircularReveal 动画** 展开筛选面板（`problemCreateFilterLayout`），动画开始时通过回调显示筛选 Fragment（`filterFragment`）。
- filterFragment显示在哪里？我没看到显示他的逻辑啊。
  - 工作台Fragment内部有以下逻辑：创建filterFragment实例，挂载到对应的FrameLayout中，然后隐藏起来。
    - ```
      filterFragment = ProblemFilterFragment.getInstance(
                  jsonList,
                  allList,
                  titleList,
      ```

```
            object : IProblemActionCallback {
                override fun onCommit() {

animateReveal(binding.problemCreateFilterLayout, false)
                    fragments[location].refreshProblemList(
                        createOrder,
                        commonOrder, priorityOrder,
expirationOrder, filterFragment!!.allList
                    )
                }

                override fun onDismiss() {

animateReveal(binding.problemCreateFilterLayout, false)
                }
            })
        addFragment(R.id.problem_create_filter_layout,
filterFragment, false)
        hideFragment(filterFragment)
```

- 对应挂载的视图位置
  - ```
    <FrameLayout
            android:id="@+id/problem_create_filter_layout"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:focusable="true"
            android:focusableInTouchMode="true"/>
    ```
  - 
    ```

```
<View
    android:layout_width="0dp"
    android:layout_weight="1"
    android:layout_height="@dimen/dp_0_5"
    android:visibility="invisible"/>
```

03.修改思路

3.1 保存筛选视图逻辑

- 需求图片:
  - 

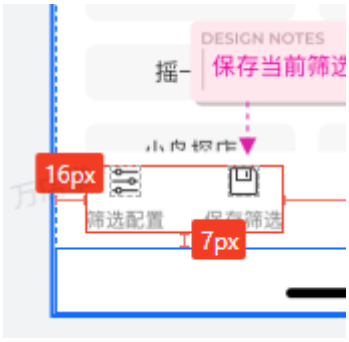
- 原代码:
  - ```
    <LinearLayout
           android:id="@+id/problem_filter_bottom_layout"
           android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:layout_alignParentBottom="true"
           android:paddingTop="@dimen/dp_7"
           android:paddingBottom="@dimen/dp_7"
           android:paddingLeft="@dimen/dp_16"
           android:paddingRight="@dimen/dp_16">
           <LinearLayout
               android:id="@+id/ll_modify"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:orientation="vertical"
               android:gravity="center">
               <ImageView
                   android:id="@+id/modify_iv"
                   android:layout_width="wrap_content"
                   android:layout_height="wrap_content"
                   android:layout_gravity="center"
                   android:padding="@dimen/dp_5"
                   android:src="@drawable/ico_configure_min" />
               <TextView
                   android:id="@+id/modify_tv"
                   android:layout_width="wrap_content"
                   android:layout_height="wrap_content"
                   android:layout_gravity="center_horizontal"
                   android:text="@string/filter_setting_title"
                   android:textColor="@color/color_FF7F7F7F"
                   android:textSize="@dimen/sp_10" />
           </LinearLayout>

           <com.ovopark.widget.BorderTextView
               android:id="@+id/problem_filter_reset"
               android:layout_width="@dimen/dp_84"
               android:layout_height="wrap_content"
               android:layout_marginLeft="@dimen/dp_80"
    ```
    ```

```
            android:gravity="center"
            android:text="@string/problem_reset"
            android:textColor="@color/main_text_black_color"
            android:textSize="@dimen/medium_text"
            app:contentBackColor="@color/transparent"
            app:strokeWidth="@dimen/dp_1"
            app:strokeColor="@color/color_FFE5E5E5"
            app:cornerRadius="@dimen/dp_6"
            android:paddingTop="@dimen/dp_7"
            android:paddingBottom="@dimen/dp_7"/>

        <com.ovopark.widget.BorderTextView
            android:id="@+id/problem_filter_commit"
            android:layout_width="@dimen/dp_84"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text="@string/confirm"
            android:textColor="@color/white"
            android:textSize="@dimen/medium_text"
            android:layout_marginLeft="@dimen/dp_12"
            app:contentBackColor="@color/main_text_yellow_color"
            app:strokeWidth="@dimen/dp_1"
            app:strokeColor="@color/color_FFE5E5E5"
            app:cornerRadius="@dimen/dp_6"
            android:paddingTop="@dimen/dp_7"
            android:paddingBottom="@dimen/dp_7"/>
    </LinearLayout>


    <com.ovopark.widget.recycleview.MaxHeightRecyclerView
        android:id="@+id/problem_filter_recyclerview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_above="@id/problem_filter_bottom_layout"
        android:layout_below="@id/ll_layout"
        android:layout_marginBottom="0dp"
        android:background="@color/white"
        android:paddingLeft="@dimen/dp_3"
        android:paddingTop="@dimen/dp_10"
        android:paddingRight="@dimen/dp_12"
        android:paddingBottom="@dimen/dp_15" />
```

- 修改预案：
  - 方法一：采用LinearLayout包裹筛选配置和保存筛选，通过权重进行分割。
    - 优点：符合代办单中的视图逻辑，两个作为一个整体，只需要考虑外部的布局，内部直接通过权重划分
    - 

        - 缺点：增加视图层级，点击事件要多走一层，绘制也要多走一层，降低了整体性能，视图优化有一个就是尽量降低视图层级，扁平化视图。
    - 方法二：保存筛选单独做一个LinearLayout，通过marginLeft和筛选配置分割。
        - 优点：
            - **减少视图层级**：相比方法一减少一层`ViewGroup`，测量，布局和点击事件的效率都有所上升，同时也避免父容器的中间层事件拦截。
            - 保存功能与筛选配置解耦，后续修改`marginLeft`即可调整间距，无需调整权重
        - 缺点：性能仍然是嵌套布局。并且比较老，没有使用上现代化的工具。
    - 方法三：采用新式的ConstraintLayout布局，重构底部导航栏的布局结构
        - 优点：通过链式约束（Chain）替代嵌套，减少层级至1层。app:layout_constraintHorizontal_bias可动态控制间距比例，无需硬编码marginLeft
        - 缺点：重构布局，可能导致代码评审风险与其他未知错误，可能还需要同步更新单元测试
- 修改方案：作为新人，应尽量避免代码重构，减少风险，同时写出高性能代码。综合考虑，采用方法二。方法三留档，后续熟悉后可以重构优化。
- 实际操作：
    - xml布局中新增如下：采用dp，而非px
        - ```
          <LinearLayout
                      android:id="@+id/ll_save"
                      android:layout_width="wrap_content"
                      android:layout_height="wrap_content"
                      android:orientation="vertical"
                      android:gravity="center"
                      android:layout_marginLeft="@dimen/dp_17">
                      <ImageView
                          android:id="@+id/save_iv"
                          android:layout_width="wrap_content"
                          android:layout_height="wrap_content"
                          android:layout_gravity="center"
                          android:padding="@dimen/dp_5"
                          android:src="@drawable/ico_save_min" />
                      <TextView
                          android:id="@+id/save_tv"
                          android:layout_width="wrap_content"
                          android:layout_height="wrap_content"
                          android:layout_gravity="center_horizontal"
                          android:text="@string/save_setting_title"
                          android:textColor="@color/color_FF7F7F7F"
                          android:textSize="@dimen/sp_10" />
                  </LinearLayout>
          ```
    - 从UI提供的网址下载对应的webp图片，存放到res中
    - 在string中增加对应的文字。
- 实际结果：小米8，Android10
          ```

3.2 保存筛选点击弹窗逻辑

- 需求图片：
  

- 修改预案：
  - 方法一：setonTouchListener
    - 优点：触发较快，考量事件分发机制，我们再dispatchTouchEvent的时候，我们会优先监听是否有onTouchListen并执行，其响应速度甚至优先于onTouchEvent
    - 缺点：触发频率高，每次MotionEvent事件到来，无论类型是Down，up都会触发，性能损耗较大。
  - 方法二：参考筛选配置的点击逻辑，针对iv_save添加onClickListener。
    - 优点：触发频率低，性能消耗小。
    - 缺点：触发慢于setonTouchListener，会在Up事件时触发。
  - 方法三：优化方法二，iv_save属于ll_save，是他的子视图。考量ll_save，其包含iv和tv，tv无需修改，也不需要点击事件，也就是iv和tv无需分别维护一个点击事件，因此我们可以针对ll_save添加onClickListener。

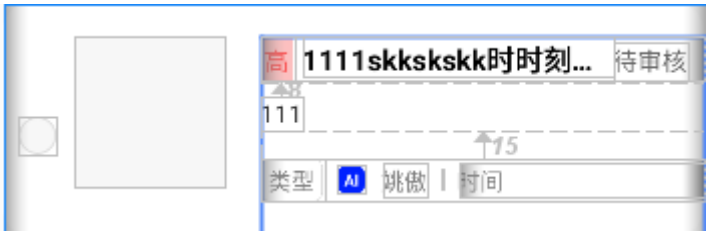
        - 优点：相对于方法二，点击事件的分发和处理层级更高了一层，性能有一定优化。
- 提问：
    - 是否要增加防抖控制：避免快速重复点击导致的多次弹窗？
    - 我的弹窗逻辑该如何写？参考哪些地方？
- 解答：
    - 防抖：如果采用：
        - ```
          //保存筛选逻辑
          binding.llSave.setOnClickListener {
              if (!CommonUtils.isFastRepeatClick(600)) { // 设置防抖间隔
                  // 实际业务逻辑
                  showToast("点击生效")
              }
          }

          @JvmStatic
          fun isFastRepeatClick(duration: Long): Boolean {
                  val time = System.currentTimeMillis()
                  val deltaTime = time - lastClickTime
                  if (0 < deltaTime && deltaTime < duration) {
                      return true
                  }
                  lastClickTime = time
                  return false
              }
          ```
        - 通过时间戳差值判断连续点击（`System.currentTimeMillis()`），当两次点击间隔小于`duration`时返回`true`
        - 当两次点击大于这个时间时，返回false，通过！返回true，执行内部弹窗逻辑。
    - 那么解答问题：有必要采用防抖吗？我点击后就会产生弹窗，连续点击也不会产生连续的弹窗啊。
    - 弹窗：自己写一个
        - https://juejin.cn/post/6844904197280923655

3.3 数据管理中子条目视图逻辑修改

- 需求图片：其实就是筛选条件不一样，拿全部的。带个30天的参数。
    - 

- 情况对比分析：
    - 当前的子条目视图逻辑：
    - 
          ```

    - 需要更改为：ItemProblemChangeOBinding
        - 

    - 存在的问题：
        - 两者元素存在很多差异，数据从哪里拿？，通过 Presenter？借助 requestDataRefresh？
- 确定和取消去掉数据库的接口，自定义Dialog