转载请注明链接: https://blog.csdn.net/feather_wch/article/details/88703295

> 从实战场景使用DataBinding才是掌握DataBinding的最好方法。

DataBinding实战场景

版本号:2019-03-20(23:23)

RecyclerView和DataBinding结合使用

1、Adapter

> 1. 重点处理 `onCreateViewHolder` 和 `onBindViewHolder`
> 2. 该Adapter配合RV的使用和寻常的用法一致

```java
public class UserListRvAdapter extends RecyclerView.Adapter {

    private List<User> mUserList;
    private Context mContext;

    public UserListRvAdapter(Context context, List<User> userList){
        mUserList = userList;
        mContext = context;
    }

    @NonNull
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, int i) {
        LayoutInflater inflater = LayoutInflater.from(mContext);
        UserItemBinding binding = DataBindingUtil.inflate(inflater, R.layout.recyclerview_user_
        return new UserViewHolder(binding.getRoot());
    }

    @Override
    public void onBindViewHolder(@NonNull RecyclerView.ViewHolder viewHolder, int position) {
        // 获取到绑定
        UserItemBinding binding = DataBindingUtil.getBinding(viewHolder.itemView);
        binding.setUser(mUserList.get(position));
        binding.executePendingBindings();
    }

    @Override
    public int getItemCount() {
        return mUserList != null ? mUserList.size() : 0;
    }


    public class UserViewHolder extends RecyclerView.ViewHolder{
        public UserViewHolder(@NonNull View itemView) {
            super(itemView);
        }
    }

    @BindingAdapter("imageurl")
    public static void loadImage(ImageView view, String url){
        Glide.with(view).load(url).into(view);
    }
}
```

2、RecyclerView的Item布局

```xml
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data class="com.hao.databinding.UserItemBinding">
        <variable
            name="user"
            type="com.hao.architecture.User"/>

    </data>
    <android.support.constraint.ConstraintLayout
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        tools:context="com.hao.architecture.DailyPlanActivity">

        <ImageView
            android:id="@+id/user_head_img"
            android:layout_width="150dp"
            android:layout_height="150dp"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toStartOf="@+id/user_name_txt"
            app:imageurl="@{user.headurl}"/>

        <TextView
            android:id="@+id/user_name_txt"
            android:layout_width="0dp"
            android:layout_height="0dp"
            app:layout_constraintStart_toEndOf="@id/user_head_img"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            android:textSize="30dp"
            android:gravity="center"
            android:text="@{user.name}">
        </TextView>

    </android.support.constraint.ConstraintLayout>

</layout>
```

ViewModel和RecyclerView

1、Activity中使用RecyclerView，并监听List的数据变更，然后更新RV的数据

> 1. 给Adapter设置点击事件
> 2. 调用Adapter的setUserList()在内部通过DiffUtil比较数据差异进行数据更新

```java
public class DailyPlanActivity extends AppCompatActivity {

    UserListRvAdapter mAdapter;
    ActivityDailyPlanBinding mBinding;
    private UserViewModel mUserViewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mBinding = DataBindingUtil.setContentView(this, R.layout.activity_daily_plan);

        // 1. 用户数据的ViewModel
        mUserViewModel = ViewModelProviders.of(this).get(UserViewModel.class);
        // 2. ViewModel绑定生命周期，在onCreate中自动load数据
        getLifecycle().addObserver(mUserViewModel);
        mUserViewModel.getUserList().observe(this, new Observer<List<User>>() {
            @Override
            public void onChanged(@Nullable List<User> userList) {
                // 内部利用DiffUtil比对，并且更新数据
                mAdapter.setUserList(userList);
            }
        });
        mUserViewModel.getSelectedUser().observe(this, new Observer<User>() {
            @Override
            public void onChanged(@Nullable User user) {
                // 选中了一个Item
                mBinding.currentUserTxt.setText(user.getName());
            }
        });
        // 3. RecyclerView相关初始化
        mAdapter = new UserListRvAdapter(this, mUserViewModel.getUserList().getValue());
        mAdapter.setUserViewModel(mUserViewModel);
        mBinding.userRecyclerview.setLayoutManager(new LinearLayoutManager(this, LinearLayoutMa
        mBinding.userRecyclerview.setAdapter(mAdapter);
    }
}
```

2、Adapter

```
public class UserListRvAdapter extends RecyclerView.Adapter {

    private List<User> mUserList;
    private Context mContext;

    private UserViewModel mUserViewModel = null;

    public UserListRvAdapter(Context context, List<User> userList){
        mUserList = userList;
        mContext = context;
    }

    @NonNull
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, int i) {
        LayoutInflater inflater = LayoutInflater.from(mContext);
        UserItemBinding binding = DataBindingUtil.inflate(inflater, R.layout.recyclerview_user_
        return new UserViewHolder(binding.getRoot());
    }

    @Override
    public void onBindViewHolder(@NonNull RecyclerView.ViewHolder viewHolder, int position) {
        // 获取到绑定
        UserItemBinding binding = DataBindingUtil.getBinding(viewHolder.itemView);
        binding.setUser(mUserList.get(position));
        binding.executePendingBindings();
        binding.getRoot().setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(mUserViewModel != null){
                    mUserViewModel.getSelectedUser().postValue(mUserList.get(position));
                }
            }
        });
    }

    @Override
    public int getItemCount() {
        return mUserList != null ? mUserList.size() : 0;
    }

    public void setUserViewModel(UserViewModel userViewModel) {
        mUserViewModel = userViewModel;
    }

    public class UserViewHolder extends RecyclerView.ViewHolder{
        public UserViewHolder(@NonNull View itemView) {
            super(itemView);
        }
    }

    public void setUserList(final List<User> userList) {
        if (mUserList == null) {
            mUserList = userList;
            notifyItemRangeInserted(0, userList.size());
        } else {
            DiffUtil.DiffResult result = DiffUtil.calculateDiff(new DiffUtil.Callback() {
                @Override
                public int getOldListSize() {
                    return mUserList.size();
                }

                @Override
                public int getNewListSize() {
                    return userList.size();
                }

                @Override
                public boolean areItemsTheSame(int oldItemPosition, int newItemPosition) {
                    return mUserList.get(oldItemPosition).getAccount()
                            .equals(userList.get(newItemPosition).getAccount());
                }

                @Override
                public boolean areContentsTheSame(int oldItemPosition, int newItemPosition) {
                    User newUser = userList.get(newItemPosition);
                    User oldUser = mUserList.get(oldItemPosition);
                    return Objects.equals(newUser.getPassword(), oldUser.getPassword())
                            && Objects.equals(newUser.getHeadurl(), oldUser.getHeadurl())
                            && Objects.equals(newUser.getName(), oldUser.getName())
                            && newUser.getAge() == oldUser.getAge();
                }
            });
            mUserList = userList;
            result.dispatchUpdatesTo(this);
        }
    }

    @BindingAdapter("imageurl")
    public static void loadImage(ImageView view, String url){
        Glide.with(view).load(url).into(view);
    }
}
```

升级版-直接使用ViewModel、内部自动刷新数据

1、Activity中RV展示用户列表，点击某一个Item后，在Activity顶部展示选中的用户信息

```
public class DailyPlanActivity extends AppCompatActivity {

    UserListRvAdapter mAdapter;
    ActivityDailyPlanBinding mBinding;
    private UserViewModel mUserViewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mBinding = DataBindingUtil.setContentView(this, R.layout.activity_daily_plan);

        // 1. 用户数据的ViewModel
        mUserViewModel = ViewModelProviders.of(this).get(UserViewModel.class);
        mBinding.setUsermodel(mUserViewModel); // 绑定到Activity上
        // 2. ViewModel绑定生命周期，在onCreate中自动load数据
        getLifecycle().addObserver(mUserViewModel);
        // 3. RecyclerView相关初始化
        mAdapter = new UserListRvAdapter(this, mUserViewModel);
        mBinding.userRecyclerview.setLayoutManager(new LinearLayoutManager(this, LinearLayoutMa
        mBinding.userRecyclerview.setAdapter(mAdapter);
        // 4. 外部响应【点击事件】
        mUserViewModel.getSelectedUser().observe(this, new Observer<User>() {
            @Override
            public void onChanged(@Nullable User user) {
                // 1. 禁止这样做：RV的点击Item事件会更改SelectedUser，Activity根据SelectedUser展示
//                mUserViewModel.getSelectedUser().setValue(user);
                // 2. 直接刷新显示即可
                mBinding.setUsermodel(mUserViewModel); // 触发刷新
            }
        });
    }
}
```

布局

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable
            name="usermodel"
            type="com.hao.architecture.UserViewModel"/>

    </data>
    <android.support.constraint.ConstraintLayout
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context="com.hao.architecture.DailyPlanActivity">

        <TextView
            android:id="@+id/current_user_txt"
            android:layout_width="0dp"
            android:layout_height="30dp"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            android:textSize="20dp"
            android:gravity="center"
            android:text="@{usermodel.getSelectedUser().getValue().getName()}"
            tools:text="图灵"/>

        <android.support.v7.widget.RecyclerView
            android:id="@+id/user_recyclerview"
            android:layout_width="0dp"
            android:layout_height="0dp"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintTop_toBottomOf="@id/current_user_txt">
        </android.support.v7.widget.RecyclerView>

    </android.support.constraint.ConstraintLayout>

</layout>
```

2、RecyclerViewAdapter

> 1. 构造时就注册观察数据列表的改变，并在改变后更新列表

```java
public class UserListRvAdapter extends RecyclerView.Adapter {

    private Context mContext;
    private List<User> mOldUserList;
    private UserViewModel mUserViewModel;

    /**=============================================================
     * @功能 1、构造方法。
     *   1. 存放User列表的LiveData进行观察
     *   2. 数据改变后，利用ViewModel中存放的LiveData<List<User>>刷新展示的数据
     * @param fragmentActivity LifecycleOwner-用于LiveData注册监听
     * @param userViewModel ViewModel
     *===================================================================*/
    public UserListRvAdapter(FragmentActivity fragmentActivity, @NonNull UserViewModel userView
        mContext = fragmentActivity;
        mUserViewModel = userViewModel;
        mUserViewModel.getUserList().observe(fragmentActivity, new Observer<List<User>>() {
            @Override
            public void onChanged(@Nullable List<User> userList) {
                // 内部利用DiffUtil比对，并且更新数据
                notifyViewModelDataChanged();
            }
        });
    }

    /**=======================================================
     * @功能 2、利用DataBinding加载出布局，构造出ViewHolder
     *=========================================================*/
    @NonNull
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, int i) {
        LayoutInflater inflater = LayoutInflater.from(mContext);
        UserItemBinding binding = DataBindingUtil.inflate(inflater, R.layout.recyclerview_user_
        return new UserViewHolder(binding.getRoot());
    }

    /**=======================================================
     * @功能 3、onBindViewHolder()方法中真正绑定数据
     *=========================================================*/
    @Override
    public void onBindViewHolder(@NonNull RecyclerView.ViewHolder viewHolder, int position) {
        /**=====================================
         * 1、【重新绑定数据】
         *=================================================*/
        UserItemBinding binding = DataBindingUtil.getBinding(viewHolder.itemView);
        // 1. 绑定User
        User user = mUserViewModel.getUserList().getValue().get(position);
        binding.setUser(user);
        // 2. 立即执行绑定
        binding.executePendingBindings();
        /**=================================
         * 2、【设置点击事件】
         *       1. 通过ViewModel中的某个LiveData将点击的Item发给外部的Observer
         *       2. 外部接收到事件后，进行相应处理
         *=====================================*/
        binding.getRoot().setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mUserViewModel.getSelectedUser().postValue(user);
            }
        });
    }

    /**============================
     * @功能 4、ViewHolder不再需要手动find控件
     *============================*/
    public class UserViewHolder extends RecyclerView.ViewHolder{
        public UserViewHolder(@NonNull View itemView) {
            super(itemView);
        }
    }

    /**================================
     * @功能 5、利用DiffUtil更新RV数据
     *     1. mOldUserList仅仅是为了比较前后差异
     *     2. 数据一直存在ViewModel中
     *================================*/
    public void notifyViewModelDataChanged() {
        List<User> userList = mUserViewModel.getUserList().getValue();
        if (mOldUserList == null) {
            mOldUserList = userList;
            notifyItemRangeInserted(0, userList.size());
        } else {
            DiffUtil.DiffResult result = DiffUtil.calculateDiff(new DiffUtil.Callback() {
                @Override
                public int getOldListSize() {
                    return mOldUserList.size();
                }

                @Override
                public int getNewListSize() {
                    return userList.size();
                }

                @Override
                public boolean areItemsTheSame(int oldItemPosition, int newItemPosition) {
                    return mOldUserList.get(oldItemPosition).getAccount()
                            .equals(userList.get(newItemPosition).getAccount());
                }

                @Override
                public boolean areContentsTheSame(int oldItemPosition, int newItemPosition) {
                    User newUser = userList.get(newItemPosition);
                    User oldUser = mOldUserList.get(oldItemPosition);
                    return Objects.equals(newUser.getPassword(), oldUser.getPassword())
                            && Objects.equals(newUser.getHeadurl(), oldUser.getHeadurl())
                            && Objects.equals(newUser.getName(), oldUser.getName())
                            && newUser.getAge() == oldUser.getAge();
                }
            });
            mOldUserList = userList;
            result.dispatchUpdatesTo(this);
        }
    }

    /**===========================
     * @功能 6、数据元素数量
     *===========================*/
    @Override
    public int getItemCount() {
        if(mUserViewModel == null){
            return 0;
        }
        List<User> users = mUserViewModel.getUserList().getValue();
        return users != null ? users.size() : 0;
    }

    /**========================================
     * @功能 7、ImageView利用imageurl属性加载网络图片
     *========================================*/
    @BindingAdapter("imageurl")
    public static void loadImage(ImageView view, String url){
        Glide.with(view).load(url).into(view);
    }
}
```

> 布局

```xml
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data class="com.hao.databinding.UserItemBinding">
        <variable
            name="user"
            type="com.hao.architecture.User"/>

    </data>
    <android.support.constraint.ConstraintLayout
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        tools:context="com.hao.architecture.DailyPlanActivity">

        <ImageView
            android:id="@+id/user_head_img"
            android:layout_width="150dp"
            android:layout_height="150dp"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toStartOf="@+id/user_name_txt"
            app:imageurl="@{user.headurl}"/>

        <TextView
            android:id="@+id/user_name_txt"
            android:layout_width="0dp"
            android:layout_height="0dp"
            app:layout_constraintStart_toEndOf="@id/user_head_img"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            android:textSize="30dp"
            android:gravity="center"
            android:text="@{user.name}">
        </TextView>
    </android.support.constraint.ConstraintLayout>
</layout>
```

参考资料

1. update RecyclerView with Android LiveData
2. googlesamples/android-architecture-components