DiffUtil Demystified

Jon Hancock

This talk has a text version available at goo.gl/yXXokW

The old way

```
public void updateItems (List<Item> newItems) {
    items.clear();
    items.addAll(newItems);
    notifyDataSetChanged();
}
```

Free Animations

notifyItemRemoved (int position)

notifyItemChanged (int position)

notifyItemInserted (int position)

notifyItemMoved (int fromPosition, int toPosition)

Fine grained changes

notifyItemChanged (int position, Object payload)

onBindViewHolder (VH holder, int position, List<Object> payloads)

It’s tedious to call all those methods

-- Jon Hancock 2017/02/21... literally just a second ago

Enter DiffUtil

```
public int getOldListSize ()

public int getNewListSize ()

public boolean areItemsTheSame (int oldItemPosition, int newItemPosition)

public boolean areContentsTheSame (int oldItemPosition, int newItemPosition)
```

```
class DiffCb extends DiffUtil.Callback {
    private final List<Item> oldItems;
    private final List<Item> newItems;

    public DiffCb(List<Item> oldItems, List<Item> newItems) {
        this.oldItems = oldItems;
        this.newItems = newItems;
    }
    ...
}
```

```
class DiffCb extends DiffUtil.Callback {
    ...
    public int getOldListSize() {
        return oldItems.size();
    }
    public int getNewListSize() {
        return newItems.size();
    }
    ...
}
```

```
class DiffCb extends DiffUtil.Callback {
    ...
    public boolean areItemsTheSame(int oldItemPosition, int newItemPosition) {
        return oldItems.get(oldItemPosition).equals(newItems.get(newItemPosition));
    }
    public boolean areContentsTheSame(int oldItemPosition, int newItemPosition) {
        return true;
    }
}
```

Apply the results

```
public void updateItems(final List<Item> newItems) {
    List<Item> oldItems = new ArrayList<>(items);
    DiffUtil.DiffResult diffResult =
                    DiffUtil.calculateDiff(new DiffCb(oldItems,newItems));
    items.clear();
    items.addAll(newItems);
    diffResult.dispatchUpdatesTo(this);
}
```

Get Threading Right

Delegate threading to a new method

```
public class MyAdapter extends RecyclerView.Adapter {
    protected List<Item> items = new ArrayList<>();

    public void updateItems(final List<Item> newItems) {
        updateItemsInternal(newItems);
    }
}
```

Diff in background, apply on main thread

```
void updateItemsInternal(final List<Item> newItems) {
    final List<Item> oldItems = new ArrayList<>(this.items);
    final Handler handler = new Handler();
    new Thread(new Runnable() {
        public void run() {
            final DiffUtil.DiffResult diffResult = DiffUtil.calculateDiff(new DiffCb(oldItems, newItems));
            handler.post(new Runnable() {
                public void run() {  applyDiffResult(newItems, diffResult); }
            });
        }
    }).start();
}
```

Apply the updates

```
public class MyAdapter extends RecyclerView.Adapter {
    protected void applyDiffResult(List<Item> newItems, DiffUtil.DiffResult diffResult) {
        dispatchUpdates(newItems, diffResult);
    }
    protected void dispatchUpdates(List<Item> newItems, DiffUtil.DiffResult diffResult) {
        items.clear();
        items.addAll(newItems);
        diffResult.dispatchUpdatesTo(this);
    }
}
```

Concurrent Updates

Handle concurrent updates

- First update wins, and others are discarded
- Latest update wins, and intermediates are discarded
- Queue them up, and apply them in order

First Wins

```
public class FirstWinsAdapter extends RecyclerView.Adapter {
    protected List<Item> items = new ArrayList<>();
    boolean operationPending;
    public void updateItems(final List<Item> newItems) {
        if (operationPending) { return; }
        operationPending = true;
        updateItemsInternal(newItems);
    }
    protected void applyDiffResult(List<Item> newItems, DiffUtil.DiffResult diffResult) {
        dispatchUpdates(newItems, diffResult);
        operationPending = false;
    }
}
```

Latest Wins

```
public class LatestWinsAdapter extends RecyclerView.Adapter {
    protected List<Item> items = new ArrayList<>();
    private Deque<List<Item>> pendingUpdates = new ArrayDeque<>();

    public void updateItems(final List<Item> newItems) {
        pendingUpdates.push(newItems);
        if (pendingUpdates.size() > 1) { return;}
        updateItemsInternal(newItems);
    }
}
```

Latest Wins

```
public class LatestWinsAdapter extends RecyclerView.Adapter {
    protected List<Item> items = new ArrayList<>();
    private Deque<List<Item>> pendingUpdates = new ArrayDeque<>();
    protected void applyDiffResult(List<Item> newItems, DiffUtil.DiffResult diffResult) {
        pendingUpdates.remove(newItems);
        dispatchUpdates(newItems, diffResult);
        if (pendingUpdates.size() > 0) {
            List<Item> latest = pendingUpdates.pop();
            pendingUpdates.clear();
            updateItemsInternal(latest);
        }
    }
}
```

Queue them up

```
public class QueueAdapter extends RecyclerView.Adapter {
    protected List<Item> items = new ArrayList<>();

    private Queue<List<Item>> pendingUpdates = new ArrayDeque<>();
    public void updateItems(final List<Item> newItems) {
        pendingUpdates.add(newItems);
        if (pendingUpdates.size() > 1) {return;}
        updateItemsInternal(newItems);
    }
}
```

Queue them up

```
public class QueueAdapter extends RecyclerView.Adapter {
    protected void applyDiffResult(List<Item> newItems, DiffUtil.DiffResult diffResult) {
        pendingUpdates.remove();
        dispatchUpdates(newItems, diffResult);
        if (pendingUpdates.size() > 0) {
            updateItemsInternal(pendingUpdates.peek());
        }
    }
}
```

Edge Cases

Questions?

@jonfhancock