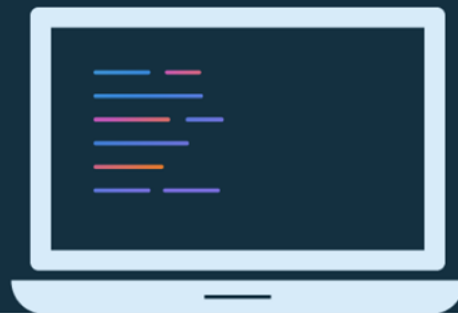




Bài học 10: Trường hợp sử dụng nâng cao của RecyclerView



Giới thiệu về bài học này

Bài học 10: Trường hợp sử dụng nâng cao của RecyclerView

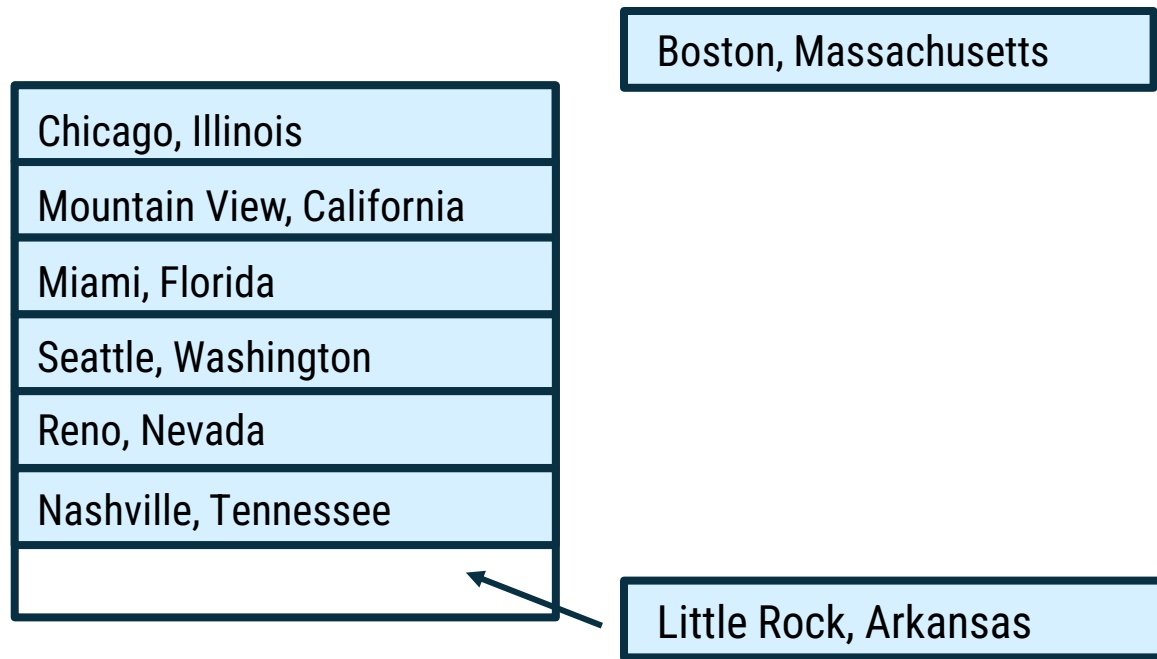
- [Tóm tắt về RecyclerView](#)
- [Liên kết nâng cao](#)
- [Nhiều loại chế độ xem mục](#)
- [Tiêu đề](#)
- [Bố cục lưới](#)
- [Tóm tắt](#)

Tóm tắt về RecyclerView

Tổng quan về RecyclerView

- Tiện ích để hiển thị danh sách dữ liệu
- "Tái chế" (tái sử dụng) các chế độ xem mục để cuộn hiệu quả hơn
- Có thể chỉ định bố cục mục danh sách cho từng mục trong tập dữ liệu
- Hỗ trợ hoạt ảnh và quá trình chuyển đổi

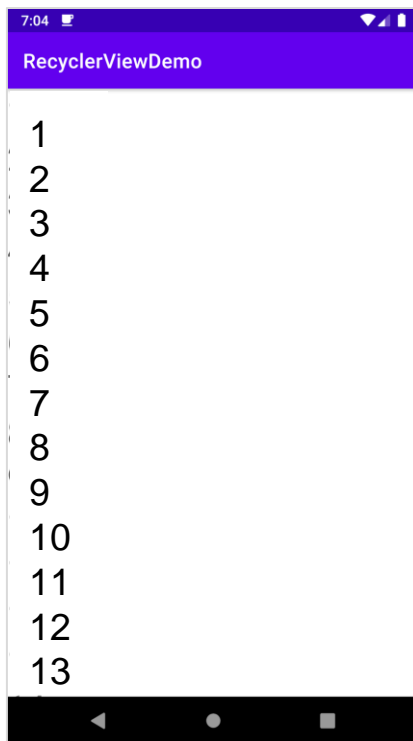
Xem tính năng tái chế trong RecyclerView



Nếu mục được cuộn ngoài màn hình, mục đó sẽ không bị hủy bỏ. Mục được đưa vào một nhóm để tái chế.

`onBindViewHolder` liên kết chế độ xem với các giá trị mới, sau đó, chế độ xem được chèn lại vào danh sách.

Ứng dụng RecyclerViewDemo



Bộ chuyển đổi cho RecyclerViewDemo

```
class NumberListAdapter(var data: List<Int>):  
    RecyclerView.Adapter<NumberListAdapter.IntViewHolder>() {  
    class IntViewHolder(val row: View): RecyclerView.ViewHolder(row) {  
        val textView = row.findViewById<TextView>(R.id.number)  
    }  
}
```

Các hàm cho RecyclerViewDemo

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):  
    IntViewHolder {  
    val layout = LayoutInflater.from(parent.context)  
        .inflate(R.layout.item_view, parent, false)  
    return IntViewHolder(layout)  
}
```

```
override fun onBindViewHolder(holder: IntViewHolder, position: Int) {  
    holder.textView.text = data.get(position).toString()  
}
```


Đặt bộ chuyển đổi lên trên RecyclerView

Trong tệp MainActivity.kt:

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
  
    val rv: RecyclerView = findViewById(R.id.rv)  
    rv.layoutManager = LinearLayoutManager(this)  
  
    rv.adapter = NumberListAdapter(IntRange(0,100).toList())  
}
```

Làm cho các mục trong danh sách có thể nhấp được

Trong tệp `NumberListAdapter.kt`:

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): IntViewHolder{
    val layout = LayoutInflater.from(parent.context).inflate(R.layout.item_view,
        parent, false)
    val holder = IntViewHolder(layout)
    holder.row.setOnClickListener {
        // Do something on click
    }
    return holder
}
```

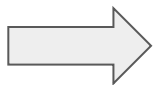
ListAdapter

- `RecyclerView.Adapter`
 - Hủy bỏ dữ liệu giao diện người dùng trong mỗi lần cập nhật
 - Có thể tốn kém và lãng phí
- `ListAdapter`
 - Tính toán sự khác biệt giữa nội dung đang hiển thị và nội dung cần hiển thị
 - Các thay đổi được tính toán trên một luồng trong nền

Sắp xếp bằng RecyclerView.Adapter

Trạng thái bắt đầu

1
5
2
6
3
7
4
8



8 lượt xóa



8 lượt chèn

1
2
3
4
5
6
7
8



16 thao tác:
8 lượt xóa
8 lượt chèn

1
2
3
4
5
6
7
8

Trạng thái kết thúc

Sắp xếp bằng ListAdapter

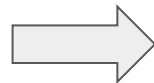
Trạng thái bắt đầu

1
5
2
6
3
7
4
8



3 lượt chèn
3 lượt xóa

1
5
2
6
3
7
4
5
6
7
8



6 thao tác:
3 lượt chèn
3 lượt xóa

Trạng thái kết thúc

1
2
3
4
5
6
7
8

Ví dụ về ListAdapter

```
class NumberListAdapter: ListAdapter<Int,  
    NumberListAdapter.IntViewHolder>(RowItemDiffCallback()) {  
  
    class IntViewHolder(val row: View): RecyclerView.ViewHolder(row) {  
        val textView = row.findViewById<TextView>(R.id.number)  
    }  
  
    ...
```

DiffUtil.ItemCallback

Xác định các phép biến đổi cần thiết để chuyển đổi giữa các danh sách

- `areContentsTheSame(oldItem: T, newItem: T): Boolean`
- `areItemsTheSame(oldItem: T, newItem: T): Boolean`

Ví dụ về DiffUtil.ItemCallback

```
class RowItemDiffCallback : DiffUtil.ItemCallback<Int>() {  
    override fun areItemsTheSame(oldItem: Int, newItem: Int): Boolean {  
        return oldItem == newItem  
    }  
  
    override fun areContentsTheSame(oldItem: Int, newItem: Int): Boolean {  
        return oldItem == newItem  
    }  
}
```


Liên kết nâng cao

ViewHolder và liên kết dữ liệu

```
class IntViewHolder private constructor(val binding: ItemViewBinding):  
    RecyclerView.ViewHolder(binding.root) {  
    companion object {  
        fun from(parent: ViewGroup): IntViewHolder {  
            val inflater = LayoutInflater.from(parent.context)  
            val binding = ItemViewBinding.inflate(inflater,  
                parent, false)  
            return IntViewHolder(binding)  
        }  
    }  
}
```

Dùng ViewHolder trong ListAdapter

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):  
    IntViewHolder {  
    return IntViewHolder.from(parent)  
}
```

```
override fun onBindViewHolder(holder: NumberListAdapter.IntViewHolder,  
    position: Int) {  
    holder.binding.num = getItem(position)  
}
```

Bộ chuyển đổi liên kết

Cho phép bạn liên kết một hàm với một thuộc tính trong tệp XML của mình

- Ghi đè hành vi hiện có của khung:

`android:text = "foo" → TextView.setText("foo")` được gọi

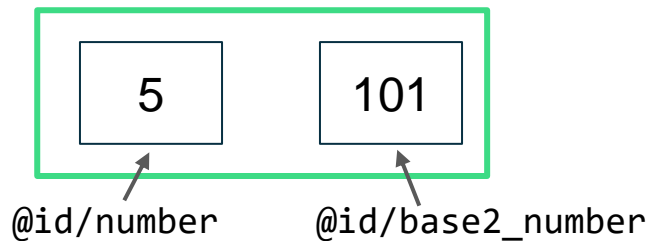
- Tạo các thuộc tính tùy chỉnh của riêng bạn:

`app:base2Number = "5" → TextView.setBase2Number("5")`
được gọi

Thuộc tính tùy chỉnh

Thêm một Chế độ xem văn bản khác trong bố cục mục danh sách sử dụng thuộc tính tùy chỉnh:

Ví dụ về mục danh sách



Thêm bộ chuyển đổi liên kết

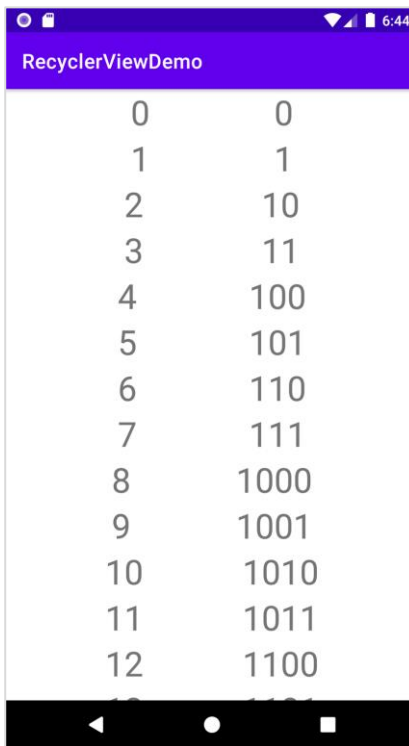
Khai báo bộ chuyển đổi liên kết:

```
@BindingAdapter("base2Number")  
fun TextView.setBase2Number(item: Int) {  
    text = Integer.toBinaryString(item)  
}
```

Trong tệp `NumberListAdapter.kt`:

```
override fun onBindViewHolder(holder: NumberListAdapter.IntViewHolder,  
    position: Int) {  
    holder.binding.num = getItem(position)  
    holder.binding.executePendingBindings()  
}
```

Ứng dụng RecyclerViewDemo cập nhật



0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100

Nhiều loại chế độ xem mục

Thêm loại mới cho chế độ xem mục

1. Tạo tệp XML mới cho bố cục mục danh sách.
2. Sửa đổi bộ chuyển đổi cơ bản để lưu giữ loại mới.
3. Ghi đè `getItemViewType` trong bộ chuyển đổi.
4. Tạo một lớp `ViewHolder` mới.
5. Thêm mã có điều kiện trong `onCreateViewHolder` và `onBindViewHolder` để xử lý loại mới.

Khai báo bố cục mới cho mục màu

```
<layout ...>
  <data>
    <variable
      name="color"
      type="android.graphics.Color" />
  </data>
  <androidx.constraintlayout.widget.ConstraintLayout ...>
    <TextView
      ...
      android:backgroundColor="@{color.toArgb()}" />
    <TextView
      ...
      android:text="@{color.toString()}" />
  </androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

Loại chế độ xem mới

- Bộ chuyển đổi cần biết về 2 loại chế độ xem mục:
 - Mục hiển thị số
 - Mục hiển thị màu

```
enum class ITEM_VIEW_TYPE { NUMBER, COLOR }
```

- Sửa đổi `getItemViewType()` để trả về loại thích hợp (ở dạng `Int`):

```
override fun getItemViewType(position: Int): Int
```

Ghi đè getItemViewType

Trong tệp `NumberListAdapter.kt`:

```
override fun getItemViewType(position: Int): Int {  
    return when(getItem(position)) {  
        is Int -> ITEM_VIEW_TYPE.NUMBER.ordinal  
        else -> ITEM_VIEW_TYPE.COLOR.ordinal  
    }  
}
```

Xác định ViewHolder mới

```
class ColorViewHolder private constructor(val binding: ColorItemViewBinding):  
    RecyclerView.ViewHolder(binding.root) {  
  
    companion object {  
        fun from(parent: ViewGroup): ColorViewHolder {  
            val inflater = LayoutInflater.from(parent.context)  
            val binding = ColorItemViewBinding.inflate(inflater,  
                parent, false)  
            return ColorViewHolder(binding)  
        }  
    }  
}
```

Cập nhật onCreateViewHolder()

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):  
    RecyclerView.ViewHolder {  
  
    return when(viewType) {  
        ITEM_VIEW_TYPE.NUMBER.ordinal -> IntViewHolder.from(parent)  
        else -> ColorViewHolder.from(parent)  
    }  
}
```


Cập nhật onBindViewHolder()

```
override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {  
    when (holder) {  
        is IntViewHolder -> {  
            holder.binding.num = getItem(position) as Int  
            holder.binding.executePendingBindings()  
        }  
        is ColorViewHolder -> {  
            holder.binding.color = getItem(position) as Color  
            holder.binding.executePendingBindings()  
        }  
    }  
}
```

Tiêu đề




Ví dụ về tiêu đề

 Món chính	
Bánh mì	5 đô la
Salad	3 đô la
Bánh sandwich	4 đô la
 Đồ uống	
Cà phê	2 đô la
Nước ngọt	1 đô la

- 2 loại chế độ xem mục:

- mục tiêu đề

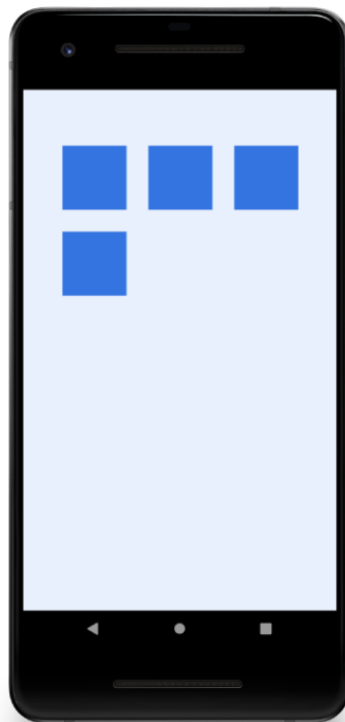
 Đồ uống

- mục thực đơn

Cà phê 2 đô la

Bố cục lưới

Danh sách và lưới



Chỉ định LayoutManager

Trong `onCreate()` của `MainActivity`, sau khi bạn có thông tin tham chiếu đến `RecyclerView`

- **Hiển thị danh sách bằng `LinearLayoutManager`:**
`recyclerView.layoutManager = LinearLayoutManager(this)`
- **Hiển thị lưới bằng `GridLayoutManager`:**
`recyclerView.layoutManager = GridLayoutManager(this, 2)`
- **Sử dụng trình quản lý bố cục khác (hoặc tạo trình quản lý của riêng bạn)**

GridLayoutManager

- Sắp xếp các mục trong lưới ở dạng bảng hàng và cột.
- Hướng có thể cuộn theo chiều dọc hoặc chiều ngang.
- Theo mặc định, mỗi mục chiếm 1 khoảng.
- Bạn có thể thay đổi số lượng khoảng của một mục (kích thước khoảng).

Đặt kích thước khoảng cho một mục

Tạo thực thể `SpanSizeLookup` và ghi đè `getSpanSize(position)`:

```
val manager = GridLayoutManager(this, 2)
manager.spanSizeLookup = object : GridLayoutManager.SpanSizeLookup() {
    override fun getSpanSize(position: Int): Int {
        return when (position) {
            0,1,2 -> 2
            else -> 1
        }
    }
}
```

Tóm tắt



Tóm tắt

Trong Bài học 10, bạn đã tìm hiểu cách:

- Dùng `ListAdapter` để giúp `RecyclerView` cập nhật danh sách hiệu quả hơn
- Tạo bộ chuyển đổi liên kết bằng logic tùy chỉnh để đặt các giá trị của Chế độ xem từ một thuộc tính XML
- Xử lý nhiều `ViewHolder` trong cùng một `RecyclerView` để hiển thị nhiều loại mục
- Dùng `GridLayoutManager` để hiển thị các mục ở dạng lưới
- Chỉ định kích thước khoảng của một mục trong lưới bằng `SpanSizeLookup`

Tìm hiểu thêm

- [Tạo một danh sách bằng RecyclerView](#)
- [RecyclerView](#)
- [ListAdapter](#)
- [Bộ chuyển đổi liên kết](#)
- [LayoutManager](#)
- [DiffUtil](#) và [ItemCallback](#)

Lộ trình

Thực hành những gì bạn đã học được bằng cách hoàn thành lộ trình này:

[Bài học 10: Trường hợp sử dụng nâng cao của RecyclerView](#)

