ANDROID NÂNG CAO

BÀI 2: Custom View

Nội dung

1. **Kiến trúc chung của View**
2. **Hiểu đúng về custom view**
3. **Các mức độ custom view**
   - Mức 1: tinh chỉnh chi tiết
   - Mức 2: viết lại một phần
   - Mức 3: viết lại phần lớn (giữ lại hành vi)
   - Mức 4: viết lại toàn bộ
4. **Ví dụ custom view mức 2: ProgressBar**
5. **Ví dụ custom view mức 3: Spinner**
6. **Ví dụ custom view mức 4: ClockView**

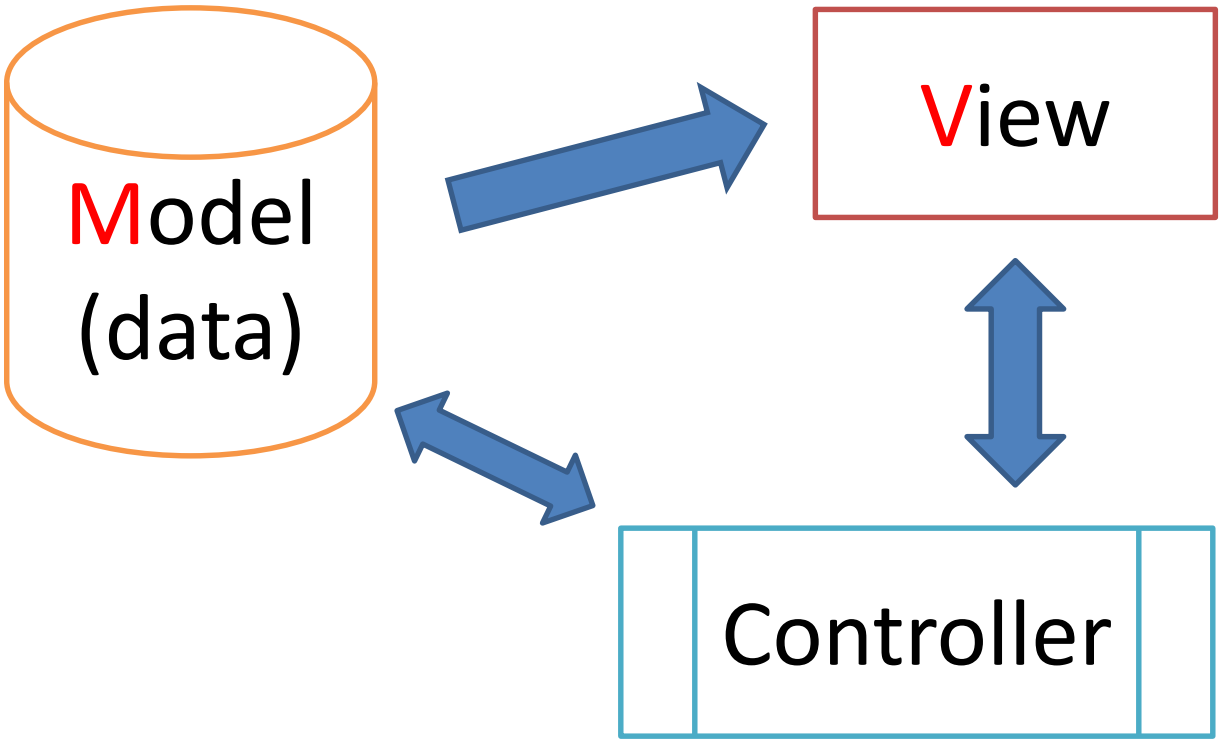
Phần 1

Kiến trúc chung của view

Kiến trúc chung của View

- **Thiết kế của View trong Android là một biến thể của cấu trúc MVC**
  - Model: dữ liệu cần thể hiện
  - View: phần màn hình giao diện người dùng
  - Controller: business logic của ứng dụng
- **Ví dụ về MVC với ListView**
  - Model: mảng các object (String)
  - View: mảng các view, mỗi view hiển thị 1 object
  - Controller: xử lý sự kiện khi người dùng scroll ListView, click vào từng view con,…

Kiến trúc chung của View



Kiến trúc chung của View

- **Tại sao lựa chọn MVC cho thiết kế giao diện?**
  - Giao diện thực chất gồm 3 phần riêng biệt: dữ liệu, minh họa và tương tác
  - Với cùng một loại dữ liệu, có nhiều cách minh họa khác nhau dẫn đến nhiều kiểu tương tác khác nhau
    - Bảng chấm công với chi tiết từng ngày làm việc của từng người
    - Bảng chấm công chỉ liệt kê số ngày làm việc của từng người
- **Kiến trúc MVC giúp lập trình viên có thể thay thế từng phần của class thay vì phải viết lại toàn bộ**
- **Thiết kế nhất quán giúp LTV nhanh chóng nắm vững việc sử dụng các thành phần UI tương tự**

Phần 2

Hiểu đúng về custom view

Hiểu đúng về custom view

- **Custom view là hoạt động bắt buộc khi xây dựng giao diện, không phải hoạt động đặc biệt**
- **Có nhiều mức độ custom view khác nhau:**
  1. Tinh chỉnh view đã có: màu, căn lề, nền,…
  2. Tinh chỉnh từng phần của view: ListView, Spinner cho phép ta thay đổi từng phần của view con
  3. Viết lại phần lớn view
  4. Viết lại hoàn toàn một view, tạo ra các loại sự kiện và trải nghiệm riêng của người dùng
- **Không nên lạm dụng custom view: với một vấn đề, sử dụng cấp độ càng thấp càng tốt**

Hiểu đúng về custom view

- **Thay đổi các thành phần trong MVC tác động như thế nào đến view?**
  - Thay đổi model: tùy theo yêu cầu của bài toán
  - Thay đổi view: sẽ thay đổi hình dạng của view
  - Thay đổi controller: thay đổi tương tác của view (thay đổi trải nghiệm người dùng)
- **Như vậy:**
  - Muốn thể hiện dữ liệu riêng: đổi Model + View
  - Muốn tạo view có hình ảnh mới: đổi View
  - Muốn có sự đặc biệt trong tương tác: đổi Controller

Phần 3

Các mức độ custom view

Custom view level 1

- **Level 1: chỉ điền thêm dữ liệu thiếu**
- **Một số thao tác khác như chỉnh vị trí, màu, lề,…**
- **Ví dụ tùy biến ListView bằng cách thêm dữ liệu:**

```
text.setAdapter(
    new ArrayAdapter<String>(
        // context của ArrayAdapter
        this,
        // chọn layout cho popup text
        android.R.layout.simple_list_item_1,
        // lấy string array ở XML
        getResources().getStringArray(R.array.listdata)
    )
);
```

Custom view level 2

- **Level 2: thay đổi một chút về thể hiện, giữ nguyên những trải nghiệm cơ bản**
- **Ví dụ tùy biến Spinner với một layout tự tạo:**

```
sp.setAdapter(
    new ArrayAdapter<String>(
        this, R.layout.my_spinner, R.id.tv1, my_sp_data
    )
);
```

- **Trong tình huống này Spinner tùy biến bằng cách:**
  - Viết layout riêng cho view con
  - Trong layout đó phải có một TextView
  - Lấy id của TextView làm tham số cho setAdapter
  - Lập trình viên phải hiểu rõ cách làm việc của view

Custom view level 3

- **Level 3: viết lại hoàn toàn giao diện, hành vi của view có thể thay đổi hoặc không**
- **Ví dụ viết lại Spinner cần hiểu cách làm việc của nó**
  - Tìm hiểu về các tình huống sử dụng các hàm trong controller của Spinner
  - Tìm hiểu về ý nghĩa các tham số của setAdapter
    - Spinner cần có các đối tượng cung cấp các view con và dữ liệu
    - Controller của Spinner xử lý các tương tác người dùng, request dữ liệu và view con mỗi khi cần thiết
    - ArrayAdapter<String> là một lớp được cung cấp sẵn, trong đó đã viết sẵn các phương thức mà Spinner cần

Custom view level 3

- **Đối với hiển thị Spinner:**
  - `public View getView(int position, View convertView, ViewGroup parent)`
  - `public View getDropDownView(int position, View convertView, ViewGroup parent)`
- **Giải thích:**
  - getView: được gọi ra khi control cần phải hiển thị view cho người dùng (khi vẽ lại hoặc cập nhật)
  - getDropDownView: được gọi ra khi control hiển thị các lựa chọn khi người dùng bấm vào Spinner
- **Mở rộng:**
  - Tách bạch dữ liệu (model) ra khỏi view như thế nào?
  - Thay đổi hành vi của Spinner với việc viết lại controller?

Custom view level 4

- **Level 4: viết lại (gần như) hoàn toàn một view**
- **Đem lại trải nghiệm mới cho người dùng**
- **Thứ tự thực hiện**
  1. Mô tả chính xác view mới gồm dữ liệu, cách thể hiện và trải nghiệm cung cấp cho người dùng
  2. Lựa chọn view mà bạn muốn viết lại (subclassing)
  3. Thiết kế các trạng thái, tương tác và thuộc tính của view
  4. Viết code
  5. Sử dụng custom view trong app
  6. Cung cấp các API cho những người khác sử dụng view của bạn

Phần 4

Ví dụ custom view mức 2: ProgressBar

Custom view mức 2: ProgressBar

- Cung cấp một số tài nguyên phù hợp, giúp view có thể thay đổi cách thức thể hiện, nhưng giữ nguyên trải nghiệm người dùng
- Ví dụ như với trường hợp ProgressBar, chúng ta cung cấp các mẫu tô để giúp thay đổi bề mặt
- ProgressBar là một trong những view được tùy biến level 2 nhiều nhất và phong phú nhất
- Khi tùy biến một view ở level 2, cần chú ý tới sự tương đồng trong giao diện giữa các view, nếu không thì sẽ tạo cảm giác giao diện thiếu sự tương đồng và trau chuốt cần thiết

Custom view mức 2: ProgressBar

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@android:id/background">
        <shape>
            <corners android:radius="5dip" />
            <gradient
                android:angle="270"
                android:centerColor="#ff5a5d5a"
                android:centerY="0.5"
                android:endColor="#ff747674"
                android:startColor="#ff9d9e9d" />
        </shape>
    </item>
```

Custom view mức 2: ProgressBar

```
    <item android:id="@android:id/progress">
        <clip>
            <shape>
                <corners android:radius="5dip" />
                    <gradient
                        android:angle="0"
                        android:endColor="#ff009900"
                        android:startColor="#ff000099" />
            </shape>
        </clip>
    </item>
</layer-list>

// lưu đoạn XML trên thành 1 file thuộc “res/drawable”
```

Custom view mức 2: ProgressBar

```
<?xml version="1.0" encoding="utf-8"?>
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromDegrees="0" android:pivotX="50%" android:pivotY="50%"
    android:toDegrees="360" >
    <shape android:shape="oval" android:useLevel="false" >
        <size android:height="48dip" android:width="48dip" />
        <gradient
            android:centerColor="#ff000000" android:centerY="0.50"
            android:endColor="#ff0000ff" android:startColor="#ff000000"
            android:type="sweep" android:useLevel="false" />
    </shape>
</rotate>

// lưu đoạn XML trên thành 1 file thuộc "res/drawable"
```

Custom view mức 2: ProgressBar

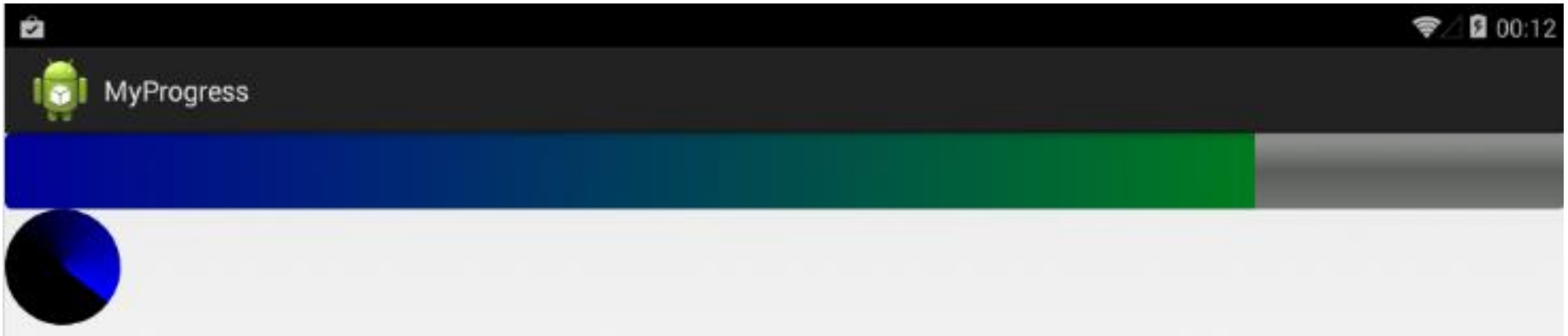
- Thiết lập thuộc tính phù hợp cho các ProgressBar minh họa

```
// ProgressBar dạng thanh ngang
<ProgressBar …
    style="?android:attr/progressBarStyleHorizontal"
    android:minHeight="50dp"
    android:progress="80"
    android:progressDrawable="@drawable/progress1" />
// ProgressBar dạng xoay tròn
<ProgressBar …
    style="?android:attr/progressBarStyleLarge"
    android:indeterminateDrawable="@drawable/progress2"
    android:progress="70" />
```

- Viết các mã minh họa hoạt động của ProgressBar

Custom view mức 2: ProgressBar

- **Giải thích hoạt động:**
  - Trong trường hợp thứ nhất: ta chỉ định 2 mẫu tô, một mẫu cho background và một mẫu cho progress, 2 mẫu này được đặt vào 1 layer-list và truyền cho ProgressBar
  - Trong trường hợp thứ hai: ta chỉ định mẫu tô cho nền của ProgressBar



Phần 5

Ví dụ custom view mức 3: Spinner

Custom view mức 3: Spinner

- **Bài toán: hiển thị danh sách các quốc gia trên một Spinner gồm có tên và cờ của quốc gia đó**
- **Bước 1: chuẩn bị dữ liệu (image cờ quốc gia)**
- **Bước 2: chuẩn bị cách bố cục dữ liệu trên 1 dòng**
  - Thiết kế layout cho một dòng của spinner
  - Nếu thích có thể thiết kế nhiều phương án
- **Bước 3: viết class Adapter phù hợp (đây là yêu cầu tối thiểu, trong thực tế có thể cần viết 3-4 class nếu viết mã đạt chuẩn chuyên nghiệp)**

Custom view mức 3: Spinner

```
class MyAdapter extends ArrayAdapter<String> {
    // hàm cung cấp view trên 1 dòng của spinner
    public View getView(int pos, View view, ViewGroup parent) {
        LayoutInflater inflater = getLayoutInflater();
        View v = inflater.inflate(layout, parent, false);
        TextView t = (TextView) v.findViewById(R.id.textView1);
        t.setText(name[pos]);
        ImageView iv = (ImageView) v.findViewById(R.id.image1);
        iv.setImageResource(icon[pos]);
        return v;
    }
```

Custom view mức 3: Spinner

```
    public View getDropDownView(int p, View v, ViewGroup g) {
        return getView(p, v, g);
    }
    int layout;
    String[] name;
    int[] icon;
    // constructor: layout l, danh sách tên n, danh sách icon i
    public MyAdapter(Context c, int l, String[] n, int[] i) {
        super(c, l, n);
        layout = l; name = n; icon = i;
    }
}
```

Custom view mức 3: Spinner

- **Giải thích:**
  - Ở level 3, chúng ta viết lại Model và View, nếu viết chuẩn mực thì về lý thì nên tách thành ít nhất 2 class độc lập, 1 class Model và 1 class View
  - Để đơn giản hóa, chúng ta ghép 2 class này thành 1 class duy nhất là MyAdapter, trong MyAdapter có chứa Model và bản thân nó là View
  - Đối với một số thiết kế có tính triết lý cao:
    - Tách thành 3 class: class View, class Data của một dòng và class Model (data của tất cả các dòng)
    - Tách thành 4 class: View, Model, RowData, DataBuilder
    - Quy luật đánh đổi: dễ viết – khó dùng, khó viết – dễ dùng

Các tùy biến level 3 thông dụng

- Tùy biến cho **ListView**: đơn giản hơn Spinner, chỉ cần viết lại "`public View getView(int position, View convertView, ViewGroup parent)`" là đủ
- Tùy biến cho **ExpandableListView**: phức tạp, cần tìm hiểu kĩ, các phương thức cần viết lại
  - public View getGroupView(int groupPosition, boolean isExpanded, View convertView, ViewGroup parentView)
  - public View getChildView(int groupPosition, int childPosition, boolean isLastChild, View convertView, ViewGroup parentView)
  - Một số phương thức về model: getChild, getChildID,…

Các tùy biến level 3 thông dụng

- **Tùy biến GridView: đơn giản, như ListView (viết lại getView)**
  - Chú ý là GridView sử dụng cực nhiều trong các ứng dụng hiện đại, vì thế cần tìm hiểu kĩ
- **Tùy biến Gallery: đơn giản, tương tự như ListView**
- **Tùy biến StackView: tương tự như ListView**
- **Chú ý: Trong tất cả các ví dụ trên, chúng ta mới chỉ tập trung vào View, một số các vấn đề cần tìm hiểu thêm**
  - Xử lý khi thêm/bớt/thay đổi dữ liệu
  - Xử lý action từ phía người dùng (tùy thuộc nhiều vào từng loại view)

Phần 6

Ví dụ custom view mức 4: ClockView

Custom view mức 4: ClockView

- **Bài toán: viết 1 view thể hiện đồng hồ thời gian**
- **Lựa chọn: để đơn giản hóa bài toán nhất, sẽ viết lại TextView, sử dụng text để hiển thị giờ**
- **Thiết kế trạng thái và các API**
  - Trạng thái ban đầu là không hoạt động
  - void setClock(hour, minute, second) để định giờ
  - void stop() để dừng
  - void start() để bắt đầu chạy
- **Lời giải dưới đây thật sự vẫn cần nâng cấp nếu bạn muốn sử dụng nó như một widget tiêu chuẩn**

Custom view mức 4: ClockView

```
public class MyClock extends TextView {
    public MyClock(Context context, AttributeSet attrs) {
        super(context, attrs);
    }
    int hour, minute, second;
    public void setClock(int h, int m, int s) {
        hour = h;
        minute = m;
        second = s;
        String str = "" + h + ":" + m + ":" + s;
        setText(str);
    }
    Handler myHandler = new Handler();
```

Custom view mức 4: ClockView

```
public void start() {
    stop();
    th = new Thread(new Runnable() {
        public void run() {
            while (true) {
                try {
                    Thread.sleep(1000);
                    addSecond();
                } catch (Exception ex) {}
            }
        }
    });
    th.start();
}
```

Custom view mức 4: ClockView

```
    void addSecond() {
        second++;
        if (second >= 60) { second = 0; minute++; }
        if (minute >= 60) { minute = 0; hour++; }
        myHandler.post(new Runnable() {
            public void run() { setClock(hour, minute, second); }
        });
    }
    Thread th = null;
    @SuppressWarnings("deprecation")
    public void stop() {
        if (th != null) th.stop();
    }
}
```

Custom view mức 4: ClockView

- **Sử dụng custom view trong app**
  - Custom view sẽ hiển thị trong "Custom & Library Views" trong giao diện thiết kế layout
  - Kéo thả như thành phần thông thường
  - Eclipse chỉ phát hiện những thuộc tính của View cha, cửa sổ Properties sẽ hiển thị các thuộc tính này
  - Có thể dùng XML để thiết lập các thuộc tính khác mà Eclipse không nhận biết được (phức tạp)
  - Trường hợp custom view là inner class, phải chỉ định tên class trong tag class của view cha

```
<view class="mobipro.MyApp$MyEditor" ...
```