4.1 Module Image Loader

Module Image Loader chịu trách nhiệm trích xuất và lưu đệm (cache) tệp ảnh. Module này gồm hai phần như sau:

4.1.1 Image Extractor

Đây thực chất là một lớp đóng gói quanh `ComicParser`. Bản thân chức năng trích xuất được thực hiện trong `ComicParser` (qua các đối tượng `ArchiveParser`), tuy nhiên Image Extractor thực hiện một số tối ưu giúp việc hiển thị ảnh nhanh chóng hơn.

Ở các mục trước, ta đã tệp lẻ trang truyện không được lưu theo thứ tự đọc. Do đó, cần mục lục tệp nén để có thể nhảy cóc đến trang truyện theo yêu cầu. Tuy nhiên, việc tải ảnh còn có thể tối ưu hơn nữa. Ta xét ví dụ trong Bảng 4.1:

Bảng 4.1: Danh sách các tệp ảnh trong một tệp truyện nén

Thứ tự trong tệp nén	Tên tệp ảnh	Dung lượng
1	`7.jpg`	100KB
2	`6.jpg`	100KB
3	`8.jpg`	100KB
4	`1.jpg`	100KB
5	`3.jpg`	100KB
6	`4.jpg`	100KB
7	`2.jpg`	100KB
8	`5.jpg`	100KB

Hiển nhiên, thứ tự ảnh cần xem là từ tệp `1.jpg` đến tệp `8.jpg`. Ta xem xét và cải tiến các chiến lược tải ảnh qua các tiểu mục tiếp theo.

Tải theo yêu cầu

Ảnh được tải theo đúng yêu cầu ngay lúc đó. Quá trình đọc tệp tin như sau:

- Đọc `1.jpg`: 100KB (bản thân ảnh) + 300KB (do trước khi đọc được `1.jpg` cần đi qua 3 ảnh `7.jpg`, `6.jpg`, `8.jpg`)
- Đọc `2.jpg`: 100KB + 600KB
- …

Vậy để đọc hết truyện, cần đọc 4500KB. Chưa hết, luồng nhập phải được tạo mới mỗi lần đọc (tức bằng số trang truyện), gây ra nhiều overhead. Lý do là bản thân SAF là một Content Provider, do đó nó nằm ở một tiến trình (process) riêng, và cần cơ chế liên lạc xuyên tiến trình (IPC) - vốn đắt đỏ về mặt tính toán trên mọi hệ điều hành - để gọi.

Nguyên nhân của cả hai điểm yếu trên là việc không sử dụng lại luồng nhập (mỗi `InputStream` chỉ đọc một ảnh). Phương án tiếp theo cần xử lí được điểm yếu này.

Tối thiểu hóa số luồng đọc

Để giảm số luồng đọc, ta cần kiểm soát một vài luồng đọc, và phân mỗi trang truyện cho một luồng đọc cụ thể. Để tối thiểu hóa số luồng đọc, ta cần dùng thêm thuật toán *chuỗi con tăng dài nhất* (longest increasing subsequence). Thuật toán cuối cùng thể hiện bằng mã giả như sau:

```
def minStream(pages):
    stream_count = 0
    map_idx_to_stream = []    # Từ điển ánh xạ số trang - số luồng

    while len(pages) != 0:
        # Dây trang tiến lên
        lis = longest_increasing_subsequence(pages)

        for page in lis:
            pages.remove_at(page)    # Bỏ trang trong dây khỏi danh sách
            map_idx_to_stream[page] = stream_count    # Gán số luồng hiện tại

        stream_count += 1

    return map_idx_to_stream
```

Thuật toán nhận vào một mảng `pages` là *thứ tự trong tệp nén* của từng trang truyện. Thuật toán trả về một từ điển như sau:

- Khóa: trang truyện số (bắt đầu từ trang 1)
- Giá trị: số luồng

Ta nhận thấy thuật toán hiển nhiên cho (xấp xỉ) số luồng ít nhất có thể, vì mỗi lần chia trang cho các luồng, ta chọn một bộ trang tăng dần dài nhất lúc đó.

Áp dụng vào ví dụ đang dùng, ta có:

- Luồng 0: đọc trang 1, 3, 4, 5
- Luồng 1: đọc trang 7, 8
- Luồng 2: đọc trang 6
- Luồng 3: đọc trang 2

Vậy để đọc hết truyện, cần đọc 2000KB, và 4 lần tạo mới luồng nhập, khá tốt so với phương pháp đầu.

Tới đây chỉ cần một số chỉnh sửa nhỏ: Giới hạn số luồng nhập. Trường hợp xấu nhất là thứ tự trong tệp ZIP ngược với thứ tự đọc, do đó có nhiều luồng mà mỗi luồng chỉ để đọc một trang truyện. yacv tránh điều này bằng cách giới hạn chỉ có 4 luồng nhập cùng lúc. Những trang không ở trong phạm vi của các luồng này quay về cách đọc nhảy cóc thông thường, không dùng lại luồng nhập chờ sẵn.

4.1.2 Image Cache

Bản thân Image Cache *không* phải là một đoạn mã, đối tượng, mà chỉ là một thư mục. yacv có 2 loại/thư mục cache, cho hai trường hợp hiển thị:

- Cache ảnh bìa
- Cache trang truyện

Lý do cần đến hai bộ cache khác nhau là vì dung lượng lớn của truyện. Các thư viện cache ảnh chọn mốc 100-200MB cho thư mục cache ảnh, đồng thời dùng cố định phương pháp thay thế LRU (nếu cache đầy sẽ xóa ảnh lâu nhất không được dùng). Khi đọc một bộ truyện dung lượng lớn hơn mốc này, toàn bộ ảnh trong cache sẽ sớm bị thay thế bởi ảnh của trang truyện. Sau khi đọc, quay về các màn hình, ảnh bìa dùng để hiển thị trong hai màn hình duyệt truyện bị mất, gây suy giảm trải nghiệm người dùng. Do đó, cần phải tách hai thư mục cache này ra để tránh ảnh hưởng đến cache trang bìa.

Cache trang truyện

Khác với cache trang truyện, cache trang bìa *không* có liên quan tới Image Loader. Thực ra ảnh bìa thì cũng được trích xuất bởi `ComicParser`, tuy nhiên với mỗi tệp truyện chỉ cần trích ra một ảnh bìa, do đó không cần cơ chế tái sử dụng luồng đọc phức tạp của Image Loader.

Cache trang bìa lại gồm 2 thư mục cache:

1. Cache ảnh hiển thị thực tế

   Cache ảnh được hiển thị bởi `ImageView`. Ảnh này là ảnh bìa đã được cắt (xem phần thiết kế màn hình duyệt truyện) và được thu phóng về chính xác kích cỡ khung nhìn. Dung lượng cache là 100MB.

   Cùng một bìa có hai kiểu hiển thị

   Thư mục cache này được quản lí bởi thư viện hiển thị ảnh. Thư viện đó nhận luồng đọc ảnh (từ `ComicParser`), ghi vào thư mục cache này, và xóa ảnh để giải phóng dung lượng khi cần.

2. Cache ảnh bìa thu nhỏ

   Cache ảnh bìa thu nhỏ, khoảng 50KB mỗi ảnh, chưa bị cắt. Dung lượng cache là 10MB.

Lí do riêng phần bìa cần hai thư mục cache là vì cache ảnh hiển thị thực tế, giống với cache trang truyện, có thể bị xóa bất cứ lúc nào. Do đó cần có một cache rất nhỏ gọn, nằm ở thư mục riêng mà Android không xóa được, chứa ảnh bìa chất lượng thấp, để khi ảnh bìa bị xóa vẫn có một bản bìa nhỏ để hiển thị trong khi chờ ảnh bìa chất lượng cao, có cắt cúp phù hợp được sinh lại.

Trong Hình 4.15, có một lớp `ImageCache` nhận `InputStream` của ảnh bìa và cache ảnh, đó chính là lớp quản lí và sinh ảnh bìa thu nhỏ.