Caching Strategy On Mobile Rich Media Engine

Xin Min, Jixian Zhang, Lei Luo
School of Computer Science and Engineering, University of Electronic Science and Technology of China, 610054
minxin1986@gmail.com, denonji@163.com, lluo@uestc.edu.cn

Abstract

With the increasing development of mobile rich media applications, it has become an important research topic on how to reduce application response time so as to run more smoothly on resource-limited devices. On the basis of the in-depth study of character drawing mode, cache replacement policy and full use of the cache resources, this paper presents a strategy to cache the characters of mobile rich media pages. The detailed structure, replacement strategies, etc. of this caching strategy are also discussed. Theoretical analysis and simulation data show that the strategy can effectively reduce application’s response time, bringing users more smooth experiences.

Keywords: character cache, mobile device, replacement strategy, rich media engine

1. The status of mobile rich media engine

Due to the significant hardware conditions, the browser engine running on the PC client could run quite smoothly almost without any optimization. However, as for the mobile rich media engine, which is constrained by the mobile terminal’s processor and memory, the applications always delay significantly and respond slowly without taking any effective optimization.

The applications based on the usual mobile rich media engine, without taking advantage of buffers, can directly read all the needed character bitmaps from the local resource files to the memory. Then they will calculate the size of the character for page layout, and eventually display on the screen. There are two obvious disadvantages about this manner:

1) Reading all the character bitmaps into the memory will bring a lot of redundant character information, which will cause a tremendous waste of memory resources. It can easily lead to memory resources exhausted, so this is extremely unrealistic for the resource-restricted devices such as cell phones.

2) In order to display a specified character, the application will have to search all the bitmaps to find out the needed one. It costs a lot of time and will definitely cause the application to display slowly.

Moreover, according to the statistics provided by reference[1], mobile games, mobile phone read, mobile search, mobile browser among all the mobile rich media applications are listed as the top four mobile phone application services used by mobile Internet users in the past six months. A large part of these applications’ contents are characters.

Based on the above several reasons, this paper presents a mechanism to add a character cache to the mobile rich media engine. In this way, the memory overhead of the mobile devices will be effectively reduced, and the display time of the applications will be greatly speeded up. Eventually, it makes the mobile rich media applications run smoothly on the mobile
phones and other mobile devices. When opening the same web page, by comparison with another mobile web engine such as UCWEB, this can indeed reduce the page display time. It turns out that the mechanism of adding the character cache can optimize the performance of the applications.

2. The cache structure and update strategy

We place the aforementioned character cache between the rich media applications and the resource files. This cache is used to store the frequently used character bitmaps so as to avoid redundant data and frequent access to the resource files. What’s more, in order to make the pages render faster, the size of the character is also stored in the cache. In this way, you don’t need to carry out operations every time but directly read out the size. Ultimately, an element of the character cache includes the following:

```c
typedef struct _FONT
{
    void *data; /*the character bitmap information*/
    int width; /*the width of the character*/
    int height; /*the height of the character*/
} FONT;
```

After the character cache is added, the exchange of information between the mobile rich media engine and the local resource files becomes as follows:

![Figure 1. The relationship between the engine, character cache and the local resource files](image)

At this time, the steps taken by the mobile rich media applications to request the character’s information are: the mobile rich media applications first search the character cache, if the character already exists in the cache, we just read out the bitmaps, width and height of the specified character from the cache and make layout display; otherwise, all these information will be read out directly from the local resource files, and then update the cache by the corresponding replacement strategies.

In the cache area, we build a hash table for the most frequently used characters. An item of this hash table contains a keyword “Key” and a pointer, which points to the information of the corresponding character. The keyword represents the code of the character. It varies with the encoding manner, such as UTF-8, ASCII, Unicode and so on. The structure of the hash table describes as follows:

![Figure 2. The structure of the hash table](image)

When the application searches the character cache, this hash table is first checked. If the requested character exists in the table, the character information will be read out by the corresponding pointer. Otherwise, the request would be submitted to the local resource files to search again.

As the bitmaps have low probability to change, the cache manager updates the information of the character in the cache in accordance with a long cycle. At the time of updating, the cache manager retrieve the buffer and update the expired information. In this way, we can guarantee the correctness of character display.

3. The replacement strategy

Since our cache is stored in limited memory, the size must be limited. Besides, as the main purpose of establishing cache is to improve the cache hit rate, the request of the applications will affect the content stored in the cache. Only the most frequently used characters are stored. As a result, a replacement strategy is necessary. Currently the mainly used replacement strategies are:

1) RAND (Random): Replace the resources randomly. Advantages: easy to implement and understand; Disadvantages: do not use the historical information, can’t reflect the Principle of Locality, so it has low hit rate;
2) FIFO (First In First Out). Select the first resource to be replaced. Advantages: easy to implement and understand; Disadvantages: too simple, can’t accurately reflect the Principle of Locality, hit rate is not high, and difficult to apply to specific applications;
3) LFU (Least Frequently Used). Select the least frequently used resource to be replaced.
Advantages: quite reasonable, make full use of the historical paging information, correctly reflect the Principle of Locality, so hit rate is high; Disadvantages: it will maintain several timers to decide which one to be replaced, so it’s quite complex to implement;

4) LRU (Least Recently Used). Select the least recently used resource to be replaced. Advantages: easier to implement than LFU, can partly reflect the Principle of Locality, hit rate is a little higher than FIFO and RAND; Disadvantages: more complex than FIFO and RAND, do not reflect the historical situation, long-term data-access features and the preference trends of the entire cache.

Because of the limited resources in the mobile rich media applications and the mobile devices, the autonomy and mobility of mobile users etc., popular cache replacement strategies are to some extent effective, but they still can’t satisfy the mobile rich media engine. In particular, they cannot apply for the features of mobile rich media applications. Therefore, according to the reference, we take an improved replacement strategy in this paper. Experiments show that the hit rate of this replacement strategy is between LRU and LFU. However, it’s much easier to implement than LFU. So, it’s quite suitable for mobile rich media engines.

In this strategy, a cache manager will maintain a log table to determine which character will be replaced in the cache. This log table includes: the character’s keyword K, the time of first use T1, the total count of use C1, the decisive value V, the time when the value is last calculated T2, the count of use in the cycle C2. Everytime a new character is requested, C1 and C2 of the corresponding character plus one. Assuming that we calculate the value at time T, then the decisive value

\[ V = d * C1 / (T - T1) + (1 - d) * C2 / (T - T2); (0 \leq d \leq 1) \]

where d is a constant used to balance the total frequency and cycle frequency of use. After the value is calculated, T2 will become T, and C2 will become 0. In this way, we can take both the total count of use and recent count of use into consideration. Thus, this strategy can not only reflect the Principle of Locality but also raise the hit rate. In order to perform replacement, we set a threshold p, if V > p, the character will get into the buffer and take the place of the character with minimum value.

4. The cache manager

The entire cache is maintained by the cache manager, which includes three modules: character management module, update management module and replacement management module. The structure is shown in Figure 3:

Figure 3. The structure of cache manager

The functions of each module are as follows:

- Character management module: Receive the requests from the application and find the corresponding character information. It will first search the cache. If hit, directly read out from the cache. Otherwise, obtain from the local resource files;
- Update management module: Check the cache periodically and update the contents;
- Replacement management module: Maintain the log table, establish the initial cache and replace the cache according to the replacement strategy.

It is the collaboration of these three modules that ensures the accuracy of data and the convenience of displaying characters.

5. The experimental results

In this paper, we take Dopod S900C which is based on Windows Mobile platform as an example. This phone's performance parameters are shown as below:

<table>
<thead>
<tr>
<th>Parameter type</th>
<th>Parameter value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone model</td>
<td>Dopod S900C</td>
</tr>
<tr>
<td>Screen parameter</td>
<td>480 × 640 pixels</td>
</tr>
<tr>
<td>Operating system</td>
<td>Windows Mobile 6.1 Professional</td>
</tr>
<tr>
<td>CPU frequency</td>
<td>528MHz</td>
</tr>
<tr>
<td>Memory</td>
<td>256MB ROM+288MB RAM</td>
</tr>
</tbody>
</table>

Take the experiment following these steps: Burn the mobile rich media engine with and without character cache respectively to this phone, request for the same web pages, note down the display time, and then exit the engine. Repeat these steps for 20 times. Record all the test results, and the final statistics are shown as Figure 4:
6. Conclusion

As shown in Figure 4, the performance after establishing the cache is quite impressive. And we can also conclude that the English pages display slightly better than the Chinese, because the English pages have less possibility of cache replacement; The more characters there are, the better the performance will be, because the cached content at this time nearly reach a steady state. As the amount of characters increases, time extend and buffer capacity grow, the engine’s effect of character cache will be increasingly evident. Ultimately, it will achieve a more ideal display status[10]. Thus, the cache will greatly optimize the overall performance of mobile rich media applications.

The establishment of character cache means a lot to reduce applications’ response time and use the memory rationally. It will be an important topic for further research on how to verify the size of cache, how to improve the performance of the cache, how to determine more reasonable value of “d” and “p” in the algorithm and so on.

7. References