**Tiện ích mở rộng Java: khái niệm nâng cao**

Trong bài viết này, chúng tôi thảo luận về các khía cạnh nâng cao của việc phát triển Tiện ích mở rộng Java:

* [**Mô hình luồng máy chủ**](http://docs2x.smartfoxserver.com/ExtensionsJava/advanced-concepts#threadModel)
* [**Nhóm luồng tự động cân bằng tải**](http://docs2x.smartfoxserver.com/ExtensionsJava/advanced-concepts#autoLoadBalance)
* [**Điều chỉnh nhóm chủ đề**](http://docs2x.smartfoxserver.com/ExtensionsJava/advanced-concepts#tuning)
* [**Duy trì trạng thái trong Tiện ích mở rộng**](http://docs2x.smartfoxserver.com/ExtensionsJava/advanced-concepts#extensionState)
* [**Kiến trúc tải lớp**](http://docs2x.smartfoxserver.com/ExtensionsJava/advanced-concepts#classLoading)
* [**Nhiệm vụ bị trì hoãn và theo lịch trình**](http://docs2x.smartfoxserver.com/ExtensionsJava/advanced-concepts#scheduler)

**Kể từ 2.9.0**

**» Mô hình phân luồng Máy chủ**

**LƯU Ý**   
Đối với tất cả người dùng SFS2X 2.8.x và các phiên bản cũ hơn, vui lòng xem **[tài liệu mô hình phân luồng cũ](http://docs2x.smartfoxserver.com/ExtensionsJava/old-thread-model)** .

SmartFoxServer 2X chạy tất cả Tiện ích mở rộng trong môi trường đa luồng. Về cơ bản có hai nhóm luồng riêng biệt hoạt động trên một Tiện ích mở rộng : **ExtensionController** và **SystemController** . Thực thể trước chịu trách nhiệm xử lý các yêu cầu của máy khách trong khi thực thể sau gửi các sự kiện hệ thống như LOGIN, USER\_DISCONNECT, ROOM\_VARIABLES\_UPDATE, v.v.

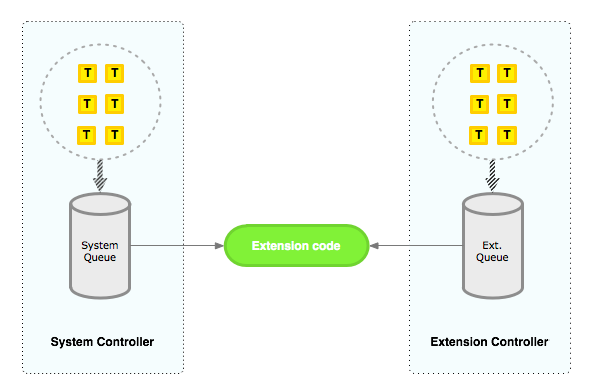
Vì nhiều luồng có thể hoạt động đồng thời trên mã Tiện ích mở rộng nên chúng tôi cần đảm bảo rằng quyền truy cập vào trạng thái chia sẻ được xử lý chính xác. Các **[bộ sưu tập đồng thời](http://www.ibm.com/developerworks/java/library/j-tiger06164.html" \t "_blank)** và **[các tính năng khóa](http://www.ibm.com/developerworks/java/library/j-jtp10264/" \t "_blank)** của JDK tiêu chuẩn cung cấp các công cụ mạnh mẽ để xử lý các sự cố đồng thời phổ biến nhất với nỗ lực tối thiểu.

Chúng tôi thực sự khuyên dùng các bộ sưu tập đồng thời thay vì các mảng gốc hoặc các bộ sưu tập kiểu cũ, chẳng hạn như Vector và Hashtable , có thể hạn chế đồng thời và tạo ra các nút cổ chai trong mã. Ngoài ra, khi cần có tính nguyên tử, chúng tôi khuyên bạn nên xem xét AtomicInteger , AtomicLong , AtomicReference như một giải pháp hợp lệ. Cuối cùng, các lớp khóa sẽ được yêu cầu khi các hoạt động phức tạp cần được chạy theo kiểu loại trừ lẫn nhau.

Chúng tôi cũng đề xuất một số tài nguyên bên ngoài nếu bạn muốn tìm hiểu sâu hơn về chủ đề này:

* [**Đồng thời Java (đa luồng)**](http://www.vogella.com/tutorials/JavaConcurrency/article.html#concurrency_amdahl)
* [**Đồng thời Java mà không gặp khó khăn**](http://www.javaworld.com/article/2078809/java-concurrency/java-101-the-next-generation-java-concurrency-without-the-pain-part-1.html)

Sơ đồ sau đây minh họa những gì chúng tôi đã mô tả. Hai bộ điều khiển chạy các nhóm luồng riêng biệt đồng thời gọi các trình xử lý sự kiện và yêu cầu trên các lớp Tiện ích mở rộng.



Ngoài ra, Mã mở rộng cũng có thể lên lịch cho bất kỳ số lượng tác vụ lặp lại hoặc bị trì hoãn nào mà lần lượt được xử lý bởi một thực thể khác, TaskScheduler , chạy nhóm luồng riêng của nó.

Điều quan trọng cần lưu ý là **API SFS2X** đã xử lý đồng thời hầu hết thời gian: tất cả các lệnh gọi được cung cấp **bởi lớp SFSApi** chính đều là luồng an toàn và điều tương tự cũng xảy ra với **[API trò chơi](http://docs2x.smartfoxserver.com/AdvancedTopics/game-api)** và API **[danh sách bạn bè](http://docs2x.smartfoxserver.com/AdvancedTopics/buddy-list-api)** . Tuy nhiên, có một vài trường hợp ngoại lệ: ví dụ: **SFSObject** và **SFSArray** không phải là luồng an toàn. Vì các đối tượng này chủ yếu được sử dụng để vận chuyển dữ liệu nên chúng thường không bị tranh chấp bởi nhiều luồng. Trong mọi trường hợp, Javadoc chỉ định đối tượng nào cần được chăm sóc thêm để đảm bảo an toàn cho luồng.

[**^menu trên cùng**](http://docs2x.smartfoxserver.com/ExtensionsJava/advanced-concepts#items)

**Kể từ 2.9.0**

**» Nhóm luồng tự động cân bằng tải**

Một trong những câu hỏi được hỏi rất thường xuyên là làm thế nào để định cấu hình số lượng luồng đang chạy trong hệ thống để mở rộng ứng dụng một cách chính xác. Trong SmartFoxServer 2.9.0, chúng tôi đã giới thiệu một cải tiến đáng kể về khả năng mở rộng bằng cách **loại bỏ nhóm luồng được cấu hình thủ công** .

Giải pháp mà chúng tôi cung cấp là một **Executor tự giám sát, tự động thay đổi quy mô** , có khả năng thay đổi quy mô theo yêu cầu, cung cấp tỷ lệ cân bằng, nhưng có thể định cấu hình, giữa độ trễ và tài nguyên được sử dụng.

Với tính năng này, số lượng cấu hình thủ công và tinh chỉnh cần thiết để chạy bất kỳ ứng dụng nào về cơ bản được giảm xuống bằng không. Quan trọng hơn, hệ thống sẽ tự động theo dõi trạng thái của hàng đợi tin nhắn và phản ứng khi thiếu chuỗi theo yêu cầu. Điều này cho phép bảo vệ máy chủ khỏi việc tạo và hủy các luồng liên tục đối với các đợt lưu lượng nhỏ. Đồng thời , nó cho phép phản ứng với sự gia tăng đáng kể về lưu lượng truy cập và/hoặc công việc I/O chậm. Các chủ đề dư thừa sẽ được giải phóng ngay khi chúng không còn cần thiết nữa.

**TÓM** LẠI : kể từ phiên bản 2.9.0, bạn không cần phải định cấu hình số lượng luồng Hệ thống/Tiện ích mở rộng tối đa theo cách thủ công, cũng như không cần lo lắng về tài nguyên luồng khi thêm các yêu cầu I/O chậm chẳng hạn như cơ sở dữ liệu bên ngoài và lệnh gọi HTTP từ xa.

[**^menu trên cùng**](http://docs2x.smartfoxserver.com/ExtensionsJava/advanced-concepts#items)

**Kể từ 2.9.0**

**» Điều chỉnh nhóm chủ đề**

Đối với hầu hết các tập quán, không cần phải chạm vào nhóm của máy chủ. Tuy nhiên, nếu bạn đang sử dụng nhiều I/O chậm trong tiện ích mở rộng của mình, chúng tôi sẽ cung cấp bảng điều khiển nâng cao trong **AdminTool** > **[Server Configurator](http://docs2x.smartfoxserver.com/GettingStarted/admintool-ServerConfigurator" \t "_blank)** cho phép các chuyên gia tinh chỉnh cấu hình luồng. Sau đây là danh sách các tham số khả dụng cho mỗi Executor.

* **Chủ đề cốt lõi** : số lượng chủ đề sẽ sử dụng khi khởi động.
* **Chủ đề sao lưu** : số lượng chủ đề được thêm mỗi khi kích hoạt sao lưu.
* **Sao lưu** tối đa: số lượng hoạt động sao lưu luồng tối đa.
* **Kích thước hàng đợi kích hoạt sao lưu** : kích thước của hàng đợi sẽ thêm một lô chuỗi sao lưu mới.
* **Thời gian kích hoạt sao lưu** : số giây sau đó sao lưu được gọi nếu trình kích hoạt kích thước hàng đợi vẫn đang hoạt động.
* **Hết hạn chủ đề sao lưu** : số giây sau đó chủ đề sao lưu bị xóa, nếu kích thước hàng đợi thấp hơn giá trị tiếp theo.
* **Kích thước hàng đợi ngăn chặn hết hạn sao lưu** : kích thước của hàng đợi cho phép xóa các chuỗi sao lưu.
* **Hoạt động nhật ký** : thêm thông báo trong tệp nhật ký mỗi khi nhóm luồng được thay đổi kích thước.
* **Khoảng thời gian cảnh báo hàng đợi đầy** : có bao nhiêu giây giữa các cảnh báo về việc hàng đợi tin nhắn đã đầy.
* **Kích thước hàng đợi yêu cầu bộ điều khiển** : tối đa. kích thước của hàng đợi bộ điều khiển.
* **lõi** : bắt đầu tất cả các luồng lõi ngay lập tức, thay vì thêm chúng khi cần (kể từ 2.13.1).

**LƯU Ý**   
Chúng tôi khuyên bạn không nên chạm vào các giá trị cấu hình mặc định trừ khi bạn có lý do chính đáng để làm như vậy. Các cài đặt mặc định đã được tinh chỉnh để cung cấp khả năng mở rộng tuyệt vời trong nhiều kịch bản và khối lượng công việc.

[**^menu trên cùng**](http://docs2x.smartfoxserver.com/ExtensionsJava/advanced-concepts#items)

**» Duy trì trạng thái trong Tiện ích mở rộng**

Khi chúng tôi sử dụng lớp **SFFSextension** làm lớp cơ sở cho Tiện ích mở rộng của mình, chúng tôi kết thúc với một đối tượng Tiện ích mở rộng chính và một loạt trình xử lý sự kiện và yêu cầu. Một câu hỏi phổ biến là: *giữ trạng thái chia sẻ của ứng dụng ở đâu?* ( điểm số , bảng xếp hạng, dữ liệu trò chơi, v.v. )

Thông thường có hai câu trả lời hợp lý:

* trong lớp Tiện ích mở rộng chính hiển thị mô hình trò chơi thông qua (các) getter/setter;
* sử dụng Singleton có thể được truy cập từ bất kỳ đâu trong mã của bạn.

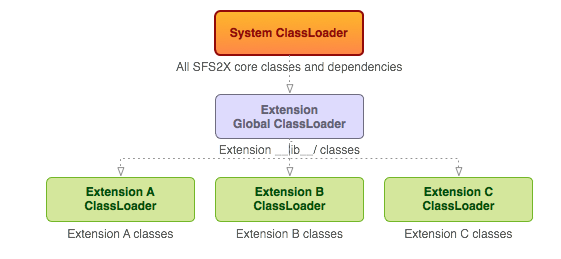
Chúng tôi rất muốn giới thiệu cách tiếp cận đầu tiên thay vì cách tiếp cận thứ hai vì một loạt lý do:

* Singletons không dễ triển khai trong môi trường có nhiều Trình tải lớp (chúng tôi giải thích tất cả các chi tiết trong phần tiếp theo của tài liệu này);
* Singletons có thể thực hiện các thủ thuật khó chịu trong nhiều lần khởi động lại Tiện ích mở rộng, do tính chất tĩnh của chúng;
* Tiện ích mở rộng chính đã hoạt động như một Singleton và rất dễ dàng truy cập nó từ bất kỳ yêu cầu hoặc trình xử lý sự kiện nào thông qua phương thức **getParentExtension ( )** ; ngoài ra, bạn không nhận được những nhược điểm của Singleton mà chúng tôi vừa đề cập.

[**^menu trên cùng**](http://docs2x.smartfoxserver.com/ExtensionsJava/advanced-concepts#items)

**» Kiến trúc tải lớp**

Trong bài viết giới thiệu về Tiện ích mở rộng , chúng tôi đã đề cập rằng mỗi Tiện ích mở rộng được tải trong một Trình nạp lớp khác nhau để cho phép triển khai lại nóng trong quá trình phát triển hoặc thậm chí sản xuất. Chúng tôi cũng minh họa rằng để chia sẻ các phụ thuộc trên nhiều Tiện ích mở rộng, các Trình tải lớp này tuân theo một hệ thống phân cấp cụ thể:



Sơ đồ cho thấy rằng mỗi Tiện ích mở rộng "thấy" tất cả các lớp đã triển khai của nó trong ClassLoader của riêng nó, nó có thể truy cập các lớp Tiện ích mở rộng toàn cầu hàng đầu nhờ ClassLoader mẹ của nó và cuối cùng, nó có thể sử dụng bất kỳ lớp nào từ khung SFS2X nhờ vào phần tử trên cùng trong hệ thống phân cấp này .

Khi một tiện ích mở rộng được tải lại, chỉ có ClassLoader dưới cùng bị hủy và được xây dựng lại. Thao tác này sẽ tạo phiên bản mới của các lớp có trong (các) tệp jar đã triển khai, trong khi các lớp còn lại ở cấp cao nhất không bị ảnh hưởng (điều này có nghĩa là không thể tải lại chúng).

Hãy xem xét một ví dụ thực tế để xem điều này có thể hữu ích như thế nào. Giả sử Tiện ích mở rộng A là Tiện ích mở rộng vùng chính của chúng tôi, trong khi B và C là hai Tiện ích mở rộng phòng, quản lý hai trò chơi khác nhau. Chúng tôi muốn Tiện ích mở rộng B và C giao tiếp với A để truy cập bảng xếp hạng trò chơi.

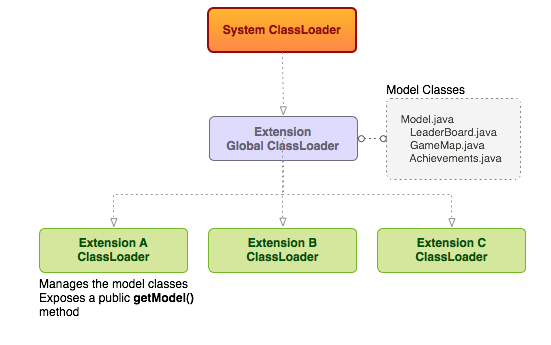
Vấn đề đầu tiên chúng ta gặp phải là chúng ta cần triển khai cùng một lớp mô hình trong cả ba Phần mở rộng (A, B và C) để chạy ví dụ đúng cách. Điều này có nghĩa là mỗi ClassLoader chứa một phiên bản khác nhau của cùng một Lớp, đây là một vấn đề nổi tiếng trong Java. ClassCastException khét tiếng được Java Runtime đưa ra khi bạn cố gắng lấy một đối tượng từ ngữ cảnh khác ngay cả khi hai lớp có cùng mã byte. Về mặt kỹ thuật, mặc dù chúng thực sự là cùng một mã byte, nhưng JVM coi chúng là các định nghĩa Lớp khác nhau tình cờ chia sẻ cùng một tên đủ điều kiện mặc dù trong ba ngữ cảnh riêng biệt ( ClassLoaders ).

**LƯU Ý**   
Nếu chúng tôi không làm mất bạn cho đến thời điểm này thì có thể bạn đã biết những kiến thức cơ bản về ClassLoading trong Java. Nếu điều này nghe có vẻ khó hiểu , chúng tôi khuyên bạn nên kiểm tra một số bài viết về chủ đề này:

* [**Một cái nhìn về Java ClassLoader**](http://www.javalobby.org/java/forums/t18345.html)
* [**Các lớp Java và tải lớp**](http://www.ibm.com/developerworks/java/library/j-dyn0429/)
* [**Tải lớp Java**](http://www.techjava.de/topics/2008/01/java-class-loading/)

May mắn thay , giải pháp khá dễ dàng: tất cả những gì bạn cần làm là triển khai các lớp mô hình trong thư mục **tiện ích mở rộng/\_\_lib\_\_/** và bạn sẽ có thể chia sẻ các đối tượng này trên tất cả các Tiện ích mở rộng.

**Ví dụ:**



Mọi Trình tải lớp mở rộng hiện có thể truy cập cùng các lớp mô hình vì chúng có thể truy cập được trong Trình tải gốc của chúng. Bằng cách cung cấp quyền truy cập vào mô hình trò chơi từ Tiện ích mở rộng Khu vực chính, chúng tôi đã tạo ra một giải pháp giống như Singleton mà không phải lo lắng về dữ liệu tĩnh.

Nếu bạn vẫn yêu cầu tạo một hoặc nhiều lớp Singleton, cần được chia sẻ trên nhiều Tiện ích mở rộng, phương pháp này cũng sẽ hiệu quả. Bạn sẽ cần triển khai (các) Singleton trong tệp jar bên trong thư mục **\_\_lib\_\_/** .

[**^menu trên cùng**](http://docs2x.smartfoxserver.com/ExtensionsJava/advanced-concepts#items)

**» Nhiệm vụ bị trì hoãn và theo lịch trình**

Thông thường, logic trò chơi yêu cầu sử dụng bộ hẹn giờ để cập nhật ứng dụng khách định kỳ ( ví dụ: thời gian kết thúc lượt, kích hoạt các sự kiện cụ thể, hành động của NPC, v.v. ).

Một giải pháp nhanh chóng cho vấn đề này là sử dụng lớp **ScheduledThreadPoolExecutor** được cung cấp trong JDK, lớp này cung cấp một trình thực thi tác vụ thuận tiện được hỗ trợ bởi một nhóm luồng. SFS2X đã chạy phiên bản riêng của Executor này (được bao bọc trong một lớp có tên là TaskScheduler ).

Đoạn mã Java sau đây cho biết cách chạy tác vụ lặp bằng cách sử dụng Trình lập lịch tác vụ của chính Máy **chủ** .

Java

[**?**](http://docs2x.smartfoxserver.com/ExtensionsJava/advanced-concepts)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | **public** **class** SchedulerTest **extends** SFSExtension  {  **private** **class** TaskRunner **implements** Runnable      {  **private** **int** runningCycles = 0;    **public** **void** run()          {              runningCycles++;              trace("Inside the running task. Cycle:  " + runningCycles);    **if** (runningCycles >= 10)              {                  trace("Time to stop the task!");                  taskHandle.cancel();              }          }      }        // Keeps a reference to the task execution      ScheduledFuture<?> taskHandle;        @Override  **public** **void** init()      {          SmartFoxServer sfs = SmartFoxServer.getInstance();            // Schedule the task to run every second, with no initial delay          taskHandle = sfs.getTaskScheduler().scheduleAtFixedRate(**new** TaskRunner(), 0, 1, TimeUnit.SECONDS);      }  } |

Phương thức **scheduleAtFixedRate** nhận bốn đối số:

1. một đối tượng *Runnable* sẽ thực thi mã của Tác vụ
2. độ trễ ban đầu trước khi bắt đầu thực hiện
3. khoảng thời gian mà nhiệm vụ sẽ được thực hiện
4. đơn vị thời gian được sử dụng để thể hiện các giá trị thời gian

Bộ lập lịch trình cũng hiển thị một phương thức **lịch trình thực thi tác vụ** *Runnable* một lần sau khoảng thời gian đã chỉ định. Cuối cùng, nhóm luồng của Trình lập lịch biểu có thể được thay đổi kích thước nhanh chóng trong thời gian chạy thông qua phương thức **resizeThreadPool ( )** .

**LƯU Ý**   
Kích thước ban đầu của nhóm luồng của hệ thống TaskScheduler có thể được điều chỉnh thông qua mô-đun Bộ **cấu hình máy chủ trong AdminTool .**