**QSP và Q2**

# Tại sao phải hiểu về QSP?

-Các ứng dụng của bạn cần một số thành phần được cấu hình trong lúc khởi động, một chuỗi các hành động đa kết nối cũng như kiểm định.

-Lấy điển hình hàm main của ứng dụng JPOS nó nhận nhiệm vụ đọc các file cấu hình và rồi tạo thể hiện của các thành phần chạy một vài Thread mới hay khởi tạo kết nối với database.v.v

-Vậy thì QSP là một cách chuẩn cho các nhiệm vụ đã được đề cập ở trên.Nó đọc tất cả các file cấu hình và trả về thể hiện của các đối tượng cho bạn sử dụng.

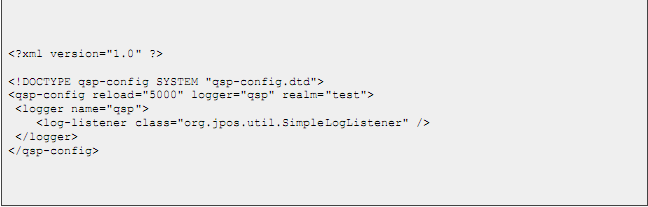
Từ phiên bản 1.4.6 QSP của Jpos đã được nâng cấp với một tên gọi đầy mới mẽ Q2.(QSP version 2). QSP vẫn được hỗ trợ một cách đầy đủ nhưng theo kinh nghiệm thì chúng ta nên sử dụng Q2 cho ứng dụng của mình bởi vì nó sẽ làm cho cuộc đời của bạn dễ dàng hơn và tốt đẹp hơn rất nhiều.Trong các trường hợp khác nhau về nguyên tắc QSP và Q2 có rất nhiều điểm tương đồng do đó để hiểu được Q2 chúng ta nên tìm hiều về QSP trước.

# QSP

Chúng ta sẽ không đi sâu vào mục đích cài đặt QSP. Trong phần này chúng ta cần nắm kỹ một số thành phần(component) được cung cấp trong Jpos sẽ được cấu hình trong file xml vì vậy yếu tố khiến chúng ta quan tâm đó là các thành phần đó sẽ được cấu hình như thế nào và chúng ta sẽ sử dụng cấu hình đó trong ứng dụng của mình như thế nào.

Bây giờ để có thể có ý niệm đầu tiên về cách thức mà QSP cấu hình chúng ta thực hiện một bài test sau:

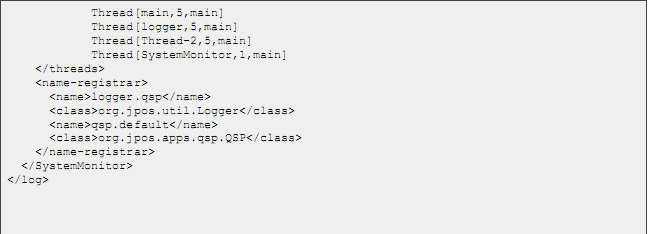
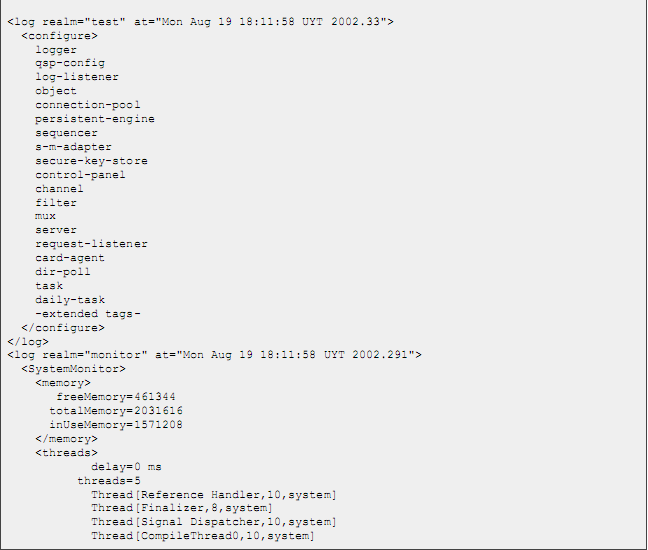
Tao file test.xml (file này được đặt cùng thư mục với jpos.jar) có nội dung như sau:



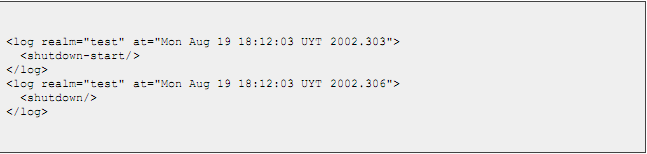
Đến đây chúng ta chưa quan tâm vội đến nội dung của file test.xml này.Vào command line gõ lệnh sau:

Java –jar jpos.jar test.xml

Bạn sẽ nhận được một thông điệp như sau:



Bây giờ bạn di chuyển file test.xml đến một nơi khác trong lúc vẫn để cửa sỗ command- line hoạt động (QSP vẫn đang chạy) . Chúng ta sẽ nhận được thông điệp sau.

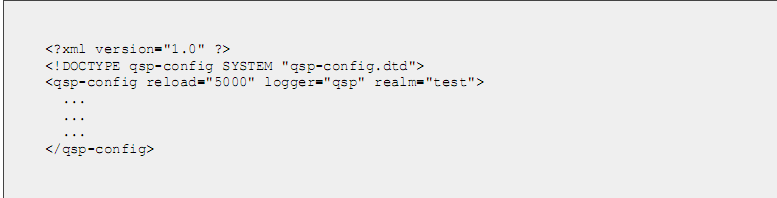


Lập tức Q2 tắt .Ye! Bây giờ tôi nghĩ bạn đã phần nào suy tưởng ra được cách thức cấu hình cho một component bất kì của Jpos(không chỉ là QSP).

Việc cấu hình động (runtime configuration) giúp cho hệ thống hoạt động liên tục mà không cần phải khởi động lại hệ thống khi có một sự câp nhật xảy ra. Điều này quả là tuyệt vời.

## Cấu hình.

### Cấu hình QSP



-Thuộc tính reload chỉ ra thời gian mà QSP kiểm tra file cấu hình cho bất kì thay đổi nào.

-logger và realm là 2 thuộc tính tùy chọn đặt cung cấp việc đặt logger cho QSP.

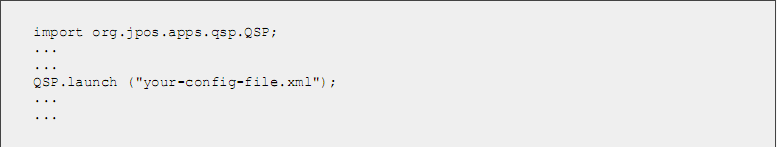
### Object.

Có ít nhất 2 cách để sử dụng QSP:

+Biểu diễn QSP từ ứng dụng của bạn

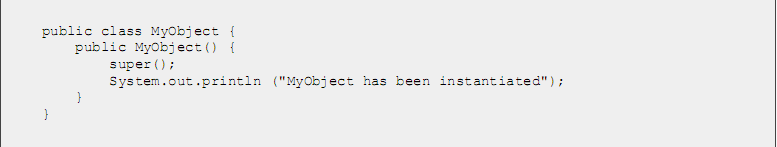
+Biểu diễn ứng dụng của bạn bằng QSP

Để khởi động QSP từ ứng dụng của bạn chúng ta thực hiện như sau:

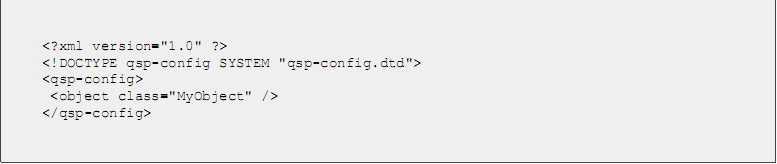


Nhưng trên thực tế thì người ta thường dùng cách thứ hai. Bạn không nên tạo quá một thể hiện của QSP trong JVM.

Để thực hiện gọi ứng dụng bằng QSP đơn giản bạn chỉ cần dùng thẻ <object> trong file cấu hình QSP. Giả sử bạn có một lớp tên là MyObject.java do bạn tự định nghĩa có nội dung như sau:

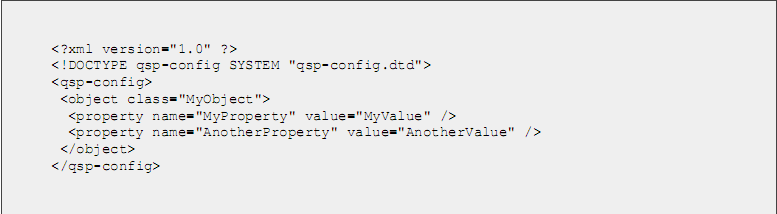


Và để sử dụng lớp này thông qua QSP chúng ta sử dụng thuộc tính class của thẻ <object> chỉ tới tên đối tương cần sử dụng.

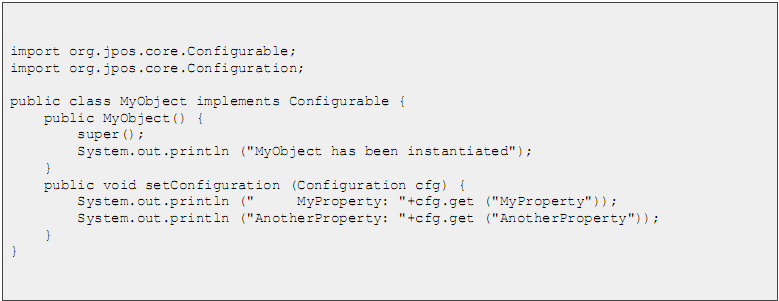


### Configuration Object

Bạn có thêm các thẻ con <properties> bên trong thẻ <object> như sau:



.Để đọc các giá trị của thẻ <properties> thì đơn giản bạn thừa kế lại interface Configuration và sữ dụng đối tượng Configuration của hàm setConfiguration().



Và khi biên dịch lại file MyObject.java sau đó chạy lại file test.xml chúng ta sẽ được kết quả sau:

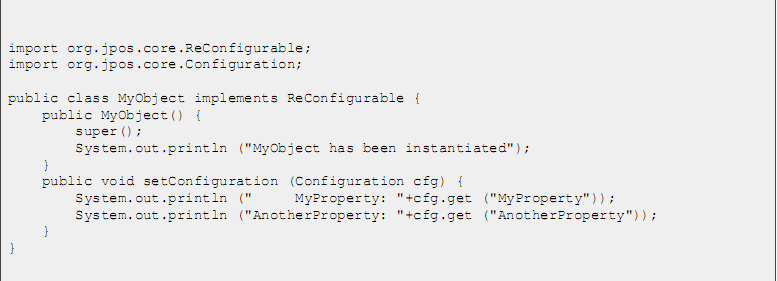


### Reconfiuration Object

Đối với các ứng dụng 24/7/365 việc khởi động lại hệ thống nếu có thay đổi xảy ra sẽ làm gián đoạn hệ thống.

Bây giờ chúng ta xem Jpos khắc phục điều này như thế nào.

Trong lớp MyObject.Java thay vì thừa kế từ Interface Configuration chúng ta sẽ thừa kế từ Interface Reconfiguration như sau:



Chỉnh sửa lại file test.xml với thuộc tính reload.



Bây giờ chúng ta sẽ biên dịch lại file MyObject.java

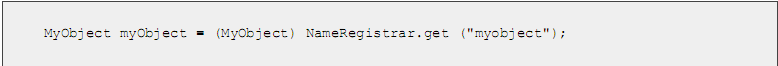
Và chạy file test.xml chúng ta sẽ thu được kết quả tương tự như phần Confiuration ở trên.

Bây giờ trong lúc Command line đang chạy thử thay đổi giá trị của <properties> từ MyValue sang MyNewValue. Và bạn sẽ thấy điều mới mẻ sau.



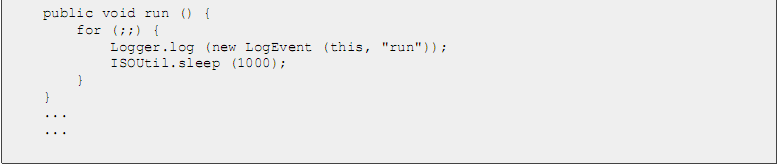
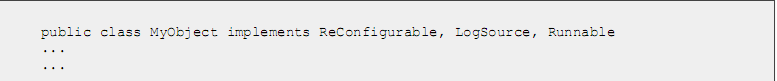
### Truy xuất đối tượng

Nếu bạn sử dụng thuộc tính name cho thẻ <object> thì QSP sẽ sử dụng NameRegistrar đăng kí <object> của bạn. Và để truy xuất đối tượng đó đơn giản như sau.



### Runable Object

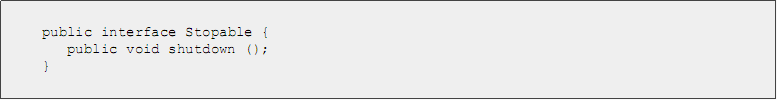
Đơn giản là bạn thực thi interface Runnable



Nếu QSP nhận ra rằng đối tượng của bạn thừa kế từ lớp **Runnable** thì đơn giản nó sẽ chạy đối tượng trong một **Thread mới**.

### Shutting Down

Nếu bạn muốn sử lý một số việc gì đó khi hệ thống tắt thì bạn có thể thực thi Interface Stopable sau.



Cụ thể

Public class MyObject implements Stopable{

…

Public void shutdown()[

System.out.println(“Good Bye”);

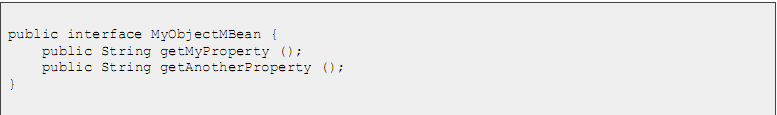
}

}

### JMX support

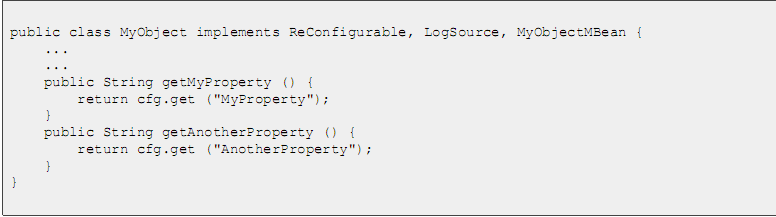
Nếu bạn muốn kiểm soát hoặc thể hiện <object> của bạn thông qua JMX[[1]](#footnote-2) , đơn giản là tạo ra một đối tượng theo chuẩn MBean và QSP sẽ đăng kí nó với MBean server cho bạn.

Cụ thể. Đầu tiên bạn tạo một interface theo chuẩn MBean.



Và sau đó là thực thi đối tượng này.

Sau đó thực thi Interface bạn mới định nghĩa.



Tuy Reconfiguration hỗ trợ bạn có thể chỉnh sửa file cấu hinh tuy nhiên việc gỡ bỏ và thêm mới file cấu hình phải khởi động lại QSP. Điều này có ảnh hưởng rất lớn đến các hệ thống 24/7.

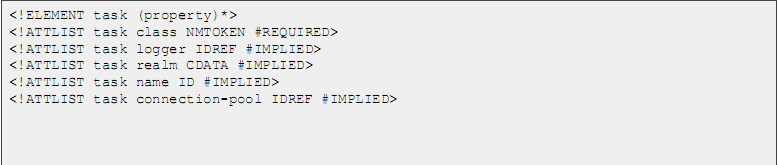
### <Task>

<task> được kiểm soát gần giống với <object> với một số khác biệt nhỏ sau:

<task> là một loại component được khởi tạo bởi QSP.

<task> được đăng kí với NameRegistar với tiền tố “task.”

Xem xét file dtd mô tả cho thẻ <task> :



Cũng giống như <object>, <task> có thể thực thi :

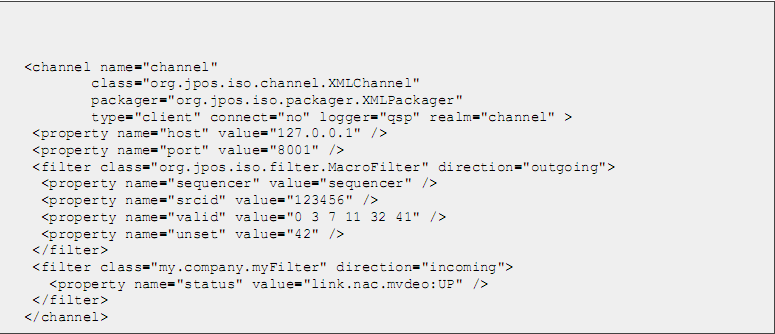
* LogSource
* [Re]Configurable
* Runable
* Stopable
* \*MBean

### <channel>

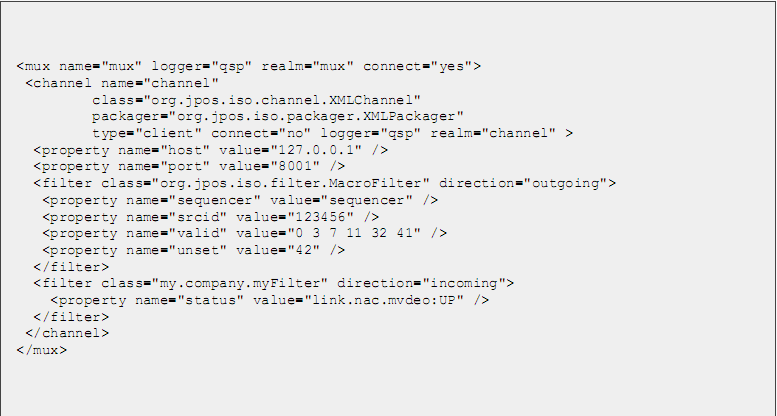
Channel được sử dụng ở hầu hết các ứng dụng của Jpos , QSP được thiết kế để cấu hình channel có thể tương tác được với các component khác như server,mux,



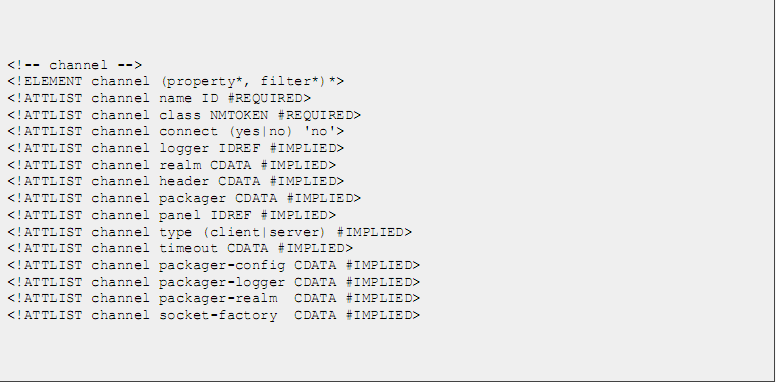
Và thật đơn giản để thêm vào một bộ filter cho channel. Filter được thêm vào bằng thẻ con <filter>.



Tương tự thật dễ dàng để cấu hình mux sử dụng channel này.

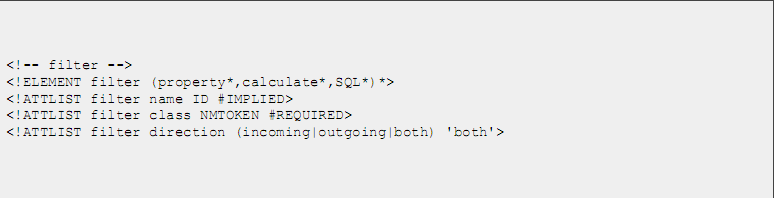


Xem xét file dtd mô tả cho channel :



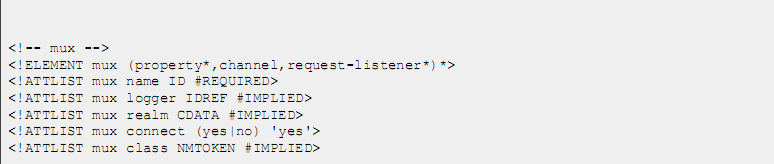
### <filter>

<filter> được QSP dùng để cấu hình cho ISOFilter. Xem xét file DTD của <filter>



### <mux>

<mux> của QSP dùng để cấu hình cho ISOMUX.

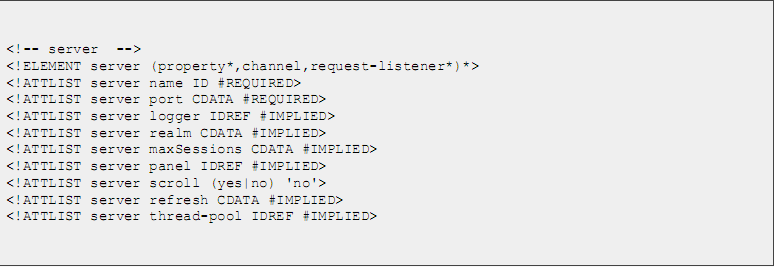


Cấu hình cần một thuộc tính di nhất name để đăng kí với NameRegistar. Một cặp logger/reaml để đặt logger cho <mux> .Thuộc tính connect dùng để cắt kết nối với <mux> bằng cách thiết lập conncet=”no” .Thuộc tính class được dùng để sử dụng các lớp ISOMUX do người dùng định nghĩa.(mặc định là org.jpos.iso.ISOMUX).

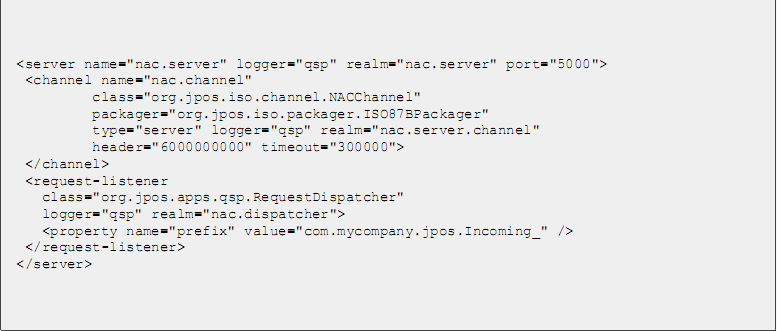


### <server>

<server> dùng để cấu hình cho đối tượng ISOServer



Nó đòi hỏi di nhất một thuộc tính *name* để đăng kí với NameRegistar.Thuộc tính *port* chỉ ra chỉ số *port* mà ISOServer sẽ lắng nghe.



Bạn có thể tham chiếu đến đối tượng ISOSERVER bằng hàm static ISOSERVER.getServer(string name).

Đối tượng *maxSessions* quy định số *session* tối đa có thể kiểm soát động thời bởi thể hiện của <server> này.

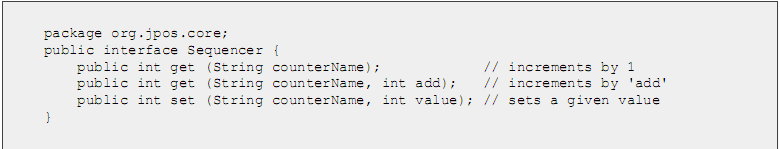
Chú ý: ISOSERVER sử dụng một ThreadPoll để kiểm soát các kết nối một cách đồng bộ.

Mội tiểu trình trong TheadPool sẽ kiểm soát các thông điệp đến sau đó chuyển chúng tới các ISORequestListener theo cách đồng bộ. Trách nhiệm của ISORequestListener là dùng một ThreadPool khác để kiểm soát một cách đồng thời. hay các giao dịch dài hơi.

### <sequencer>

Các ứng dụng thường yêu cầu một chuỗi các số tự tăng chẳng hạn để định nghĩa cho thông điệp.(trace number hoặc retrival number)

Sau đây là intreface của Sequencer:

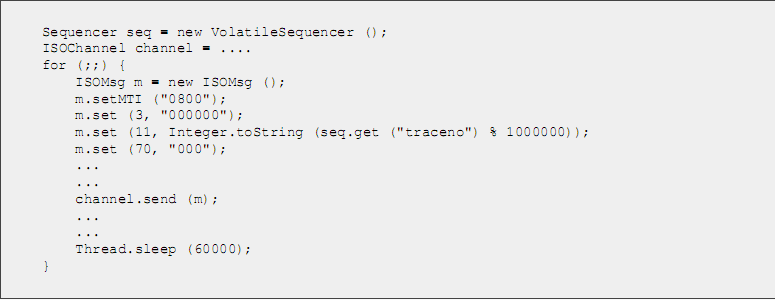


Có 2 loại Sequencer:

VolatileSequencer : Mất đi khi tắt JVM do được lưu trên bộ nhớ.

ReliableSequenser : được lưu trên ổ cứng.

Ví dụ sử dụng:



# Q2

Việc tích hợp tất cả các file cấu hình trong một file làm cho việc cấu hình trở nên cồng kềnh.Vì thế Jpos đã nghĩ đến chuyện phải tách nó ra.

Một số component hỗ trợ Reconfiguration nhưng cũng có một số không hỗ trợ kết quả là khi có một sự thay đổi trong hệ thống thì phải khở động lại QSP.

Khi chúng ta ngắt file cấu hình hoặc thêm vào một user library mới chúng ta phải khởi động lại hệ thống.

Vì vậy Jpos quyết định một file cấu hình trên một component và một vòng lặp đơn giản để thực thi các component đó.

## Biên dịch Q2 [[2]](#footnote-3)

Khi chúng ta khởi chạy Q2 thì nó sẽ kiểm tra thư mục delog để phát hiện file cấu hình cho các component cũng như thư mục deloy/lib cho các file jars mới.

Bây giờ chúng ta tìm hiểu cách mà Qbean hoạt động.

Qbean đại diện bằng đối tượng QbeanSupport. QBeanSupport thực chất là một service và thực sự mang “bản chất” của một service.QBeanSupport cung cấp cho bạn 4 hàm và 4 hàm này có thể override lại ở các lớp con thừa kế.

initService()

startService()

stopService()

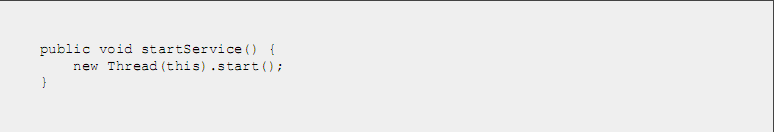
destroyService()

Những phương thức này cung cấp các kiểm soát tới file cấu hình xml (đặt trong thư mục deloy)

Thêm vào đó initService(),startService(),stopService(),destroyService() có thể bắt và log lại tất cả các ngoại lệ trong quá trình thực thi.

Bằng cách mở rộng QBeanSupport bạn không cần phải thực thi tất cả các hàm trong vòng đời của QBean.

Nếu bạn muốn khởi tạo lớp QBean là một đối tượng Runnable thì đơn giản bạn cho đối tượng của bạn chạy trên một Thread mới.



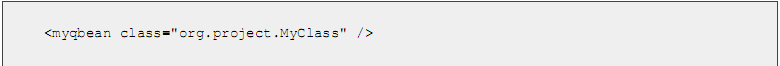
JMX yêu cầu mở rộng các interface \*MBean để các thuộc tính thì sẽ được quản lý bởi JMX và thông qua MBeanServer.

Các phương thức setPersist(Element) và Element getPersist() được cung cấp bởi QbeanSupport để QBean có thể nhận về file cấu hình xml mà nó sở hữu từ đó có thể duy trùy kết nối cũng như nhận biết các sự thay đổi trên file cấu hình đó.

## QBean description

File xml của Qbean cực kì linh động.

Một cấu hình đơn giản nhất của qbean với tên thành phần và thuộc tính class chỉ tới lớp thực hiện.

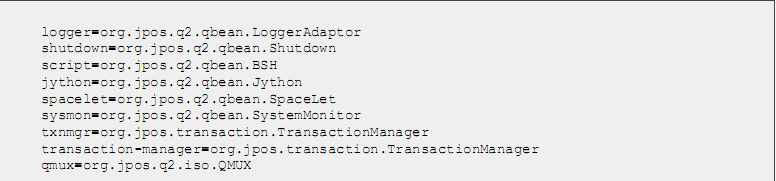


Lớp org.project.MyClass của bạn được giả định rằng thực thi interface Qbean bằng cách dễ dàng thực hiện bằng cách mở rộng lớp org.jpos.q2.QBeanSupport.

Q2 sẽ cố gắng khởi tạo thể hiện lớp của bạn và đăng ký chúng với MbeanServer thông qua JMX dưới tên “Q2: type=qbean name=mybean”

Điều quan trọng là nếu lớp của bạn có phuơng thức void setName(string name) thì Q2 sẽ đăng kí lớp của bạn với NameRegistrar

Nếu bạn không chỉ ra thuộc tính class, Q2 sẽ cố gắng lấy những lớp thích hợp trong file QFactory.properties tương thích với tên thẻ (ví dụ là mybean như trên).



Vây để tắt Qbean thì đơn giản bạn chỉ cần thiết kế file cấu hình shutdow.xml như sau.



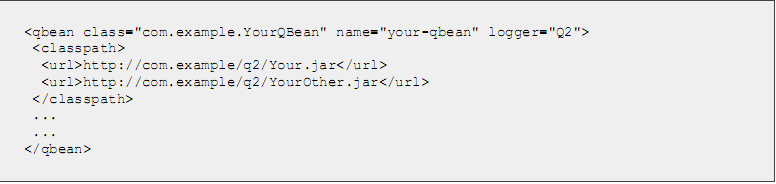
Điều này tương đương với <shutdown class=”org.jpos,q2.qbean.Shutdown />

Một điểm rất hay của Jpos là file cấu hình xml có thể có bất kỳ định dạng nào bằng cách thực thi org.jpos.core.XmlConfiguaration interface. (Q2 sẽ đưa JDom Element tới QBean để nó có thể truy xuất bất kì thành phần nào của file cấu hình xml)

## Q2 Dynamic class loading

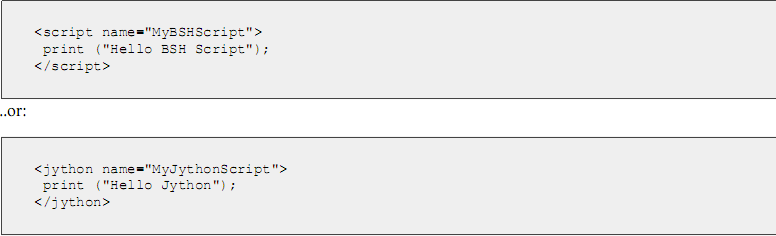
Nếu bạn phải thêm bất kì chức năng mới nào vào hệ thống đang chạy thì bạn hãy biên dịch nó trở thành file .jar và để ở thư mục deloy/lib và sau đó đặt cấu hình xml ở deloy nhầm giúp Q2 khởi động QBean của bạn.

Ngoài ra không chỉ nạp các fle cố định có trên máy tính của bạn Q2 còn có thể nạp các file jar từ các host trên mạng.

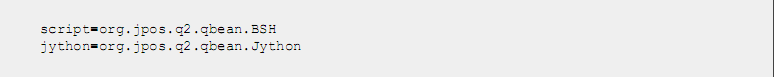


## Q2 Script

Q2 cung cấp những cách dễ dàng để chạy BSH hay JyThon-Script



Nếu bạn nhìn vào org.jpos.q2.Qfactory.properties chúng ta sẽ thấy rằng :



script chỉ tới class org.jpos.q2.qbean.BSH và jython chỉ tới class org.jpos.q2.qbean.Jython

# Tài liệu tham khảo

[1] JPos Programming Guide - Alejandro Revilla – jpos.org

[2] <http://java.sun.com/developer/technicalArticles/J2SE/jmx.html> - Sun Development Network

1. <http://java.sun.com/developer/technicalArticles/J2SE/jmx.html> [↑](#footnote-ref-2)
2. Xem lại file biên dịch Jpos trên window.docx [↑](#footnote-ref-3)