it20one-service-framework v.20050204

Project Documentation

IT20one **07 February 2005**

Table of Contents

1.1 Main

What is YAAFI?

The Yet Another Avalon Framework Implementation.

YAAFI is a light-weight implementation of a service framework using the Avalon service lifecycle interfaces. There are a few other implementations out there such as Excalibur, Fortress and most notably Merlin but YAAFI gives you a lot of bells and whistles with minimal baggage.

What we left out to

- logger manager implementation
- run-time instrumentation to monitor application health
- service implementation versioning
- service initialization in a background thread
- support for singleton lifestyle only

1.2 Overview

What is YAAFI?

The Yet Another Avalon Framework Implementation.

YAAFI is a light-weight implementation of a service framework using the Avalon service lifecycle interfaces. There are a few other implementations out there such as Excalibur, Fortress and most notably Merlin but YAAFI gives you a lot of bells and whistles with minimal baggage.

What we left out to

- logger manager implementation
- run-time instrumentation to monitor application health
- service implementation versioning
- service initialization in a background thread
- support for singleton lifestyle only

1.3 Lifecycle

Service Lifecycle

The service lifecycle contains of a bunch of interfaces covering the following aspects of a service

- incarnation
- reconfiguration
- decommissioning

These interfaces are the contract between your service implementation and the service container. And this is the reason why we can deploy a service implementation using different service framework implementations such as Excalibur or Merlin.

1.3.1 Incarnation

Incarnation

The incarnation of a service covers the creation and configuration of a service

The following methods are invoked:
- Constructor()
- LogEnabled.enableLogging(Logger)
- Contextualizable.contextualize(Context)
- Serviceable.service(ServiceManager)
- Configurable.configure(Configuration)
- Parameterizable.parameterize(Parameters)
- Initializable.initialize()
- Executable.execute()
- Startable.start()

The good news are that you don't have to implement all these interfaces if you have a simple service. The bad news are that you might need all of this interfaces in a complex application ... :-)

Constructor()

This doesn't come as a surprise

LogEnabled.enableLogging(Logger)

Here you get the logger for your service implementation. This is again an interface to an implementation of a logger provided by the caller of the service framework.

Contextualizable.contextualize(Context)

The context contains information about your application environment. The following entries are guaranteed to be available since they are supplied by YAAFI

Name	Type	Description
urn:avalon:home	File	The home directory of the application. This is usually the current working directory or WEB-INF. It is assumed that your application has read access.
urn:avalon:temp	File	The temp directory of the application. It is assumed that temporary files will be dumped there and therefore read/write access is required.

Name	Type	Description
componentAppRoot	String	The absolute path home directory of the application again. This is provided for backward compatibility with Fulcrum and might be depracted in the future.

Serviceable.service(ServiceManager)

At this point you get a reference to the service container. This is the right moment to lookup all dependent services just to make sure that everything is fine.

Configurable.configure(Configuration)

A common task is to access configuration information whereas the Configuration instance is a light-weight XML DOM tree. This means you can use nested XML files for the configuration of your service.

Parameterizable.parameterize(Parameters)

Quite frankly I'm not sure why this method is needed. The only reason I can think of is a command-line application ...

Initializable.initialize()

This method is used for initializing your service implementation since you have all your configuration information by now.

Executable.execute()

If the component implements Executable the execute method will be invoked before the component instance is exposed to any other component.

Startable.start()

The Startable interface is used by any component that is constantly running for the duration of its life.

1.3.2 Reconfiguration

Decommision

The reconfiguration of a service covers the following methods

- Suspendable.suspend()
- Reconfigurable.reconfigure(Configuration)
- Suspendable.resume()

Suspendable.suspend()

Suspend the service since it is guarenteed that no client will invoke the service.

Reconfigurable.reconfigure(Configuration)

Reconfigure the service with the Configuration instance.

Suspendable.resume()

Resume the service - afterwards clients will invoke the service again.

1.3.3 Decommision

Decommision

The decommision of a service covers the shutdown procedure a service

- Startable.stop()
- Disposable.dispose()
- Finalizer

Startable.stop()

Stop all of the service activities since it is guaranteed that no client will invoke the service.

Disposable.dispose()

Free all resources hold by the service implementation.

Finalizer

Well, it might be never called but all resouces have been released before.

1.4 Services

YAAFI Services

YAAFI comes already with the following services since they are generally useful and do not add any dependencies

Name	Desciption
ServiceManagerService	Keeps a reference to the YAAFI instance
SystemPropertyService	Copies system properties during startup

1.4.1 ServiceManagerService

Overview

This service keeps an YAAFI instance.

Configuration

Role Configuration

```
<role
    default-class="org.apache.fulcrum.yaafi.service.servicemanager.ServiceManagerService"
    early-init="true"
/>
```

Usage

ServiceManagerService.getServiceManager() returns you an instance of ServiceManager which allows to lookup other services.

```
FOO foo = (FOO) ServiceManagerService.getServiceManager().lookup("FOO");
```

1.4.2 SystemPropertyService

Overview

This service copies entries from the configuratio.xml into the SystemProperties.

Quite useful since you can avoid tinkering with system properties in start scripts.

Configuration

Role Configuration

```
<role
    name="org.apache.fulcrum.yaafi.service.systemproperty.SystemPropertyService"
    shorthand="SystemPropertyService"
    default-class="org.apache.fulcrum.yaafi.service.systemproperty.SystemPropertyServiceImpl"
    early-init="true"
/>
```

Component Configuration

```
<SystemPropertyService>
    <property name="FOO">BAR</property>
</SystemPropertyService>
```

Usage

This service does not expose any methods

1.5 How To

How To

How to write my own service?

- Write your service interface and implementation using the Avalon Lifecycle interfaces.
- Add an entry to the role configuration file. This entry contains the information how YAAFI can instantiate and access the service
- Add an entry to the component configuratino file if you need to configure your service.

How can I embed YAAFI in an application?

The embedding is done by creating a YAAFI instance using the ServiceManagerFactory.create() method.

The following example creates a fully initialized and running YAAFI container with the given configuration parameters using a LOG4J logger.

```
ServiceContainer manager = null;
Logger logger = new Log4JLogger( org.apache.log4j.Logger.getLogger("YAAFI");
manager = ServiceManagerFactory.create(
    logger,
    "roleConfiguration.xml",
    "componentConfiguration.xml",
    "parameters.xml"
    );
```

At the end of day you have to terminate YAAFI

```
manager.dispose();
```

How can I embed YAAFI into Turbine?

In the 'contrib' directory there is a ready-to-use Turbine service which needs the following configuration (for Turbine 2.2)

```
services.YaafiComponentService.classname=org.apache.turbine.services.yafficomponent.TurbineYaafiComponentService
services.YaafiComponentService.componentRoles=./conf/componentRoles.xml
services.YaafiComponentService.componentConfiguration=./conf/componentConfiguration.xml
services.YaafiComponentService.parameters=
```

1.6 Todo's

TODO

Support for encrytped configuration files

Encryption/decryption of role and component configuration file using BlowfishJ from Markus Hahn (see http://www.lassekolb.info/bfacs.htm).

Adding a component personality

Based on the component personality a proper Context will be created (e.g. 'merlin', 'fortress', 'phoenix') and passed to the component. This would allow to reuse Avalon service written for other containers (I'm not jokink)

Adding a container personality

This would allow to embed YAAFI cleanly in another Avalon Container, e.g. within James.

Add automatic reconfiguration

It would be nice to poll the configuration files and trigger a reconfiguration if they were changed