alma.acs.container

Interface ComponentLifecycle

All Known Implementing Classes:
HelloComponent

public interface **ComponentLifecycle**

Interface that every component has to implement.

The methods are used by the container to start and stop the component, that is, to manage its lifecycle.

Note that the constructor of the component implementation should be very small, preferably empty. Use the methods `initialize()` and `execute()` instead.

TODOs:

- we still have to think about the exception mechanism.
- new methods for more advanced behavior, e.g. restarting a component transparently for its client, will be added later. Their definition requires more analysis, and their implementation will also require changes to the ACS Manager etc.

Author:
hsommer

Method Summary

`void`	`aboutToAbort()` Called when due to some error condition the component is about to be forcefully removed some unknown amount of time later (usually not very much...).
`void`	`cleanUp()` Called after the last functional call to the component has finished.
`void`	`execute()` Called after `initialize()` to tell the component that it has to be ready to accept incoming functional calls any time.

`String`	`getComponentName()` Gets the name
`void`	`initialize()` Called to give the component time to initialize itself.
`void`	`setComponentName(String name)` Sets the name that is assigned at deployment time or dynamically by the framework.
`void`	`setContainerServices(ContainerServices containerServices)` Sets a callback handle to the container.

Method Detail

setComponentName

```
public void setComponentName(String name)
```

Sets the name that is assigned at deployment time or dynamically by the framework.

---

getComponentName

```
public String getComponentName()
```

Gets the name

---

setContainerServices

```
public void setContainerServices(ContainerServices containerServices)
```

Sets a callback handle to the container.

Parameters:

`containerServices` - the interface of the container offering services to the component.

---

initialize

```
public void initialize()
```

Called to give the component time to initialize itself. For instance, the component could retrieve connections, read in configuration files/parameters, build up in-memory tables, ...

Called after `setContainerServices(alma.acs.container.ContainerServices)` but before `execute()`. In fact, this method might be called quite some time before functional requests can be sent to the component.

Must be implemented as a synchronous (blocking) call.

Note: for the more advanced feature of on-the-fly reinitialization (e.g. to read in modified config parameters from the CDB, we could either have a separate method like `reinitialize`, or just call `initialize` again on the running component. While implementing this method, please think about which of these alternatives would be better for you so that you can give qualified feedback later.

---

execute

`public void **execute**()`

Called after `initialize()` to tell the component that it has to be ready to accept incoming functional calls any time.

Examples:

- last-minute initializations for which `initialize` seemed too early
- component could start actions which aren't triggered by any functional call, e.g. the Scheduler could start to rank SBs in a separate thread.

Must be implemented as a synchronous (blocking) call (can spawn threads though).

---

cleanUp

`public void **cleanUp**()`

Called after the last functional call to the component has finished. The component should then orderly release resources etc.

---

aboutToAbort

`public void **aboutToAbort**()`

Called when due to some error condition the component is about to be forcefully removed some unknown amount of time later (usually not very much...).

The component should make an effort to die as neatly as possible.

Because of its urgency, this method will be called asynchronously to the execution of any other method of the component.

---

alma.acs.container

Interface ContainerServices

All Known Implementing Classes:
ComponentAdapter

public interface **ContainerServices**

created on Oct 24, 2002 12:56:36 PM

Author:
hsommer $Id$

Method Summary

`void`	`assignUniqueEntityId(alma.entity.xmlbinding.commonentity.EntityT entity)` Creates a unique id and sets it on the given (binding class) entity object.
`Object`	`getComponent(String componentId, String somethingElseTodo)` The encapsulation of requests for other components is not yet specified.
`Logger`	`getLogger(String subNamespace)` Method getLogger.

Method Detail

getLogger

`public Logger getLogger(String subNamespace)`

Method getLogger.

Parameters:
`subNamespace` - sub-namespace to be chosen by the component. uses same '.' separated format as in java.util.Logger#getLogger(String name); may be null.
Returns:
Logger

assignUniqueEntityId

`public void assignUniqueEntityId(alma.entity.xmlbinding.commonentity.EntityT entity)`

Creates a unique id and sets it on the given (binding class) entity object. The id will be redundantly stored in an encrypted format so that later it can be verified that the id is indeed the one originally assigned.

Parameters:

`entity` - must be freshly created and

---

getComponent

```
public Object getComponent(String componentId,
                           String somethingElseTodo)
```

The encapsulation of requests for other components is not yet specified. Some considerations: 1) this method is necessarily generic and will require a cast to the requested interface. 2) we attempt to come up with some additional way (e.g. a code-generated singleton class) to give type safe access to other components, like done with EJB xxxHome. 3) the Container will shortcut calls for components inside the same process. Like in EJB and CORBA, the implementation must therefore not assume receiving a by-value copy of any parameter from the other component.

```java
package alma.testcomponents;

import java.util.logging.Logger;

import alma.HelloApp.HelloComponentOperations;
import alma.acs.container.ComponentLifecycle;
import alma.acs.container.ContainerServices;

/**
 * @author hsommer
 */
public class HelloComponent implements ComponentLifecycle, HelloComponentOperations
{
    private String m_name;
    private ContainerServices m_containerServices;
    private Logger m_logger;

    /**
     * @see alma.HelloApp.HelloComponentOperations#sayHello()
     */
    public String sayHello()
    {
        m_logger.info("sayHello called...");
        return "hello";
    }



    /**
     * @see alma.acs.container.ComponentLifecycle#aboutToAbort()
     */
    public void aboutToAbort()
    {
    }

    /**
     * @see alma.acs.container.ComponentLifecycle#cleanUp()
     */
    public void cleanUp()
    {
    }

    /**
     * @see alma.acs.container.ComponentLifecycle#execute()
     */
    public void execute()
    {
    }

    /**
     * @see alma.acs.container.ComponentLifecycle#getComponentName()
     */
    public String getComponentName()
    {
        return m_name;
    }

    /**
     * @see alma.acs.container.ComponentLifecycle#initialize()
     */
    public void initialize()
    {
    }
    /**
     * @see alma.acs.container.ComponentLifecycle#setComponentName(java.lang.String)
     */
    public void setComponentName(String name)
    {
        m_name = name;
    }

    /**
     * @see
alma.acs.container.ComponentLifecycle#setContainerServices(alma.acs.container.ContainerServices)
     */
    public void setContainerServices(ContainerServices containerServices)
    {
        m_containerServices = containerServices;
        m_logger = m_containerServices.getLogger("hello");
    }

}
```