FOM%Cocoon

FOM: The Cocoon Object

The Cocoon object is the only possible way the flow can communicate with Cocoon. In a sense, it's a gateway between the two realms. This protects the flow from abusing the Cocoon internals (for example, there is no way the flow can compose a sitemap pipeline since the cocoon object doesn't give it that power)

Properties

cocoon.request -> the request object

cocoon.response -> the response object

cocoon.context -> the context object

cocoon.session -> the session object

cocoon.log -> the log object

cocoon.parameters -> the sitemap parameters

Methods

void sendPage(uri,map)

returns control to the sitemap, invoking the pipeline that will match the given URI and passing the given map as model.

void sendPageAndWait(uri,map)

same as above, but creates one (or more) continuation objects and makes their IDs available as part of the model passed.

void processPipelineTo(uri,map,outputStream)

invokes the pipeline that will match the given URI, passing the map as the model but connecting the output of the pipeline to the given output stream.

Note: Additionally you can find the probably more useful helper methods

- `processToStream(uri,viewData,outputStream)`
- `processToSAX(uri,viewData, contentHandler)`
- `processToDOM(uri,viewData)`

in the class `o.a.c.components.flow.util.PipelineUtil`. *For more info read the announcement thread or the JavaDocs. In the announcement thread you can also read, that the* `processPipelineTo(uri,map,outputStream)` *method might be deprecated in the future.*

void redirectTo(uri)

triggers a client-side redirect to the given URI.

Object getComponent(id)

obtains the component indicated by the given ID.

releaseComponent(component)

release pooled components.

load(script)

load scripts dynamically.

Methods that are planned for future releases but are currently not part of the FOM

void addEventListener(eventName,eventHandler)

adds an event listener to the given event name (for example, session expiration).

void removeEventListener(eventName,eventHandler)

removes the given handler from listening the tiven event name.

These two methods define the hooks between the FOM and the FOM Event Model that is scheduled to be designed and implemented for future FOM versions.

Methods that were left out

input/output module support

the reason for the first goes together with callAction(). Input/output modules were designed to overcome limitations in the scriptability of the sitemap.

see Flow%LegacySupport

Map callAction(name,map)

invoques the action indicated by the given name and pass the given map as model

NOTE (SM): *I personally believe that the getComponent() method removes all needs for the callAction() method. I foresee a future where the callAction() method will be deprecated. I would personally go the extra mile and avoid having it there altogether, but since there is no easy way to plugin new avalon components at runtime (at least, not as easier as plugging in different actions into the sitemap), I'm in favor of leaving it for now, until 'real blocks' will make it unnecessary.*

NOTE (RP): *I removed this method as there haven't been any objections. But before we implement the FOM there will be a vote on this. So don't worry!.*

see Flow%LegacySupport

Session getSession()

NOTE(SM/RR): *both Ricardo and I believe that the flow should always be associated with a Session. Thus the use of the semantics "getSession" instead of "createSession". We are, in fact, against the concept of having the flow behave differently when the session has been created or when it has not been. This implicit behavior is potentially very dangerous from a user perspective and should be avoided. Moreover, it has been pointed out how continuations can be see as a way to "extend" sessions rather than replacing them. This would allow us to reuse the session-handling machinery already in place for things like load-balancing and fault-tollerance.*

Note(RP): *I removed it in favour of the session object as property of the cocoon object.*