JOGOmap

NAME

JOGOmapSpider - a JavaScript Class for Spidering Web pages

SYNOPSIS

```
<script type="text/javascript" src="CCallWrapper.js"></script>
<script type="text/javascript" src="JOGOmapSpider.js"></script>
<script type="text/javascript">

function JOGOmapPageLoader()  {
  // customize per loader
}

JOGOmapPageLoader.load =
function JOGOmapPageLoader_loadPage(/* String */ url, /* String */ referer) {
  // customize to initiate loading a new page
};

JOGOmapPageLoader.cancel =
function JOGOmapPageLoader_cancel() {
  // customize to cancel loading a page
};

JOGOmapPageLoader.getDocument =
function JOGOmapPageLoader_getDocument() {
  // customize to obtain a reference to
  // the loaded document.
};

var aUrl = 'http://example.com';
var aRestrictUrl = true;
var aDepth = 1;
var aPageLoader = new JOGOmapPageLoader();
var aOnLoadTimeoutInterval = 120;

var spider = new JOGOmapSpider(aUrl,
                    aDomain,
                    aRestrictUrl,
                    aDepth,
                    aPageLoader,
                    aOnLoadTimeoutInterval,
                    aExtraPrivileges,
                    aRespectRobotRules,
                    aUserAgent);
```

```
spider.mOnStart       = function() { /* customize */ };
spider.mOnBeforePage  = function() { /* customize */ };
spider.mOnAfterPage   = function() { /* customize */ };
spider.mOnPageTimeout = function() { /* customize */ };
spider.mOnStop        = function() { /* customize */ };
spider.mOnPause       = function() { /* customize */ };
spider.mOnRestart     = function() { /* customize */ };

spider.run();

</script>
```

DESCRIPTION

JOGOmapSpider is a JavaScript class which is intended to be used as in-browser web spidering application. JOGOmapSpider's results are processed and stored in Resource Document Format (RDF) and can be analyzed, transformed and displayed in an easy way.

JOGOmapSpider uses a helper class to abstract and manage requests to load external documents which must implement the following methods:

* load(String url)
* cancel()
* Document getDocument()

Constructor

```
function JOGOmapSpider(/* String */ aUrl,
        /* String */ aDomain,
        /* Boolean */ aRestrictUrl,
        /* Number */ aDepth,
        /* JOGOmapPageLoader */ aPageLoader,
        /* Seconds */ aOnLoadTimeoutInterval,
        /* Boolean */ aExtraPrivileges,
        /* Boolean */ aRespectRobotRules,
        /* String */ aUserAgent)
```

Constructs an instance of JOGOmapSpider which will begin spidering at the location aUrl, optionally restricting visited pages to include the string aDomain (if aRestrictUrl is true) up to a depth of aDepth while waiting for aOnLoadTimeoutInterval seconds for each page to load before timing out. aPageLoader is a special helper object used to manager loading external pages. aExtraPrivilges can be set to true in Gecko in order to use netscape.security.PrivilegeManager to request additional security privileges in order to obtain cross-domain access to loaded page's DOM.

If aRespectRobotRules is true, JOGOmapSpider will load robots.txt from each domain it visits using isRobotBlocked.js and will not add pages to its pending list if they are blocked by the robot rules.

aUserAgent is currently passed to isRobotBlocked() however is not (yet) used to set the user agent of the client. If not passed as an argument, aUserAgent is assumed to be 'Gecko/'.

Properties

mUrl
String containing the initial URL where JOGOmapSpider will begin. If the initial value obtained from aUrl in the constructor does not begin with 'http://' or 'https://', 'http://' will be prepended to aUrl before it is saved in mUrl.

mDomain
String which will be used (if mRestrictUrl is true) to filter links in loaded documents. If aDomain is not specified, aDomain will be set to the value of aUrl with any leading protocol ('http://' or 'https://') removed.

mRestrictUrl
Boolean which if true will cause JOGOmapSpider to only follow links which contain the String mDomain.

mDepth
Number indicating the depth (number of clicks) to which JOGOmapSpider will follow.

mCallWrapperOnLoadPageTimeout
An instance of CCallWrapper which is used to detect timeout conditions when loading pages.

mOnLoadTimeoutInterval
Number (in milliseconds) that JOGOmapSpider will wait for a page to load before calling mCallWrapperOnLoadpageTimeout.

mExtraPrivileges
Boolean which invokes netscape.security.PrivilegeManager.enablePrivilege to enable cross-domain spidering.

mRespectRobotRules
Boolean which determines if JOGOmapSpider will load and check urls of pages to be loaded against a site's robots.txt file using isRobotBlocked().

mUserAgent
String which is passed to isRobotBlocked() to check the robot rules however currently defaults to 'Gecko/'.

mPagesVisited
An Array of Strings of the URLS visited by JOGOmapSpider.

mPagesPending
An Array of CUrl instances representing the pages to be visited.

mPagesHash
A hash of URL strings which are used to determine pages which have already been visited and which do not need to be revisited.

mState
A String indicating the state of the spider. One of 'ready', 'running', 'pausing', 'paused', 'timeout', 'stopping', 'stopped'.

mCurrentUrl
An instance of CUrl representing the page currently being loaded. CUrl is an internal class with two properties (mDepth - the depth of the URL and mUrl a String containing the actual URL).

mDocument
A reference to the currently loaded Document. This property is set to null in loadPage() and set to the currently loaded document by onLoadPage().

Controller Methods

These methods are used to control the spider.

run
run() begins spidering at mUrl. run() enters the 'running' state, then calls the user specified mOnStart() before loading any pages. mOnStart controls the execution of the spider by returning true to begin loading pages or false to prevent loading pages and re-enter the 'ready' state.

restart
If the spider is in either the 'paused' or 'timeout' states, restart() will enter the 'running' state, then call the user specified. mOnRestart() before loading any pages. mOnRestart controls the execution of the spider by returning true to begin loading pages or false to prevent loading pages and enter the 'paused' state.

pause
If mCallWrapperOnLoadPageTimeout is not null, pause() enters the 'pausing' state and then returns to allow either onLoadPage or onLoadPageTimeout to complete before entering the 'paused' state. If mCallWrapperOnLoadPageTimeout is null, pause() enters the 'paused' state then calls mOnPause()

to control the execution of the spider by returning true to stay in the paused state or false to enter the 'running' state and call loadPage() to continue spidering.

stop
If mCallWrapperOnLoadPage is not null, stop() enters the 'stopping' sate and then returns to allow either onLoadPage or onLoadPageTimeout to complete before entering the 'stopped' state. If mCallWrapperOnLoadPageTimeout is null, stop() enters the 'stopped' state, then calls mOnStop() to control the execution of the spider by returning true to stay in the stopped state or false to enter the 'running' state and call loadPage() to continue spidering.
If stop() enters 'stopped' state the RDF data collection is processed by an XUL template. The result of the XUL processor is displayed in an extra window containing the spidered site structure as a tree.

Internal Methods

These methods are used internally by JOGOmapSpider.

init
init() is a private method used to initialize the JOGOmapSpider and the required RDF services. A in-memory RDF datasource is perpared and initialized for holding the spider's data.

addPage
addPage(aUrl) is a private method called by onLoadPage() to add the URL aUrl to the list of pending URLs to be visited. addPage limits the pages to be visited by rejecting any URLs which

* do not have either 'http' or 'https' protocols
* have already been added
* exceed the specified depth
* if mRestrictUrl is true, rejecting any URLs which do not contain mDomain
* end in one of a set of 'bad extensions'. These extensions are currently hard-coded into addPage

loadPage
loadPage() initiates the process of loading the next page from the list maintained in mPagesPending. If the spider is not in the 'running' state, loadPage does nothing. loadPage removes the next CUrl object from the mPagesPending stack and calls stop() if there are no more pages pending. mOnBeforePage() is called to control execution of the spider by returning true to begin the load or false to prevent the load and to enter the 'paused' state. If mOnBeforePage prevents the load, the CUrl object is placed back on the mPagesPending stack.
An instance of a CCallWrapper for onLoadPageTimeout is created, saved to mCallWrapperOnLoadPageTimeout and asynchronously executed to detect page timeouts and the mPageLoader's load() method is called to initiate the loading of the next page.

onLoadPageTimeout
If the mCallWrapperOnLoadPageTimeout has not been cancelled by onLoadPage, onLoadPageTimeout will execute.

If the spider is in the 'pausing' state, pause() is called and the spider enters the 'paused' state. If the spider is in the 'stopping' state, stop() is called and the spider enters the 'stopped' state.
The user defined mOnPageTimeout() is called to control the execution of the spider. If mOnPageTimeout returns true the spider will enter the 'timeout' state, otherwise the spider enters the 'running' state and loadPage will be called to continue spidering the site.

onLoadPage
onLoadPage() is called by the implementation of the page loader when each page has completed loading. onLoadPage will cancel mCallWrapperOnLoadPageTimeout. If the spider is in the 'pausing' state, pause() is called and the spider enters the 'paused' state. If the spider is in the 'stopping' state, stop() is called and the spider enters the 'stopped' state. onLoadPage adds the current mCurrentUrl to the mPagesVisited array, saves a reference to the loaded document in mDocument and call addPage for each link, frame and iframe in the document.
onLoadPage calls the user specified mOnAfterPage to control the execution of the spider. If mOnAfterPage returns true, onLoadPage will call loadPage to begin loadingthe next page. if mOnAfterPage returns false, the spider will enter the 'paused' state.

cancelLoadPage
cancelLoadPage() calls mCallWrapperOnLoagePageTimeout.cancel() to cancel any pending page load timeouts and mPageLoader.cancel() to cancel any pages being currently loaded by the page loader and puts the current URL back on the mPagesPending stack.

seqContainer
seqContainer() creates an RDF container that can be used to store RDF statements. It is used by mOnAfterPage to process and store the spidered data in RDF.

Processing Methods

The following methods are intended to be customized in order to provide extended functionality to the spider.

mOnStart
mOnStart() is called by run(). Return true to begin loading pages, false to re-enter the 'ready' state.

mOnBeforePage
mOnBeforePage() is called by loadPage(). Return true to continue loading the page, false to enter the 'paused' state. Note that when onBeforePage is called, mCurrentUrl will contain the CUrl object for the next page to be loaded.

mOnAfterPage
mOnAfterPage() is called by onLoadPage after all new links have been added to mPagesPending. Return true to load the next page or false to enter the 'paused' state. Note that when mOnAfterPage is called, mDocument will contain a reference to the currently loaded document.
mOnAfterPage also processes the spidered data from onLoadPage() and puts the data in the RDF in-memory datasource. To create containers for every spidered site mOnAfterPage uses seqContainer().

mOnPageTimeout
mOnPageTimeout() will be called if the page does not complete loading in mOnLoadTimeoutInterval milliseconds. Return true to enter the 'timeout' state, false to attempt to load the next page.

mOnStop
mOnStop() is called by stop. Return true to enter the 'stopped' state, false to attempt to load the next page.

mOnPause
mOnPause() is called by pause. Return true to enter the 'paused' state, false to attempt to load the next page.

mOnRestart
mOnRestart() is called by restart. Return true to load the next page, false to re-enter the 'paused' state.