jakarta.servlet

Interface Ramfile

All Known Implementing Classes:

RAMfileArray

---

public interface **Ramfile**

javax.servlet.Ramfile interface and org.apache.catalina.core.RAMfileArray class were developed and written by Mr. Samuel A Marchant ,
July / August 2020 Sydney Australia nicephotog@gmail.com (this preview October 2020 - this class reshuffled of methods between this modification set of three classes)
javax.servlet.Ramfile / jakarta.servlet.Ramfile is implemented by **org.apache.catalina.core.RAMfileArray**

The RAM file array is "a multi dimensional byte[][] array" used to store "commonly used / served" files in memory as a ready indexed byte[] array to place immediately into servlet output, e.g. web page header GIF or JPG e.t.c. , UI web page image parts , or as much PDF , .DOC or .ZIP documents or archive for download, all that require expensive disk in/out sequence as either hardware BUS hops and file-system lookup or disk buffer access.
note that if the output of the server is customised to individual requests and not continuously the same choice of resource then either a servlet or a JSP page should be "threadsafe".

For most, with browsers and the IMG html tag, the browser UA does not need a file extension just the bare bytes of the file and usually accross all browser UA know how to handle JPEG, BMP,PNG and GIF images.

An instance of this Ramfile interface must be obtained from the ServletContext object.

This interface is the user end set of tools of the RAMfileArray class that is started when the Tomcat Server itself is started.
It is called from within a Servlet application context.

There is , only one instance of RAMfileArray each loaded application (e.g. .war) and to use it must be instantiated first before all other activities.

The best method to start and load these persistent classes is to use an Application Context listener in their application
EXAMPLE:

```
import jakarta.servlet.ServletContextEvent;
import jakarta.servlet.ServletContextListener;
import jakarta.servlet.Ramfile;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletContext;
import jakarta.servlet.RamStringB64;

public class Contextinit implements ServletContextListener{

public void contextInitialized(ServletContextEvent sce) {

// initialise required vars
String loadname = "C:\\Program Files (x86)\\Apache Software Foundation\\Tomcat 10.0\\webapps\\RAMFileLoader\\WEB-INF\\filefile.txt";
String load64name = "C:\\Program Files (x86)\\Apache Software Foundation\\Tomcat 10.0\\webapps\\RAMFileLoader\\WEB-INF\\file64file.txt";

Ramfile ramm = sce.getServletContext().getRamFileInstance();
RamStringB64 r64 = sce.getServletContext().getRamStringB64Instance();
try {
java.io.File effar = new java.io.File(loadname);
ramm.loadPreLoadingList(effar);
//
effar = new java.io.File(load64name);
r64.loadPreLoadingListB64(effar,false);
//
// https://docs.oracle.com/javaee/6/api/?javax/servlet/ServletContextListener.html
System.out.println("Completed context listener calls"); // to STDOUT log in Apache Tomcat
}catch(Exception ecces){
ecces.printStackTrace();
}
}//enmeth

public void contextDestroyed(ServletContextEvent sce) {
// on destroy
}//enmeth
```

} // enclss

A good simple test of the time required for the files to load onto the array at initialisation is using all the documents for the array in a folder and copy and paste all of them to another folder, the time it requires to complete from the start of the copying sequence is approximately the time to wait for the files to complete loading onto the RAMfileArray[][].

To reload the RAMfileArray will require stopping and undeploying the application that it is operated through because when it is loaded it is impossible to change that.
It is only for secure purpose that it is not made easier (attempting reload of array when it has been loaded only causes reload attempt to bypass), so it is suggested, only a loader servlet, an inventory servlet and a byte-download-output servlet are present in the application it is initiated for use from. *An instance of the interface is obtained this way using getRamFileInstance() method of javax.servlet.ServletContext interface:*

```
// first way to obtain it
ServletContext sxt = ((ServletConfig)this.getServletConfig()).getServletContext();
Ramfile ramm = sxt.getRamFileInstance();

// Second way to obtain it from the ServletContext
Ramfile ramm = getServletContext().getRamFileInstance();

// a file stored on the array at some index
int array_file_index = 27;

// get the file as a byte[] array from the Ramfile object
byte[] attach_download = ramm.getRAMByteArray(array_file_index);
long ram1 = ramm.getOutWriteRAMbyteLength(array_file_index);
```

Example: To get the application context from another application and write an image as byte[] array to a stream

1. First the META-INF/context.xml in both applications must be changed. for crossContext ability

"WebApplication3" will obtain the Ramfile object interface instance from the application with the RAMfileArray instance, "RAMFileLoader" and bring back an image byte[] array in the "RAMFileLoader" application

```
<Context path="/RAMFileLoader" crossContext="true">
<WatchedResource>WEB-INF/web.xml</WatchedResource>
</Context>

<Context path="/WebApplication3" crossContext="true">
<WatchedResource>WEB-INF/web.xml</WatchedResource>
</Context>

Ramfile ramx = request.getServletContext().getContext("/RAMFileLoader").getRamFileInstance();
byte[] attach_download = ramx.getRAMByteArray(array_file_index);
long ram1 = ramx.getOutWriteRAMbyteLength(array_file_index);
```

User Agent browser page image tag in the htlm would appear something such as this for the byte array outputing downloader servlet call

```
<img src="http://somedomain.com/WebApplication3/RamfileByteOutput?index=37&">
```

..."The RAM file array is "a multi dimensional byte[][] array" used to store commonly used files in memory, e.g. web page header GIF or JPG e.t.c. , UI web page image parts , or as much PDF , .DOC or .ZIP documents for download all that require expensive disk in/out sequence"...

By storing commonly used files on the array they can be accessed and sent without jamming the speed of the server and more critical I/O disk services such as databases, this trick is particularly good in small single site personal business servers that do not have high BUS speed between devices and slower CPU clock,too, the number of read heads and number of platters in a disk is of signifigance, the less of these the slower and more blocked disk IO is and the slower and more compounded the start stop thread pipelining effect on serving data to the network will be.
Because ISP provider communication requires the number of bytes can be collected and ready to receive, **the best assurance for purpose here is to use the write(byte[]) method** inherited from the ServletOutputStream, this ensures the whole send size of the array is known to the system controlling the stream.

If the Tomcat server is X32 (not X64) it must be set to use 1024 Mb of RAM and the other settings such as Object stack 512 Mb and Thread 256 Mb so the server and JVM will keep the object instance in RAM without disk buffering.
This will allow around 100 MegaBytes of RAM to hold the files implicitly to memory without buffering.
If the Tomcat is an X64 then the maximum RAM for storing RAM loaded files is 1/10th of the total RAM assigned to the Server.

In the real world scenario this interface and its class object in the server are used with a servlet and its response output stream e,g in a browser, the
example **<IMG src="http://localhost:8080/WebApplication3/NewServlet1?index=37&">**
image tag in the user agent does not look at the file-type to determine the image type, it looks in the known binary header part of the file to determine what to do.

CGI http headers for sending JPEG image file bytes directly to the html src tag attribute from a servlet:

```
response.setContentType("image/jpeg");
response.setContentLength((int)ram1);
response.addHeader("Content-Transfer-Encoding"," binary");
response.addHeader("Content-Disposition"," inline; filename=\""+ramm.getLoadedFileNameAtIndex(1)+"\""); // straight into SRC= of the IMG tag
```

Too, a server sending data has its own specifications, and in J2EE Java servers, the response stream sent to a browser is implicitly to

output "text/html" Content-Type by default as either an HTML page or a JSP page.This means *any other data stream type sent* **must first explicitly change this server default content type by using setContentType() method of the response object**

Example of a "file download" to a browser:

```
response.setContentType("application/octet-stream");
response.setContentLength((int)ram1);
response.addHeader("Content-Transfer-Encoding"," binary")
// "attachment" pops a download box in the browser , default for Content-Disposition header is "inline"
response.addHeader("Content-Disposition",(" attachment; filename=\"+array_file_index+".jpeg\""));

// get the relevant server pipeline bundle stream for response
javax.servlet.ServletOutputStream speem = response.getOutputStream();
// write(byte[]) inherited method of ServletOutputStream
speem.write(attach_download);
```

See also: an associate similar mechanism ***javax.servlet.RamStringB64 / jakarta.servlet.RamStringB64*** and ***org.apache.catalina.core.RAMStringB64Array***

See also: an associate similar mechanism ***javax.servlet.RamHttpTools / jakarta.servlet.RamHttpTools*** and ***org.apache.catalina.core.RAMfilingHTTPtools***

Method Summary

All Methods | **Instance Methods** | **Abstract Methods**

Modifier and Type	Method and Description
`RamfileExtensible`	**`getRAMfileExtensible`**`()` Obtain the **jakarta.servlet.RamfileExtensible interface** for access to a dynamic byte[] that can be changed during application operation.
`java.lang.String`	**`getInterfaceReadyFlag`**`()` Returns a status string relating the byte[][] array loaded status.
`boolean`	**`getIsInterfaceInstanceReady`**`()` Returns true if RAMfile array is completely loaded.
`java.lang.String`	**`getLineCRLF`**`()` Returns a newline character, either a default machine string "newline character" has been set or the compliment set method of this get method can be used to custom set a "newline character".
`java.lang.String`	**`getLoadedCanonicalFileNameAtIndex`**`(int idxName)` Gets the full path and name of the file at a particular index on the Ramfile array.
`java.lang.String`	**`getLoadedFileNameAtIndex`**`(int idxName)` Gets the name only of the file at a particular index on the Ramfile array.
`java.lang.String`	**`getLoadedRAMFileNamesInventory`**`(boolean HtmlTXT)` Print out tool. Gets a String of names of the files were sent to be loaded onto the RAM byte[][] array.
`long`	**`getOutWriteRAMbyteLength`**`(int arrLenIdX)` Returns a primitive long of the number of bytes of the particular file loaded at the index.
`byte[]`	**`getRAMByteArray`**`(int idxAr)` Brings back a byte[] array of a loaded file from the specified index on the loaded RAMfileArray byte[][] array.
`int`	**`getTotalLoadedFiles`**`()` Returns the number of files loaded on the RAMFileArray

void	**loadPreLoadingList**(java.io.File setlist) Load the RAMfileArray by what MUST BE a TXT file - A list of full path filenames (separated by commas or by newline characters of the platform the server operates).
void	**loadPreLoadingList**(java.io.File[] fistset) **If an ApplicationContextListener is not used to load these persistent Tomcat server classes on application deployment start , this is the only "recommended way" to load the RAMFileArray (loop the file array first to check if File.exists())** Load the files for the objects RAMfileArray byte[][] array by a java.io.File[] array object from some other program in the server i.e. a special custom servlet.
void	**loadPreLoadingList**(java.lang.String listset) Load the files by a String passed to the RAM file object from another program i.e. *a servlet*. - A list of full path filenames (separated by commas or by newline characters of the platform the server operates).
void	**loadPreLoadingList**(java.lang.String[] fls) Load the files by passing a String array of full path filename elements to the RAM file object the bare full path names as allowed by programming language syntax). It is best to apply String trim method to the ready names as assurance.
java.lang.String	**pingMessageTest**() Returns a String message of version and particulars as output about the RAMfileArray class object for test and debug purpose.
void	**setLineCRLF**(java.lang.String CR_LF) sets a new line character according to chosen requirement.

Method Detail

getRAMfileExtensible

```
RamfileExtensible getRAMfileExtensible()
```

Obtain the **jakarta.servlet.RamfileExtensible interface** for access to a dynamic byte[] that can be changed during application operation.

getIsInterfaceInstanceReady

```
boolean getIsInterfaceInstanceReady()
```

Returns true if RAMfileArray is completely loaded based on whether the files are all loaded ready for use by the string process report steps, not the finish of the loading procedure sequence.

getInterfaceReadyFlag

```
java.lang.String getInterfaceReadyFlag()
```

Returns a status string relating the byte[][] array loaded status. There are four flags that can be used in a servlet to check the applications RAMfileArray instance if loading has been attempted.
UNASSIGNED_ARRAY - nothing occurred, the object is simply a created instance of / with the recently loaded web application.
LOADING_ARRAY - Loading is occurring.
FAILED_LOAD - Some problem has occurred such as an exception, see the Tomcat stderr logs (or server error logs of any other java server it is fitted to the application context of)
ARRAY_LOADED_READY - All loading has finished , the object is now ready to use.

note: For testing the time it will take to load the object to ready, it is not dissimilar to the time it takes on a desktop to copy all files from one folder to another.

getLoadedFileNameAtIndex

```
java.lang.String getLoadedFileNameAtIndex(int idxName)
```

Gets the name only of the file with no path attached at a particular index on the Ramfile array.

getLoadedCanonicalFileNameAtIndex

```
java.lang.String getLoadedCanonicalFileNameAtIndex(int idxName)
```

Gets the full path and name of the file at a particular index of the RAMfileArray[][] from its inventory record filenames array.

setLineCRLF

```
void setLineCRLF(java.lang.String CR_LF)
```

Sets a global newline character. (See its compliment retrieval method - getLineCRLF())

getLineCRLF

```
java.lang.String getLineCRLF()
```

Retrieves a newline character, either a default CR LF "\r\n" string "newline character" has been set or the compliment set method of this get method can be used to custom set a "newline character".

getTotalLoadedFiles

```
int getTotalLoadedFiles()
```

Returns the number of files loaded on the RAMfileArray

loadPreLoadingList

```
void loadPreLoadingList(java.io.File setlist)
```

Load the RAM byte[][] array by what MUST BE a TXT file - A list of filenames with full path on newlines or comma separated full path names (NO quotations around them , ordinary text) saved in the TXT file that has been loaded by a program and *is a java.io.File*

loadPreLoadingList

```
void loadPreLoadingList(java.lang.String listset)
```

Load the files by a String passed to the RAM file object from another program i.e. *a servlet* (the bare full path names as allowed by programming language syntax separated by commas or concatenated to the appropriate platform newline character).

loadPreLoadingList

```
void loadPreLoadingList(java.lang.String[] fls)
```

Load the files by passing a String[] array of full path filename elements to the RAM file object the bare full path names as allowed by programming language syntax).
It is best to apply String trim method to the ready names as assurance.

loadPreLoadingList

```
void loadPreLoadingList(java.io.File[] fistset)
```

This is the recommended way to load the RAMFileArray (loop the file array first to check if File.exists())
Load the files for the objects RAMfileArray byte[][] array by a java.io.File[] array object from some other program in the server i.e.

a special custom servlet.

note: *While this is the recommended loader method, the best method to use to start the loading is a "ServletContextListener" class that will start the loading when the application is loaded into Tomcat, inside that is the best place to put the loading code. called Loader Servlets are too unsecure for the purpose.*

getOutWriteRAMbyteLength

```
long getOutWriteRAMbyteLength(int arrLenIdX)
```

int arrLenIdX The index required to obtain the array/file-size length no different to
e.g. only: RAMfile[63].length property call on an array

Returns a primitive long of the number of bytes of the particular file loaded at the index, the "file byte size" is identical to the files byte[] array length.

getRAMByteArray

```
byte[] getRAMByteArray(int idxAr)
```

getLoadedRAMFileNamesInventory

```
java.lang.String getLoadedRAMFileNamesInventory(boolean HtmlTXT)
```

Brings back a byte[] array of a loaded file from the specified index on the loaded RAMfile byte[][] array.
Returns a byte[] array object from the input argument *int idxAr* index of the files' byte[] array for use to acquire for output in a servlet

pingMessageTest

```
java.lang.String pingMessageTest()
```

Should contain details such as Version , date compiled, OS Platform compiled on , JDK level used for compile, and any changes from previous versions
All in all a debug ping message to understand if it is operative without triggering any dangerous mechanisms.