The Cache Module

Michael Gohde

June 5, 2015

1 Overview

The Cache Module provides caching and internet download services for one or more Experiment Clients. As such it requires only pure network code without any XML parsing or any other such features.

The CM as implemented here consists of a highly threaded design as is typical of server software written in Java. There are three main threads of execution, namely the Management Server Thread, the Cache Request Server Thread, and the Database Saver Thread. Their names should be fairly self-explanatory, and each will be documented in the following sections.

2 The Cache Database

The Cache Database is a shared object containing references to source URLs and were they are cached in the local filesystem. The interface that the Cache Database has with the rest of the program is intended to allow for different data structures to be used to store the database, as the default binary search tree may eventually be replaced with another storage solution if necessary. The database's file format should be considered implementation-dependent, as it is specific to how each implementer chooses to format the file. As RSSE is still in a very early development state, the CM has a very simple file format defined below:

VER *RSSE version code*
First URL
Location on filesystem
Second URL
Location on filesystem
...

The "VER" tag placed in the header of the file is intended to allow forward compatibility with additional capabilities that may be added to the Cache Module.

3 Initialization

In order to start itself, the CM undergoes the following procedure:

1. Parse command line arguments.
2. Check if a configuration file exists in the default location.
3. Read the configuration file if it exists, else use defaults.
4. Check if a DB snapshot file exists.
5. Read the DB snapshot if it exists, else initialize the database as blank.
6. Start the DB Saver thread.
7. Start the Management Server thread.
8. Block on the Management Terminal until it exits.

4 The Management Server Thread

The Management Server Thread enables users to remotely or locally manage their Cache Modules. It uses the Management Protocol to communicate and execute management transactions. This specific implementation of the Management Server enables the server to execute a small set of commands related to testing, starting, and stopping the Cache Request Server.

During each connection, the Management Server is passed an object containing a command and one or more arguments. It then proceeds to execute the query passed and returns another object of the same type to represent any results or output relevant to the query.

Queries are transmitted over network streams as follows:

COMMAND arg1 arg2 arg3 ...

Where "COMMAND" is any of the commands recognized by the Management Server. Each query is terminated with a newline ('backslashcharn') character, which makes the network transfers used by the Management Server very simple:

Client: COMMAND arg1 arg2 arg3 ...
Server: RESPCMD arg1 arg2 arg3 ...

Where "Client:" and "Server:" are not part of the stream sent. "RESPCMD" in this case can be any command understood by the client. At this time, the CMTerminal (the Management Protocol client) can understand the commands "PRINT" and "PRINTERR" in addition to any other commands used to connect to a server and start sending queries.

5 The Cache Request Server Thread

The Cache Request Server is quite a bit more complex than the Management Server, as it has to deal with the possibility of adding and searching elements in the database as well as managing several different connection types. All connection types are listed in the following table, though several points should be clarified first.

Firstly, every message sent to the client or server must be terminated with a single newline character. Secondly, all capitalization shown should be kept coherent unless the implementation is designed such that all commands are parsed in a case-insensitive manner. Finally, for most of the commands sent between the client and server, it is expected that the other will respond with either "OK" or "ERR". Please also note that all commands are written in a human-readable format to facilitate debugging via telnet.

The following subsections detail all of the connection types known to the Cache Module at the time of this writing.

5.1 Cache Request with No Data

This is the simplest request type as it merely requests that the server cache a file rather than print its location or dump its contents.

Client	**Server**
CACHE	
	OK *or* ERR
URL http://example.server/thing	
	OK *or* ERR

5.2 Fetch Request with Known Local URI

This allows a client that is aware that it is running on the same physical machine as the Cache Module to fetch a file's location rather than the binary contents of that file.

Client	**Server**
FETCH LOCAL	
	OK *or* ERR
URL http://example.server/thing	
	OK *or* ERR URI file://path/to/thing

5.3 Fetch Request with Known Remote

This connection type allows the server to send a file over the connection rather than just a reference to that file on the current filesystem. It thus allows for connections to be made from remote machines.

Client	**Server**
FETCH BINARY	
	OK *or* ERR
	FILENAME thing
	FILESIZE 1024
OK *or* ERR	
	Binary data stream

5.4 Fetch Request with Forced Binary Transfer

See previous subsection.

5.5 Fetch Request with Auto-detect

This connection type allows the client and server to automatically detect whether or not they are running on the same machine (ie. discern between Local and Remote) so that redundant files are not created. Once the connection type is detected and returned by the server, the client should then proceed to use the new connection type explicitly to connect to the server again. This feature isn't currently used by the Experiment Client, though it will be in future releases.

Client	**Server**
FETCH AUTODETECT	
	REMOTE *or* LOCAL

5.6 Request to Check if a File is Cached

This connection type is mostly just for debugging and administrative purposes, as it is unlikely that it will prove useful for Experiment Clients. As such, it is likely possible for this to remain unimplemented, at least for the first release version of RSSE. It is, however, implemented in this specific implementation as it was used for the aforementioned debugging and testing purposes.

Client	Server
CHECK	
	OK
URL http://example.server/thing	
	OK *or* ERR 0

5.7 Error Codes

During connections, errors may be thrown for a number of reasons. The error codes listed below need not necessarily be checked for or implemented, but should be considered a standard if you are to implement this specific caching protocol.

Error Code	Definition
0	Unknown or generic error; Thrown during checks to show that the file isn't parsed.
1	Thrown when the caching server has run out of storage space and cannot honor the request.
2	Thrown when the Caching Server is already honoring that request in another connection.
3	Thrown when the caching server cannot fetch the requested URL for some reason.
1000	Thrown when the client can't store a file with explicit FETCH BINARY.
1002	Thrown when the URI provided is bad in FETCH LOCAL.

6 The Database Saver Thread

The Database Saver thread saves the database at specified intervals to a snapshot file that can be loaded later on startup.

7 Directory Structure

Due to the multi-platform nature of Java, one single directory structure shouldn't be provided. Implementations are individually free to put files where they see fit or where is appropriate for the given platform, however this implementation targeted at *NIX platforms installs itself in the following directories:

7.1 /var/rsse

The general storage and configuration location for RSSE is in the /var/rsse directory on *NIX platforms. Within this, each component of RSSE may maintain its own subdirectory in order to store any files that should be logically separated from all of the other components.

7.2 /usr/sbin

The RSSE .jar files should all be placed in the /usr/sbin directory, while any loaders for RSSE (such as the one used in the sample installer script) should be placed anywhere else that is appropriate.

8 Installation

This is, unfortunately, one of the most difficult parts of RSSE at the present. In order to install this implementation of the Cache Module, several requirements must be met:

1. A functional installation of Java (versions 7 and 8 should likely work without problems).
2. A *NIX-based operating system.
3. The ability to modify system directories.
4. Time and patience.

If the following requirements are met, then please follow the following step-by-step guide to installing the CM:

1. Start by ensuring that you have CacheModule.jar.
2. Create the following directories: /var/rsse/cache, /var/rsse/cache/storage, and if it doesn't already exist, /var/rsse.
3. Chown and chmod the directories as appropriate for the intended user of the CM.
4. Copy CacheModule.jar to /usr/sbin
5. Ensure that the executable flag is set on CacheModule.jar, as some Java implementations refuse to work otherwise.
6. Optionally build the cm.c wrapper for the Cache Module.
7. Run "java -jar /usr/sbin/CacheModule.jar –genconfig" go generate a default configuration file.
8. Edit ./cm.conf to ensure that the defaults are suitable for your configuration.
9. Copy or move cm.conf to /var/rsse/cache
10. Ensure that it's readable to whoever will run the cache module.

Now that installation is complete, the service needs to be run manually as there has not yet been time to write an rc and/or sysv init script for it.

Start the Cache Module. You should be greeted with:

localhost-¿
If you see this prompt, then installation was likely successful. Enter the command "startservice" to start the caching server. When you have finished with the Cache Module, enter the command "stopservice" to stop the caching server.

9 Using the CMTerminal

By default or when running the cache module with the "–terminal" argument, you will be presented with a command line interface to the Management Server. This section will be broken into two subsections, one for the CMTerminal that appears by default and one for remote administration through the "–terminal" argument.

9.1 Default CMTerminal

By default, the Cache Module will launch a command line interpreter so that users can easily manage the CM's operation. This means that no commands need to be issued to connect the CMTerminal to its host, as it is already pre-configured to communicate with the local Cache Module. As such, very few commands should be used in this mode. They are listed in the following table:

Command	**Description**
cache *url*	Cache the URL passed.
report	List all cached files
startservice	Start the Cache Request thread.
stopservice	Stop the Cache Request thread.
loadcfg *file path*	Load configuration from the specified file.
savecfg *file path*	Save configuration to the specified file.

9.2 Remote CMTerminal

When the Cache Module is launched with the "–terminal" argument, it will act as a management terminal for some remote CM. When used remotely, several commands need to be run in order to specify which server to connect to as well as which port to use. All commands that are required in order to connect to a server are bolded in the following table. Please note that when used remotely, all of the commands from the table in "Default CMTerminal" are still valid and can be sent to the remote server once connected.

Command	Description
setsrvr *server name or ip addr*	Sets the server to send commands to.
setport *port number*	Sets the port to use on the server specified.
help *command*	Prints out help for the specified command.
cd *dir*	Changes the current directory. This and other directory manipulation commands will eventually be used to directly send and change files locally and on the connected server.
pwd	Prints the current working directory.
ls	Lists the contents of the current working directory.

10 Known Bugs

1. It appears that the Cache Request protocol can't currently handle the client sending an error at the end of a connection.

2. The auto-detect feature of the "Fetch Request with Auto detect" isn't yet able to accurately discern whether a client is operating on the same physical machine as the server. As such, it shouldn't be used yet.