Using the Infinispan Memcached Server

Table of Contents

Chapter 1. Memcached Server

The Infinispan server distribution contains a server module that implements the Memcached text protocol. This allows Memcached clients to talk to one or several Infinispan backed Memcached servers. These servers can be standalone just like Memcached, where each server acts independently and does not communicate with the rest. They can also be clustered, where the servers replicate or distribute their contents to other Infinispan backed Memcached servers, providing clients with failover capabilities.

1.1. Client Encoding

The Memcached text protocol assumes data values read and written by clients are raw bytes. The support for type negotiation will come with the Memcached binary protocol implementation.

It's not possible for a Memcached client to negotiate the data type to obtain data from the server or send data in different formats. The server can optionally be configured to handle values encoded with a certain Media Type. By setting the `client-encoding` attribute in the `memcached-connector` element, the server will return content in this configured format, and clients can also send data in this format.

The `client-encoding` is useful when a single cache is accessed from multiple remote endpoints (Rest, HotRod, Memcached) and it allows users to tailor the responses/requests to Memcached text clients. For more infomarmation on interoperability between endpoints, consult the endpoint interoperability documentation.

1.2. Command Clarification

1.2.1. Flush All

Even in a clustered environment, flush_all command leads to the clearing of the Infinispan Memcached server where the call lands. There is no attempt to propagate this flush to other nodes in the cluster. This is done so that the `flush_all` with delay use case can be reproduced with the Infinispan Memcached server. The aim of passing a delay to `flush_all` is so that different Memcached servers can be flushed at different times, and hence avoid overloading the database with requests as a result of all Memcached servers being empty. For more info, check the Memcached text protocol section on flush_all .

1.2.2. Unsupported Features

This section details features of the Memcached text protocol that are currently not supported by the Infinispan based Memcached implementation.

Individual Stats

There are differences in the nature of the original memcached implementation, which is C/C\\ based, and the Infinispan implementation which is Java based. There are some general purpose statistics that are not supported. For these unsupported stats, Infinispan Memcached server always

returns 0.

Unsupported statistics

- pid
- pointer_size
- rusage_user
- rusage_system
- bytes
- curr_connections
- total_connections
- connection_structures
- auth_cmds
- auth_errors
- limit_maxbytes
- threads
- conn_yields
- reclaimed

Statistic Settings

The statistics setting section of the text protocol has not been implemented due to its volatility.

Settings with Arguments Parameter

Since the arguments that can be sent to the Memcached server are not documented, Infinispan Memcached server does not support passing any arguments to the stats command. If any parameters are passed, the Infinispan Memcached server will respond with a `CLIENT_ERROR` .

Delete Hold Time Parameter

Memcached no longer honors the optional hold time parameter for the delete command. The Infinispan based Memcached server does not implement the feature.

Verbosity Command

The verbosity command is not supported since Infinispan logging cannot be simplified to defining the logging level alone.

Chapter 2. Talking To Infinispan Memcached Servers From Non-Java Clients

This section shows how to talk to Infinispan Memcached server via a non-Java client, such as a Python script.

2.1. Multi Clustered Server Tutorial

This example showcases the distribution capabilities of Infinispan Memcached severs that are not available in the original Memcached implementation.

Procedure

1. Start two clustered nodes: This configuration is the same one used for the GUI demo:

   ```
   $ ./bin/standalone.sh -c clustered.xml -Djboss.node.name=nodeA
   $ ./bin/standalone.sh -c clustered.xml -Djboss.node.name=nodeB
   -Djboss.socket.binding.port-offset=100
   ```

   Alternatively, use:

   ```
   $ ./bin/domain.sh
   ```

   Which automatically starts two nodes.

2. Execute the test_memcached_write.py script. This basically executes several write operations against the Infinispan Memcached server bound to port 11211. If the script is executed successfully, you should see an output similar to this:

   ```
   Connecting to 127.0.0.1:11211
   Testing set ['Simple_Key': Simple value] ... OK
   Testing set ['Expiring_Key' : 999 : 3] ... OK
   Testing increment 3 times ['Incr_Key' : starting at 1 ]
   Initialise at 1 ... OK
   Increment by one ... OK
   Increment again ... OK
   Increment yet again ... OK
   Testing decrement 1 time ['Decr_Key' : starting at 4 ]
   Initialise at 4 ... OK
   Decrement by one ... OK
   Testing decrement 2 times in one call ['Multi_Decr_Key' : 3 ]
   Initialise at 3 ... OK
   Decrement by 2 ... OK
   ```

3. Execute the test_memcached_read.py script which connects to server bound to 127.0.0.1:11311 and verifies that it can read the data that was written by the writer script to the first server. If

the script is executed successfully, you should see an output similar to this:

```
Connecting to 127.0.0.1:11311
Testing get ['Simple_Key'] should return Simple value ... OK
Testing get ['Expiring_Key'] should return nothing... OK
Testing get ['Incr_Key'] should return 4 ... OK
Testing get ['Decr_Key'] should return 3 ... OK
Testing get ['Multi_Decr_Key'] should return 1 ... OK
```