LEARN MEMCACHED

memory caching system

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Memcached is an open source, high-performance, distributed memory object caching system.

This tutorial provides a basic understanding of all the relevant concepts of Memcached needed to create and deploy a highly scalable and performance-oriented system.

Audience

This tutorial is designed for software professionals who wish to learn and apply the concepts of Memcached in simple and easy steps.

Prerequisites

Before proceeding with this tutorial, you need to know the basics of data structures.

Copyright & Disclaimer

Table of Contents

Part 1

Basics

1. OVERVIEW

Memcached is an open source, high-performance, distributed memory caching system intended to speed up dynamic web applications by reducing the database load. It is a key-value dictionary of strings, objects, etc., stored in the memory, resulting from database calls, API calls, or page rendering.

Memcached was developed by Brad Fitzpatrick for LiveJournal in 2003. However, it is now being used by Netlog, Facebook, Flickr, Wikipedia, Twitter, and YouTube among others

The key features of Memcached are as follows:

- It is open source.
- Memcached server is a big hash table.
- It significantly reduces the database load.
- It is perfectly efficient for websites with high database load.
- It is distributed under Berkeley Software Distribution (BSD) license.
- It is a client-server application over TCP or UDP.

Memcached is not:

- a persistent data store
- a database
- application-specific
- a large object cache
- fault-tolerant or highly available

2. ENVIRONMENT

Installing Memcached on Ubuntu

To install Memcached on Ubuntu, go to terminal and type the following commands:

```
$sudo apt-get update

$sudo apt-get install memcached
```

Confirming Memcached Installation

To confirm if Memcached is installed or not, you need to run the command given below. This command shows that Memcached is running on the default port **11211**.

```
$ps aux | grep memcached
```

To run Memcached server on a different port, execute the command given below. This command starts the server on the TCP port 11111 and listens on the UDP port 11111 as a daemon process.

```
$memcached -p 11111 -U 11111 -d
```

You can run multiple instances of Memcached server through a single installation.

Memcached Java Environment Setup

To use Memcached in your Java program, you need to download spymemcached-2.10.3.jar and setup this jar into the classpath.

3. CONNECTION

To connect to a Memcached server, you need to use the telnet command on HOST and PORT names.

Syntax

The basic syntax of Memcached telnet command is as shown below:

```
$telnet HOST PORT
```

Here, **HOST** and **PORT** are machine IP and port number respectively, on which the Memcached server is executing.

Example

The following example shows how to connect to a Memcached server and execute a simple set and get command. Assume that the Memcached server is running on host 127.0.0.1 and port 11211.

```
$telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
// now store some data and get it from memcached server
set tutorialspoint 0 900 9
memcached
STORED
get tutorialspoint
VALUE tutorialspoint 0 9
memcached
END
```

Connection from Java Application

To connect the Memcached server from your java program, you need to add the Memcached jar into your classpath as shown in the previous chapter. Assume that the Memcached server is running on host 127.0.0.1 and port 11211.

Example

```
import net.spy.memcached.MemcachedClient;
public class MemcachedJava {
   public static void main(String[] args) {

       // Connecting to Memcached server on localhost
      MemcachedClient mcc = new MemcachedClient(new
      InetSocketAddress("127.0.0.1", 11211));
      System.out.println("Connection to server sucessfully");

      //not set data into memcached server
      System.out.println("set status:"+mcc.set("tutorialspoint", 900,
      "memcached").done);

      //Get value from cache
      System.out.println("Get from Cache:"+mcc.get("tutorialspoint"));
   }
}
```

Output

On compiling and executing the program, you get to see the following output:

```
Connection to server successfully
set status:true
Get from Cache:memcached
```

Part 2

Storage Commands

4. SET DATA

Memcached **set** command is used to set a new value to a new or existing key.

Syntax

The basic syntax of Memcached **set** command is as shown below:

```
set key flags exptime bytes [noreply]

value
```

The keywords in the syntax are as described below:

- **key:** It is the name of the key by which data is stored and retrieved from Memcached.
- **flags:** It is the 32-bit unsigned integer that the server stores with the data provided by the user, and returns along with the data when the item is retrieved.
- **exptime:** It is the expiration time in seconds. 0 means no delay. If exptime is more than 30 days, Memcached uses it as UNIX timestamp for expiration.
- **bytes:** It is the number of bytes in the data block that needs to be stored. This is the length of the data that needs to be stored in Memcached.
- **noreply (optional):** It is a parameter that informs the server not to send any reply.
- **value:** It is the data that needs to be stored. The data needs to be passed on the new line after executing the command with the above options.

Output

The output of the command is as shown below:

```
STORED
```

- **STORED** indicates success.
- **ERROR** indicates incorrect syntax or error while saving data.

Example

In the following example, we use tutorialspoint as the key and set value Memcached in it with an expiration time of 900 seconds.

```
set tutorialspoint 0 900 9
memcached
STORED
get tutorialspoint
VALUE tutorialspoint 0 9
memcached
END
```

Set Data Using Java Application

To set a key in Memcached server, you need to use Memcached **set** method.

Example

```
import net.spy.memcached.MemcachedClient;
public class MemcachedJava {
   public static void main(String[] args) {

      // Connecting to Memcached server on localhost
      MemcachedClient mcc = new MemcachedClient(new
      InetSocketAddress("127.0.0.1", 11211));
      System.out.println("Connection to server sucessfully");
      System.out.println("set status:"+mcc.set("tutorialspoint", 900,
      "memcached").done);

      // Get value from cache
      System.out.println("Get from Cache:"+mcc.get("tutorialspoint"));
   }
}
```

Output

On compiling and executing the program, you get to see the following output:

```
Connection to server successfully
set status:true
Get from Cache:memcached
```

5. ADD DATA

Memcached **add** command is used to set a value to a new key. If the key already exists, then it gives the output NOT_STORED.

Syntax

The basic syntax of Memcached **add** command is as shown below:

```
add key flags exptime bytes [noreply]
value
```

The keywords in the syntax are as described below:

- **key:** It is the name of the key by which data is stored and retrieved from Memcached.
- **flags:** It is the 32-bit unsigned integer that the server stores with the data provided by the user, and returns along with the data when the item is retrieved.
- **exptime:** It is the expiration time in seconds. 0 means no delay. If exptime is more than 30 days, Memcached uses it as a UNIX timestamp for expiration.
- **bytes:** It is the number of bytes in the data block that needs to be stored. This is the length of the data that needs to be stored in Memcached.
- **noreply (optional):** It is a parameter that informs the server not to send any reply.
- **value:** It is the data that needs to be stored. The data needs to be passed on the new line after executing the command with the above options.

Output

The output of the command is as shown below:

```
STORED
```

- **STORED** indicates success.
- **NOT_STORED** indicates the data is not stored in Memcached.

Example

In the following example, we use ‘key’ as the key and add the value Memcached in it with an expiration time of 900 seconds.

```
add key 0 900 9
memcached
STORED
get key
VALUE key 0 9
memcached
END
```

Failure Output

```
add key 0 900 5
redis
NOT_STORED
```

Add Data Using Java Application

To add data in a Memcached server, you need to use the Memcached **add** method.

Example

```
import net.spy.memcached.MemcachedClient;
public class MemcachedJava {
   public static void main(String[] args) {

      // Connecting to Memcached server on localhost
      MemcachedClient mcc = new MemcachedClient(new
      InetSocketAddress("127.0.0.1", 11211));
      System.out.println("Connection to server successful");
      System.out.println("add status:"+mcc.add("tutorialspoint", 900,
      "redis").done);
      System.out.println("add status:"+mcc.add("tp", 900, "redis").done);

      // Get value from cache
      System.out.println("Get from Cache tp:"+mcc.get("tp"));
   }
}
```

Output

On compiling and executing the program, you get to see the following output:

```
Connection to server successful
add status:false
add status:true
Get from Cache tp:redis
```

6. REPLACE DATA

Memcached **replace** command is used to replace the value of an existing key. If the key does not exist, then it gives the output NOT_STORED.

Syntax

The basic syntax of Memcached **replace** command is as shown below:

```
replace key flags exptime bytes [noreply]
value
```

The keywords in the syntax are as described below:

- **key:** It is the name of the key by which data is stored and retrieved from Memcached.
- **flags:** It is the 32-bit unsigned integer that the server stores with the data provided by the user, and returns along with the data when the item is retrieved.
- **exptime:** It is the expiration time in seconds. 0 means no delay. If exptime is more than 30 days, Memcached uses it as a UNIX timestamp for expiration.
- **bytes:** It is the number of bytes in the data block that needs to be stored. This is the length of the data that needs to be stored in the Memcached.
- **noreply (optional):** It is a parameter that informs the server not to send any reply.
- **value:** It is the data that needs to be stored. The data needs to be passed on the new line after executing the command with the above options.

Output

The output of the command is as shown below:

```
STORED
```

- **STORED** indicates success.
- **NOT_STORED** indicates the data is not stored in Memcached.

Example

In the following example, we use ‘key’ as the key and store memcached in it with an expiration time of 900 seconds. After this, the same key is replaced with the value ‘redis’.

```
add key 0 900 9
memcached
STORED
get key
VALUE key 0 9
memcached
END
replace key 0 900 5
redis
get key
VALUE key 0 5
redis
END
```

Replace Data Using Java Application

To replace data in a Memcached server, you need to use the Memcached **replace** method.

Example

```
import net.spy.memcached.MemcachedClient;
public class MemcachedJava {
   public static void main(String[] args) {
      // Connecting to Memcached server on localhost
      MemcachedClient mcc = new MemcachedClient(new
      InetSocketAddress("127.0.0.1", 11211));
      System.out.println("Connection to server sucessfully");
      System.out.println("set status:"+mcc.set("tutorialspoint", 900,
      "memcached").done);

      // Get value from cache
      System.out.println("Get from Cache:"+mcc.get("tutorialspoint"));

      // now replace the existing data
      System.out.println("Replace cache:"+mcc.replace("tutorialspoint",
      900, "redis").done);

      // get the updated data
      System.out.println("Get from Cache:"+mcc.get("tutorialspoint"));
   }
}
```

Output

On compiling and executing the program, you get to see the following output:

```
Connection to server successfully
set status:true
Get from Cache:memcached
Replace cache:true
Get from Cache:redis
```

7. APPEND DATA

Memcached **append** command is used to add some data in an existing key. The data is stored after the existing data of the key.

Syntax

The basic syntax of Memcached **append** command is as shown below:

```
append key flags exptime bytes [noreply]
value
```

The keywords in the syntax are as described below:

- **key:** It is the name of the key by which data is stored and retrieved from Memcached.
- **flags:** It is the 32-bit unsigned integer that the server stores with the data provided by the user, and returns along with the data when the item is retrieved.
- **exptime:** It is the expiration time in seconds. 0 means no delay. If exptime is more than 30 days, Memcached uses it as a UNIX timestamp for expiration.
- **bytes:** It is the number of bytes in the data block that needs to be stored. This is the length of the data that needs to be stored in Memcached.
- **noreply (optional):** It is a parameter that informs the server not send any reply.
- **value:** It is the data that needs to be stored. The data needs to be passed on the new line after executing the command with the above options.

Output

The output of the command is as shown below:

```
STORED
```

- **STORED** indicates success.
- **NOT_STORED** indicates the key does not exist in the Memcached server.
- **CLIENT_ERROR** indicates error.

Example

In the following example, we try to add some data in a key that does not exist. Hence, Memcached returns **NOT_STORED**. After this, we set one key and append data into it.

```
append tutorials 0 900 5
redis
NOT_STORED
set tutorials 0 900 9
memcached
STORED
get tutorials
VALUE tutorials 0 14
memcached
END
append tutorials 0 900 5
redis
STORED
get tutorials
VALUE tutorials 0 14
memcachedredis
END
```

Append Data Using Java Application

To append data in a Memcached server, you need to use the Memcached **append** method.

Example

```
import net.spy.memcached.MemcachedClient;
public class MemcachedJava {
   public static void main(String[] args) {

      // Connecting to Memcached server on localhost
      MemcachedClient mcc = new MemcachedClient(new
      InetSocketAddress("127.0.0.1", 11211));
      System.out.println("Connection to server successful");
      System.out.println("set status:"+mcc.set("tutorialspoint", 900,
      "memcached").isDone());

      // Get value from cache
      System.out.println("Get from Cache:"+mcc.get("tutorialspoint"));

      // now append some data into existing key
      System.out.println("Append to cache:"+mcc.append("tutorialspoint",
      "redis").isDone());

      // get the updated key
      System.out.println("Get from Cache:"+mcc.get("tutorialspoint"));
   }
}
```

Output

On compiling and executing the program, you get to see the following output:

```
Connection to server successful
set status:true
Get from Cache:memcached
Append to cache:true
Get from Cache:memcachedredis
```

8. PREPEND DATA

Memcached **prepend** command is used to add some data in an existing key. The data is stored before the existing data of the key.

Syntax

The basic syntax of Memcached **prepend** command is as shown below:

```
prepend key flags exptime bytes [noreply]
value
```

The keywords in the syntax are as described below:

- **key:** It is the name of the key by which data is stored and retrieved in Memcached.
- **flags:** It is the 32-bit unsigned integer that the server stores with the data provided by the user, and returns along with the data when the item is retrieved.
- **exptime:** It is the expiration time in seconds. 0 means no delay. If exptime is more than 30 days, Memcached uses it as a UNIX timestamp for expiration.
- **bytes:** It is the number of bytes in the data block that needs to be stored. This is the length of the data that needs to be stored in Memcached.
- **noreply (optional):** It is a parameter that informs the server not send any reply.
- **value:** It is the data that needs to be stored. Data needs to be passed on the new line after executing the command with the above options.

Output

The output of the command is as shown below:

```
STORED
```

- **STORED** indicates success.
- **NOT_STORED** indicates the key does not exist in the Memcached server.
- **CLIENT_ERROR** indicates error.

Example

In the following example, we add some data in a key that does not exist. Hence, Memcached returns **NOT_STORED**. After this, we set one key and prepend data into it.

```
prepend tutorials 0 900 5
redis
NOT_STORED
set tutorials 0 900 9
memcached
STORED
get tutorials
VALUE tutorials 0 14
memcached
END
prepend tutorials 0 900 5
redis
STORED
get tutorials
VALUE tutorials 0 14
redismemcached
END
```

Prepend Data Using Java Application

To prepend data in a Memcached server, you need to use the Memcached **prepend** method.

Example

```
import net.spy.memcached.MemcachedClient;
public class MemcachedJava {
   public static void main(String[] args) {

      // Connecting to Memcached server on localhost
      MemcachedClient mcc = new MemcachedClient(new
      InetSocketAddress("127.0.0.1", 11211));
      System.out.println("Connection to server successful");
      System.out.println("set status:"+mcc.set("tutorialspoint", 900,
      "memcached").isDone());

      // Get value from cache
      System.out.println("Get from Cache:"+mcc.get("tutorialspoint"));

      // now append some data into existing key
      System.out.println("Prepend to
      cache:"+mcc.prepend("tutorialspoint", "redis").isDone());

      // get the updated key
      System.out.println("Get from Cache:"+mcc.get("tutorialspoint"));
   }
}
```

Output

On compiling and executing the program, you get to see the following output:

```
Connection to server successful
set status:true
Get from Cache:memcached
Prepend to cache:true
Get from Cache:redismemcached
```

9. CAS COMMAND

CAS stands for Check-And-Set or Compare-And-Swap. Memcached **CAS** command is used to set the data if it is not updated since last fetch. If the key does not exist in Memcached, then it returns **NOT_FOUND**.

Syntax

The basic syntax of Memcached **CAS** command is as shown below:

```
set key flags exptime bytes unique_cas_key [noreply]

value
```

The keywords in the syntax are as described below:

- **key:** It is the name of the key by which data is stored and retrieved from Memcached.
- **flags:** It is the 32-bit unsigned integer that the server stores with the data provided by the user, and returns along with the data when the item is retrieved.
- **exptime:** It is the expiration time in seconds. 0 means no delay. If exptime is more than 30 days, Memcached uses it as a UNIX timestamp for expiration.
- **bytes:** It is the number of bytes in the data block that needs to be stored. This is the length of the data that needs to be stored in Memcached.
- **unique_cas_key:** It is the unique key get from gets command.
- **noreply (optional):** It is a parameter that informs the server not to send any reply.
- **value:** It is the data that needs to be stored. Data needs to be passed on new line after executing the command with the above options.

Output

The output of the command is as shown below:

```
STORED
```

- **STORED** indicates success.
- **ERROR** indicates error while saving data or wrong syntax.

- **EXISTS** indicates that someone has modified the CAS data since last fetch.
- **NOT_FOUND** indicates that the key does not exist in the Memcached server.

Example

To execute a CAS command in Memcached, you need to get a CAS token from the Memcached gets command.

```
cas tp 0 900 9
ERROR
cas tp 0 900 9 2
memcached
set tp 0 900 9
memcached
STORED
gets tp
VALUE tp 0 9 1
memcached
END
cas tp 0 900 5 2
redis
EXISTS
cas tp 0 900 5 1
redis
STORED
get tp
VALUE tp 0 5
redis
END
```

CAS Using Java Application

To get CAS data from a Memcached server, you need to use Memcached **gets** method.

Example

```
import net.spy.memcached.MemcachedClient;
public class MemcachedJava {
   public static void main(String[] args) {
      // Connecting to Memcached server on localhost
      MemcachedClient mcc = new MemcachedClient(new
      InetSocketAddress("127.0.0.1", 11211));
      System.out.println("Connection to server successful");
      System.out.println("set status:"+mcc.set("tutorialspoint", 900,
      "memcached").isDone());

      // Get cas token from cache
      long castToken = mcc.gets("tutorialspoint").cas;
      System.out.println("Cas token:"+castToken);

      // now set new data in memcached server
      System.out.println("Now set new data:"+mcc.cas("tutorialspoint",
      castToken, 900, "redis"));
      System.out.println("Get from Cache:"+mcc.get("tutorialspoint"));
   }
}
```

Output

On compiling and executing the program, you get to see the following output:

```
Connection to server successful
set status:true
Cas token:3
Now set new data:OK
Get from Cache:redis
```

Part 3

Retrieval Commands

10. GET DATA

Memcached **get** command is used to get the value stored at key. If the key does not exist in Memcached, then it returns nothing.

Syntax

The basic syntax of Memcached **get** command is as shown below:

```
get key
```

Example

In the following example, we use tutorialspoint as the key and store memcached in it with an expiration time of 900 seconds.

```
set tutorialspoint 0 900 9
memcached
STORED
get tutorialspoint
VALUE tutorialspoint 0 9
memcached
END
```

Get Data Using Java Application

To get data from a Memcached server, you need to use the Memcached **get** method.

Example

```
import net.spy.memcached.MemcachedClient;
public class MemcachedJava {
   public static void main(String[] args) {

      // Connecting to Memcached server on localhost
      MemcachedClient mcc = new MemcachedClient(new
      InetSocketAddress("127.0.0.1", 11211));
      System.out.println("Connection to server sucessfully");
      System.out.println("set status:"+mcc.set("tutorialspoint", 900,
      "memcached").done);

      // Get value from cache
      System.out.println("Get from Cache:"+mcc.get("tutorialspoint"));
   }
}
```

Output

On compiling and executing the program, you get to see the following output:

```
Connection to server successfully
set status:true
Get from Cache:memcached
```

11. GET CAS DATA

Memcached **gets** command is used to get the value with CAS token. If the key does not exist in Memcached, then it returns nothing.

Syntax

The basic syntax of Memcached **gets** command is as shown below:

```
gets key
```

Example

```
set tutorialspoint 0 900 9
memcached
STORED
gets tutorialspoint
VALUE tutorialspoint 0 9 1
memcached
END
```

In this example, we use tutorialspoint as the key and store memcached in it with an expiration time of 900 seconds.

Get CAS Data Using Java Application

To get CAS data from a Memcached server, you need to use the Memcached **gets** method.

Example

```
import net.spy.memcached.MemcachedClient;
public class MemcachedJava {
   public static void main(String[] args) {

      // Connecting to Memcached server on localhost
      MemcachedClient mcc = new MemcachedClient(new
      InetSocketAddress("127.0.0.1", 11211));
```

```
        System.out.println("Connection to server sucessfully");
        System.out.println("set status:"+mcc.set("tutorialspoint", 900,
        "memcached").done);

        // Get value from cache
        System.out.println("Get from Cache:"+mcc.gets("tutorialspoint"));
    }
}
```

Output

On compiling and executing the program, you get to see the following output:

```
Connection to server successfully
set status:true
Get from Cache:{CasValue 2/memcached}
```

12. DELETE DATA

Memcached **delete** command is used to delete an existing key from the Memcached server.

Syntax

The basic syntax of Memcached **delete** command is as shown below:

```
delete key
```

If the key is successfully deleted, then it returns DELETED. If the key is not found, then it returns NOT_FOUND, otherwise it returns ERROR.

Example

In this example, we use tutorialspoint as a key and store memcached in it with an expiration time of 900 seconds. After this, it deletes the stored key.

```
set tutorialspoint 0 900 9
memcached
STORED
get tutorialspoint
VALUE tutorialspoint 0 9
memcached
END
delete tutorialspoint
DELETED
get tutorialspoint
END
delete tutorialspoint
NOT_FOUND
```

Delete Data Using Java Application

To delete data from a Memcached server, you need to use the Memcached **delete** method.

Example

```
import net.spy.memcached.MemcachedClient;
public class MemcachedJava {
   public static void main(String[] args) {

      // Connecting to Memcached server on localhost
      MemcachedClient mcc = new MemcachedClient(new
      InetSocketAddress("127.0.0.1", 11211));
      System.out.println("Connection to server successful");
      System.out.println("set status:"+mcc.set("tutorialspoint", 900,
      "memcached").done);

      // Get value from cache
      System.out.println("Get from Cache:"+mcc.get("tutorialspoint"));

      // delete value from cache
      System.out.println("Delete from
      Cache:"+mcc.delete("tutorialspoint").isDone());

      // check whether value exists or not
      System.out.println("Get from Cache:"+mcc.get("tutorialspoint"));
   }
}
```

Output

On compiling and executing the program, you get to see the following output:

```
Connection to server successful
set status:true
Get from Cache:memcached
Delete from Cache:true
Get from Cache:null
```

13. INCREMENT DECREMENT DATA

Memcached **incr** and **decr** commands are used to increment or decrement the numeric value of an existing key. If the key is not found, then it returns **NOT_FOUND**. If the key is not numeric, then it returns **CLIENT_ERROR cannot increment or decrement non-numeric value**. Otherwise, **ERROR** is returned.

Syntax - incr

The basic syntax of Memcached **incr** command is as shown below:

```
incr key increment_value
```

Example

In this example, we use visitors as key and set 10 initially into it, thereafter we increment the visitors by 5.

```
set visitors 0 900 2
10
STORED
get visitors
VALUE visitors 0 2
10
END
incr visitors 5
15
get visitors
VALUE visitors 0 2
15
END
```

Syntax - decr

The basic syntax of Memcached **decr** command is as shown below:

```
decr key decrement_value
```

Example

```
set visitors 0 900 2
10
STORED
get visitors
VALUE visitors 0 2
10
END
decr visitors 5
5
get visitors
VALUE visitors 0 1
5
END
```

Incr/Decr Using Java Application

To increment or decrement data in a Memcached server, you need to use Memcached **incr or decr** methods respectively.

Example

```
import net.spy.memcached.MemcachedClient;
public class MemcachedJava {
   public static void main(String[] args) {

      // Connecting to Memcached server on localhost
      MemcachedClient mcc = new MemcachedClient(new
      InetSocketAddress("127.0.0.1", 11211));
      System.out.println("Connection to server sucessfully");
      System.out.println("set status:"+mcc.set("count", 900,
      "5").isDone());

      // Get value from cache
      System.out.println("Get from Cache:"+mcc.get("count"));

      // now increase the stored value
      System.out.println("Increment value:"+mcc.incr("count", 2));

      // now decrease the stored value
      System.out.println("Decrement value:"+mcc.decr("count", 1));

      // now get the final stored value
      System.out.println("Get from Cache:"+mcc.get("count"));
   }
}
```

Output

On compiling and executing the program, you get to see the following output:

```
Connection to server successfully
set status:true
Get from Cache:5
Increment value:7
Decrement value:6
Get from Cache:6
```

Part 4

Statistical Commands

14. STATS

Memcached **stats** command is used to return server statistics such as PID, version, connections, etc.

Syntax

The basic syntax of Memcached **stats** command is as shown below:

```
stats
```

Example

```
stats
STAT pid 1162
STAT uptime 5022
STAT time 1415208270
STAT version 1.4.14
STAT libevent 2.0.19-stable
STAT pointer_size 64
STAT rusage_user 0.096006
STAT rusage_system 0.152009
STAT curr_connections 5
STAT total_connections 6
STAT connection_structures 6
STAT reserved_fds 20
STAT cmd_get 6
STAT cmd_set 4
STAT cmd_flush 0
STAT cmd_touch 0
STAT get_hits 4
STAT get_misses 2
STAT delete_misses 1
STAT delete_hits 1
STAT incr_misses 2
STAT incr_hits 1
STAT decr_misses 0
STAT decr_hits 1
STAT cas_misses 0
STAT cas_hits 0
STAT cas_badval 0
STAT touch_hits 0
STAT touch_misses 0
STAT auth_cmds 0
STAT auth_errors 0
STAT bytes_read 262
STAT bytes_written 313
STAT limit_maxbytes 67108864
STAT accepting_conns 1
STAT listen_disabled_num 0
STAT threads 4
STAT conn_yields 0
STAT hash_power_level 16
STAT hash_bytes 524288
STAT hash_is_expanding 0
STAT expired_unfetched 1
STAT evicted_unfetched 0
STAT bytes 142
STAT curr_items 2
STAT total_items 6
STAT evictions 0
STAT reclaimed 1
END
```

Stats Using Java Application

To get stats from a Memcached server, you need to use the Memcached **stats** method.

Example

```
import net.spy.memcached.MemcachedClient;
public class MemcachedJava {
   public static void main(String[] args) {

      // Connecting to Memcached server on localhost
      MemcachedClient mcc = new MemcachedClient(new
      InetSocketAddress("127.0.0.1", 11211));
      System.out.println("Connection to server successful");
      System.out.println("Stats:"+mcc.stats);
   }
}
```

Output

On compiling and executing the program, you get to see the following output:

```
Connection to server successful
Stats:[/127.0.0.1:11211:[delete_hits:0, bytes:71, total_items:4,
rusage_system:0.220013, touch_misses:0, cmd_touch:0,
listen_disabled_num:0, auth_errors:0, evictions:0, version:1.4.14,
pointer_size:64, time:1417279366, incr_hits:1, threads:4,
expired_unfetched:0, limit_maxbytes:67108864, hash_is_expanding:0,
bytes_read:170, curr_connections:8, get_misses:1, reclaimed:0,
bytes_written:225, hash_power_level:16, connection_structures:9,
cas_hits:0, delete_misses:0, total_connections:11, rusage_user:0.356022,
cmd_flush:0, libevent:2.0.19-stable, uptime:12015, reserved_fds:20,
touch_hits:0, cas_badval:0, pid:1138, get_hits:2, curr_items:1,
cas_misses:0, accepting_conns:1, evicted_unfetched:0, cmd_get:3,
cmd_set:2, auth_cmds:0, incr_misses:1, hash_bytes:524288, decr_misses:1,
decr_hits:1, conn_yields:0]]
```

15. STATS ITEMS

Memcached **stats items** command is used to get items statistics such as count, age, eviction, etc. organized by slabs ID.

Syntax

The basic syntax of Memcached **stats items** command is as shown below:

```
stats items
```

Example

```
stats items
STAT items:1:number 1
STAT items:1:age 7
STAT items:1:evicted 0
STAT items:1:evicted_nonzero 0
STAT items:1:evicted_time 0
STAT items:1:outofmemory 0
STAT items:1:tailrepairs 0
STAT items:1:reclaimed 0
STAT items:1:expired_unfetched 0
STAT items:1:evicted_unfetched 0
END
```

16. STATS SLABS

Memcached **stats slabs** command displays slabs statistics such as size, memory usage, commands, count etc. organized by slabs ID.

Syntax

The basic syntax of Memcached **stats slabs** command is as shown below:

```
stats slabs
```

Example

```
stats slabs
STAT 1:chunk_size 96
STAT 1:chunks_per_page 10922
STAT 1:total_pages 1
STAT 1:total_chunks 10922
STAT 1:used_chunks 1
STAT 1:free_chunks 10921
STAT 1:free_chunks_end 0
STAT 1:mem_requested 71
STAT 1:get_hits 0
STAT 1:cmd_set 1
STAT 1:delete_hits 0
STAT 1:incr_hits 0
STAT 1:decr_hits 0
STAT 1:cas_hits 0
STAT 1:cas_badval 0
STAT 1:touch_hits 0
STAT active_slabs 1
STAT total_malloced 1048512
END
```

17. STATS SIZES

Memcached **stats sizes** command provides information about the sizes and number of items of each size within the cache. The information is returned in two columns. The first column is the size of the item (rounded up to the nearest 32 byte boundary), and the second column is the count of the number of items of that size within the cache.

Syntax

The basic syntax of Memcached **stats sizes** command is as shown below:

```
stats sizes
```

Example

```
stats sizes
STAT 96 1
END
```

The item size statistics are useful only to determine the sizes of the objects you are storing. Since the actual memory allocation is relevant only in terms of the chunk size and page size, the information is only useful during a careful debugging or diagnostic session.

18. CLEAR DATA

Memcached **flush_all** command is used to delete all data (key-value pairs) from the Memcached server. It accepts an optional parameter called **time** that sets a time after which the Memcached data is to be cleared.

Syntax

The basic syntax of Memcached **flush_all** command is as shown below:

```
flush_all [time] [noreply]
```

The above command always returns OK.

Example

In the following example, we store some data into the Memcached server and then clear all the data.

```
set tutorialspoint 0 900 9
memcached
STORED
get tutorialspoint
VALUE tutorialspoint 0 9
memcached
END
flush_all
OK
get tutorialspoint
END
```

Clear Data Using Java Application

To clear data from a Memcached server, you need to use the Memcached **flush** method.

Example

```
import net.spy.memcached.MemcachedClient;
public class MemcachedJava {
   public static void main(String[] args) {

      // Connecting to Memcached server on localhost
      MemcachedClient mcc = new MemcachedClient(new
      InetSocketAddress("127.0.0.1", 11211));
      System.out.println("Connection to server sucessfully");
      System.out.println("set status:"+mcc.set("count", 900,
      "5").isDone());

      // Get value from cache
      System.out.println("Get from Cache:"+mcc.get("count"));

      // now increase the stored value
      System.out.println("Increment value:"+mcc.incr("count", 2));

      // now decrease the stored value
      System.out.println("Decrement value:"+mcc.decr("count", 1));

      // now get the final stored value
      System.out.println("Get from Cache:"+mcc.get("count"));

      // now clear all this data
      System.out.println("Clear data:"+mcc.flush().isDone());
   }
}
```

Output

On compiling and executing the program, you get to see the following output:

```
Connection to server successfully
set status:true
Get from Cache:5
Increment value:7
Decrement value:6
```

```
Get from Cache:6
Clear data:true
```