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 and/or UDP.

**Memcached is not** –

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

**MEMCACHED - 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 check if Memcached is presently running or not, run the command given below –

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

This command should show that Memcached is running on the default port **11211**.

To run Memcached server on a different port, run the command given below –

```
$memcached -p 11111 -U 11111 -u user -d
```
It should start the server and listen on TCP port 11111 and UDP port 11111 as a daemon process.

This command is explained below –

- `-p` is for TCP port number
- `-U` is for UDP port number
- `-u` is for user name
- `-d` runs memcached as daemon process

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.

**MEMCACHED - CONNECTION**

To connect to a Memcached server, you need to use telnet command providing HOST name and PORT number.

**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 running.

**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. The following program assumes that Memcached server is running on host 127.0.0.1 and port 11211 –

**Example**

```
import net.spy.memcached.MemcachedClient;
import java.net.*;

public class MemcachedJava {
```
```java
public static void main(String[] args) {
    try{
        // Connecting to Memcached server on localhost
        MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));
        System.out.println("Connection to server successful.");
        // Shutdowns the memcached client
        mcc.shutdown();
    }catch(Exception ex){
        System.out.println(ex.getMessage());
    }
}
```

This program tries to connect memcached server on IP 127.0.0.1 and port number 11211 using `InetSocketAddress`.

**Output**

On compiling and executing the program, memcached client should connect to memcached server listening on port 11211 and the following should be the output –

```
Connection to server successful.
```

The terminal may show few informational messages too, those can be ignored.

**MEMCACHED - SET DATA**

Memcached `set` command is used to set a value to key; if the key does not exist, a new key is created and value is assigned to that 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 unique key by which data is accessed.
- **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 (seconds) of data stored in cache. A 0 value means "never expire", i.e. it should not be removed from the cache unless required. If the exptime is more than 30 days then Memcached interprets it as UNIX timestamp for expiration.
- **bytes** - This is the length of the data in bytes that needs to be stored in Memcached.
- **noreply (optional)** - This parameter informs the server not to send any reply.
- **value** - It is the data that needs to be stored. The data needs to be given on the new line after executing the command with the above options.

**Example**

In the following example, we use –

```
key → tutorialspoint
flag → 0
```
exptime → 900 (expiry time in seconds)
bytes → 9 (size of data value in bytes)
value → memcached

```
set tutorialspoint 0 900 9
memcached
STORED

get tutorialspoint
VALUE tutorialspoint 0 9
memcached
END
```

Output

The output of the command is as shown below –

```
STORED
```

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

Set Data Using Java Application

To set a key in Memcached server, we need to use Memcached set method. The set method returns `Future Object`. We need to include `java.util.concurrent.Future` interface in order to store output of set method.

Example

```
import java.net.InetSocketAddress;
import java.util.concurrent.Future;
import net.spy.memcached.MemcachedClient;

public class MemcachedJava {
    public static void main(String[] args) {
        try{
            // Connecting to Memcached server on localhost
            MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));
            System.out.println("Connection to server sucessful.");

            // now set data into memcached server
            Future fo = mcc.set("tutorialspoint", 900, "Free Education");

            // print status of set method
            System.out.println("set status:" + fo.get());

            // retrieve and check the value from cache
            System.out.println("tutorialspoint value in cache - " + mcc.get("tutorialspoint");

            // Shutdowns the memcached client
            mcc.shutdown();
        }
        catch(Exception ex){
            System.out.println( ex.getMessage() );
        }
    }
}
This program should set the value **memcached** with key **tutorialspoint**. If the key-value pair is set successfully, the **get** method of Future object should return **true**.

**Output**

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

```
Connection to server successful.
set status: true
tutorialspoint value in cache - Free Education
```

**MEMCACHED - 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 unique key by which data is accessed.
- **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 (seconds) of data stored in cache. A 0 value means "never expire", i.e. it should not be removed from the cache unless required. If the exptime is more than 30 days then Memcached interprets it as UNIX timestamp for expiration.
- **bytes** - This is the length of the data in bytes that needs to be stored in Memcached.
- **noreply (optional)** - This parameter informs the server not to send any reply.
- **value** - It is the data that needs to be stored. The data needs to be given on the **new line** after executing the command with the above options.

**Example**

In the following example, we use –

```
key → new_key
flag → 0
exptime → 900 (expiry time in seconds)
bytes → 10 (size of data value in bytes)
value → data_value
```

```
add new_key 0 900 10
STORED
data_value
get new_key
VALUE new_key 0 10
data_value
END
```
Output

If the data is stored successfully, the output should be –

STORED

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

Failure Output

Now, if we try to add 'new_key' again, it should give the following error –

```
add new_key 0 900 5
redis
NOT_STORED
```

Add Data Using Java Application

To store data in a Memcached server, we use the Memcached **add** method.

Example

In last example, we have set key **tutorialspoint**. In this example, we shall try to demonstrate adding data having existing key.

```java
import java.net.InetSocketAddress;
import java.util.concurrent.Future;
import net.spy.memcached.MemcachedClient;

public class MemcachedJava {
    public static void main(String[] args) {
        try {
            // Connecting to Memcached server on localhost
            MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));
            System.out.println("Connection to server sucessful.");
            // add data to memcached server
            Future fo = mcc.set("tutorialspoint", 900, "Free Education");
            // print status of set method
            System.out.println("set status:" + fo.get());
            // retrieve and check the value from cache
            System.out.println("tutorialspoint value in cache - " + mcc.get("tutorialspoint");
            // try to add data with existing key
            Future fo = mcc.add("tutorialspoint", 900, "memcached");
            // print status of set method
            System.out.println("add status:" + fo.get());
            // adding a new key to memcached server
            fo = mcc.add("codingground", 900, "All Free Compilers");
            // print status of set method
            System.out.println("add status:" + fo.get());
            // retrieve and check the value from cache
            System.out.println("codingground value in cache - " + mcc.get("codingground");
        }
    }
}
```
// Shutdowns the memcached client
mcc.shutdown();

} catch (Exception ex) {
    System.out.println(ex.getMessage());
}

Output

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

Connection to server successful.
set status:true
tutorialsvalue in cache - Free Education
add status:false
add status:true
codingground value in cache - All Free Compilers

The first 'add status' displays false because the key tutorials already exists in memcached server. The second 'add status' displays true indicates that the key is successfully stored.

MEMCACHED - 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 unique key by which data is accessed.
- **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 (seconds) of data stored in cache. A 0 value means "never expire", i.e. it should not be removed from the cache unless required. If the exptime is more than 30 days then Memcached interprets it as UNIX timestamp for expiration.
- **bytes** - This is the length of the data in bytes that needs to be stored in Memcached.
- **noreply (optional)** - This parameter informs the server not to send any reply.
- **value** - It is the data that needs to be stored. The data needs to be given on the new line after executing the command with the above options.

Example

For example we shall use −

    key → mykey
    flag → 0
    exptime → 900
    bytes → 10 (expiry time in seconds)
Here, we use 'mykey' as the key and store data_value in it. After this, the same key is replaced with 'some_other_value'.

```
add mykey 0 900 10
data_value
STORED
get mykey
VALUE mykey 0 10
data_value
END
replace mykey 0 900 16
some_other_value
get key
VALUE mykey 0 16
some_other_value
END
```

**Output**

The output of the command is as shown below –

```
STORED
```

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

**Replace Data Using Java Application**

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

**Example**

```java
import java.net.InetSocketAddress;
import java.util.concurrent.Future;

import net.spy.memcached.MemcachedClient;

public class MemcachedJava {
    public static void main(String[] args) {
        try {
            // Connecting to Memcached server on localhost
            MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));
            System.out.println("Connection to server successful.");

            // First add a key and value
            Future fo = mcc.set("tutorialspoint", 900, "Free Education");

            // print status of add method
            System.out.println("add status:" + fo.get());

            // retrieve and check the value from cache
            System.out.println("tutorialspoint value in cache - " + mcc.get("tutorialspoint");

            // adding a new key to memcached server
            fo = mcc.replace("tutorialspoint", 900, "Largest Tutorials' Library");

            // print status of set method
            System.out.println("replace status:" + fo.get());

            // retrieve and check the value from cache
```
```java
System.out.println("tutorialspoint value in cache - " +
mcc.get("tutorialspoint");

    // Shutdowns the memcached client
    mcc.shutdown();
}
}
```

**Output**

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

```
Connection to server successful.
set status:true
tutorialspoint value in cache - Free Education
Replace status:true
tutorialspoint value in cache - Largest Tutorials' Library
```

**MEMCACHED - APPEND DATA**

Memcached **append** command is used to add data in an existing key. This data is added at the end of the previous value.

**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 unique key by which data is accessed.
- **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 (seconds) of data stored in cache. A 0 value means "never expire", i.e. it should not be removed from the cache unless required. If the exptime is more than 30 days then Memcached interprets it as UNIX timestamp for expiration.
- **bytes** - This is the length of the data in bytes that needs to be stored in Memcached.
- **noreply (optional)** - This parameter informs the server not to send any reply.
- **value** - It is the data that needs to be stored. The data needs to be given on the new line after executing the command with the above options.

**Example**

In the following example -

- First we store a key (tutorials) and value (memcached) to Memcached
- Then, We retrieve the value using get command and check it
- Then, we **append** "redis" to the key tutorials
- and, we retrieve it and check again.

```
set tutorials 0 900 9
memcached
```
Output

The output of the `append` 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.

Append Data Using Java Application

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

Example

```java
import java.net.InetSocketAddress;
import java.util.concurrent.Future;
import net.spy.memcached.MemcachedClient;

public class MemcachedJava {
    public static void main(String[] args) {
        try{
            // Connecting to Memcached server on localhost
            MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));
            System.out.println("Connection to server successful.");
            // add data to memcached server
            Future fo = mcc.set("tutorialspoint", 900, "Free Education");
            // print status of set method
            System.out.println("set status:" + fo.get());
            // retrieve and check the value from cache
            System.out.println("tutorialspoint value in cache - " + mcc.get("tutorialspoint"));

            // try to add data with existing key
            Future fo = mcc.append("tutorialspoint", 900, " for All");
            // print status of set method
            System.out.println("append status:" + fo.get());
            // retrieve and check the value from cache
            System.out.println("tutorialspoint value in cache - " + mcc.get("codingground"));
        }
    }
}
```
// Shutdowns the memcached client
mcc.shutdown();

} catch(Exception ex) {
    System.out.println(ex.getMessage());
}

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

Connection to server successful.
set status:true
tutorialspoint value in cache - Free Education
append status:true
tutorialspoint value in cache - Free Education for All

MEMCACHED - PREPEND DATA

Memcached `prepend` command is used to add data in an existing key. This data is added 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 unique key by which data is accessed.
- **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 (seconds) of data stored in cache. A 0 value means "never expire", i.e. it should not be removed from the cache unless required. If the exptime is more than 30 days then Memcached interprets it as UNIX timestamp for expiration.
- **bytes** - This is the length of the data in bytes that needs to be stored in Memcached.
- **noreply (optional)** - This parameter informs the server not to send any reply.
- **value** - It is the data that needs to be stored. The data needs to be given on the new line after executing the command with the above options.

Example
In the following example -

- First we set a key (tutorials) and value (memcached) in Memcached
- Then, We retrieve the value using get command and check it
- Then, we `prepend"redis"` to the key tutorials
- and, we retrieve it and check again.

```
set tutorials 0 900 9
memcached
STORED
get tutorials
```
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.

Prepend Data Using Java Application

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

Example

```java
import java.net.InetSocketAddress;
import java.util.concurrent.Future;
import net.spy.memcached.MemcachedClient;

public class MemcachedJava {
    public static void main(String[] args) {
        try {
            // Connecting to Memcached server on localhost
            MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));
            System.out.println("Connection to server successful.");
            // add data to memcached server
            Future fo = mcc.set("tutorialspoint", 900, "Education for All");
            // print status of set method
            System.out.println("set status:" + fo.get());
            // retrieve and check the value from cache
            System.out.println("tutorialspoint value in cache - " + mcc.get("tutorialspoint"));
            // try to add data with existing key
            Future fo = mcc.prepend("tutorialspoint", 900, "Free ");
            // print status of set method
            System.out.println("prepend status:" + fo.get());
            // retrieve and check the value from cache
            System.out.println("tutorialspoint value in cache - " + mcc.get("codingground"));
        }
        // Shutdowns the memcached client
        mcc.shutdown();
    }
}
```
catch(Exception ex)
    System.out.println(ex.getMessage());
}

Output

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

<table>
<thead>
<tr>
<th>Connection to server successful.</th>
</tr>
</thead>
<tbody>
<tr>
<td>set status: true</td>
</tr>
<tr>
<td>tutorialspoint value in cache - Education for All</td>
</tr>
<tr>
<td>prepend status: true</td>
</tr>
<tr>
<td>tutorialspoint value in cache - Free Education for All</td>
</tr>
</tbody>
</table>

MEMCACHED - CAS COMMAND

CAS stands for Check-And-Set or Compare-And-Swap. Memcached CAS command 'checks' and 'set' data item if and only if, no other client process has updated it since last read by this client.

For example, if in the past, we retrieved the value <key, value> as <total, 3784>. If we now try to set or modify this value using CAS command, then CAS command will first check if it is the same value or is changed by some other client. If it is changed, then we should not change it to avoid any race anomalies in the system. If the value is not modified since our last fetch, CAS now sets the updated value.

Memcached server assigns a unique 64-bit CAS token to all items stored in it. We use gets command to retrieve CAS number of any item.

Syntax

The basic syntax of Memcached CAS command is as shown below −

cas key flags exptime bytes unique_cas_token [noreply]

The keywords in the syntax are as described below −

- **key** - It is the name of the unique key by which data is accessed.

- **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 (seconds) of data stored in cache. A 0 value means "never expire", i.e. it should not be removed from the cache unless required. If the exptime is more than 30 days then Memcached interprets it as UNIX timestamp for expiration.

- **bytes** - This is the length of the data in bytes that needs to be stored in Memcached.

- **unique_cas_token** – It is a unique token number obtained from gets command.

- **noreply (optional)** - This parameter informs the server not to send any reply.

- **value** - It is the data that needs to be stored. The data needs to be given on the new line after executing the command with the above options.

Example

To execute a CAS command in Memcached, you need to obtain a CAS token from the Memcached gets command. gets command is variation of get command, it is explained in forthcoming chapter.

In this example we should see −

- Incorrect cas statement i.e. *unique cas key missing*

- *cas*ing a non-existing key
- add a key value item
- obtain unique cas key using **gets** command
- use **cas** and **unique_cas_key** to update data item
- use **get** command and check if data is updated

```plaintext
 cas tp 0 900 9  <-- unique cas key missing
   ERROR
 cas tp 0 900 9 2
 memcached
 NOT_FOUND  <-- the key *tp* does not exist

 set tp 0 900 9
 memcached
 STORED

 gets tp
 VALUE tp 0 9 1
 memcached
 END

 cas tp 0 900 5 1
 redis
 STORED

 get tp
 VALUE tp 0 5
 redis
 END
```

**Output**

CAS command may produce one of the following result –

- **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.

**CAS Using Java Application**

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

**Example**

```java
import java.net.InetSocketAddress;
import java.util.concurrent.Future;

import net.spy.memcached.CASValue;
import net.spy.memcached.CASResponse;
import net.spy.memcached.MemcachedClient;

public class MemcachedJava {
    public static void main(String[] args) {
        try{
            // Connecting to Memcached server on localhost
            MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));
            System.out.println("Connection to server sucessful.");
        }
    }
}
```
// add data to memcached server
Future fo = mcc.set("tutorialspoint", 900, "Free Education");

// print status of set method
System.out.println("set status:" + fo.get());

// retrieve value stored for tutorialspoint from cache
System.out.println("tutorialspoint value in cache - " + mcc.get("tutorialspoint"));

// obtain CAS token value using gets method
CASValue casValue = mcc.gets("tutorialspoint");

// display CAS token value
System.out.println("CAS token - " + casValue);

// try to update data using memcached cas method
CASResponse casresp = mcc.cas("tutorialspoint", casValue.getCas(), 900, "Largest Tutorials-Library");

// display CAS Response
System.out.println("CAS Response - " + casresp);

// retrieve and check the value from cache
System.out.println("tutorialspoint value in cache - " + mcc.get("tutorialspoint"));

// Shutdowns the memcached client
mcc.shutdown();

} catch(Exception ex)
    System.out.println(ex.getMessage());

}

Output

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

Connection to server successful.
set status:true
tutorialspoint value in cache - Free Education
CAS - {CasValue 34/Free Education}
CAS Response - OK
tutorialspoint value in cache - Largest Tutorials-Library

MEMCACHED - GET DATA

Memcached get command is used to get the value stored at key. No value is returned if the key does not exist.

Syntax

The basic syntax of Memcached get command is as shown below −

get key

More than one key can be supplied with space between them as shown here −

get key1 key2 key3

Example

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

To get data from a Memcached server, we use Memcached `get` method.

Example

```java
import java.net.InetSocketAddress;
import java.util.concurrent.Future;
import net.spy.memcached.MemcachedClient;

public class MemcachedJava {
    public static void main(String[] args) {
        try {
            // Connecting to Memcached server on localhost
            MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));
            System.out.println("Connection to server sucessful.");

            // add data to memcached server
            Future fo = mcc.set("tutorialspoint", 900, "Free Education");

            // print status of set method
            System.out.println("set status:" + fo.get());

            // retrieve value from cache using get method
            System.out.println("tutorialspoint value in cache - " + mcc.get("tutorialspoint").get());

            // Shutdowns the memcached client
            mcc.shutdown();
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

Output

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

```
Connection to server sucessful.
set status:true
tutorialspoint value in cache - Free Education
```

Memcached `gets` command is used to get the value with CAS token. No value is returned if the key does not exist.

Syntax

The basic syntax of Memcached `gets` command is as shown below –

```
gets key
```
More than one key can be supplied with space between them as shown here –

```
gets key1 key2 key3
```

**Example**

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

In the output of `gets` command, we see 1 at the end. This 1 is unique CAS token associated with key `tutorialspoint`.

**Get CAS Data Using Java Application**

Memcached `gets` method is used to retrieve data items stored with key.

**Example**

```java
import java.net.InetSocketAddress;
import java.util.concurrent.Future;
import net.spy.memcached.CASValue;
import net.spy.memcached.CASResponse;
import net.spy.memcached.MemcachedClient;

public class MemcachedJava {
    public static void main(String[] args) {
        try {
            // Connecting to Memcached server on localhost
            MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));
            System.out.println("Connection to server sucessful.");

            // add data to memcached server
            Future fo = mcc.set("tutorialspoint", 900, "Free Education");

            // print status of set method
            System.out.println("set status:" + fo.get());

            // retrieve value stored for tutorialspoint from cache
            System.out.println("tutorialspoint value in cache - " + mcc.get("tutorialspoint"));

            // obtain CAS token value using gets method
            CASValue casValue = mcc.gets("tutorialspoint");

            // display CAS token value
            System.out.println("CAS value in cache - " + casValue);

            // Shoutdowns the memcached client
            mcc.shutdown();
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```
On compiling and executing the program, you get to see the following output –

```
Connection to server successful.
set status:true
tutorialspoint value in cache - Free Education
CAS - {CasValue 34/Free Education}
```

### MEMCACHED - DELETE KEY

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 [noreply]
```

### Output

CAS command may produce one of the following result –

- **DELETED** indicates successful deletion.
- **ERROR** indicates error while deleting data or wrong syntax.
- **NOT_FOUND** indicates that the key does not exist in the Memcached server.

### 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
ger tutorialspoint
VALUE tutorialspoint 0 9
memcached
END
delete tutorialspoint
DELETED
ger 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 java.net.InetSocketAddress;
import java.util.concurrent.Future;

import net.spy.memcached.MemcachedClient;

public class MemcachedJava {
    public static void main(String[] args) {
        try{
```
// Connecting to Memcached server on localhost
MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));
System.out.println("Connection to server sucessful.");

// add data to memcached server
Future fo = mcc.set("tutorialspoint", 900, "World's largest online tutorials library");

// print status of set method
System.out.println("set status:" + fo.get());

// retrieve and check the value from cache
System.out.println("tutorialspoint value in cache - " + mcc.get("tutorialspoint"));

// try to add data with existing key
Future fo = mcc.delete("tutorialspoint");

// print status of delete method
System.out.println("delete status:" + fo.get());

// retrieve and check the value from cache
System.out.println("tutorialspoint value in cache - " + mcc.get("codingground"));

// Shutdowns the memcached client
mcc.shutdown();

} catch(Exception ex)
    System.out.println(ex.getMessage());
}

Output

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

Connection to server successful
set status:true
tutorialspoint value in cache - World's largest online tutorials library
delete status:true
tutorialspoint value in cache - null

MEMCACHED - 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

Output

incr command may produce one of the following result −

- NOT_FOUND indicates that the key does not exist.
- CLIENT_ERROR indicates that value is not numaical.
- ERROR indicates any other error such as syntax error.
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
```

Output

incr command may produce one of the following result –

- **NOT_FOUND** indicates that the key does not exist.
- **CLIENT_ERROR** indicates that value is not numerical.
- **ERROR** indicates any other error such as syntax error.

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

        try {
            // Connecting to Memcached server on localhost
            MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));
            System.out.println("Connection to server sucessful.");
            // add numerical data to memcached server
            Future fo = mcc.set("number", 900, "1000");
            // print status of set method
            System.out.println("set status:" + fo.get());
            // retrieve and check the value from cache
            System.out.println("value in cache - "+mcc.get("number");
            // increment and check the value from cache
            System.out.println("value in cache after increment - "+mcc.incr("number", 111));
            // decrement and check the value from cache
            System.out.println("value in cache after decrement - "+mcc.decr("number", 112));

            // Shutdowns the memcached client
            mcc.shutdown();
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}

Output

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

```
Connection to server successful.
set status:true
value in cache - 1000
value in cache after increment - 1111
value in cache after decrement - 999
```

**MEMCACHED - 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
```
Stats Using Java Application

To get stats from a Memcached server, you need to use the Memcached `getStats()` method.

**Example**

```java
import java.net.InetSocketAddress;
import net.spy.memcached.MemcachedClient;

public class MemcachedJava {
    public static void main(String[] args) {
        try {
            // Connecting to Memcached server on localhost
            MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));

            System.out.println("Connection to server successful.");

            // display statistics about memcached
            System.out.println("Memcached Statistics - "+ mcc.getStats());

            // Shutdowns the memcached client
            mcc.shutdown();
        }
    }
}
```
```java
} catch (Exception ex) {
    System.out.println(ex.getMessage());
}
```

**Output**

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

```
Connection to server successful.
Memcached Statistics - {/127.0.0.1:11211={delete_hits=3, bytes=72, total_items=64, rusage_system=1.918095, touch_misses=0, cmd_touch=0, listen_disabled_num=0, auth_errors=0, evictions=0, version=1.4.14 (Ubuntu), pointer_size=64, time=1426883478, incr_hits=6, threads=4, expired_unfetched=0, limit_maxbytes=67108864, hash_is_expanding=0, bytes_read=3739, curr_connections=5, get_misses=9, reclaimed=12, bytes_written=3379, hash_power_level=16, connection_structures=6, cas_hits=8, delete_misses=0, total_connections=47, rusage_user=2.534625, cmd_flush=0, libevent=2.0.21-stable, uptime=40839, reserved_fds=20, touch_hits=0, cas_badval=8, pid=1204, get_hits=71, curr_items=1, cas_misses=0, accepting_conns=1, evicted_unfetched=0, cmd_get=80, cmd_set=71, auth_cmds=0, incr_misses=0, hash_bytes=524288, decr_misses=0, decr_hits=6, conn_yields=0}}
```

### MEMCACHED - 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
```

### MEMCACHED - 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
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.

---

**MEMCACHED - 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

```java
import java.net.InetSocketAddress;
import java.util.concurrent.Future;
import net.spy.memcached.MemcachedClient;

public class MemcachedJava {
    public static void main(String[] args) {
        try {
            // Connecting to Memcached server on localhost
            MemcachedClient mcc = new MemcachedClient(new InetSocketAddress("127.0.0.1", 11211));
            System.out.println("Connection to server sucessful.");

            // add data to memcached server
            Future fo = mcc.set("tutorialspoint", 900, "Free Education");

            // print status of set method
            System.out.println("set status:" + fo.get());

            // retrieve and check the value from cache
            System.out.println("tutorialspoint value in cache - " + mcc.get("tutorialspoint"));

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

            // Shutdowns the memcached client
            mcc.shutdown();
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

Output

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

Connection to server successful.
set status:true
tutorialspoint value in cache - Free Education
Clear data:true