GetCache Client User Guide

version 1.3.0

Introduction

GetCache is an in-memory key-value store developed with .Net 4.5 very simple to use and manage.
GetCache is a distributed in-memory cache that supports data sharding on multiple nodes and data replication.
GetCache stores any type of data, objects can be serialized using JSON or XML, row object like images or videos can be stored as byte arrays.

GetCache Client is a set of libraries that allow access to storage by applications developed by users.

.Net Client library

Distribution

GetCache client library can be downloaded from the website www.getcache.net or via NuGet.
The package contains the DLL file *GetCache.Client.dll* .

The .Net client is also distributed with the server package.

The .Net client is developed using .Net 4.0 for better compatibility.

Client API

The class that exposes the client API is **GetCacheShardingClient**. The API offers methods to operate with the remote GetCache server. GetCacheShardingClient can connect a single node or a cluster of nodes.

API List

T Get<T>(String key)
Get an object of by the input key and returns the object of type T or null if it's not found or the object is expired.

void Put(String key, Object obj, Int32 ttl = 60, PersistenceMethod persistence = PersistenceMethod.SINGLE)
Put an object in the storage. if the key is already present, the object is replaced.
The time to live value is expressed in seconds, a zero or negative number is equivalent to infinite. The time to live is evaluated from the last time the object is read calling the Get method

or from the creation time if the object is never read.
The persistence method can be Single or Replicated and it indicates how the GetCache cluster must store the data in a single node (determined by the sharding algorithm) or store the data in all the nodes.

Byte[] GetRawData(String key)
Get a serialized object by key and returns the object or null if it's not found or the object is expired.

void PutRawData(String key, Byte[] obj, Int32 ttl = 60, PersistenceMethod persistence = PersistenceMethod.SINGLE)
Put pre-serialized object in the storage. if the key is already present, the object is replaced. The time to live value expressed in seconds, a zero or negative number is equivalent to infinite. The persistence method can be Single or Replicated data on the nodes of the cluster.

Boolean Remove(String key)
Remove an object by key. Returns true if the object is removed else false.

Connect the server

Create the GetCacheShardingClient using the constructor and passing the input parameters:

- ***host*** represents the hostname (or IP address) of one available GetCache server node (see *WcfServerHost* in the server user guide)
- ***port*** indicates the http or the tcp port of the node (see *WcfServerPort* and *WcfServerPortTcp* in the server user guide)
- ***method*** is the serialization method used by the client, can be JSON (default value) or XML
- ***binding*** is the binding type used to connect the server, it can be HTTP or NET_TCP (default value from version 1.1) . This parameter must be used in accord with the *port* parameter.

Example of client initialization:

```
// Create GetCache client with default serialization method (JSON) and default binding //
(NET_TCP) calling localhost server at port 8282
GetCacheShardingClient client = new GetCacheShardingClient("localhost", 8282);

// Create GetCache client using JSON and HTTP binding
GetCacheShardingClient clientHttp = new GetCacheShardingClient("localhost", 9292,
SerializationMethod.JSON, BindingType.HTTP);
```

At creation time the client connect the remote node and ask for availables nodes in the cluster.

The clients maintains the list of nodes and send the data to the nodes using a sharding data algorithm.
It is a good practice to connect the cluster using an instance of the *GetCacheClientConfig* object, configuring at least two nodes of the cluster to have a better guarantee that the first connection to the cluster is carried out on an available node.

```
// Create the client configuration
GetCacheClientConfig config = new GetCacheClientConfig() {
Binding = BindingType.NET_TCP,
Method = SerializationMethod.JSON,
};
config.AddNode("localhost", 8282); // add connection data for node 1
config.AddNode("localhost", 8484); // add connection data for node 2
// Create the GetCache client
GetCacheShardingClient client = new GetCacheShardingClient(config);
```

The initial list provided to the client configuration can also contain all nodes in the cluster, or just one, the important thing is that the list contains at least one active node that can be queried by the client.

From release 1.3.0 the GetCacheClientConfig class has a new property *DisableClientTransportSecurity* that allows you to disable security of the transport of the client-server communication. This property is true by default because from release 1.3.0 also the server has the security transport disable by default.

Bindings

NET_TCP binding offers in general better performance than HTTP binding. This is the binding used by default in the client configuration.

HTTP binding can be used across firewalls and load balancers easily.

Example of usage

Define the class Customer

```
public class Customer
{
	public Customer()
	{
		CreationDate = DateTime.Now;
	}
```

```
        public String Id { get; set; }
        public String Name { get; set; }
        public String LastName { get; set; }
        public Int32 Wallet { get; set; }
        public DateTime CreationDate { get; set; }
}
```

Put, get and remove the data

```
// create GetCache client
GetCacheShardingClient client = new GetCacheShardingClient("localhost", 9292);

// create the object
Customer cust = new Customer()
{
        Id = "1001",
        Name = "Max",
        LastName = "Smith",
        Wallet = 100,
};

// put the object in the cache using the Id as key and using default ttl = 60 secs
// and persistence method Single by default
client.Put(cust.Id, cust);

// get the object from the cache using the key
Customer myCust = client.Get<Customer>("1001");
Console.WriteLine(myCust.Name);

// remove the object from the cache (or wait 60 seconds for automatic expiration)
client.Remove("1001");
```

Best practices

Create the client once and use it many times in the application can reduce the number of call needed by the client to know the state of the cluster, but use different clients for different threads because *GetCacheShardingClient* is not thread-safe.

Recreate the client when the cluster configuration changes (node was added or removed), this is not really mandatory but this practice allows to make better use of the mechanism of data sharding.

Other client libraries

At the moment there are not still available libraries developed on different platforms. from Net but in the future will be provided client libraries in other .Net versions and in other language.