

TayzGrid Open Source Guide

June 25, 2015

Contents

1. Introduction to TayzGrid.....	1
1.1 Edition Comparison	1
2. Install & Configure TayzGrid	5
2.1 Install TayzGrid	5
2.1.1 UNIX Platform.....	5
2.1.2 Windows Platform.....	5
2.2 Cache Server Hardware Requirement	5
2.2.1 Adequate RAM in Cache Servers.....	5
2.2.2 1Gbit Network Interface Card in Cache Servers	5
2.2.3 Dual-CPU, Quad-Core or higher.....	6
2.2.4 Disk.....	6
2.3 Cache Server Software Requirement.....	6
2.3.1 Windows 2008/2012 Server (64-bit)	6
2.3.2 Unix Server (64-bit).....	6
2.4 Configure TCP Port for TayzGrid Clients	6
2.4.1 Modify client.conf	7
2.4.2 Modify server.properties	7
2.5 Map Cache Server to a Network Card	7
2.6 Configure Cache Size Notification Threshold	8
3. Cache Administration.....	9
3.1 Create a Cache.....	9
3.1.1 Create a Local Cache.....	9
3.1.2 Create a Replicated Cache	10
3.1.3 Create a Partitioned Cache.....	11
3.2 Start/Stop Cache	12
3.3 Add/Remove Remote Clients.....	12
3.4 Add/Remove Cache Servers	12
3.5 Test the Cache Cluster	13
3.6 Create Indexes for SQL Queries	13
4. Cache Monitoring.....	15
4.1 Java Monitoring and Management Console	15
4.1.1 Monitoring Cache Server Counters using JConsole Tool.....	15
4.2 View Cluster Health.....	21
4.3 Server Log Files.....	21
4.4 Java Session Store Provider Logs.....	22
4.5 Client Side API Logging.....	22
5. TayzGrid Java Session Module.....	23
5.1 Conceptual Overview	23
5.1.1 Session Persistence in TayzGrid.....	23
5.1.2 Using Java Session through TayzGrid with No Code Change.....	23

5.1.3	Session Expiration	24
5.1.4	Handling Multiple Session Requests and Data Integrity	24
5.2	Configuring Application to Use Java Session Module.....	24
5.2.1	Adding Libraries.....	24
5.2.2	Defining Filter	24
6.	TayzGrid as Hibernate Second Level Cache.....	27
6.1	Conceptual Overview	27
6.1.1	Hibernate 1st Level Cache	27
6.1.2	TayzGrid as Hibernate 2nd level Cache	27
6.2	How to Use TayzGrid as Hibernate Second Level Cache.....	29
6.2.1	How to Configure Hibernate Application	29
6.2.2	How to Configure Cacheable Objects	30
6.2.3	How to Configure Cache Regions.....	31
6.2.4	How to Use Query Caching	33
6.2.5	How to Enable Query Caching	33
7.	TayzGrid for Object Caching.....	35
7.1	Connect to the Cache.....	35
7.1.1	Using Basic Initialize Method.....	35
7.1.2	Initializing Cache using CacheInitParams.....	35
7.2	Adding Items	36
7.2.1	Adding Objects to Cache	36
7.2.2	Adding Objects Using CacheItem	36
7.2.3	Adding Items to Cache in Bulk.....	36
7.2.4	Adding New Data with Absolute Expiration	37
7.3	Updating Items	37
7.3.1	Updating Objects in Cache.....	37
7.3.2	Updating Objects Using CacheItem.....	38
7.3.3	Updating Items in the Cache in Bulk	38
7.3.4	Updating Data with Absolute Expiration.....	39
7.4	Fetching Items.....	39
7.4.1	Retrieving Data using Get Method.....	39
7.4.2	Retrieve a CacheItem	40
7.4.3	Checking If an Item Exists in Cache.....	40
7.5	Lock/Unlock Items.....	40
7.5.1	Locking an Item Explicitly.....	40
7.5.2	Locking an Item during Fetch Operation	41
7.5.3	Lock Expiration	41
7.5.4	Releasing Lock with Update Operation	41
7.5.5	Releasing Lock Explicitly	42
7.6	Locking (Optimistic locking)	42
7.6.1	Conceptual Overview.....	42
7.7	How to Utilize CacheItemVersion.....	43
7.7.1	Saving Item Version for the First Time.....	43
7.7.2	Updating Item with Specific Version	43

7.7.3	Retrieving Cached Item with Specific Version.....	44
7.7.4	Retrieving Cacheltem with Specific Version.....	44
7.7.5	Retrieve Item if a Newer Version Exists in Cache.....	45
7.7.6	Removing an Item with Specified Item Version	45
7.7.7	Delete an Item with Item Version	45
7.8	How to Use Group Cache Data	46
7.8.1	Adding an Item to Data Group	46
7.8.2	Adding Data Group with Cacheltem.....	46
7.8.3	Updating an Item in a Data Group.....	46
7.8.4	Retrieving Keys Pertaining to a Particular Group.....	46
7.8.5	Retrieving Keys and Values Pertaining to a Particular Group	47
7.8.6	Removing a Data Group.....	47
7.9	Tags.....	48
7.9.1	Conceptual Overview.....	48
7.9.2	How to Tag Cached Data	48
7.10	Understanding Named Tags	50
7.10.1	Creating Named Tags.....	50
7.10.2	Adding item to cache with Named Tags	50
7.10.3	Adding Named Tags through Cacheltem	50
7.10.4	Querying Cache Items with respect to Named Tags.....	51
7.11	Searching Items with SQL & LINQ	51
7.11.1	Adding and Updating Indexed Items.....	51
7.11.2	Cache Search Returning Keys	52
7.11.3	Cache Search Returning Items.....	52
7.11.4	Querying Samples for Operators.....	53
7.12	Removing Items	54
7.12.1	Difference between Delete and Remove methods.....	54
7.12.2	Using Remove Method.....	54
7.12.3	Using Delete Method.....	55
7.13	Item Based Event Notification	55
7.13.1	Types of Notifications.....	55
7.13.2	Registering Event with a Particular Item.....	55
7.13.3	Registering Events with Cacheltem	56
7.13.4	Un-Registering Item Level Notification.....	56
7.14	Using Cache Level Events	56
7.14.1	Registering Cache Level Events.....	57
7.14.2	Un-registering Cache Level Events	58
7.15	Using Custom Notifications.....	58
7.16	Disconnect from the Cache.....	59
8.	Configuring as Memcached Wrapper.....	60
8.1	Memcached Protocol Server	60
8.1.1	Configure TayzGrid Memcached Gateway Service.....	60
8.1.2	Start Memcached Gateway Server.....	61
9.	How to Use TayzGrid as Spring Data Cache.....	62

9.1	How to Configure Spring Application to Use TayzGrid.....	62
9.2	How to Configure Caches.....	65

1. Introduction to TayzGrid

TayzGrid is an elastic In-Memory Data Grid with a peer to peer dynamic clustering architecture that makes sharing and managing data in a cluster as simple as on a single server. TayzGrid also provides a highly scalable Java Session State storage solution for applications running in multi-server configurations. Distributed caching enables you to scale your Java applications to handle extreme transaction loads.

TayzGrid has the following editions:

1. **TayzGrid Enterprise:** All features included. Includes regular phone and email support and free upgrades, timely bug-fixes, and frequently patch releases. Allows you to purchase 24x7 supports as well.
2. **TayzGrid Professional:** Same features as TayzGrid Open Source plus TayzGrid Manager. Includes regular phone and email support and free upgrades, timely bug-fixes, and frequently patch releases. 24x7 support purchase not available.
3. **TayzGrid Open Source:** Released under Apache 2.0 license and freely available along with its source code. No support provided by Alachisoft. Only community based support is available.

Below is an edition comparison of all TayzGrid. For a more detailed understanding of all these features, please see the [TayzGrid features](#) page.

1.1 Edition Comparison

Feature	Open Source	Professional	Enterprise
Caching Topologies			
Local Cache	X	X	X
Client Cache			X
Mirrored Cache			X
Replicated Cache	X	X	X
Partitioned Cache	X	X	X
Partition-Replica Cache (Async)			X
Partition-Replica Cache (Sync)			X
Bridge Topology			X
Data Grid Clients			
Local/Remote .NET Clients			X
Local/Remote Java Clients	X	X	X
Big Data Processing & Analytics			
MapReduce	X	X	X
Aggregator	X	X	X
Entry Processor	X	X	X

Feature	Open Source	Professional	Enterprise
Web Applications			
Java Web Sessions (Basic)	X	X	X
Java Web Sessions (Advanced)			X
Container Specific Session Replication Modules			X
ASP.NET Sessions			X
ASP.NET View State			X
ASP.NET Output Cache			X
Third-Party Integrations			
100% JCache (JSR-107) Compliant	X	X	X
Hibernate 2nd Level Cache	X	X	X
Spring Integration	X	X	X
Memcached Protocol Server	X	X	X
Memcached Wrapper Client (Java)	X	X	X
Memcached Wrapper Client (.NET)			X
NHibernate 2nd Level Cache (Basic)			X
NHibernate 2nd Level Cache (Advanced)			X
Entity Framework 2nd Level Cache Provider			X
Data Expirations			
Absolute Expirations	X	X	X
Sliding Expirations	X	X	X
Cache Dependencies			
Key Based Dependency			X
File Based Dependency			X
Custom Dependency			X
Multi-cache Key Dependency			X
Cache Sync Dependency			X
Synchronize Cache with Database			
SqlDependency (SQL Server DB Events)			X
OracleDependency (Oracle DB Events)			X
Db Dependency (OLEDB Polling)			X
Event Notifications			
Item based (onUpdate/onRemove)	X	X	X
Cache level (Add/Update/Remove)	X	X	X
Custom Events (fired by clients)	X	X	X
Continuous Query			X

Feature	Open Source	Professional	Enterprise
Object Caching Features			
Basic operations (Get, Add, Insert, Remove)	X	X	X
Bulk operations (Get, Add, Insert, Remove)	X	X	X
Async operations (Add, Insert, Remove)	X	X	X
Streaming API			X
Lock/Unlock (exclusive locking)	X	X	X
Item Versioning (optimistic locking)	X	X	X
Tags	X	X	X
Named Tags	X	X	X
Groups/Subgroups	X	X	X
Object Query Language - OQL	X	X	X
LINQ (.NET)			X
Portable Data Types	X	X	X
Read-Thru, Write-Thru, Write-Behind	X	X	X
Cache Loader	X	X	X
Evictions			
Priority Eviction	X	X	X
Least Recently Used (LRU) Eviction	X	X	X
Least Frequently Used (LFU) Eviction	X	X	X
Do Not Evict Option	X	X	X
Cache Management			
TayzGrid Manager (GUI tool)		X	X
TayzGrid Monitor (GUI tool)			X
Command line tools	X	X	X
Cache Management Java API	X	X	X
Cache Management .NET API			X
JMX/SNMP Java Client Counters	X	X	X
TayzGrid Email Alerts	X	X	X
Auto Restart & Join Cluster on Reboot	X	X	X
Multiple NIC Mapping in Cache Server & Client	X	X	X
Event Notifications on Cluster Changes	X	X	X
General Features			
Active Directory/LDAP Authentication			X
Authorization			X
Encryption (3DES, AES, ...)			X
Compression			X
Fast Compact Serialization			X
Indexing on Object Attributes	X	X	X

Feature	Open Source	Professional	Enterprise
Installation Package			
Client & Server Installer (.msi & .tar.gz)	X	X	X
.NET Client Installer (.msi)			X

2. Install & Configure TayzGrid

2.1 Install TayzGrid

You can install TayzGrid both on UNIX and Windows platforms through various installation modes, based on your needs and convenience.

2.1.1 *UNIX Platform*

On the UNIX platform you can install TayzGrid only in Command Line Mode through the following installation mode:

- 1. Custom Installation (.tar.gz):** In custom installation, you can specify customized parameters for installing TayzGrid. Without customization, TayzGrid is installed according to default values and in the same directory where .tar.gz is extracted.

Terminal Installation: In order to install TayzGrid Open Source extract the contents of the .tar.gz and execute the install script and provide the required parameters

2.1.2 *Windows Platform*

On the Windows platform, you can install TayzGrid Open Source through the command-line tool provided:

Command Line Installation Mode: TayzGrid provides a Command Line installation facility that you can either run from a Command Prompt or add to your script files (.bat files). To install TayzGrid from Command Prompt you use the msixec.exe utility. Through command line installation you can install/uninstall TayzGrid silently, without user intervention.

2.2 Cache Server Hardware Requirement

2.2.1 *Adequate RAM in Cache Servers*

Each TayzGrid server uses around 40-50MB memory as a JVM process and then puts 15-20% overhead on whatever you cache. Please keep this in mind when deciding how much memory to have in your cache servers. The total memory you need depends on how much data you plan to store in the cache. If you're storing JSP Servlet Sessions, then figure out your average session size and maximum number of sessions you will ever have in the cache. You can do the same for application data caching.

2.2.2 *1Gbit Network Interface Card in Cache Servers*

Cache servers should have a dedicated 1 Gbit or faster NIC. It is recommended that you use this NIC only for TayzGrid cluster and client communication in order to maximize its usage.

2.2.3 Dual-CPU, Quad-Core or higher

TayzGrid is highly multi-threaded and takes full advantage of extra cores and CPUs. The most common configuration for TayzGrid is a dual-CPU quad-core machine (meaning a total of 8 physical cores or 16 virtual cores per server).

You may need stronger processing power if you have higher transaction loads and/or larger amount of data being stored in TayzGrid.

2.2.4 Disk

TayzGrid does not make heavy use of disk space. Hence, you don't need any extra ordinary disk space in your cache server machines.

2.3 Cache Server Software Requirement

2.3.1 Windows 2008/2012 Server (64-bit)

TayzGrid requires Windows 2008/2012 Server 64-bit platform for cache servers.

2.3.2 Unix Server (64-bit)

TayzGrid is officially supported on Red Hat Enterprise Linux Server (v 6.4), Fedora (v 18), CentOS (6.3) and Ubuntu Server (12.10).

- **UNIX**

2. JRE/JDK 1.5 or above for Clients
3. JRE/JDK 1.6 or above for Servers

- **Windows**

4. .NET 2.0 or .NET 4.0 is required (required for tools and clients)
5. VC redistributable (JNI - .NET to C++ to Java)
6. VC redistributable is needed for read-through write-through feature
7. JRE/JDK 1.5 or above for Clients
8. JRE/JDK 1.6 or above for Servers

Note: It is recommended that you use the latest JDK/JRE for the Server.

2.4 Configure TCP Port for TayzGrid Clients

By default, all remote clients talk to TayzGrid servers on TCP port 9600. If your web/app servers are accessing the caching tier in a different subnet or through a firewall and you need to open specific ports to allow them to connect to cache servers, please open up this port in your firewall configuration.

If you want to change this port, then you must modify two TayzGrid configuration files. The first file is `[InstallDirectory]\config\server.properties` on each cache server machine and the second is `[InstallDirectory]\config\client.conf` on all your remote client machines.

2.4.1 Modify client.conf

Change the "port" in the cache-server tag below to specify a different port based on your environment preferences. You need to specify the following in `server.properties` which can be found at `[InstallDirectory]\config` for Windows and Linux.

```
<configuration>
  <cache-server port="9600" connection-retries="5" retry-interval="1" retry-
connection-delays="0" client-request-timeout="90" connection-timeout="5" local-
server-ip="/">
    ...
</configuration>
```

2.4.2 Modify server.properties

Change `CacheServer.Port` setting in `[InstallDirectory]\bin\service\server.properties` files of all servers. You need to specify the following in `server.properties` which can be found at `[InstallDirectory]\config` for Windows and Linux.

```
...
# Specify the port at which Cache Server will listen for the incoming connection
requests
CacheServer.Port                                =9600
...
```

The port value in both files (in `client.conf` and in `server.properties`) must match.

NOTE: Whenever you update `server.properties`, you must do it on all cache servers and then you must restart TayzGrid service afterwards. Otherwise, your changes will not take effect.

2.5 Map Cache Server to a Network Card

When your machine has more than one network interface cards (NIC), you can specify which NIC to use for cluster-wide communication as well as client/server communication between TayzGrid remote clients and the cache servers. Each NIC has its own IP-address. You need to specify the following in `server.properties` which can be found at `[InstallDirectory]\config` for Windows and Linux.

```
...

# When your machine has more than one network cards, you can inform TayzGrid
# which interface to use for cluster-wide communication. In order to do so,
# provide the IP Address you want the cluster server to bind with.
CacheServer.BindToClusterIP                      =20.200.20.235

# When your machine has more than one network cards, you can inform TayzGrid
# which interface to use for client communication. In order to do so, please
# provide the IP Address you want the client's (TayzGrid client) server to bind
with.
CacheServer.BindToClientServerIP                 =20.200.20.235

...
```

NOTE: Whenever you update `server.properties`, you must do it on all cache servers and then you must restart TayzGrid service afterwards. Otherwise, your changes will not take effect.

2.6 Configure Cache Size Notification Threshold

Cache size threshold specifies the size of cache in percentage of maximum cache size at which point TayzGrid should notify you. TayzGrid notifies you by logging an event in Event Log. You can then use third party tools to monitor Event Log and be notified through a variety of mediums.

This notification helps you increase cache capacity in-time either by increasing the maximum cache size if you have more memory available on cache servers or by adding a new cache server to the cluster to add more storage capacity. Without this ability, your cache would become full and start evicting items which you may not want.

You need to make the following change in `server.properties`.

```
...
# CacheSizeThreshold is the size of cache in percentage of total cache size. When
this threshold is reached a warning message is logged into system
# event log. No warning message is logged if the following line is commented.
CacheServer.CacheSizeThreshold           =80
...
```

NOTE: Whenever you update `server.properties`, you must do it on all cache servers and then you must restart TayzGrid service afterwards. Otherwise, your changes will not take effect.

3. Cache Administration

Once TayzGrid has been installed and you've specified all environment related settings, you're now ready to create a cache. TayzGrid Open Source edition allows you to create a Local Cache (meaning a stand-alone cache), Partitioned Cache, and Replicated Cache.

For other caching topologies, you need to use TayzGrid Enterprise. See more details on [edition comparison](#) between TayzGrid Open Source and Enterprise.

3.1 Create a Cache

3.1.1 Create a Local Cache

A local cache can be created through a command line tool called 'CreateCache'. This tool can be found under "[InstallDirectory]\bin".

Following is the command to create a local cache named 'locCache' of size 512 MB on cache server 20.200.20.20.

```
createcache locCache -s 20.200.20.20 -S 512 -t local
```

"-s" expects name or IP-address of a cache server as an argument. And, "-S" expects cache size in MB. And, "-t" expects you to specify a caching topology name like "local", "replicated", or "partitioned".

This command creates a configuration for the cache 'locCache' in "[InstallDirectory]\config\cache.config" file. You can always modify this config file directly if you need to change some values specified in this config. However, TayzGrid service needs to be restarted whenever this file is manually modified. Modifying it through createcache tool does not require you to restart TayzGrid service.

```
<configuration>
  <cache-config config-id="0">
    <cache-settings cache-name="locCache" alias="" inproc="False" bridge-target-
cache="False" last-modified="" auto-start="False" data-format="Binary" client-port="9625">
      <logging enable-logs="True" trace-errors="True" trace-notice="False" trace-
warnings="False" trace-debug="False"/>
      <performance-counters enable-counters="True"/>
      <compression enable-compression="False" threshold="500kb"/>
      <expiration-policy duration="0" default-policy="none"/>
      <cache-notifications item-remove="False" item-add="False" item-update="False"
cache-clear="False"/>
      <cleanup interval="15sec"/>
      <storage type="heap" cache-size="1024mb"/>
      <eviction-policy enable-eviction="True" default-priority="normal"
policy="priority" eviction-ratio="5%"/>
      <expiration-policy default-policy="none" duration="0"/>
      <cache-topology topology="local-cache">
    </cache-topology>
  </cache-settings>
</cache-config>
</configuration>
```

3.1.2 Create a Replicated Cache

Using `creatcache.exe`, a replicated cache named `repCache` can be created on one or more cache servers. The example below creates it on two cache servers (nodes) 20.200.20.20 and 20.200.20.125 of size 1024 MB using following command:

```
creatcache repCache -t replicated -s 20.200.20.20,20.200.20.125 -S 1024 -C 8700
```

"-t" expects you to specify a caching topology name like "local", "replicated", or "partitioned". And "-s" expects name or IP-address of a cache server as an argument. And, "-S" expects cache size in MB. "-C" is used to specify a TCP client port. Each client uses this port to connect to the cache.

This command creates a configuration for the cache `repCache` in `cache.config` file under `[InstallDir]\config` folder. You can always modify this config file if you need to change some values specified in this config. However, service needs to be restarted whenever the file is manually modified. Modifying it through `creatcache.exe` does not require you to restart TayzGrid service.

```
<configuration>
  <cache-config config-id="0">
    <cache-settings cache-name="repCache" alias="" inproc="False" last-modified="" auto-
start="False" data-format="Binary" client-port="8700" >
      <logging enable-logs="True" trace-errors="True" trace-notice="False" trace-
warnings="False" trace-debug="False"/>
      <performance-counters enable-counters="True"/>
      <compression enable-compression="False" threshold="500kb"/>
      <expiration-policy duration="0" default-policy="none"/>
      <cache-notifications item-remove="False" item-add="False" item-update="False"
cache-clear="False"/>
      <cleanup interval="15sec"/>
      <storage type="heap" cache-size="1024mb"/>
      <eviction-policy enable-eviction="True" default-priority="normal"
policy="priority" eviction-ratio="5%"/>
      <expiration-policy default-policy="none" duration="0"/>
      <cache-topology topology="replicated-server">
        <cluster-settings operation-timeout="60sec" stats-repl-interval="600sec" use-
heart-beat="False">
          <data-replication replication-strategy="False"/>
          <cluster-connection-settings port-range="1" connection-retries="10"
connection-retry-interval="2secs" join_retry_count="24" join_retry_timeout="5"/>
        </cluster-settings>
      </cache-topology>
    </cache-settings>
    <cache-deployment>
      <servers>
        <server-node ip="20.200.20.20" active-mirror-node="False"/>
        <server-node ip="20.200.20.125" active-mirror-node="False"/>
      </servers>
    </cache-deployment>
  </cache-config>
</configuration>
```

Note: The client port that is provided should be as such so as to make sure that the subsequent 5 ports are also unused by either TayzGrid service or any other configured cache. It is recommended that the client port allotment be left to TayzGrid.

3.1.3 Create a Partitioned Cache

Similarly, we can create a partitioned cache 'partCache' of one or more cache servers. The example below creates cache on two nodes using following command.

```
createcache partCache -t partitioned -s 20.200.20.20,20.200.20.125 -S 1024 -c 8710
```

"-t" expects you to specify a caching topology name like "local", "replicated", or "partitioned". And "-s" expects name or IP-address of a cache server as an argument. And, "-S" expects cache size in MB. "-c" is used to specify a TCP client port. This is the port used by the client to communicate to the Server.

Please note that in case of Partitioned Cache "-S" specifies not the total cache size but the size of each individual partition. This means that the total cache size is the sum of all partitions.

The above command produces following cache entry in the config file.

```
<configuration>
<cache-config config-id="0">
  <cache-settings cache-name="partCache" alias="" inproc="False" bridge-target-
cache="False" last-modified="" auto-start="False" data-format="Binary" client-port="8710">
    <logging enable-logs="True" trace-errors="True" trace-notice="False" trace-
warnings="False" trace-debug="False"/>
    <performance-counters enable-counters="True"/>
    <compression enable-compression="False" threshold="500kb"/>
    <expiration-policy duration="0" default-policy="none"/>
    <cache-notifications item-remove="False" item-add="False" item-update="False"
cache-clear="False"/>
    <cleanup interval="15sec"/>
    <storage type="heap" cache-size="1024mb"/>
    <eviction-policy enable-eviction="True" default-priority="normal"
policy="priority" eviction-ratio="5%"/>
    <expiration-policy default-policy="none" duration="0"/>
    <cache-topology topology="partitioned">
      <cluster-settings operation-timeout="60sec" stats-repl-interval="60sec" use-
heart-beat="False">
        <data-replication replication-strategy="False"/>
        <cluster-connection-settings port-range="1" connection-retries="10"
connection-retry-interval="2secs" join_retry_count="24" join_retry_timeout="5"/>
      </cluster-settings>
    </cache-topology>
  </cache-settings>
</cache-deployment>
<servers>
  <server-node ip="20.200.20.125" active-mirror-node="False"/>
</servers>
</cache-deployment>
</cache-config>
</configuration>
```

Note: The client port that is provided should be as such so as to make sure that the subsequent 5 ports are also unused by either TayzGrid service or any other configured cache. It is recommended that the client port allotment be left to TayzGrid to avoid conflicts between caches.

3.2 Start/Stop Cache

You can start and stop a cache using tools `'startcache'` and `'stopcache'` located under `[InstallDirectory]\bin\tools` for Windows and `[InstallDirectory]\bin` for Linux. For example, following commands start and stop one or more named caches.

```
startcache repCache partCache
startcache repCache -s 20.200.20.125

stopcache repCache partCache
stopcache repCache -s 20.200.20.125
```

The first command start/stops cache on the server where this command is run. For a "local cache", this stops the entire cache. But, for a Replicated or Partitioned Cache with multiple nodes in the cluster, this stops the cache only on one server.

You can also start/stop the cache on a remote cache server by specifying the IP-address of that cache server with `"-s"` switch. So, if you want to start/stop the cache on all servers in the cluster, you must run start/stop command against each cache server.

3.3 Add/Remove Remote Clients

You can use TayzGrid command line tool `'addclientnode'` to add a client node to the cache and `'addclientnode'` to remove a remote client from the cache. Following command adds and removes 20.200.20.31 as a client node to the cache cluster `'repCache'`.

```
addclientnode repCache -e 20.200.20.31
addclientnode repCache -e 20.200.20.31 -s 20.200.20.20

removeclientnode repCache -e 20.200.20.31
removeclientnode repCache -e 20.200.20.31 -s 20.200.20.20
```

`"-e"` expects you to specify the client node. `"-s"` expects you to specify a remote cache server and if you don't specify it then local machine is assumed to be the cache server.

3.4 Add/Remove Cache Servers

You can always add a cache server to the cluster at any time using the tool `'addnode'` which is located under `[InstallDirectory]\bin\tools` for windows and `[InstallDirectory]\bin` for Linux. You can use `'removenode'` to remove a server from the cluster.

Following command adds and removes a cache node 20.200.20.54 from the named cache `'repCache'`.

```
addnode repCache -x 20.200.20.20 -N 20.200.20.54
```

`"-x"` expects you to specify the existing cache server, `"-N"` expects you to specify the new cache server to add. You can use remove a cache server 20.200.20.54 from the cache cluster `'repCache'` using following command.

```
removenode repCache -s 20.200.20.54
```

"-s" expects you specify the cache server to remove from the cluster.

3.5 Test the Cache Cluster

Before you should start using the cache with your application, you need to first test to make sure the cache is working properly. In order to do that you need to add some test data to the cache and then monitor to see if the data has been added or not. Here is how you can do this.

1. Start cache on all cache servers in the cluster.
2. Verify the Cluster Health by using 'listcaches' command against cache server 20.200.20.20. It shows all the caches on registered on that server and for the ones that are running it shows which cache servers are currently part of the cache cluster. Here it is (see details on this in "View Cluster Health" section later in this document):

```
listcaches -s 20.200.20.20 -a
```

"-s" asks you to specify name or IP-address of a cache server. "-a" says to display details for all caches.

- For each running cache you can use the JMX port to view the counters for it. The JMX port can be used in the jconsole.exe to view the counters of the cache. In order to determine the JMX port for a cache you can add 1502 in the client port. Then simply connect to the process via remote login through jconsole.exe.
3. Run "stresstesttool" command-line program provided with TayzGrid to add, update, and fetch data from the cache. It is located under [InstallDirectory]\bin\tools folder. Run it as follows from either a remote client or a cache server machine:

```
stresstesttool repCache
```

4. Verify that the above mentioned counters show the right values according to the test data you added.

If the above happens, then you can rest assured that you've configured the cache correctly. You can either let the "test data" expire in 5 minutes or manually clear the cache using tool 'clearcache' located under [InstallDirectory]\bin\tools.

```
clearcache repCache
```

3.6 Create Indexes for SQL Queries

TayzGrid requires you to define indexes on all searchable attributes used in SQL queries WHERE clause. This is because, without indexing, TayzGrid would have to traverse the entire cache in order to find items. This would be very costly operation with potential of slowing down the entire cache.

TayzGrid provides its own indexing mechanism. First you define an index on an object attribute and this becomes part of the cache configuration. Then, when cache is started and the items are added or updated in the cache, TayzGrid uses Java Reflection to extract data from these items and populates the

index with it. And, when items are removed from the cache, their corresponding data is also removed from the index.

Once the index is populated, then when you run SQL queries, they are executed first against the indexes to find the corresponding data and then returned to your application very quickly.

Suppose that cache contains Product object where the definition of Product is as below:

```
class Product implements Serializable {  
  
    int ProductID;  
    String ProductName;  
    int UnitsInStock;  
    int Category;  
}
```

You can add query indexes on selective attributes of any type using command line tool

'addqueryindex.exe' located under [InstallDirectory]\bin\tools as follows:

```
addqueryindex.exe repCache -a C:\Program Files\TayzGrid\samples\data\dist\data.jar -c  
TayzGrid.Sample.Data.Product -L ProductID$ProductName$Category$UnitsInStock
```

"-a" switch asks for a jar with its path. "-c" asks for the class name. "-L" asks you to specify one or more attribute names separated by '\$' sign.

The above command adds query indexes on attributes 'ProductID', 'ProductName', 'Category' and 'UnitsInStock' of above mentioned class 'Product' defined in an assembly 'C:\Program Files\TayzGrid\samples\data\dist\data.jar'. Please note that all attributes are separated by '\$' sign.

You must restart the cache after defining or removing an index definition (not the data but only the index definition).

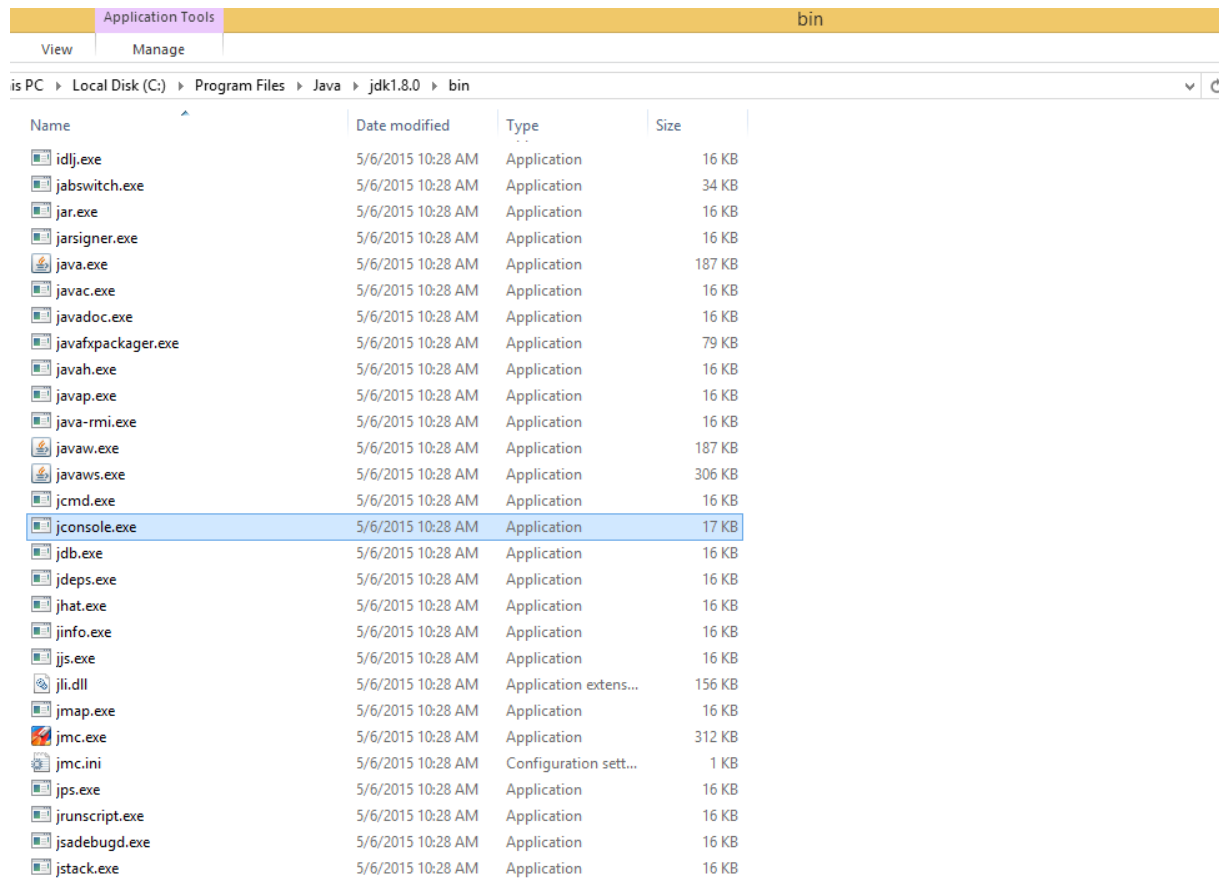
4. Cache Monitoring

4.1 Java Monitoring and Management Console

TayzGrid publishes its performance counters through JConsole. You can monitor TayzGrid counters for a specific cache on any Windows machine using JConsole tool that comes with the java installation.

4.1.1 Monitoring Cache Server Counters using JConsole Tool

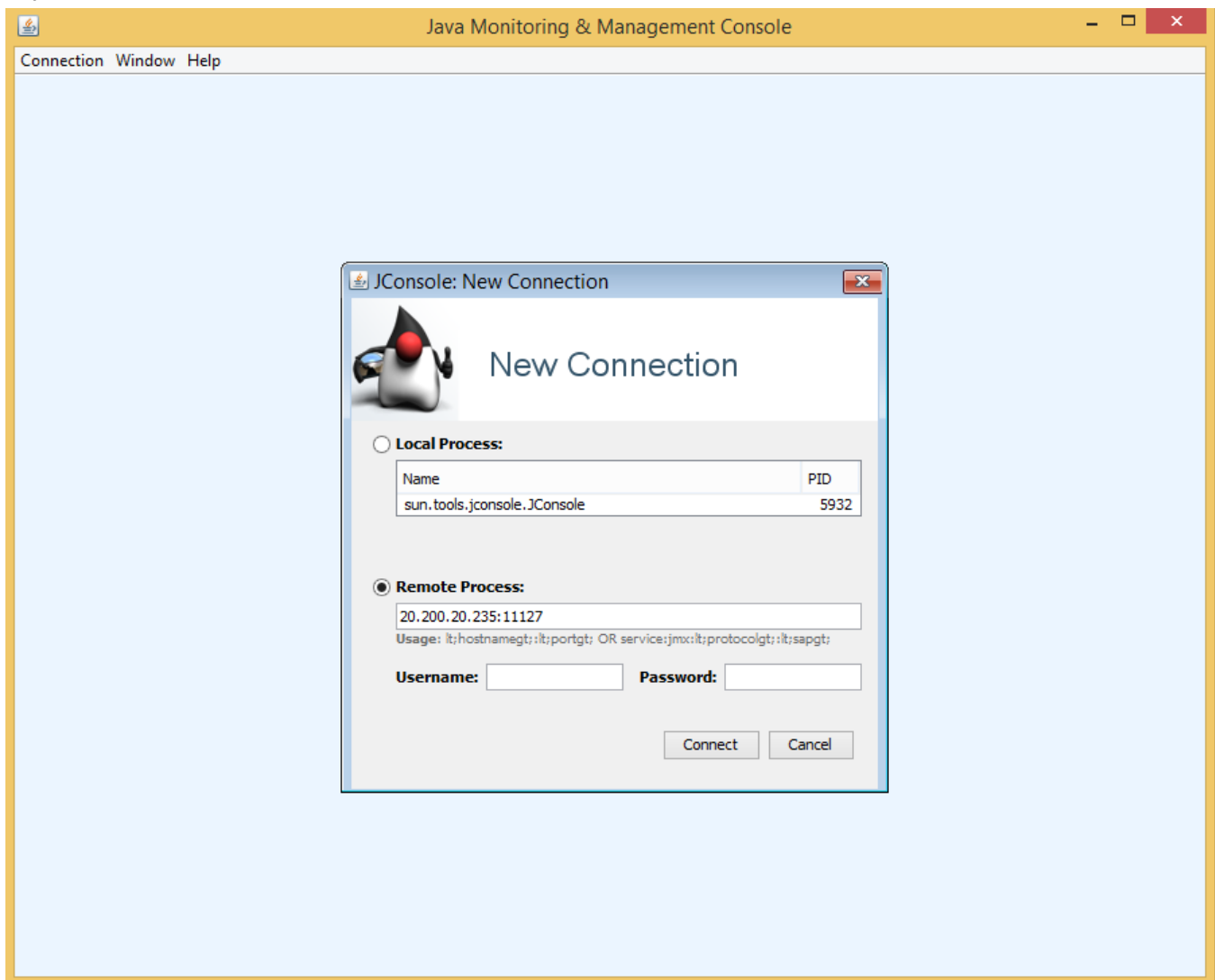
TayzGrid publishes cache server counters in MBeans under category TayzGrid. This category has all counters related to the cache server. Follow the steps given below to monitor the TayzGrid counters through JConsole tool:



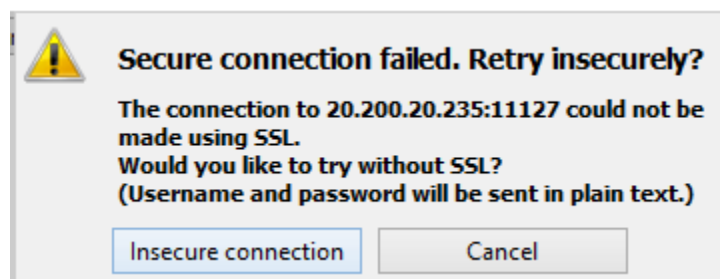
The screenshot shows a Windows Explorer window titled 'Application Tools' with the address bar displaying 'bin'. The breadcrumb path is 'is PC > Local Disk (C:) > Program Files > Java > jdk1.8.0 > bin'. The file list below shows various Java executables and configuration files. The file 'jconsole.exe' is highlighted in blue.

Name	Date modified	Type	Size
idlj.exe	5/6/2015 10:28 AM	Application	16 KB
jabswitch.exe	5/6/2015 10:28 AM	Application	34 KB
jar.exe	5/6/2015 10:28 AM	Application	16 KB
jarsigner.exe	5/6/2015 10:28 AM	Application	16 KB
java.exe	5/6/2015 10:28 AM	Application	187 KB
javac.exe	5/6/2015 10:28 AM	Application	16 KB
javadoc.exe	5/6/2015 10:28 AM	Application	16 KB
javafxpackager.exe	5/6/2015 10:28 AM	Application	79 KB
javah.exe	5/6/2015 10:28 AM	Application	16 KB
javap.exe	5/6/2015 10:28 AM	Application	16 KB
java-rmi.exe	5/6/2015 10:28 AM	Application	16 KB
javaw.exe	5/6/2015 10:28 AM	Application	187 KB
javaws.exe	5/6/2015 10:28 AM	Application	306 KB
jcmd.exe	5/6/2015 10:28 AM	Application	16 KB
jconsole.exe	5/6/2015 10:28 AM	Application	17 KB
jdb.exe	5/6/2015 10:28 AM	Application	16 KB
jdeps.exe	5/6/2015 10:28 AM	Application	16 KB
jhat.exe	5/6/2015 10:28 AM	Application	16 KB
jinfo.exe	5/6/2015 10:28 AM	Application	16 KB
jjs.exe	5/6/2015 10:28 AM	Application	16 KB
jli.dll	5/6/2015 10:28 AM	Application extens...	156 KB
jmap.exe	5/6/2015 10:28 AM	Application	16 KB
jmc.exe	5/6/2015 10:28 AM	Application	312 KB
jmc.ini	5/6/2015 10:28 AM	Configuration sett...	1 KB
jps.exe	5/6/2015 10:28 AM	Application	16 KB
jrunscript.exe	5/6/2015 10:28 AM	Application	16 KB
jsadebugd.exe	5/6/2015 10:28 AM	Application	16 KB
jstack.exe	5/6/2015 10:28 AM	Application	16 KB

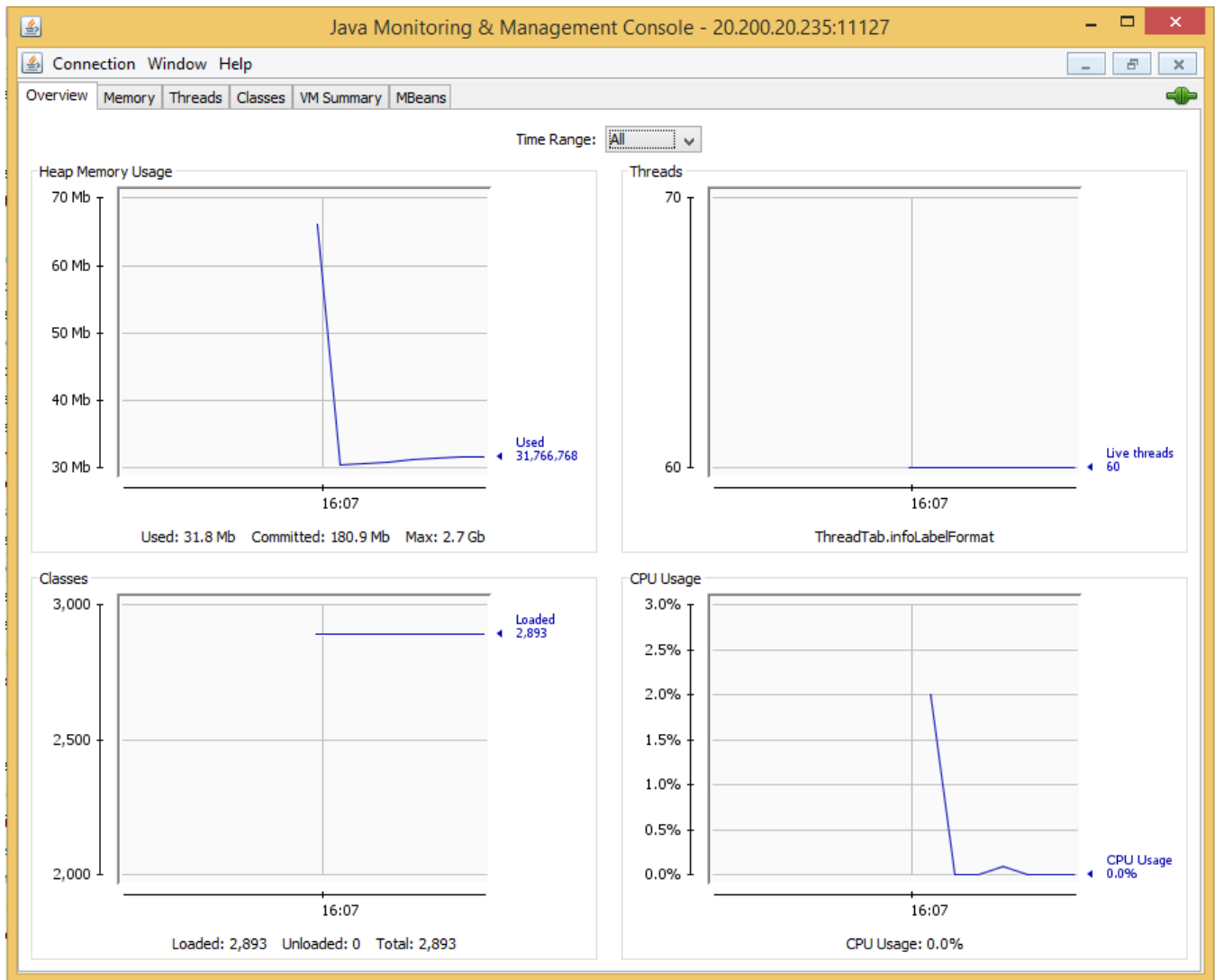
- Go to the java installation directory, for Windows it will be something like C:\Program Files\Java\jdk(version)\bin. Run the jconsole.exe application.



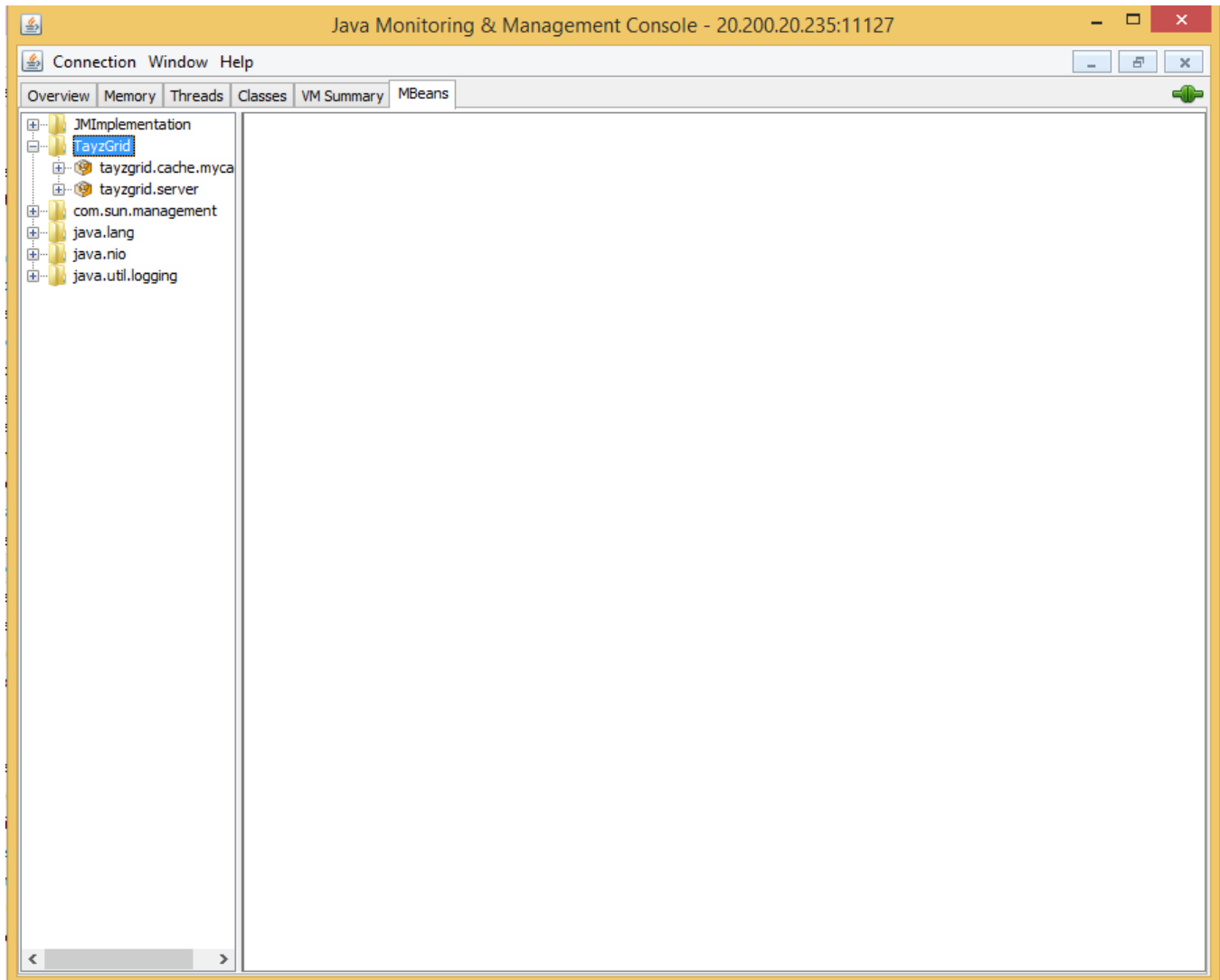
- Select the remote connection option and input the host address and the calculated JMX port, i.e. Client port + 1502. This can be a Windows or Linux machine.



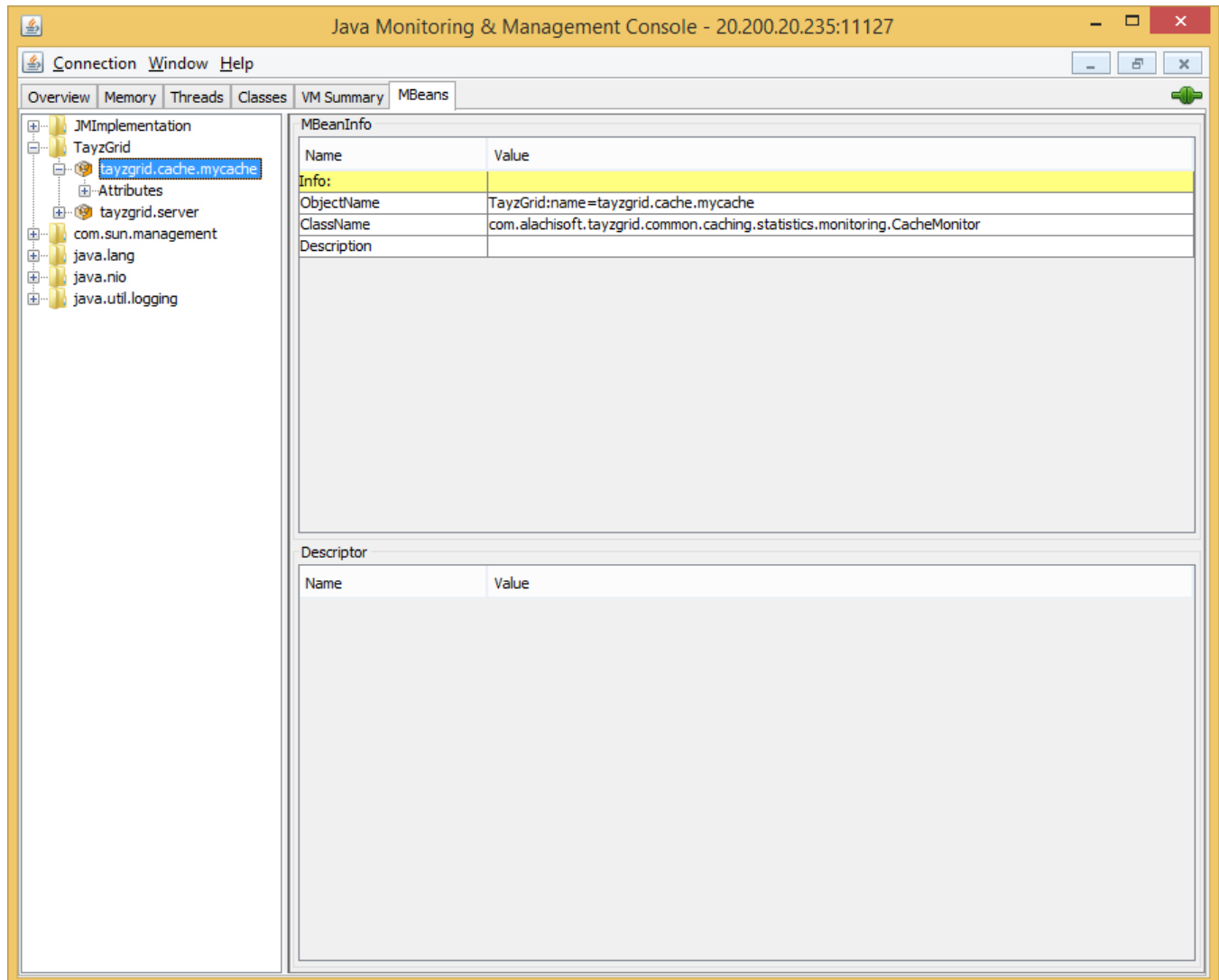
- If prompted for insecure connection, continue anyway.



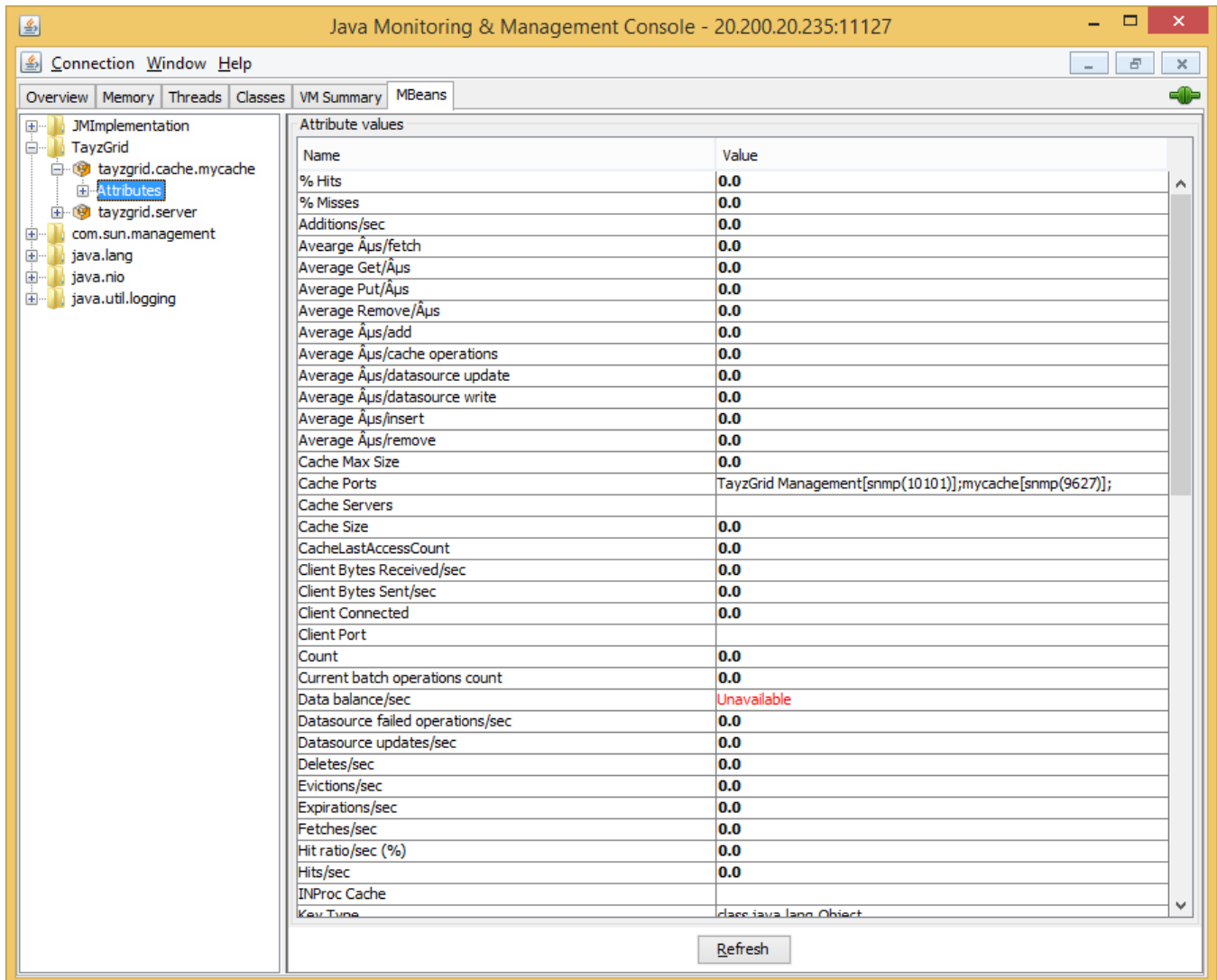
- Graph view will be shown on successful connection.



- Press the MBeans tab under the toolbar to display the registered counters.



- Expand the TayzGrid folder and click on the target which has the cache name in the end, e.g. tayzgrid.cache.mycache in case of mycache instance running.



Java Monitoring & Management Console - 20.200.20.235:11127

Connection Window Help

Overview Memory Threads Classes VM Summary MBeans

Tree View:

- JMImplementation
 - TayzGrid
 - tayzgrid.cache.mycache
 - Attributes**
 - tayzgrid.server
 - com.sun.management
 - java.lang
 - java.nio
 - java.util.logging

Attribute values

Name	Value
% Hits	0.0
% Misses	0.0
Additions/sec	0.0
Average Âµs/fetch	0.0
Average Get/Âµs	0.0
Average Put/Âµs	0.0
Average Remove/Âµs	0.0
Average Âµs/add	0.0
Average Âµs/cache operations	0.0
Average Âµs/datasource update	0.0
Average Âµs/datasource write	0.0
Average Âµs/insert	0.0
Average Âµs/remove	0.0
Cache Max Size	0.0
Cache Ports	TayzGrid Management[snmp(10101)];mycache[snmp(9627)];
Cache Servers	
Cache Size	0.0
CacheLastAccessCount	0.0
Client Bytes Received/sec	0.0
Client Bytes Sent/sec	0.0
Client Connected	0.0
Client Port	
Count	0.0
Current batch operations count	0.0
Data balance/sec	Unavailable
Datasource failed operations/sec	0.0
Datasource updates/sec	0.0
Deletes/sec	0.0
Evictions/sec	0.0
Expirations/sec	0.0
Fetches/sec	0.0
Hit ratio/sec (%)	0.0
Hits/sec	0.0
INProc Cache	
Key Type	class java.lang.Object

Refresh

- Select the Attributes property listed under the cache, the window will display all the available counters for the cache.

4.2 View Cluster Health

Cluster health can be verified using 'listcaches.exe' tool located under [InstallDirectory]\bin\tools.

Following command displays cluster health of all caches registered on node 20.200.20.20.

```
listcaches.exe -a -s 20.200.20.20
```

This command generates an output similar to the following.

```
Alachisoft (R) TayzGrid Utility ListCaches. Version 4.4.0.0
Copyright (C) Alachisoft 2015. All rights reserved.
```

```
Listing registered caches on server 20.200.20.20:8250
```

```
Cache-ID:      repCache
Scheme:        replicated-server
Status:        Running
Cluster size:  2
                20.200.20.125:8710
                20.200.20.20:8710
UpTime:        1/13/2015 12:41:23 PM
Capacity:      1024 MB
Count:         1000000
```

The list of cache servers shown above indicates that these cache servers are successfully part of the cluster. Any cache server that has failed to join the cluster will not show up here. Additionally, if you have a partially connected cluster (meaning some nodes have formed one cluster and others have formed a second one), you'll only see a subset of cache servers in the above list.

So, this command is very useful in quickly seeing whether all the desired servers are part of the cluster or not.

4.3 Server Log Files

Server logs for each started cache are created under the [InstallDirectory]\log folder. These logs are very helpful in finding out the possible causes of any failures or undesired behaviors in cache. Log file name is created in following format *CacheName_timeStamp_CacheServerIP*. All Cluster level information/error will be logged in server logs.

TIMESTAMP	THREADNAME	LEVEL	MESSAGE
2015-01-22 11:15:46,225	28	INFO	gms_id :
2015-01-22 11:15:46,249	28	INFO	ConnectionTable.Start operating parameters -> [bind_addr :20.200.20.21:
2015-01-22 11:15:46,292	28	INFO	TCP.start operating parameters -> [heart_beat:False ;heart
2015-01-22 11:15:46,325	28	INFO	pb.ClientGmsImpl.join() no initial members discovered: creating group as

You don't have to enable server logging. It is done automatically.

4.4 Java Session Store Provider Logs

Java log files are generated in `[InstallDirectory]\log\SessionStoreProvider` folder. You have to set the `enableLogs` attribute `true` in `web.config` to generate these log files.

```
<providers>
  <add name="TayzGridSessionProvider"
    type="Alachisoft.TayzGrid.Web.SessionState.NSessionStoreProvider"
    exceptionsEnabled="true" enableSessionLocking="true" sessionAppId="TayzGridTest"
    useInProc="false" enableLogs="true" cacheName="mypartitionedcache" AsyncSession="false"
    />
</providers>
```

Java Session state log file name is created in following format *CacheName.Timestamp_NodeIPAddress*.

4.5 Client Side API Logging

TayzGrid Client Side API Logging is a diagnostic feature which logs all TayzGrid API calls made by an application. This is a configurable logging mechanism that creates several log files.

The purpose of API logging is to identify which API and the sequences causing the problem, so that the problem can be reproduced easily at your end. Client API logging saves an appropriate overload of the method called along with *key*, *data size*, *cache status*, *encryption* and *compression* information.

Note: Since logging has a performance impact, it is recommended to enable the logging only for diagnostic purposes.

5. TayzGrid Java Session Module

JSP session provides the mechanism by which a user is identified across more than one page request. For this purpose, various servlet containers provide session replication mechanisms. For instance, Tomcat web server provides sessions handling through sticky sessions or session replication over tomcat web servers cluster. Sticky sessions persist on a server, which receives the web request, which means you can't guarantee failover recovery if that server node goes down. Also session replication over multiple server instances may cause memory and data transfer overhead. Session clustering replicates the session but is slow. And, both options have serious scalability issues. Similarly Web logics provides in memory session replication (replicates session state from one instance to another) or JDBC-based persistence which may cause performance issues. In short, available options in different servlet containers are either slow in performance, low or no scalability, or lack of reliability. And, these problems become more severe when your Java servlet application runs in a load-balanced web server farm.

An alternative option is to use an in-memory distributed data grid like TayzGrid for persistence of your Java Servlet Session. TayzGrid supports session clustering without compromising on high performance and scalability.

5.1 Conceptual Overview

5.1.1 *Session Persistence in TayzGrid*

TayzGrid provides you with a "no code change" option for java applications to persist sessions in TayzGrid distributed caching. If your application is running in a web server farm with load balanced configured and you need a reliable and scalable storage for your session persistence, then TayzGrid provides you with advantages like:

- **Performance:** TayzGrid has high performance as it provides extremely fast response time.
- **Scalability:** It's an in-memory distributed data grid which provides linear scalability. As you increase your cache cluster nodes to tackle extreme transaction load, performance also improves and hence you can achieve as much scalability as needed.
- **Session replication topologies:** TayzGrid provides different caching topologies through which you persist your sessions according to your needs. For instance you can use replicated topology for cache clusters which provide more failover support.
- **Reliability:** TayzGrid lets you intelligently replicate java servlet sessions across multiple servers according to used topology without compromising reliability and performance. So, you won't lose any session data even if a cache server goes down.
- **Availability:** TayzGrid provides strong distributed cache clusters with no single point of failure. In this way, you can change your cluster span adding or removing cache servers at runtime without stopping your application or the cache.

5.1.2 *Using Java Session through TayzGrid with No Code Change*

TayzGrid provides you with the ability to configure a caching cluster for session persistence without any code change in your application. All you need to do is to modify your application configuration file and add TayzGrid session storage support as a Filter to your java servlet application.

TayzGrid session provider will be configured as a filter in your web application. You also need to change session persistence settings in *session.xml* file for cache id and session mode settings. TayzGrid session provider filter will intercept http request/response for sessions' management and eventually store this session in configured cache.

5.1.3 Session Expiration

TayzGrid handles session expiration and removes sessions from cache when specified session time out has elapsed. You can specify session timeout value through servlet containers and TayzGrid will internally use that interval to expire sessions from cache. TayzGrid uses sliding expiration, one of its own features, to invalidate session. Session are inserted in cache with a sliding expiration of specified "session time out" interval. This interval will be reset every time that session is accessed. Sessions will be removed from cache if it will remain inactive for the specified "session time out" interval.

5.1.4 Handling Multiple Session Requests and Data Integrity

Parallel HTTP request for the same session in case of AJAX can cause session to be mutated independently. To avoid such a situation, TayzGrid session persistence uses locking mechanism so that no two parallel requests can change the same session. First requests will acquire lock on the session and other requests will wait, until the first request is completed and it releases the lock.

TayzGrid provides a no-code change option for Java based web applications to store sessions in TayzGrid by implementing session module *tg-sessionprovider.jar* as a Java filter. In order to use TayzGrid java session module, you must be running a Servlet 2.3 compatible container (web server). The user only needs to reference a few libraries and add a filter in *web.xml* of the application so that all the requests will come to the application through the TayzGrid filter.

Following J2EE platforms are supported:

- Tomcat
- JBoss
- Web Logic
- WebSphere

5.2 Configuring Application to Use Java Session Module

TayzGrid provides a no-code change option for Java applications for storing sessions in TayzGrid distributed caching. Following are the steps that the user needs to follow to configure a Java application for using TayzGrid as a distributed session store.

5.2.1 Adding Libraries

The user needs to add all libraries located at *%TG_HOME%/lib* and *%TG_HOME%/lib/resources* and *%TG_HOME%/lib/integrations/tg-sessionprovider.jar*. If you are using IDE just add references to these files in the project, if you are not using IDE then you need to copy these files to '*WEB-INF/lib*' folder under the application root folder.

- **On Linux:** *\$TG_HOME/lib/*
- **On Windows:** *%TG_HOME% /lib/*

5.2.2 Defining Filter

After adding the required libraries in your application, you need to add the filter in the deployment descriptor of your web application. The deployment descriptor is a XML file named '*web.xml*' located in

the WEB-INF directory under the application root directory. The class file containing the filter implementation is

'com.alachisoft.tayzgrid.web.sessionstate.tayzgridSessionStoreProvider'

To ensure the correct caching of sessions, you must apply this filter as the first filter in your deployment descriptor. Filters are executed in the order they are defined in deployment descriptor.

To configure a filter, you must define it first under the <filter> tag and then provide a URL mapping for the filter using <filter-mapping> tag. The following filter configuration means that the filter will be applied to all the URLs in a web application.

```
<filter>
    <filter-name>TayzGridSessionProvider</filter-name>
    <filter-
class>com.alachisoft.tayzgrid.web.session.tayzgridSessionProvider</fil
ter-class>
</filter>
<filter-mapping>
    <filter-name>TayzGridSessionProvider</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

TayzGrid Session filter can be configured with the settings specified in 'session.xml' file located at %TG_HOME%/config. You must specify configPath with filter initialization parameter in order to load the configuration settings.

```
<init-param>
    <param-name>configPath</param-name>
    <param-value>%TG_HOME%/config</param-value>
</init-param>
```

Note: Please note that the value of configPath must be the installation directory of TayzGrid.

The following is an example of how the deployment descriptor should look like after defining the filter.

```
<session-config>
    <session-timeout>30</session-timeout>
</session-config>
<filter>
    <filter-name>TayzGridSessionProvider</filter-name>
    <filter-
class>com.alachisoft.tayzgrid.web.session.tayzgridSessionProvider</fil
ter-class>
    <init-param>
        <param-name>configPath</param-name>
        <param-value>%TG_HOME%/config</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>TayzGridSessionProvider</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

You can configure session module as single-regional or multi-regional module.

1. Single-Regional Session State Provider:

For single-regional session state provider specify 'session.xml' as:

```
<?xml version="1.0" encoding="UTF-8"?>
<servlet-session-config>
  <cache id="mycache"/>
  <log log-props="log4j.properties"/>
  <locking enable-session-locking="true" retries-count="2" retry-
interval="500ms" lock-timeout="360000ms" empty-session-when-
locked="false"/>
</servlet-session-config>
```

The following is the description of different key-value pairs specified above:

Member	Description
Id	Name of running cache that will be used to persist session state
log-props	Name of the properties file to load logging settings.
enable-session-locking	If set to true, TayzGrid Session Store Provider exclusively locks the session-store item for which multiple concurrent requests are made.
retries-count	If <code>enable-session-locking</code> is set to true and this integer is not less than 0, TayzGrid Session Store Provider will return empty session after <code>retries-count</code> , which specify the number of retries to acquire a lock.
retry-interval	It specifies the time interval between each operation retry, in case the requested session is locked.
lock-timeout	It specifies the time after which a locked session is unlocked, i.e session lock is released.
empty-session-when-locked	If <code>enable-session-locking</code> and <code>empty-session-when-locked</code> are both set to true, TayzGrid Session Store Provider will return empty session after <code>retries-count</code> , otherwise throw and log an exception notifying that item is locked even after retries count.

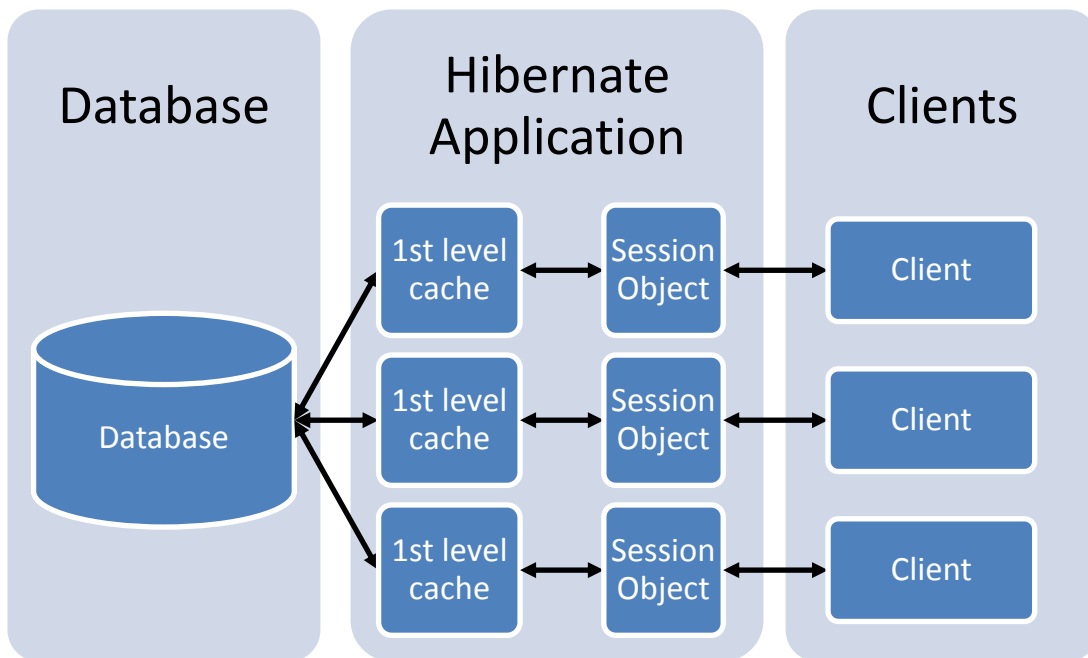
6. TayzGrid as Hibernate Second Level Cache

6.1 Conceptual Overview

Hibernate is an open source **Object-Relational Mapping** solution which provides a framework for mapping relational databases on an object oriented representation of data. This reduces significant developer's efforts for retrieving data from the database. Furthermore, to improve applications' performance and reduce load on the database, Hibernate provides the features of caching to reduce repetitive database calls.

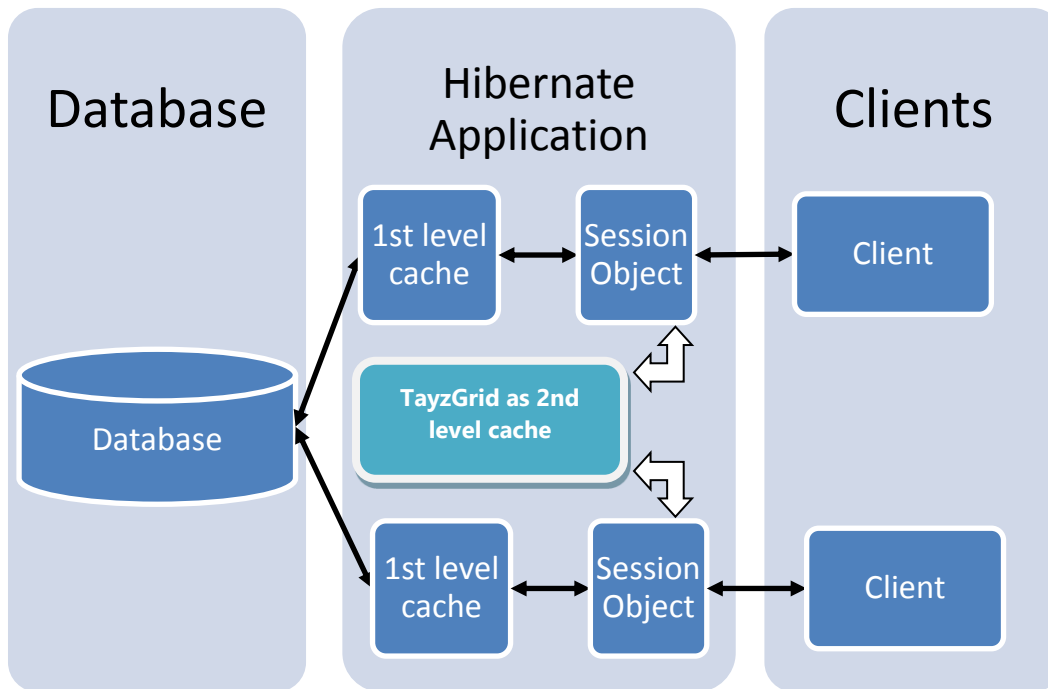
6.1.1 *Hibernate 1st Level Cache*

Hibernate's 1st level caching module provides session level in the process caching. Whenever a session is created, a session cache is associated with it. When an object is loaded from the database, a copy of the object is maintained in the session cache. If the object is to be fetched again in the same session, the cached copy of object is returned. In a particular transaction, if the same object is updated multiple times, it is only updated in session cache. At committing transaction, only the final state of object is persisted in the database, thus avoiding repetitive update calls.



6.1.2 *TayzGrid as Hibernate 2nd level Cache*

TayzGrid provides **Hibernate's** 2nd level cache provider. TayzGrid provider can be configured as Hibernate's 2nd level cache **without any code change**. Using TayzGrid as 2nd level cache for Hibernate enables applications to use TayzGrid's distributed caching features.



TayzGrid provides the following features with 2nd level cache provider:

- **Specify Multiple Regions:**

Multiple Hibernate regions can be specified and configured in TayzGrid's configuration file for Hibernate. These regions are identified using region names and can be configured with multiple TayzGrid features.

Each region can be configured with TayzGrid caching features.

- **TayzGrid's Cache for Region:**

TayzGrid allows you to specify TayzGrid's cache instance for each region separately. Each region can be configured on separate TayzGrid's cache instance or one instance can also be used for multiple regions.

- **Cache Item Priority:** Data in each region can be configured to have a priority in cache. This priority is then used for data eviction policies.
- **Absolute Expiration:** TayzGrid's configurable absolute expiration feature keeps data in cache up to date by expiring old data after a configured time interval.
- **Sliding Expiration:** Cache region can also be configured to have an absolute expiration for items in cache. Using sliding expiration data which is not used since the configured interval will be removed automatically by TayzGrid.

- **Configure Multiple Regions Even on Single TayzGrid's Cache Instance:**

TayzGrid provider for Hibernate allows you to use a single TayzGrid's instance for multiple Hibernate cache instances. TayzGrid keeps record of data and its associated Hibernate region.

- **Database Dependencies:**

To ensure data consistency with database, TayzGrid provides a database dependency feature for Hibernate applications that can be configured on entity bases. Database dependency synchronizes cache data with the database such that any change in the database record

invalidates the respective entity in cache. Thus Hibernate retrieves an up to date record even if the database is updated by an application other than Hibernate application.

- **Query Caching Feature to Cache Large Queries' Results:**

Complex and large queries if executed repeatedly with no result change can put a huge load on the database. To avoid multiple executions of such queries, TayzGrid allows you to cache such queries using Hibernate's query cache feature. So that such query and its results executed for one Hibernate application become available to all Hibernate applications using that database without querying the database.

- **No Code Change Required:**

TayzGrid allows you to use all features through configuration files with no code change required.

6.2 How to Use TayzGrid as Hibernate Second Level Cache

Second level cache can be configured by enabling the use of L2 cache and configuring L2 cache provider in Hibernate configurations. Please refer to the following steps to use TayzGrid as Hibernate's second level cache:

1. Configure Hibernate application to use TayzGrid as 2nd level cache
2. Configure application objects as cacheable
3. Configure cache regions in TayzGrid configuration file
4. Use query caching if needed

Following sections explain in detail each of the above step:

6.2.1 How to Configure Hibernate Application

TayzGrid has to be configured in application's Hibernate configuration file as second level cache. Following is a sample hibernate's configuration file **hibernate.cfg.xml** with TayzGrid caching configuration.

```
<hibernate-configuration>
  <session-factory>
    ...
    <!-- Enable the second-level cache -->
    <property name="hibernate.cache.use_second_level_cache">true</property>
    <!-- Configure TayzGrid as second level cache -->
    <property name =
"hibernate.cache.region.factory_class">com.alachisoft.tayzgrid.integrations.hiberna
te.cache.TayzgridRegionFactory</property>
    <!-- Specify application id for TayzGrid -->
    <property name="tayzgrid.application_id">myapp</property>
    <!-- Enable the query cache -->
    <property name="hibernate.cache.use_query_cache">true</property>
    ...
  </session-factory>
</hibernate-configuration>
```

Properties added in hibernate's configuration file are explained as under:

1. Enable use of second level cache in your application by adding the following property in Hibernate configuration section's **session-factory** tag:

```
<property name="hibernate.cache.use_second_level_cache">true</property>
```
2. Configure TayzGrid as Hibernate's second level cache provider.

For Hibernate 3.6 and above add following property in Hibernate's configuration file under **session-factory** tag:

```
<property name =
  "hibernate.cache.region.factory_class">com.alachisoft.tayzgrid.integrations.hibe
  rnate.cache.TayzGridRegionFactory
</property>
```

hibernate.cache.region.factory_class: This option lets you specify TayzGrid's region cache factory class Hibernate 3.6 and above. Hibernate will use this cache factory class to initialize and use TayzGrid as second level cache.

For Hibernate versions 3.5 and below instead of `hibernate.cache.region.factory_class` add following property in Hibernate's configuration file under **session-factory** tag :

```
<property
  name="hibernate.cache.provider_class">com.alachisoft.tayzgrid.integrations.hiber
  nate.cache.TayzGridProvider
</property>
```

hibernate.cache.provider_class: This option lets you specify TayzGrid as second level cache provider. You need to mention TayzGrid's cache provider class for Hibernate. This is how Hibernate knows how to call second level cache.

3. TayzGrid provider for Hibernate identifies each application by an application id that is later used to find appropriate configuration for that application from TayzGrid configuration file for Hibernate "TayzGridHibernate.xml". This application id must be specified in hibernate's configuration file. Add following property in hibernate's configuration **session-factory** tag for this purpose:

```
<property name="tayzgrid.application_id">myapp</property>
```

4. To use TayzGrid as query cache, enable query cache by adding following property:

```
<property name="hibernate.cache.use_query_cache">true</property>
```

After configuring Hibernate application to use TayzGrid, place TayzGrid's Hibernate provider jar **tg-hibernate.jar** located at %TayzGridInstallDir%/lib/integration/ in application's class path.

In case of Hibernate version 3.5 and below, build source located at %TayzGridInstallDir%/integration/hibernate-3.5 for required jar. Similarly add all jar files located at following directories to application's class path:

- %TayzGridInstallDir%/lib
- %TayzGridInstallDir%/lib/resources

6.2.2 How to Configure Cacheable Objects

Enabling use of second level cache does not cache each class's object by default. Instead classes that need to be cached are to be marked cacheable in class's mapping (.hbm.xml) file. Following is a sample mapping file **Customer.hbm.xml** with caching configuration:

```
<hibernate-mapping>
  <class name="hibernator.BLL.Customer" table="customers">
    <cache usage="read-write" region="AbsoluteExpirationRegion"/>
    <id column="CustomerID" name="CustomerID">
      <generator class="assigned"/>
    </id>
    <property name="Address" type="java.lang.String">
      <column name="Address"/>
    </property>
    <set cascade="all" lazy="true" name="Orders">
      <cache usage="read-write"/>
      <key column="CustomerID"/>
      <one-to-many class="hibernator.BLL.Orders"/>
    </set>
  </class>
```

```
...
</class>
```

```
</hibernate-mapping>
```

cache tag in the above configuration file makes the class cacheable:

```
<cache usage="read-write" region="AbsoluteExpirationRegion"/>
```

In above tag, properties **usage** and **region** can be changed.

Members	Description
Region	Specifies name of second level cache region to be used for this class's objects. If no region is specified, fully qualified class name will be used as region name with default region configurations.
Usage	Specifies caching concurrency strategy to be used for this class. Hibernate allows following three concurrency strategies to be used for caching: <ul style="list-style-type: none"> • read-write • nonstrict-read-write • read-only

See Hibernate documentation for further details about caching strategies.

6.2.3 How to Configure Cache Regions

Hibernate uses cache regions to store objects. TayzGrid allows cache regions to be configured with different properties. For this purpose TayzGrid has a configuration file named "TayzGridHibernate.xml", which contains all region's configurations and other related configurations used by TayzGrid. This file can be placed at the root directory of application or in config folder of TayzGrid Installation directory.

Following is a sample **TayzGridHibernate.xml** configuration file:

TayzGrid Hibernate.xml contains following configurable options:

```
<configuration>
<application-config application-id="myapp" enable-cache-exception="true" default-
region-name=" DefaultRegion " key-case-sensitivity="false">
  <cache-regions>
    <region name="AbsoluteExpirationRegion" cache-name="mycache"
priority="Default" expiration-type="absolute" expiration-period="10"/>
    <region name=" DefaultRegion " cache-name="mycache" priority="default"
expiration-type="none" expiration-period="0"/>
  </cache-regions>
  <database-dependencies>
    <dependency entity-name="hibernator.BLL.Customer" type="oledb" sql-
statement="select ContactName from dbo.Customers where CustomerID =?" cache-key-
format="dependency.customer:[pk]"/>
  </database-dependencies>
</application-config>
</configuration>
```

application-config: This section lets you specify configuration for the use of TayzGrid for a particular application.

Multiple application configurations can be specified in TayzGridHibernate.xml. Any Hibernate application will use one of the specified configuration based on the application-id specified in hibernate's configuration file.

Following are the configurable options in application-config section.

Members	Description
application-id	This option lets you specify the unique id for current application-config. This id is used to select appropriate configuration for a particular application.
enable-cache-exception	Identifies whether the exceptions if occurred in TayzGrid will be propagated to Hibernate provider.
default-region-name	Allows the user to specify a default region for the application. If a particular region's configurations are not found default region's configurations will be used. Specified default region's configuration must exist in cache-regions section.
key-case-sensitivity	This option allows the user to specify whether cache keys will be case sensitive or not. This option has to be configured according to the database used. If database is case-sensitive set this option true, otherwise false.
cache-regions	This section lets you configure multiple cache regions for Hibernate application. Each region's configuration is specified in region tag.
region	<p>This tag contains the configurations for one particular region. Following options can be configured for any particular region.</p> <p>name: Each region is identified by its name. Name of the region should be unique.</p> <p>cache-name: This option allows the user to specify TayzGrid's cache name to be used for this region.</p> <p>priority: The priority you want to use for items cached. The possible values are :</p> <ul style="list-style-type: none"> • Default • Low • BelowNormal • Normal • AboveNormal • High • NotRemovable <p>expiration-type: this option allows the user to specify the type of expiration for any item in this region. Possible values are absolute/sliding/none.</p> <p>expiration-period: This option allows the user to specify expiration period if absolute/sliding expiration is configured. Its value should be greater than 0.</p>
Database dependencies	<p>This section lets you specify database dependencies for the current Hibernate application.</p> <p>dependency: Each dependency is configured in a dependency tag. Multiple tags can be configured for multiple dependencies. Following options can be configured for each dependency.</p> <p>entity-name: Dependencies are added based on the fully qualified</p>

	<p>name of classes. This option allows the user to specify name of class for which current dependency is to be added. Any entity can have maximum one dependency.</p> <p>type: This option allows the user to specify database dependency type. Possible values are oracle/oledb. Since TayzGrid provider uses connection string specified in Hibernate's configuration, use dependency type appropriate to connection string.</p> <p>sql-statement: Allows the user to specify sql statement to be used for building TayzGrid's database dependency.</p> <p>cache-key-format: This option is used to configure cache key format for any item of current entity type. Cache key format should include "[pk]" in it, which will be replaced with the primary key of record. cache-key-format may also include "[en]" which will be replaced with entity name. Default value of cache key format is "HibernateTayzGrid:[en]#[pk]". Same cache key format should be used while writing database trigger in case of oledb database dependency. See TayzGrid help for writing database trigger for oledb dependency.</p> <p>composite-key-separator: If the records of current entity have composite key as primary key, cache-key is formed by combining all keys separated by composite-key-separator. composite-key-separator must be of one character length. Its default value is "\$".</p>
--	---

6.2.4 How to Use Query Caching

Query caching makes use of L2 cache to store queries, so that whenever the same query is needed to be executed again, it can be fetched from the cache. Please note that any object retrieved as a result of query are cached to their respective regions, therefore objects must be marked as cacheable for efficient use of query caching. However query along with the primary keys of result sets is stored in a default query region named as a fully qualified name of hibernate's StandardQueryCache class "*org.hibernate.cache.StandardQueryCache*".

Note: Since not every query's results remain the same for a period of time, therefore query caching should only be used with queries whose results are less likely to be changed frequently.

6.2.5 How to Enable Query Caching

To enable query caching, add property `hibernate.cache.use_query_cache` with value true in Hibernate configuration section's **session-factory** tag in **hibernate.cfg.xml** file:

```
<hibernate-configuration>
  <session-factory>
    ...
    <!-- Enable the second-level cache -->
    <property name="hibernate.cache.use_second_level_cache">true</property>
    <!-- Configure TayzGrid as second level cache -->
    <property name =
"hibernate.cache.region.factory_class">com.alachisoft.tayzgrid.integrations.hiberna
te.cache.TayzGridRegionFactory</property>
    <!-- Specify application id for TayzGrid -->
    <property name="tayzgrid.application_id">myapp</property>
    <!-- Enable the query cache -->
```

```

    <property name="hibernate.cache.use_query_cache">true</property>
    ...
</session-factory>
</hibernate-configuration>

```

Enabling query cache does not cache each query by default. Instead queries needed to be cached are to be set cacheable in code. To set a query cacheable, call **setCacheable(true)** function of query while creating query. e.g.:

```

List customerEnumerator = session.createQuery("from Customer
c").setCacheable(true).list();
Following is a sample code of using a cacheable query:
List customers = new ArrayList();
Transaction tx = null;
try
{
    if (!session.isConnected())
    {
        session = factory.openSession();
    }
    tx = session.beginTransaction();
    List customerEnumerator = session.createQuery("from Customer
c").setCacheable(true).list();
    for (Customer cust : (List<Customer>) customerEnumerator)
    {
        customers.add(cust);
    }
    tx.commit();
}
catch (Exception ex)
{
    tx.rollback();
    session.clear();
    session.disconnect();
    throw ex;
}

```

7. TayzGrid for Object Caching

You can cache application data by making TayzGrid API calls directly from your application. Here are the steps to do that.

Before you can do this, you need to first make sure you've installed TayzGrid and configured your cache. See [Install TayzGrid & Configure Cache](#) section for more details. After that, follow these steps.

For detailed understanding on all TayzGrid APIs, please read [Java API Reference](#) of TayzGrid.

7.1 Connect to the Cache

After successfully configuring TayzGrid, you can start developing the application by embedding TayzGrid API calls. TayzGrid client applications can communicate with the servers with an `InitializeCache` method. It establishes connection between client and server and returns a single cache handler per cache for a single application. However an application can contain multiple separate cache instances.

7.1.1 Using Basic Initialize Method

```
Cache cache = null;
try{
    cache = TayzGrid.initializeCache("mycache");
}
catch(Exception e){
    //handle exception
    // Possible Exceptions:
    //1. No server available to process the request
    //2. client.conf does not contain current cache information
}
```

7.1.2 Initializing Cache using CacheInitParams

`CacheInitParams` allows editing values of the properties at initialization time. In this example, the values of `RetryInterval` and `ConnectionRetries` properties can be changed; for this application these values will be used instead of the ones specified in client configuration files.

```
Cache cache = null;

try {
    // Create new InitParam instance
    CacheInitParams initParam = new CacheInitParams();
    initParam.setRetryInterval(3);
    initParam.setConnectionRetries(2);
    cache = TayzGrid.initializeCache("mycache", initParam);
} catch (Exception exp) {
    // handle exception
}
```


7.2 Adding Items

After successful initialization of cache and gaining valid cache handle, Add operation can be performed. Add is the basic operation provided by TayzGrid; data can be added/updated to the cache using multiple API calls.

7.2.1 Adding Objects to Cache

In order to add data in cache it must be ensured that the object implements the Java Serializable interface. In this example, a new object of Product class is created and added to cache.

```
Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";
product.UnitsInStock = 15;

String key = "Product:" + product.ProductID;

try {
    cache.add(key, product);
} catch (Exception ex) {
    // handle exception
    // usually thrown if key already exist in cache
    // however verify the failure reason
}
```

7.2.2 Adding Objects Using CacheItem

CacheItem is a custom class provided by TayzGrid which can be used to add data to the cache. This class encapsulates data as its value property. CacheItem also lets you set multiple metaInfo associated with an object in single operation.

```
Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";
product.UnitsInStock = 15;

CacheItem cacheItem = new CacheItem(product);
String key = "Product:" + product.ProductID;

try {
    cache.add(key, cacheItem);
} catch (Exception ex) {
    // handle exception
}
```

7.2.3 Adding Items to Cache in Bulk

A collection of items can be added in cache using AddBulk method. This method returns a dictionary of all the keys that failed to be added along with the exception.

```
Product product1 = new Product();
product1.ProductID = 1001;
```

```

product1.ProductName = "Chai";
product1.UnitsInStock = 15;
String key1 = "Product:" + product1.ProductID;

Product product2 = new Product();
product2.ProductID = 1002;
product2.ProductName = "Chang";
product2.UnitsInStock = 25;
String key2 = "Product:" + product2.ProductID;

String[] keys = {key1, key2};

CacheItem[] items = new CacheItem[2];
items[0] = new CacheItem(product1);
items[1] = new CacheItem(product2);

try {
    HashMap result = cache.addBulk(keys, items);
} catch (Exception ex) {
    // handle exception
}

```

7.2.4 Adding New Data with Absolute Expiration

In this example Add operation provided by TayzGrid is used to add Absolute Expiration to item. The expiration is set by using Calendar class. Also if Absolute Expiration is used NoSlidingExpiration is to be specified.

```

Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";
String key = "Product:" + product.ProductID;
Calendar calendar = Calendar.getInstance();
calendar.add(Calendar.MINUTE, 5);

try {
    // adding absolute expiration of 5 min through Add API.
    cache.add(key, product, calendar.getTime(),
        TimeSpan.ZERO, CacheItemPriority.Normal);
} catch (Exception ex) {
    // handle exception
}

```

7.3 Updating Items

Once you've initialized the cache, you can use the cache handle later in your application to Insert() items to the cache. An Insert() call adds the item if the item does not exist. Otherwise, it updates the existing item. Here is the sample code:

7.3.1 Updating Objects in Cache

Similarly, in order to update data previously added in cache, it must be ensured that the object is serialized. In this example, a new object is added with an existing key.

```

Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";
product.UnitsInStock = 5; // updated units

```

```
String key = "Product:" + product.ProductID;
try {
    //precondition: Cache is already initialized and item exists
    cache.insert(key, product);
} catch (Exception ex) {
    // handle exception
}
```

7.3.2 Updating Objects Using CacheItem

In this example, a key is updated that has already been existing in cache with object set as a property of CacheItem. CacheItem is a custom class provided by TayzGrid which can be used to add data to the cache. This class encapsulates data as its value property. CacheItem also lets you set multiple metaInfo associated with an object in single operation.

```
Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";
product.UnitsInStock = 5; // updated units
CacheItem cacheItem = new CacheItem(product);
String key = "Product:" + product.ProductID;
try {
    cache.insert(key, cacheItem);
} catch (Exception ex) {
    // handle exception
}
```

7.3.3 Updating Items in the Cache in Bulk

An existing collection of items can be updated with the help of InsertBulk method.

```
Product product1 = new Product();
product1.ProductID = 1001;
product1.ProductName = "Chai";
product1.UnitsInStock = 5; //updated units
String key1 = "Product:" + product1.ProductID;

Product product2 = new Product();
product2.ProductID = 1002;
product2.ProductName = "Chang";
product2.UnitsInStock = 6; //updated units

String key2 = "Product:" + product2.ProductID;
String[] keys = {key1, key2};

CacheItem[] items = new CacheItem[2];
items[0] = new CacheItem(product1);
items[1] = new CacheItem(product2);

try {
    cache.insertBulk(keys, items);
} catch (Exception ex) {
    // handle exception
}
```

7.3.4 Updating Data with Absolute Expiration

In this example, Insert provided by TayzGrid is used to add/update Absolute Expiration to item added to the key. The expiration is set by using Calendar class. Also, as Absolute Expiration is being used, DefaultSlidingExpiration is used for Sliding Expiration.

```
Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";
product.UnitsInStock = 5; //updated units
String key = "Product:" + product.ProductID;
Calendar calendar = Calendar.getInstance();
calendar.add(Calendar.MINUTE, 5);

try {
    // adding absolute expiration of 5 min through Add API.
    cache.insert(key, product, calendar.getTime(),
        Cache.DefaultSlidingExpiration,
        CacheItemPriority.Normal);
} catch (Exception ex) {
    // handle exception
}
```

Insert() operation does not return until all the cache servers are updated based on your caching topology.

7.4 Fetching Items

Primarily a cache is only considered as effective as its ability to retrieve data. TayzGrid utilizes the key-value architecture of its store to maximize the ways of data retrieval. Get method is the primary API provided to serve previously cached data; however basic indexer approach can also be used. Both of these methods are explained below.

7.4.1 Retrieving Data using Get Method

In this example, the basic Get method is used to retrieve item "Product:1001". This item has previously been added to the cache. Get method returns general object which needs to be casted accordingly. If a key does not exist in cache **null** value is returned.

```
String key = "Product:1001";
Product product = null;

try{
    //null is returned if key does not exist in the cache.
    Object result = cache.get(key);
    if ((result != null) && (result instanceof Product)){
        product = (Product)result;
    }
}
catch (Exception ex){
    // handle exception
}
```

7.4.2 Retrieve a CacheItem

In this example, the basic GetCacheItem method is used to retrieve item "Product:1001". This item has been added to the cache. This method returns CacheItem whose value property encapsulates the data; the Object value need to be cast accordingly.

```
String key = "Product:1001";
Product product = null;

try {
    CacheItem result = cache.GetCacheItem(key);
    if (result != null) {
        if (result.GetValue() instanceof Product) {
            product = (Product) result.GetValue();
        }
    }
} catch (Exception ex) {
    // handle exception
}
```

7.4.3 Checking If an Item Exists in Cache

In order to verify if a key exists in the cache or not, the Contains() method can be used. This method determines whether cache contains the specific key and returns true if key already exists in cache and false if not.

```
String key = "Product:1001";

try {
    if (cache.Contains(key)) {
        // do something
    } else {
        // do something
    }
} catch (Exception ex) {
    // handle exception
}
```

7.5 Lock/Unlock Items

A lockHandle is associated with item to be locked. This ensures that the particular item remains inaccessible throughout the cache. Two primary properties **lockId** and **lockDate** are used for locking while performing various cache operations.

TayzGrid provides method calls exclusively for locking as well as numerous overloads that manipulate the locking mechanism.

7.5.1 Locking an Item Explicitly

Lock on an item can be acquired explicitly before performing any operation. Lock method requires a TimeSpan to lock an item for a specified time. However, if you do not want the acquired lock to expire simply specify a new TimeSpan().

The lock method used in this example associates a lockHandle. Kindly ensure that the single LockHandle is associated with a single key. Release the lock before re-using the handle; otherwise it might lead to inconsistency of behavior.

```

    try {
        // Specifiying the time span of 10 sec for which the item
remains locked
        boolean locked = cache.lock(key, new TimeSpan(10000),
lockHandle);
    } catch (Exception ex) {
        // handle exception
    }

```

7.5.2 Locking an Item during Fetch Operation

You can lock an item while it is being retrieved. The item will be inaccessible for others unless it is released. In case of a mismatch of key, **null** value is returned.

In this example, a key and a lockHandle for the key should be provided to fetch the cached object and lock it. "True" should be given if the user wants to acquire lock.

```

LockHandle lockHandle = new LockHandle();

//Specify time span of 10 sec for which the item remains locked
TimeSpan lockSpan = new TimeSpan(0, 0, 10);
String key = "Product:1001";

try {
    Object result = cache.get(key, lockSpan, lockHandle, true);
    if (result != null) {
        if (result instanceof Product) {
            Product product = (Product) result;
        }
    }
} catch (Exception ex) {
    // handle exception
}

```

7.5.3 Lock Expiration

If time is specified as a value of TimeSpan(), TayzGrid would lock the item for that specified duration.

```

LockHandle lockHandle = new LockHandle();

//Specify time span of 10 sec for which the item remains locked
TimeSpan lockSpan = new TimeSpan(0, 0, 10);
String key = "Product:1001";

try {
    boolean lockAcquired = cache.lock(key, lockSpan, lockHandle);
    //Verify that the lock is released automatically after this
time period.
} catch (Exception ex) {
    // handle exception
}

```

7.5.4 Releasing Lock with Update Operation

While updating an item, the lock can be released allowing other cache clients to use the cached data. In order to successfully release the locked item, you will lockHandle that was

initially used to lock the item must be specified. In case of an invalid or different handle, TayzGrid would throw an `OperationFailedException`.

```
Product updatedProduct = new Product();
updatedProduct.ProductID = 1001;
updatedProduct.ProductName = "Chai";
updatedProduct.Category = 4;
String key = "Product" + updatedProduct.ProductID;
LockHandle lockHandle = new LockHandle();

try {
    // lock existing item for the time span of 30 seconds
    boolean locked = cache.lock(key, new TimeSpan(0, 30),
lockHandle);

    if (locked) {
        cache.insert(key, new CacheItem(updatedProduct),
lockHandle, true);
    }
} catch (Exception ex) {
    // handle exception
}
```

7.5.5 Releasing Lock Explicitly

In order to release lock forcefully on a previously locked cached item, `lockHandle` that was used to lock the key should be specified.

```
LockHandle lockHandle = new LockHandle();
String key = "Product:1001";

try {
    // lock an existing item and save the lockHandle for 10 seconds
    boolean locked = cache.lock(key, new TimeSpan(0, 0, 10),
lockHandle);

    if (locked) {
        // unlock locked item using saved LockHandle
        cache.unlock(key, lockHandle);
    }
} catch (Exception ex) {
    // handle exception
}
```

7.6 Locking (Optimistic locking)

7.6.1 Conceptual Overview

There are two types of locking available with TayzGrid. For Pessimistic locking kindly review "How to lock Cache Data" section.

Optimistic locking

- In optimistic locking, TayzGrid uses cache item versioning. **Cache Item Version** is a property associated with every cache item. It is basically a numeric value that is used to represent the version of the cached item which changes with every update to an item. This property allows you to track whether any change occurred in an item or

not. When you fetch an item from cache, you also fetch its current version in the cache

- You can specify the cache item version in data update call. This update call succeeds only if the current version of the cache item is the same as passed by the client, otherwise it fails. This is used when you want to update a cache item which you have previously read from cache. If someone else has updated the cache item, the update should fail. In this way each client will update only the latest copy of the data and hence maintain data integrity throughout the cache.

7.7 How to Utilize CacheItemVersion

You can also retrieve the latest version of item from cache through GetIfNewer API. Through this method you will provide the existing version and ask for the latest version if any. In this way you can track whether an item is updated in the cache or not.

7.7.1 Saving Item Version for the First Time

An add operation returns CacheItemVersion. If the item is added for the first time, '1' is returned. This version will be updated in future operations on this key.

In the following example, you will add a custom data in cache with key "Product:1001" and save its item version. Kindly ensure that the object is either serialized or registered with TayzGrid Compact serialization framework.

```
Product updatedProduct = new Product();
updatedProduct.SetProductID(1001);
updatedProduct.SetProductName("Chai");
updatedProduct.SetCategory(4);
String key = "Product" + updatedProduct.GetProductID();

try {
    CacheItemVersion itemVersion = cache.add(key, updatedProduct);
    //Save this item Version for future utilization
} catch (Exception ex) {
    // handle exception
}
```

7.7.2 Updating Item with Specific Version

Item versioning can be used to ensure consistency of data in an environment in which the cache is being accessed by multiple applications. The item version can be used to ensure that the data being updated is the same as in the cache and has not been updated prior to the update operation by any other cache client.

In the following example, an update is performed on a previously cached item. If the specified version is equal in the cache item, then it will be updated else an Exception will be thrown by TayzGrid.

```
//precondition: itemVersion is saved when item was added in cache
Product updatedProduct = new Product();
updatedProduct.SetProductID(1001);
updatedProduct.SetProductName("Chai");
updatedProduct.SetCategory(5); // updated category
```



```

CacheItem cacheItem = new CacheItem(updatedProduct);
String key = "Product" + updatedProduct.getProductID();

//saved itemVersion from add item call
cacheItem.setVersion(itemVersion);
try {
    CacheItemVersion newVersion = cache.insert(key, cacheItem);
    //save new version for future usage
    //verify the updated version and value in cache
} catch (Exception ex) {
    // handle exception
}

```

7.7.3 Retrieving Cached Item with Specific Version

You can retrieve a cached item by specifying the item version. In case of a mismatch of version, "Null" value is returned to you.

In the following example you need to specify key and previously saved item version of that key to fetch the cached object.

```

//precondition: itemVersion is saved when item was added or inserted
in cache
try {
    String key = "Product:1001";
    //saved itemVersion from add or insert item call
    Object result = cache.get(key, itemVersion);
    if (result != null) {
        if (result instanceof Product) {
            Product product = (Product) result;
        }
    }
} catch (Exception ex) {
    // handle exception
}

```

7.7.4 Retrieving CacheItem with Specific Version

GetCacheItem method is used to retrieve item "Product:1001". This item has previously been added to the cache. This method returns CacheItem whose value property encapsulates the data; you need to cast the Object value accordingly. In case a key does not exist in cache, an exception is thrown.

If the item version varies from the one specified, **Null** would be returned.

```

try {
    String key = "Product:1001";
    //saved itemVersion from add or insert item call
    CacheItem retrievedData = cache.getCacheItem(key, itemVersion);
    if (retrievedData != null) {
        if (retrievedData.getValue() instanceof Product) {
            Product product = (Product) retrievedData.getValue();
        }
    }
} catch (Exception ex) {
    // handle exception
}

```

7.7.5 Retrieve Item if a Newer Version Exists in Cache

getIfNewer() can be used to fetch the existing item if a newer version is available in cache. By specifying the current version as an argument of the method call, the cache returns appropriate result.

If the version specified is less than the one in cache, only then the method returns a new Item else **null** would be returned.

```
String key = "Product:1001";
try {
    //saved itemVersion from add or insert item call
    Object result = cache.getIfNewer(key, itemVersion);
    if (result != null) {
        if (result instanceof Product) {
            Product product = (Product) result;
        }
    }
} catch (Exception ex) {
    // handle exception
}
```

7.7.6 Removing an Item with Specified Item Version

The remove method is a basic method which removes the key from cache and returns the removed object to the cache. If a custom object is added to the cache, the remove method will return Object.

If the item version is different from the one in the cache or if the key does not exist, an OperationFailedException will be thrown by TayzGrid.

```
String key = "Product:1001";
try {
    //saved itemVersion from add or insert item call
    Object result = cache.remove(key, itemVersion);
    if (result != null) {
        if (result instanceof Product) {
            Product product = (Product) result;
        }
    }
} catch (Exception ex) {
    // handle exception
}
```

7.7.7 Delete an Item with Item Version

A delete method is a basic method which deletes the key from cache. If the item version is different from the one in the cache or if the key does not exist, an Exception will be thrown by TayzGrid.

```
String key = "Product:1001";
try {
    //saved itemVersion from add or insert item call
    cache.delete(key, itemVersion);
} catch (Exception ex) {
    // handle exception
}
```

7.8 How to Use Group Cache Data

7.8.1 Adding an Item to Data Group

The following example adds an item to cache with the data group identifier "Product" and sub group identifier "Beverages". In this example, you use addOperation to set group and sub-group of the item.

```
Product product = new Product();
product.setProductID(1001);
product.setProductName("Chai");
String key = "Product:" + product.getProductID();
try {
    cache.add(key, product, "Product", "Beverages");
} catch (Exception ex) {
    // handle exception
}
```

7.8.2 Adding Data Group with CacheItem

In the following example, data group is set by assigning it to a property of CacheItem.

```
Product product = new Product();
product.setProductID(1001);
product.setProductName("Chai");
String key = "Product:" + product.getProductID();
CacheItem cacheItem = new CacheItem(product);
cacheItem.setGroup("Product");
cacheItem.setSubGroup("Beverages");
try {
    cache.add(key, cacheItem);
} catch (Exception ex) {
    // handle exception
}
```

7.8.3 Updating an Item in a Data Group

If you add item without any group or want to add group to an already cached item then you have to first remove that item from the cache and then insert/Add that item with the desired group and sub group identifiers. Otherwise, the insert operation throws a data groups mismatch exception

7.8.4 Retrieving Keys Pertaining to a Particular Group

To return list of keys that belongs to a specified group and sub-group, you can use the getGroupKeys method. This method returns a list of keys.

```
try {
    ArrayList keys = (ArrayList) cache.getGroupKeys("Product", "Beverages");
    for (Object key : keys) {
        // do something
    }
} catch (Exception ex) {
    // handle exception
}
```

7.8.5 Retrieving Keys and Values Pertaining to a Particular Group

To return dictionary of keys and values included in specified group and sub-group, you can use the `getGroupData` method. This method returns a list of key and value pairs.

```
try {
    HashMap map = cache.getGroupData("Product", "Beverages");
    Set<String> keys = map.keySet();
    Iterator<String> keyIter = keys.iterator();
    while (keyIter.hasNext()) {
        String key = keyIter.next();
        String value = (String) map.get(key);
        // do something
    }
} catch (Exception ex) {
    // handle exception
}
```

7.8.6 Removing a Data Group

The data items pertaining to group "Product" and sub group "Beverages" is removed by this sample code, which has been added previously.

```
try {
    cache.removeGroupData("Product", "Beverages");
} catch (Exception ex) {
    // handle exception
}
```

7.9 Tags

7.9.1 Conceptual Overview

Using Tags, you can associate keywords(s) with your cache items. You can mark your data with tags which act as identifiers for your cache items. For instance, in an online library there are different genres of books. For each different category a tag identifier can be added, also tags for authors, year of publication etc. can also be added. Note here that a single author can have two books written under two different categories. Also multiple different authors may have a publication in the same year, thus cache search can be made for each tag filter separately i.e. for all publications in year 2000 or for all publications of john keats, etc. Also these filter tags can be used in combination with each other like all publications of john keats in year 2000.

- **Retrieve/Remove using Tags:** These identifying marks can help you in finding and/or removing items from the cache.
- **Support for Multiple Tags:** You can tag more than one keyword with any cache item.
- **Many-to-Many Grouping:** Tags provide a many-to-many grouping where one tag can contain multiple cached items and one cached item can belong to multiple tags. TayzGrid provides you with Tag class in which you can specify one or more tags associated with cached items.
- **Case Sensitive:** Tags are case sensitive.
- **Query using Tags:** TayzGrid also supports tags in queries.

For example if you tag your cache item with a list of tags i.e. "bookinformation" having tags like author, yearofpublication category, you can fetch items through API which match with ANY ONE tag (e.g. Book where author or yearofpublication or category is matched) from the list or you can fetch items where ALL tags (e.g. book where author and yearofpublication and category are matched) in list match. Furthermore you can also search on the basis of ONLY ONE tag (i.e. only author is a match). You can also remove items from cache on the basis of these marks of identification. You can use the same filters of tags for removal of items. You can use these different filters according to your business needs.

7.9.2 How to Tag Cached Data

7.9.2.1 Creating Tags

A Tag is a class provided by TayzGrid so that you can associate these tags with a particular data object.

```
// Create a Tag array and specify tags.
Tag[] productTags = new Tag[2];
productTags[0] = new Tag("Product");
productTags[1] = new Tag("Beverages");
```

7.9.2.2 Adding Items to Cache with Tags

In this example you need to use the [Add Operation overload](#) to associate previously created tags.

```
Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";
String key = "Product:" + product.ProductID;
try {
    cache.add(key, product, productTags);
} catch (Exception ex) {
    // handle exception
}
```

7.9.2.3 Tagging Data through CacheItem

In this example tags are set by assigning them to the [CacheItem](#).

```
CacheItem cacheItem = new CacheItem(product);
cacheItem.setTags(productTags);
try {
    cache.add(key, cacheItem);
} catch (Exception ex) {
    // handle exception
}
```

7.9.2.4 Retrieving Previously Tagged Data

Retrieving Keys Associated with a Tag

```
//For finding keys related to one tags
Collection keys = cache.getKeysByTag(productTags[0]);
//For Finding keys containing at least any one tag from tag list.
Collection keys = cache.getKeysByAnyTag(productTags);
//For finding keys containing all tags from tag list.
Collection keys = cache.getKeysByAllTags(productTags);
```

Retrieving Key from Cache Containing Tag

```
//For finding keys and values related to one tag.
HashMap result = cache.getByTag(productTags[0]);
//For finding keys and values containing at least any one tag in the tag
list
HashMap result = cache.getByAnyTag(productTags);
//For finding keys and values related to all tags from the tag list
HashMap result = cache.getByAllTags(productTags);
```

Querying Cached Items with respect to Tags

```
Tag[] productTags = new Tag[2];
productTags[0] = new Tag("Product");
productTags[1] = new Tag("Beverages");

HashMap queryValue = new HashMap();
//Add items to cache containing tags in ArrayList if multiple tags value is
required.
ArrayList queryTagList = new ArrayList();
queryTagList.add("Product");
queryTagList.add("Beverages");
// Add tag list to Hashtable for query searching values.
queryValue.put("$Tag$", queryTagList);
try {
    //Query string, for searching Cache items with respect to Tags.
    Collection keysResultSet = cache.search("SELECT Product WHERE
this.$Tag$=? AND this.$Tag$=?", queryValue);
    // keysResultSet contains all keys related to both tags.
} catch (Exception ex) {
    // handle exception
}
```

In the above code example, please specify the fully qualified name of your custom class instead of 'Product' in the query string.

7.9.2.5 Removing Items from Cache Containing Tag

```
//For removing keys containing specified tag
cache.removeByTag(productTags[0]);
//For removing keys containing at least one tag from tags list.
cache.removeByAnyTag(productTags);
//For removing keys containing all tags from list
cache.removeByAllTags(productTags);
```

7.10 Understanding Named Tags

In TayzGrid you can associate one or more keywords with cache items. These keywords will act as an identifying mark for cache items. You can retrieve that data later on the basis of specified keywords through "Tags". But if you require a higher level of tagging where your tags can have types or names and your requirement is to query data related to specific type of tag then you need "Named Tags".

The "Named Tags" feature set is the enhancement of the "Tags" feature set.

'NamedTagsDictionary' class will be used to define Named Tags which will be added with the cache items for further use. Only the string data type is allowed as a Named Tag "key" and primitive data types are allowed as valid Named Tag "values".

7.10.1 Creating Named Tags

A NameTagsDictionary is a class provided by TayzGrid so you can associate these tags with a particular data.

```
// Creating NameTagsDictionary and adding custom named tags.
NamedTagsDictionary productNamedTag = new NamedTagsDictionary();
productNamedTag.add("UnitsAvailable", 4);
productNamedTag.add("Category", 25);
productNamedTag.add("Supplier", "Alex");
```

7.10.2 Adding item to cache with Named Tags

In the following example you need to use the [Add Operation overload](#) to associate named tags previously created.

```
Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";
String key = "Product:" + product.ProductID;
try {
    cache.add(key, product, productNamedTag);
} catch (Exception ex) {
    // handle exception
}
```

7.10.3 Adding Named Tags through CachelItem

In the following example named tags are set by assigning them as a property of [CachelItem](#).

```
CacheItem cacheItem = new CacheItem(product);
cacheItem.setNamedTags(productNamedTag);
```

```

try {
    cache.add(key, cacheItem);
} catch (OperationFailedException ex) {
    // handle exception
}

```

7.10.4 Querying Cache Items with respect to Named Tags

```

String queryString = "SELECT Product WHERE this.UnitsAvailable = ? AND
this.Supplier = ?";

HashMap values = new HashMap();
values.put("UnitsAvailable", 4);
values.put("Supplier", "Alex");
try {
    //if only related cache keys are required than use Search
    Collection keysResultSet = cache.search(queryString, values);
    // if both keys and values are required than use SearchEntries
    HashMap resultSet = cache.searchEntries(queryString, values);

    //Iterate through list of all products having UnitsAvailable as 4 and Supplier name
    is "Alex"
    Iterator iterator = resultSet.entrySet().iterator();
    while (iterator.hasNext()) {
        Map.Entry entry = (Map.Entry) iterator.next();
        String key = entry.getKey().toString();
        if (entry.getValue() instanceof Product) {
            Product product = (Product) entry.getValue();
        }
    }
    //do something
}
} catch (Exception ex) {
    // handle exception
}

```

In the above code example please specify the fully qualified name of your custom class instead of 'Product' in the query string.

Note: If you have multiple applications that are sharing the same cache and all of them are supposed to add named tags, then make sure that same named tags have homogenous data types. E.g. if one client is adding named tag "ProductID" with String data type then all other clients should add values of "ProductID" in String format not in Integer or other for same cache.

7.11 Searching Items with SQL & LINQ

7.11.1 Adding and Updating Indexed Items

Before searching, items need to be indexed and added in cache. For adding indexed items, the basic APIs of add and insert as mentioned in TayzGrid Basic Operations should be used.

```

Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";

```



```
String key = "Product:" + product.ProductID;

try {
    cache.add(key, product);
} catch (Exception ex) {
    // handle exception
}
```

7.11.2 Cache Search Returning Keys

The following code searches the cache and returns a collection of keys. Then, it iterates over all the returned keys and individually fetches the corresponding cached items from the cache.

```
HashMap values = new HashMap();
values.put("ProductName", "Chai");
values.put("UnitsInStock", 250);

try {
    // Instead of Product, specify fully qualified name of your custom
class.

    String query = "SELECT Product where this.ProductName = ?";
    Collection result = cache.search(query, values);

    query = "SELECT Product WHERE this.UnitsInStock > ?";
    result = cache.search(query, values);

    query = "SELECT Product WHERE this.ProductName = ? and
this.UnitsInStock > ?";
    result = cache.search(query, values);

    query = "SELECT Product WHERE Not(this.ProductName = ? and
this.UnitsInStock > ?)";
    result = cache.search(query, values);
} catch (Exception ex) {
    // handle exception
}
```

The benefit of this approach is that the query returns keys only. Then, the client application can determine which key it wants to fetch. The drawback of this approach is increased number of trips to cache as most of the items are fetched from cache anyway. And, if the cache is distributed, this may eventually become costly.

7.11.3 Cache Search Returning Items

In some situations when the intention is to fetch all or most of the cached items associated with cache keys, it is better to fetch all the keys and items in one call. Below is an example of such a scenario. It searches the cache and returns a dictionary containing both keys and values. This way, all the data based on the search criteria is fetched in one call to TayzGrid. This is much more efficient way of fetching all the data from the cache. Example is mentioned below.

```
String queryString = "SELECT Product WHERE this.UnitsAvailable = ? AND
this.Supplier = ?";
HashMap values = new HashMap();
values.put("UnitsAvailable", 4);
```

```

values.put("Supplier", "Alex");
try { //if only related cache keys are required than use Search
    Collection keysResultSet = cache.search(queryString, values);
    // if both keys and values are required than use SearchEntries
    HashMap resultSet = cache.searchEntries(queryString, values);
    //Iterate through list of all products having UnitsAvailable as 4 and
Supplier name is "Alex"
    Iterator iterator = resultSet.entrySet().iterator();
    while (iterator.hasNext()) {
        Map.Entry entry = (Map.Entry) iterator.next();
        String key = entry.getKey().toString();

        if (entry.getValue() instanceof Product) {
            Product product = (Product) entry.getValue();
        } //do something
    }
} catch (Exception ex) {
    // handle exception
}

```

7.11.4 Querying Samples for Operators

Some code samples for different queries are given below:

- **Using Equal Operator**

```

String queryString = "SELECT Product where this.ProductID = ?";
HashMap values = new HashMap();
values.put("ProductID", 1001);

try {
    Collection keys = cache.search(queryString, values);
    HashMap items = cache.searchEntries(queryString, values);
} catch (Exception ex) {
    // handle exception
}

```

- **Using Multiple Operators**

```

String queryString = "SELECT Product where this.ProductID < ? AND
this.Category = ?";

HashMap values = new HashMap();
values.put("ProductID", 1001);

try {
    Collection keys = cache.search(queryString, values);
    HashMap items = cache.searchEntries(queryString, values);
} catch (Exception ex) {
    // handle exception
}

```

- **Using IN Operator**

```

String queryString = "SELECT Product where this.ProductID IN (?, ?, ?)";

```

```

ArrayList idList = new ArrayList();
idList.add(10);
idList.add(4);
idList.add(7);

HashMap values = new HashMap();
values.put("ProductID", idList);
try {
    Collection keys = cache.search(queryString, values);
    HashMap items = cache.searchEntries(queryString, values);
} catch (Exception ex) {
    // handle exception
}

```

• Using LIKE Operator

```

String queryString = "SELECT Product where this.ProductName LIKE ?";
ArrayList list = new ArrayList();
list.add("Ch*");

HashMap values = new HashMap();
values.put("ProductName", list);
try {
    Collection keys = cache.search(queryString, values);
    HashMap items = cache.searchEntries(queryString, values);
} catch (Exception ex) {
    // handle exception
}

```

7.12 Removing Items

Deleting item(s) from the cache is also considered as one of the basic operations. TayzGrid primarily provides two (remove, delete) methods to delete an item from cache.

7.12.1 Difference between Delete and Remove methods

delete	remove
Returns nothing after deletion.	Deletes data from the cache and returns deleted data to the application.

7.12.2 Using Remove Method

A remove method is a basic method which removes the key from the cache and returns the removed object to the cache.

```

String key = "Product:1001";
Product product = null;
try {
    // null is returned if key does not exist in cache
    Object result = cache.remove(key);
    if (result != null) {
        if (result instanceof Product) {
            product = (Product) result;
        }
    }
}

```

```

    }
} catch (Exception ex) {
    // handle exception
}

```

7.12.3 Using Delete Method

A delete method is a basic method which deletes the key from the cache.

```

String key = "Product:1001";
try {
    cache.delete(key);
} catch (Exception ex) {
    // handle exception
}

```

7.13 Item Based Event Notification

TayzGrid features event driven fetching of data whenever the data in cache is modified. This allows client applications to receive data without any explicit data fetch request to the cache server. Events work on push mechanism for whenever there is some activity of interest takes place in cache. It publishes events to its subscribed clients.

7.13.1 Types of Notifications

- **Add Notification**

Add Notification is triggered when a new item is added to the cache. All applications that have registered this event will receive a notification when data is added to the cache.

- **Update Notification**

Update Notification is triggered when an existing item is updated in the cache. All applications that have registered this event will receive a notification when data is updated in the cache.

- **Remove Notification**

Remove Notification is triggered when an item is removed from the cache. All applications that have registered this event will receive a notification when data is removed from the cache.

7.13.2 Registering Event with a Particular Item

TayzGrid provides a separate classification of event which can be registered with individual keys. To register update or remove event with a particular key(s) `addCacheDataModificationListener` method should be used.

In this example the cache will generate notification when the appropriate operation is performed on the key.

```

//Register Listener
try {
    CacheEventListener eventListener = new CacheEventListener(null, null,
null);

```

```

        cache.addCacheDataModificationListener("key", eventListener,
EnumSet.of(EventType.ItemAdded), EventDataFilter.Metadata);
    } catch (Exception ex) {
        // handle exception
    }

```

7.13.3 Registering Events with CacheItem

An event can also be registered with a particular key by registering as a property of `CacheItem`. The `addCacheDataNotificationListener` method in `CacheItem` can be used to provide all appropriate information for registering the notification.

```

try {
    CacheEventListener eventListener = new CacheEventListener(null, null,
null);

    cache.addCacheDataModificationListener("key", eventListener,
        EnumSet.of(EventType.ItemAdded), EventDataFilter.Metadata);
    Product product = new Product();
    product.ProductID = 1001;
    product.ProductName = "Laptop";
    product.UnitsInStock = 15;
    CacheItem cacheItem = new CacheItem(product);
    // Registering event with cacheItem
    cacheItem.addCacheDataNotificationListener(eventListener,
EnumSet.of(EventType.ItemAdded, EventType.ItemRemoved, EventType.ItemUpdated),
EventDataFilter.Metadata);
    String key = "Product:" + product.ProductID;
    // Registering
    cache.insert(key, cacheItem);
} catch (Exception ex) {
    // handle exception
}

```

7.13.4 Un-Registering Item Level Notification

You can also unregister a previously registered item level notification using the `removeCacheDataModificationListener` method. In this example, the appropriate key, callback as `CacheEventListener` and `EnumSet` of `EventType` for which the callback needs to be unregistered must be specified.

```

try {
    cache.removeCacheDataModificationListener("key", eventListener,
        EnumSet.of(EventType.ItemRemoved));
} catch (Exception ex) {
    // handle exceptio
}

```

7.14 Using Cache Level Events

By default they are disabled (except for Cache Cleared) for any cache configuration and must be enabled for events to be published.

The client application that is supposed to receive events must include the specific event handler method to respond to the event(s) raised. Whenever events are raised, the target event handler method executes the entire code written in the method body.

Note: Please note that the application will not be able to receive events unless it registers itself against cache using the specific event registration API call.

7.14.1 Registering Cache Level Events

Cache Level notifications are generic events which will be triggered upon every execution of the appropriate registered event type. To register these events for notification, you need to implement `CacheStatusEventListener`, and then pass its instance to `addCacheStatusEventListener` method with `EnumSet` of `CacheStatusNotificationType`.

You need to specify the appropriate `EventType` to be registered. Single as well as multiple event types can be specified in a single method call.

Every target method returns a key and a `ClusterEvent` object which contains information relating to the event that has been triggered.

```
EnumSet<CacheStatusNotificationType> _eventType =
EnumSet.noneOf(CacheStatusNotificationType.class);

CacheStatusEventListener listener = new CacheStatusEventListener() {
    @Override
    public void memberJoined(ClusterEvent ce) {
        // Do appropriate actions
    }

    @Override
    public void memberLeft(ClusterEvent ce) {
        // Do appropriate actions
    }

    @Override
    public void cacheStopped(ClusterEvent ce) {
        // Do appropriate actions
    }
};

Cache cache = null;
try{
    cache = TayzGrid.initializeCache("mycache");
    _eventType.add(CacheStatusNotificationType.ALL);
    cache.addCacheStatusEventListener(listener, _eventType);
}
catch(Exception e){
    //handle exception
    // Possible Exceptions:
    //1. No server available to process the request
    //2. client.conf does not contain current cache information
}
```

7.14.2 Un-registering Cache Level Events

You can also unregister a previously registered cache level notification using the following `removeCacheStatusEventListener` method. In this example you need to specify the `EnumSet` of `CacheStatusNotificationType` for which you want to unregister the event.

```
try{
    cache.removeCacheStatusEventListener(listener, _eventType);
}
catch(Exception e){
    //handle exception
}
```

7.15 Using Custom Notifications

Registering Custom Notification

You can create a custom event according to your requirement. You would need to register it with `addCustomEventListener`. TayzGrid does not raise the custom events on its own, in order to trigger it you would need to call `raiseCustomEvent` method.

```
import com.alachisoft.tayzgrid.event.CustomEvent;
import com.alachisoft.tayzgrid.event.CustomListener;

public class CustomEventListener implements CustomListener {

    @Override
    public void customEventOccured(CustomEvent ce) {
        //perform appropriate operations
    }
}

// Code in main class to register custom event
CustomEventListener customEventListener = new CustomEventListener();
Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Laptop";
product.UnitsInStock = 15;

try {
    //Registering custom event
    cache.addCustomEventListener(customEventListener);
    //perform appropriate tasks
    //Raise Custom event upon according to need
    cache.raiseCustomEvent("mycache", product);
} catch (Exception ex) {
    //handle exception
}
```

7.16 Disconnect from the Cache

At the end of your application when you no longer need to use the cache, you should call `dispose()` on it. That frees up the connection to the cache. If your cache was created `InProc`, this actually disposes your cache instance which in a `Replication Cache` means one of the cache nodes is leaving the cluster. Here is the code:

```
Cache cache = null;
try{
    cache = TayzGrid.initializeCache("myReplicatedCache");
    cache.dispose();
}
catch(Exception e){
    //handle exception
    // Possible Exceptions:
    //1. No server available to process the request
    //2. client.conf does not contain current cache information
}
```


8. Configuring as Memcached Wrapper

8.1 Memcached Protocol Server

TayzGrid can be configured to act as a Memcached Protocol Server so all Memcached applications in any language can use it without any code changes. TayzGrid Memcached Protocol Server implements both "text" and "binary" Memcached protocols.

Architecturally, TayzGrid Memcached Protocol Server is a service process that becomes a proxy to TayzGrid cache cluster and routes all Memcached client requests to the TayzGrid cluster and also sends back any responses accordingly. Due to this, you can configure the Memcached Protocol Server on a different set of servers than the TayzGrid cluster if you wish.

8.1.1 Configure TayzGrid Memcached Gateway Service

Open "memcachegateway.properties" file from [InstallDirectory]\ config" on cache client machine or cache server where you want to run your Gateway server.

It contains information on IP addresses and ports where it is listening to receive Memcached client requests and also target cache name, where it is going to route these requests.

You need to provide target cache name and also setup gateway IP and Port for receiving client requests. You will need to change application config to talk to this service later on.

```
#MemcacheGateway Service Properties
```

```
CacheName=mycache  
TextProtocolIP=20.200.20.235  
TextProtocolPort=11214  
BinaryProtocolIP=20.200.20.235  
BinaryProtocolPort=11215  
MaxCommandLength=1024
```

Here are details on configurations.

Members	Description
CacheName	Name of clustered cache on which all of your operations from Memcached end user applications are going to be performed via TayzGrid Memcached gateway service.
TextProtocolIP, TextProtocolPort	IP and port where gateway service will run and listen all client requests for applications using Text protocol of Memcached. Here you can specify either remote client address or cache server address depending upon client gateway and server gateway deployments respectively.

BinaryProtocolIP , BinaryProtocolPort	IP and port where gateway service will run and listen all client requests for applications using Binary protocol of Memcached.
MaxCommandLength	Specifies the maximum length of command in KBs for gateway server Text protocol.

8.1.2 Start Memcached Gateway Server

Start Memcached Wrapper for TayzGrid service from services. Memcached Wrapper for TayzGrid Service is not set to automatic by default. You must start the service manually at first and can change startup option to "Automatic" later on.

9. How to Use TayzGrid as Spring Data Cache

Note: TayzGrid supports cache integration with spring framework version 3.1 and later.

Caching can be configured with Spring application in two ways:

Using configurations only: This method uses spring's aspect oriented programming feature to add caching as an aspect in spring application. No code change is required for this method.

Annotation Driven: This method uses caching annotations defined in spring framework. These annotations are to be added before definitions of functions to be cached.

For more details on both methods, see Spring Framework documentation.

TayzGrid can be used with Spring for caching by following these two steps:

1. Configure Spring Application to use TayzGrid
2. Configure caches in TayzGrid configuration file.

Both steps are described in details in following sections:

9.1 How to Configure Spring Application to Use TayzGrid

In application context file of spring application, enable caching and specify TayzGrid as cache manager for spring.

Following is a sample context file for enabling cache:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:cache="http://www.springframework.org/schema/cache"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/aop
                           http://www.springframework.org/schema/aop/spring-aop.xsd
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/tx/spring-tx.xsd
                           http://www.springframework.org/schema/cache
                           http://www.springframework.org/schema/cache/spring-cache.xsd">
    <bean id="cacheManager"
        class="com.alachisoft.tayzgrid.integrations.spring.TayzGridCacheManager">
        <property name="tayzGridConfigurationManager" ref="tayzgridConfigManager"/>
        <property name="logFilePath" value="C:\CustomerDBSpringLogs"/>
    </bean>
    <bean id="tayzgridConfigManager"
        class="com.alachisoft.tayzgrid.integrations.spring.configuration.TayzGridConfigurationManager">
        <property name="configFile" value="C:\Program
Files\TayzGrid\samples\spring\tayzgrid-spring.xml" />
    </bean>

    <bean id="collectionKeyGenerator"
        class="cachekeygenerator.TayzGridCollectionKeyGenerator"/>
    <bean id="entityKeyGenerator" class="cachekeygenerator.TayzGridEntityKeyGenerator"/>
    <cache:advice id="collectionCacheAdvice" key-generator="collectionKeyGenerator">
        <cache:caching cache="CustomerCollectionCache">
```

```

<cache:cache-evict all-entries="true" method="create"/>
<cache:cache-evict all-entries="true" method="remove"/>
<cache:cache-evict all-entries="true" method="edit"/>
<cache:cacheable method="findAll"/>
<cache:cacheable method="findRange"/>
<cache:cacheable method="count"/>
</cache:caching>
</cache:advice>
<cache:advice id="entityCacheAdvice" key-generator="entityKeyGenerator">
  <cache:caching cache="CustomerEntityCache">
    <cache:cacheable method="find"/>
    <cache:cache-evict method="edit"/>
    <cache:cache-evict method="remove"/>
  </cache:caching>
</cache:advice>
<aop:config>
  <aop:advisor advice-ref="collectionCacheAdvice" pointcut="execution(*
customerdb.service.CustomerRESTFacade.*(..))"/>
  <aop:advisor advice-ref="entityCacheAdvice" pointcut="execution(*
customerdb.service.CustomerRESTFacade.*(..))"/>
</aop:config>
...
</beans>

```

Different sections to be configured in the above application context file for caching with TayzGrid are as under:

1. Configuration based usage of TayzGrid with Spring require AOP namespace to be defined in application context. Add `xmlns:aop="http://www.springframework.org/schema/aop"` in beans tag and specify following aop schema location in `xsi:schemaLocation`:
<http://www.springframework.org/schema/aop>
<http://www.springframework.org/schema/aop/spring-aop.xsd>
 This step is not required if annotation driven caching is used.
2. To enable use of caching in spring add cache namespace in application context. Add `xmlns:cache="http://www.springframework.org/schema/cache"` in beans tag and following cache schema location in `xsi:schemaLocation` before specifying caching configuration:
<http://www.springframework.org/schema/cache>
<http://www.springframework.org/schema/cache/spring-cache.xsd>
3. Add bean with id `cacheManager` and specify `tayzgrid` cache manager class. Following properties are needed to be specified for `cacheManager`:
 - **tayzGridConfigurationManager**: Reference to `tayzgrid` configuration manager bean
 - **logFilePath**: Fully qualified path for cache manager logs

Following is a sample bean definition:

```

<bean id="cacheManager"
class="com.alachisoft.tayzgrid.integrations.spring.TayzGridCacheManager">
  <property name="tayzGridConfigurationManager" ref="tayzgridConfigManager"/>
  <property name="logFilePath" value="C:\CustomerDBSpringLogs"/>
</bean>

```

4. Define `tayzgrid` configuration manager bean referenced in `cacheManager`. Specify `com.alachisoft.tayzgrid.integrations.spring.configuration.TayzGridConfigurationManager` as class. Specify fully qualified path of `tayzgrid` configuration file for spring **tayzgrid-spring.xml** as `configFile` property of configuration manager. Following is a sample configuration manager bean:

```
<bean id="tayzgridConfigManager"
class="com.alachisoft.tayzgrid.integrations.spring.configuration.TayzGridConfigurationManager">
<property name="configFile" value="C:\Program Files\TayzGrid\samples\spring\tayzgrid-spring.xml" />
</bean>
```

5. If you want to use custom key generator for cache key generation define beans for key generator. Following is sample bean for key generator:

```
<bean id="collectionKeyGenerator"
class="cachekeygenerator.TayzGridCollectionKeyGenerator"/>
```

Specify your own key generator class as `class`. See Spring Framework help for implementing custom key generators.

6. Add cache advice and specify caching properties and methods to be cached. Following is a sample cache advice tag:

```
<cache:advice id="collectionCacheAdvice" key-generator="collectionKeyGenerator">
<cache:caching cache="CustomerCollectionCache">
<cache:cache-evict all-entries="true" method="create"/>
<cache:cache-evict all-entries="true" method="remove"/>
<cache:cache-evict all-entries="true" method="edit"/>
<cache:cacheable method="findAll"/>
<cache:cacheable method="findRange"/>
<cache:cacheable method="count"/>
</cache:caching>
</cache:advice>
```

For further details on cache advice see Spring Framework documentation.

If annotation based caching is used, add following tag instead of `cache:advice` tag:

```
<cache:annotation-driven />
```

And add spring caching annotations on functions to be cached. For further details on caching annotations see Spring Framework documentation.

7. Finally to apply cache advice on appropriate points in spring application, specify above defined advice in AOP config tag. E.g.:

```
<aop:config>
<aop:advisor advice-ref="collectionCacheAdvice" pointcut="execution(*
customerdb.service.CustomerRESTFacade.*(..))"/>
</aop:config>
```

This step is not required for annotation driven caching.

For further details on app configuration and pointcuts, see Spring framework documentation.

After configuring Spring application to use TayzGrid, place TayzGrid's Spring provider jar `tg-spring.jar` located at `%TayzGridInstallDir%/lib/integration/` in application's class path.

Similarly add all jar files located at following directories to application's class path:

- `%TayzGridInstallDir%/lib`
- `%TayzGridInstallDir%/lib/resources`

9.2 How to Configure Caches

TayzGrid cache manager uses `tayzgrid-spring.xml`, provided in `cacheManager` bean to configure caches. Each cache has its own properties for expiration and eviction of items. Following is a sample **tayzgrid-spring.xml** file:

```
<application-config default-cache-name="default">
  <caches>
    <cache name="CustomerCollectionCache" tayzgrid-instance="mycache"
priority="normal" expiration-type="absolute" expiration-period="10"/>
    <cache name="CustomerEntityCache" tayzgrid-instance="mycache" priority="normal"
expiration-type="absolute" expiration-period="10"/>
    <cache name="default" tayzgrid-instance="mycache" priority="normal" expiration-
type="sliding" expiration-period="15" />
  </caches>
</application-config>
```

Following are configurable option in `tayzgrid-spring.xml`:

Members	Description
default-cache-name	This lets you specify a default cache for spring application. If required cache configuration does not exist in configuration file, default cache will be used. Default cache's configuration must be defined.
caches	Multiple caches can be defined under <code>caches</code> tag. Each cache is configured using a separate <code>cache</code> tag.
name	This lets you specify unique cache name for current cache configuration. This cache name is also used in application context file while specifying cache configurations for spring application.
tayzgrid-instance	Each spring cache is mapped on a specific TayzGrid cache instance. Multiple spring caches can be configured to use the same TayzGrid cache instance. TayzGrid <code>cacheManager</code> for spring will keep record of each spring's cache data in <code>tayzgrid</code> cache.
priority	Each item added in spring cache is added in TayzGrid cache with this specified priority. This priority is used by TayzGrid for expirations. The possible values are : <ul style="list-style-type: none"> • Default • Low • BelowNormal • Normal • AboveNormal • High • NotRemovable
expiration-type	This option allows specifying type of expiration for any item in this cache. Possible values are <code>absolute/sliding/none</code> .
expiration-period	Allows specifying expiration period. If <code>absolute/sliding</code> expiration is configured, its value should be greater than 0.