Monitoring {brandname} 10.0

Table of Contents

Chapter 1. Management

1.1. CLI

You can use the CLI to perform management operations on a standalone node or a domain controller.

```
$ bin/ispn-cli.sh
[disconnected /] connect
[standalone@localhost:9990 /] cd subsystem=datagrid-infinispan
[standalone@localhost:9990 subsystem=datagrid-infinispan] cd cache-container=local
[standalone@localhost:9990 cache-container=local] cd local-cache=default
[standalone@localhost:9990 local-cache=default]
```

The CLI is extremely powerful and supports a number of useful features to navigate the management resource tree as well as inspecting single resources or entire subtrees. It is also possible to batch multiple commands together so that they are applied as a single operation.

1.2. Console

You can use the web console to perform management operations on servers running in either standalone or domain mode. The console only supports a subset of the operations provided by the CLI, however you can perform the following actions:

- View/Edit Cache Container Configuration
- Execute Tasks across Containers
- View/Edit Cache Configurations
- Create/Destroy Cache Instances
- View Cluster/Server/Cache Statistics
- View event logs
- Start/Stop servers/clusters (domain mode only)

To access the console start your server(s) in the required mode, navigate to `http://localhost:9990` and enter your user credentials. If you would like to contribute to the development of the console, the source code can be found here.

Before you can use the web console, you must first setup at least one user account via the `./bin/add-user.sh` script. Detailed instructions of this process are presented in your browser if you attempt to access the console before creating any user accounts.

1.3. JMX

You can monitor an {brandname} Server over JMX in two ways:

- Use JConsole or VisualVM running locally as the same user. This will use a local jvmstat connection and requires no additional setup
- Use JMX remoting (aka JSR-160) to connect from any host. This requires connecting through the management port (usually 9990) using a special protocol which respects the server security configuration

To setup a client for JMX remoting you need to add the $ISPN_HOME/bin/client/jboss-client.jar to your client's classpath and use one of the following service URLs:

- `service:jmx:remote-http-jmx://host:port` for plain connections through the management interface
- `service:jmx:remote-https-jmx://host:port` for TLS connections through the management interface (although this requires having the appropriate keys available)
- `service:jmx:remoting-jmx://localhost:port` for connections through the remoting interface (necessary for connecting to individual servers in a domain)

The JMX subsystem registers a service with the Remoting endpoint so that remote access to JMX can be obtained over the exposed Remoting connector. This is switched on by default in standalone mode and accessible over port 9990 but in domain mode it is switched off so it needs to be enabled. In domain mode the port will be the port of the Remoting connector for the Server instance to be monitored.

```
<subsystem xmlns="urn:jboss:domain:jmx:1.3">
    <expose-resolved-model/>
    <expose-expression-model/>
    <remoting-connector use-management-endpoint="false"/>
</subsystem>
```

1.4. Prometheus

You can also expose JMX beans using Prometheus. In order to do this, just run the server with additional parameter `--jmx`, for example: `./standalone.xml -c cloud.xml --jmx`. Prometheus configuration is stored in `prometheus_config.yaml` file. It is possible to override this file by specifying it after `--jmx` parameter. For example: `./standalone.sh -c cloud.xml --jmx my-config.yaml`.

1.5. Access Logs

Hot Rod and REST endpoints can record all inbound client requests as log entries with the following categories:

- `org.infinispan.HOTROD_ACCESS_LOG` logging category for the Hot Rod endpoint.
- `org.infinispan.REST_ACCESS_LOG` logging category for the REST endpoint.

1.5.1. Enabling Access Logs

Access logs for Hot Rod and REST endpoints are disabled by default. To enable either logging category, set the level to `TRACE` in the server configuration file, as in the following example:

```
<logger category="org.infinispan.HOTROD_ACCESS_LOG" use-parent-handlers="false">
    <level name="TRACE"/>
    <handlers>
       <handler name="HR-ACCESS-FILE"/>
    </handlers>
</logger>
```

1.5.2. Access Log Properties

The default format for access logs is as follows:

```
%X{address} %X{user} [%d{dd/MMM/yyyy:HH:mm:ss z}] "%X{method} %m %X{protocol}" %X{status} %X{requestSize} %X{responseSize} %X{duration}%n
```

The preceding format creates log entries such as the following:

```
127.0.0.1 - [30/Oct/2018:12:41:50 CET] "PUT /rest/default/key HTTP/1.1" 404 5 77 10
```

Logging properties use the `%X{name}` notation and let you modify the format of access logs. The following are the default logging properties:

Property	Description
`address`	Either the `X-Forwarded-For` header or the client IP address.
`user`	Principal name, if using authentication.
`method`	Method used. `PUT`, `GET`, and so on.
`protocol`	Protocol used. `HTTP/1.1`, `HTTP/2`, `HOTROD/2.9`, and so on.
`status`	An HTTP status code for the REST endpoint. `OK` or an exception for the Hot Rod endpoint.
`requestSize`	Size, in bytes, of the request.
`responseSize`	Size, in bytes, of the response.
`duration`	Number of milliseconds that the server took to handle the request.

Use the header name prefixed with `h:` to log headers that were included in requests; for example, `%X{h:User-Agent}`.

Chapter 2. Management Tooling

Management of {brandname} instances is all about exposing as much relevant statistical information that allows administrators to get a view of the state of each {brandname} instance. Taking in account that a single installation could be made up of several tens or hundreds {brandname} instances, providing clear and concise information in an efficient manner is imperative. The following sections dive into the range of management tooling that {brandname} provides.

2.1. JMX

Over the years, JMX has become the de facto standard for management and administration of middleware and as a result, the {brandname} team has decided to standardize on this technology for the exposure of management and statistical information.

2.1.1. Understanding The Exposed MBeans

By connecting to the VM(s) where {brandname} is running with a standard JMX GUI such as JConsole or VisualVM you should find the following MBeans:

- For CacheManager level JMX statistics, without further configuration, you should see an MBean called *org.infinispan:type=CacheManager,name="DefaultCacheManager"* with properties specified by the CacheManager MBean .
- Using the cacheManagerName attribute in globalJmxStatistics XML element, or using the corresponding GlobalJmxStatisticsConfigurationBuilder.cacheManagerName(String cacheManagerName) call, you can name the cache manager in such way that the name is used as part of the JMX object name. So, if the name had been "Hibernate2LC", the JMX name for the cache manager would have been: *org.infinispan:type=CacheManager,name="Hibernate2LC"* . This offers a nice and clean way to manage environments where multiple cache managers are deployed, which follows JMX best practices .
- For Cache level JMX statistics, you should see several different MBeans depending on which configuration options have been enabled. For example, if you have configured a write behind cache store, you should see an MBean exposing properties belonging to the cache store component. All Cache level MBeans follow the same format though which is the following: `org.infinispan:type=Cache,name="${name-of-cache}(${cache-mode})",manager="${name-of-cache-manager}",component=${component-name}` where:
- ${name-of-cache} has been substituted by the actual cache name. If this cache represents the default cache, its name will be `___defaultCache`.
- ${cache-mode} has been substituted by the cache mode of the cache. The cache mode is represented by the lower case version of the possible enumeration values shown here.
- ${name-of-cache-manager} has been substituted by the name of the cache manager to which this cache belongs. The name is derived from the *cacheManagerName* attribute value in `globalJmxStatistics` element.
- ${component-name} has been substituted by one of the JMX component names in the JMX reference documentation .

For example, the cache store JMX component MBean for a default cache configured with synchronous distribution would have the following name: `org.infinispan:type=Cache,name="___defaultcache(dist_sync)",manager="DefaultCacheManager",component=CacheStore`

Please note that cache and cache manager names are quoted to protect against illegal characters being used in these user-defined names.

2.1.2. Enabling JMX Statistics

The MBeans mentioned in the previous section are always created and registered in the MBeanServer allowing you to manage your caches but some of their attributes do not expose meaningful values unless you take the extra step of enabling collection of statistics. Gathering and reporting statistics via JMX can be enabled at 2 different levels:

CacheManager level

The CacheManager is the entity that governs all the cache instances that have been created from it. Enabling CacheManager statistics collections differs depending on the configuration style:

- If configuring the CacheManager via XML, make sure you add the following XML under the `<cache-container />` element:

```
<cache-container statistics="true"/>
```

- If configuring the CacheManager programmatically, simply add the following code:

```
GlobalConfigurationBuilder globalConfigurationBuilder = ...
globalConfigurationBuilder.globalJmxStatistics().enable();
```

Cache level

At this level, you will receive management information generated by individual cache instances. Enabling Cache statistics collections differs depending on the configuration style:

- If configuring the Cache via XML, make sure you add the following XML under the one of the top level cache elements, such as `<local-cache />`:

```
<local-cache statistics="true"/>
```

- If configuring the Cache programmatically, simply add the following code:

```
ConfigurationBuilder configurationBuilder = ...
configurationBuilder.jmxStatistics().enable();
```

2.1.3. Monitoring cluster health

It is also possible to monitor {brandname} cluster health using JMX. On CacheManager there's an additional object called `CacheContainerHealth`. It contains the following attributes:

- cacheHealth - a list of caches and corresponding statuses (HEALTHY, UNHEALTHY or REBALANCING)
- clusterHealth - overall cluster health
- clusterName - cluster name
- freeMemoryKb - Free memory obtained from JVM runtime measured in KB
- numberOfCpus - The number of CPUs obtained from JVM runtime
- numberOfNodes - The number of nodes in the cluster
- totalMemoryKb - Total memory obtained from JVM runtime measured in KB

2.1.4. Multiple JMX Domains

There can be situations where several CacheManager instances are created in a single VM, or Cache names belonging to different CacheManagers under the same VM clash.

Using different JMX domains for multi cache manager environments should be last resort. Instead, it's possible to name a cache manager in such way that it can easily be identified and used by monitoring tools. For example:

- Via XML:

```
<cache-container statistics="true" name="Hibernate2LC"/>
```

- Programmatically:

```
GlobalConfigurationBuilder globalConfigurationBuilder = ...
globalConfigurationBuilder.globalJmxStatistics()
    .enable()
    .cacheManagerName("Hibernate2LC");
```

Using either of these options should result on the CacheManager MBean name being: `org.infinispan:type=CacheManager,name="Hibernate2LC"`

For the time being, you can still set your own jmxDomain if you need to and we also allow duplicate domains, or rather duplicate JMX names, but these should be limited to very special cases where different cache managers within the same JVM are named equally.

2.1.5. Registering MBeans In Non-Default MBean Servers

Let's discuss where {brandname} registers all these MBeans. By default, {brandname} registers them in the standard JVM MBeanServer platform . However, users might want to register these

MBeans in a different MBeanServer instance. For example, an application server might work with a different MBeanServer instance to the default platform one. In such cases, users should implement the MBeanServerLookup interface provided by {brandname} so that the getMBeanServer() method returns the MBeanServer under which {brandname} should register the management MBeans. Once you have your implementation ready, simply configure {brandname} with the fully qualified name of this class. For example:

- Via XML:

```
<cache-container statistics="true">
   <jmx mbean-server-lookup="com.acme.MyMBeanServerLookup" />
</cache-container>
```

- Programmatically:

```
GlobalConfigurationBuilder globalConfigurationBuilder = ...
globalConfigurationBuilder.globalJmxStatistics()
    .enable()
    .mBeanServerLookup(new com.acme.MyMBeanServerLookup());
```

2.1.6. Available MBeans

For a complete list of available MBeans, refer to the JMX reference documentation

2.2. Command-Line Interface (CLI)

{brandname} offers a simple Command-Line Interface (CLI) with which it is possible to interact with the data within the caches and with most of the internal components (e.g. transactions, cross-site backups, rolling upgrades).

The CLI is built out of two elements: a server-side module and the client command tool. The server-side module (`infinispan-cli-server-$VERSION.jar`) provides the actual interpreter for the commands and needs to be included alongside your application. {brandname} Server includes CLI support out of the box.

Currently the server (and the client) use the JMX protocol to communicate, but in a future release we plan to support other communication protocols (in particular our own Hot Rod).

The CLI offers both an interactive and a batch mode. To invoke the client, run the *bin/ispn-cli.[sh|bat]* script.

The following is a list of command-line switches which affect how the CLI can be started:

```
-c, --connect=URL       connects to a running instance of Infinispan.
                        JMX over RMI
jmx://[username[:password]]@host:port[/container[/cache]]
                        JMX over JBoss remoting
remoting://[username[:password]]@host:port[/container[/cache]]
-f, --file=FILE         reads input from the specified file instead of using

                        interactive mode. If FILE is '-', then commands will be read
                        from stdin
-h, --help              shows this help page
-v, --version           shows version information
```

- JMX over RMI is the traditional way in which JMX clients connect to MBeanServers. Please refer to the JDK Monitoring and Management documentation for details on how to configure the process to be monitored
- JMX over JBoss Remoting is the protocol of choice when your {brandname} application is running within WildFly or EAP.

The connection to the application can also be initiated from within the CLI using the connect command.

```
[disconnected//]> connect jmx://localhost:12000
[jmx://localhost:12000/MyCacheManager/>
```

The CLI prompt will show the active connection information, including the currently selected CacheManager. Initially no cache is selected so, before performing any cache operations, one must be selected. For this the *cache* command is used. The CLI supports tab-completion for all commands and options and for most parameters where it makes sense to do so. Therefore typing *cache* and pressing TAB will show a list of active caches:

```
[jmx://localhost:12000/MyCacheManager/> cache
___defaultcache  namedCache
[jmx://localhost:12000/MyCacheManager/]> cache ___defaultcache
[jmx://localhost:12000/MyCacheManager/___defaultcache]>
```

Pressing TAB at an empty prompt will show the list of all available commands:

```
alias       cache       container   encoding    get         locate      remove
site        upgrade
abort       clearcache  create      end         help        put         replace
start       version
begin       commit      disconnect  evict       info        quit        rollback
stats
```

The CLI is based on Æsh and therefore offers many keyboard shortcuts to navigate and search the

history of commands, to manipulate the cursor at the prompt, including both Emacs and VI modes of operation.

2.2.1. Commands

abort

The *abort* command is used to abort a running batch initiated by the *start* command

```
[jmx://localhost:12000/MyCacheManager/namedCache]> start
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> abort
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
null
```

begin

The *begin* command starts a transaction. In order for this command to work, the cache(s) on which the subsequent operations are invoked must have transactions enabled.

```
[jmx://localhost:12000/MyCacheManager/namedCache]> begin
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b b
[jmx://localhost:12000/MyCacheManager/namedCache]> commit
```

cache

The *cache* command selects the cache to use as default for all subsequent operations. If it is invoked without parameters it shows the currently selected cache.

```
[jmx://localhost:12000/MyCacheManager/namedCache]> cache ___defaultcache
[jmx://localhost:12000/MyCacheManager/___defaultcache]> cache
___defaultcache
[jmx://localhost:12000/MyCacheManager/___defaultcache]>
```

clearcache

The *clearcache* command clears a cache from all content.

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> clearcache
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
null
```

commit

The *commit* command commits an ongoing transaction

```
[jmx://localhost:12000/MyCacheManager/namedCache]> begin
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b b
[jmx://localhost:12000/MyCacheManager/namedCache]> commit
```

container

The *container* command selects the default container (cache manager). Invoked without parameters it lists all available containers

```
[jmx://localhost:12000/MyCacheManager/namedCache]> container
MyCacheManager OtherCacheManager
[jmx://localhost:12000/MyCacheManager/namedCache]> container OtherCacheManager
[jmx://localhost:12000/OtherCacheManager/]>
```

create

The *create* command creates a new cache based on the configuration of an existing cache definition

```
[jmx://localhost:12000/MyCacheManager/namedCache]> create newCache like namedCache
[jmx://localhost:12000/MyCacheManager/namedCache]> cache newCache
[jmx://localhost:12000/MyCacheManager/newCache]>
```

deny

When authorization is enabled and the role mapper has been configured to be the ClusterRoleMapper, principal to role mappings are stored within the cluster registry (a replicated cache available to all nodes). The *deny* command can be used to deny roles previously assigned to a principal:

```
[remoting://localhost:9999]> deny supervisor to user1
```

disconnect

The *disconnect* command disconnects the currently active connection allowing the CLI to connect to another instance.

```
[remoting://localhost:9999]> deny supervisor to user1
```

encoding

The *encoding* command is used to set a default codec to use when reading/writing entries from/to a cache. When invoked without arguments it shows the currently selected codec. This command is useful since currently remote protocols such as HotRod and Memcached wrap keys and values in specialized structures.

```
[jmx://localhost:12000/MyCacheManager/namedCache]> encoding
none
[jmx://localhost:12000/MyCacheManager/namedCache]> encoding --list
memcached
hotrod
none
rest
[jmx://localhost:12000/MyCacheManager/namedCache]> encoding hotrod
```

end

The *end* command is used to successfully end a running batch initiated by the *start* command

```
[jmx://localhost:12000/MyCacheManager/namedCache]> start
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> end
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
a
```

evict

The *evict* command is used to evict from the cache the entry associated with a specific key.

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> evict a
```

get

The *get* command is used to show the value associated to a specified key. For primitive types and Strings, the *get* command will simply print the default representation. For other objects, a JSON representation of the object will be printed.

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
a
```

grant

When authorization is enabled and the role mapper has been configured to be the ClusterRoleMapper, principal to role mappings are stored within the cluster registry (a replicated

cache available to all nodes). The *grant* command can be used to grant new roles to a principal:

```
[remoting://localhost:9999]> grant supervisor to user1
```

info

The *info* command is used to show the configuration of the currently selected cache or container.

```
[jmx://localhost:12000/MyCacheManager/namedCache]> info
GlobalConfiguration{asyncListenerExecutor=ExecutorFactoryConfiguration{factory=org.inf
inispan.executors.DefaultExecutorFactory@98add58},
asyncTransportExecutor=ExecutorFactoryConfiguration{factory=org.infinispan.executors.D
efaultExecutorFactory@7bc9c14c},
evictionScheduledExecutor=ScheduledExecutorFactoryConfiguration{factory=org.infinispan
.executors.DefaultScheduledExecutorFactory@7ab1a411},
replicationQueueScheduledExecutor=ScheduledExecutorFactoryConfiguration{factory=org.in
finispan.executors.DefaultScheduledExecutorFactory@248a9705},
globalJmxStatistics=GlobalJmxStatisticsConfiguration{allowDuplicateDomains=true,
enabled=true, jmxDomain='jboss.infinispan',
mBeanServerLookup=org.jboss.as.clustering.infinispan.MBeanServerProvider@6c0dc01,
cacheManagerName='local', properties={}},
transport=TransportConfiguration{clusterName='ISPN', machineId='null', rackId='null',
siteId='null', strictPeerToPeer=false, distributedSyncTimeout=240000, transport=null,
nodeName='null', properties={}},
serialization=SerializationConfiguration{advancedExternalizers={1100=org.infinispan.se
rver.core.CacheValue$Externalizer@5fabc91d,
1101=org.infinispan.server.memcached.MemcachedValue$Externalizer@720bffd,
1104=org.infinispan.server.hotrod.ServerAddress$Externalizer@771c7eb2},
marshaller=org.infinispan.marshall.VersionAwareMarshaller@6fc21535, version=52,
classResolver=org.jboss.marshalling.ModularClassResolver@2efe83e5},
shutdown=ShutdownConfiguration{hookBehavior=DONT_REGISTER}, modules={},
site=SiteConfiguration{localSite='null'}}
```

locate

The *locate* command shows the physical location of a specified entry in a distributed cluster.

```
[jmx://localhost:12000/MyCacheManager/namedCache]> locate a
[host/node1,host/node2]
```

put

The *put* command inserts an entry in the cache. If the cache previously contained a mapping for the key, the old value is replaced by the specified value. The user can control the type of data that the CLI will use to store the key and value.

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b 100
[jmx://localhost:12000/MyCacheManager/namedCache]> put c 4139l
[jmx://localhost:12000/MyCacheManager/namedCache]> put d true
[jmx://localhost:12000/MyCacheManager/namedCache]> put e { "package.MyClass": {"i": 5,
"x": null, "b": true } }
```

The put command can optionally specify a lifespan and a maximum idle time.

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a expires 10s
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a expires 10m maxidle 1m
```

replace

The *replace* command replaces an existing entry in the cache. If an old value is specified, then the replacement happens only if the value in the cache coincides.

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> replace a b
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
b
[jmx://localhost:12000/MyCacheManager/namedCache]> replace a b c
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
c
[jmx://localhost:12000/MyCacheManager/namedCache]> replace a b d
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
c
```

roles

When authorization is enabled and the role mapper has been configured to be the ClusterRoleMapper, principal to role mappings are stored within the cluster registry (a replicated cache available to all nodes). The *roles* command can be used to list the roles associated to a specific user, or to all users if one is not given:

```
[remoting://localhost:9999]> roles user1
[supervisor, reader]
```

rollback

The *rollback* command rolls back an ongoing transaction

```
[jmx://localhost:12000/MyCacheManager/namedCache]> begin
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b b
[jmx://localhost:12000/MyCacheManager/namedCache]> rollback
```

site

The *site* command performs operations related to the administration of cross-site replication. It can be used to obtain information related to the status of a site and to change the status (online/offline)

```
[jmx://localhost:12000/MyCacheManager/namedCache]> site --status NYC
online
[jmx://localhost:12000/MyCacheManager/namedCache]> site --offline NYC
ok
[jmx://localhost:12000/MyCacheManager/namedCache]> site --status NYC
offline
[jmx://localhost:12000/MyCacheManager/namedCache]> site --online NYC
```

start

The *start* command initiates a batch of operations.

```
[jmx://localhost:12000/MyCacheManager/namedCache]> start
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b b
[jmx://localhost:12000/MyCacheManager/namedCache]> end
```

stats

The *stats* command displays statistics about a cache

```
[jmx://localhost:12000/MyCacheManager/namedCache]> stats
Statistics: {
  averageWriteTime: 143
  evictions: 10
  misses: 5
  hitRatio: 1.0
  readWriteRatio: 10.0
  removeMisses: 0
  timeSinceReset: 2123
  statisticsEnabled: true
  stores: 100
  elapsedTime: 93
  averageReadTime: 14
  removeHits: 0
  numberOfEntries: 100
  hits: 1000
}
LockManager: {
  concurrencyLevel: 1000
  numberOfLocksAvailable: 0
  numberOfLocksHeld: 0
}
```

upgrade

The *upgrade* command performs operations used during the rolling upgrade procedure.

```
[jmx://localhost:12000/MyCacheManager/namedCache]> upgrade --synchronize=hotrod --all
[jmx://localhost:12000/MyCacheManager/namedCache]> upgrade --disconnectsource=hotrod
--all
```

version

The *version* command displays version information about both the CLI client and the server

```
[jmx://localhost:12000/MyCacheManager/namedCache]> version
Client Version x.x.x.Final
Server Version x.x.x.Final
```

Data Types

The CLI understands the following types:

- string strings can either be quoted between single (') or double (") quotes, or left unquoted. In this case it must not contain spaces, punctuation and cannot begin with a number e.g. 'a string', key001
- int an integer is identified by a sequence of decimal digits, e.g. 256

- long a long is identified by a sequence of decimal digits suffixed by 'l', e.g. 1000l
- double
  - a double precision number is identified by a floating point number(with optional exponent part) and an optional 'd' suffix, e.g.3.14
- float
  - a single precision number is identified by a floating point number(with optional exponent part) and an 'f' suffix, e.g. 10.3f
- boolean a boolean is represented either by the keywords true and false
- UUID a UUID is represented by its canonical form XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
- JSON serialized Java classes can be represented using JSON notation, e.g. {"package.MyClass":{"i":5,"x":null,"b":true}}. Please note that the specified class must be available to the CacheManager's class loader.

Time Values

A time value is an integer number followed by time unit suffix: days (d), hours (h), minutes (m), seconds (s), milliseconds (ms).

2.3. Hawt.io

Hawt.io, a slick, fast, HTML5-based open source management console, also has support for {brandname}. Refer to Hawt.io's documentation for information regarding this plugin.

2.4. Writing plugins for other management tools

Any management tool that supports JMX already has basic support for {brandname}. However, custom plugins could be written to adapt the JMX information for easier consumption.