Hazelcast Code Reference Card

Before Getting Started

Be sure that `Hazelcast-[version].zip` is downloaded from **www.hazelcast.org**, unzipped, `lib/hazelcast-[version].jar` is added to the classpath, and Hazelcast libraries are imported.

Or, if you are using Maven, add the below dependency to your pom.xml:

```
<dependency>
<groupId>com.hazelcast</groupId>
<artifactId>hazelcast</artifactId>
<version><version_number></version>
</dependency>
```

Please check the Hazelcast website to learn what the most recent version is.

Start Your First Nodes

To start a single node and create a map and queue named customers, run the below sample code.

```
public class GettingStarted {
    public static void main(String[] args) {
        Config cfg = new Config();
        HazelcastInstance instance = Hazelcast.newHazelcastInstance(cfg);
        Map<Integer, String> mapCustomers = instance.getMap("customers");
        System.out.println("Initial Map Size:" + mapCustomers.size());
        mapCustomers.put(1, "Joe");
        mapCustomers.put(2, "Ali");
        mapCustomers.put(3, "Avi");
        System.out.println("Customer with key 1: "+ mapCustomers.get(1));
        System.out.println("Map Size:" + mapCustomers.size());
        Queue<String> queueCustomers = instance.getQueue("customers");
        System.out.println("Initial Queue size: " + queueCustomers.size());
        queueCustomers.offer("Tom");
        queueCustomers.offer("Mary");
        queueCustomers.offer("Jane");
        System.out.println("First customer: " + queueCustomers.poll());
        System.out.println("Second customer: "+ queueCustomers.peek());
        System.out.println("Queue size: " + queueCustomers.size());
    }
}
```

If you run this code once more, a second node starts and these two nodes will form a cluster so that the map and queues will be clustered.

Moreover, below code starts a Hazelcast client, connects this client to the 2-node cluster created above and prints the size of `customers` map, after of course `hazelcast-client-[version].jar` is added to the classpath.

```
public class GettingStartedClient {
    public static void main(String[] args) {
        HazelcastInstance client = HazelcastClient.newHazelcastClient();
        IMap map = client.getMap("customers");
        System.out.println("Map Size:" + map.size());
    }
}
```

Hazelcast at a Glance

Thread Safe

Hazelcast codes manipulate shared data structures in a way that guarantees safe execution by multiple threads at the same time.

```
HazelcastInstance instance = Hazelcast.newHazelcastInstance();
Map<String, Employee> = instance.getMap("employees");
List<Users> = instance.getList("users");
```

Multiple Instances on the Same JVM

More than one instance (node) can be created on the same Java Virtual Machine.

```
Config config = new Config();
HazelcastInstance h1 = Hazelcast.newHazelcastInstance(config)
HazelcastInstance h2 = Hazelcast.newHazelcastInstance(config)
```

Serializable Objects

All objects stored in Hazelcast needs to be serialized. The easiest way is to implement java.io.Serializable interface. For a better performance, Hazelcast specific interfaces can be used, such as DataSerializable.

```
public interface DataSerializable extends Serializable {
    void writeData(DataOutput out) throws IOException;
    void readData(DataInput in) throws IOException;
}
```

If none of these meets the need, Hazelcast supports pluggable serialization. You can make Hazelcast to use any Serialization framework you wish.

Cluster Interface

Hazelcast allows registering for membership events to get notified when members are added or removed. Cluster members can also be retrieved.

```
Cluster cluster = hazelcastInstance.getCluster();
cluster.addMembershipListener(new MembershipListener() {
    @Override
    public void memberAdded(MembershipEvent membershipEvent) {
        System.out.println("New member added: " + membershipEvent.getMember());
    }
    @Override
    public void memberRemoved(MembershipEvent membershipEvent) {
        System.out.println("Member left the cluster: " + membershipEvent.getMember());
    }
});
Member localMember = cluster.getLocalMember();
System.out.println (localMember.getInetAddress());
Set<Member> setMembers = cluster.getMembers();
```

"Distributed World" of Hazelcast

Here are sample usages of distributed data structures.

Distributed Map

```
Config config = new Config();
HazelcastInstance h = Hazelcast.newHazelcastInstance(config);
ConcurrentMap<String, String> map = h.getMap("my-distributed-map");
map.put("key", "value");
map.get("key");
//Concurrent Map methods
map.putIfAbsent("somekey", "somevalue");
map.replace("key", "value", "newvalue");
```

Distributed Queue

```
Config config = new Config();
HazelcastInstance h = Hazelcast.newHazelcastInstance(config);
BlockingQueue<String> queue = h.getQueue("my-distributed-queue");
queue.offer("item");
String item = queue.poll();
//Timed blocking Operations
queue.offer("anotheritem", 500, TimeUnit.MILLISECONDS);
String anotherItem = queue.poll(5, TimeUnit.SECONDS);
//Indefinitely blocking Operations
queue.put("yetanotheritem");
String yetanother = queue.take();
```

Distributed Lock

```
Config config = new Config();
HazelcastInstance h = Hazelcast.newHazelcastInstance(config);
Lock lock = h.getLock("my-distributed-lock");
lock.lock();
try {
    //do something here
} finally {
    lock.unlock();
}
//Lock on Map
IMap<String, String> map = h.getMap("customers");
map.lock("1");
try {
    // do something here
} finally {
    map.unlock("1");
}
```

"Distributed World" of Hazelcast (Continued)

Distributed Topic

```
public class DistributedTopic implements MessageListener<String> {
    public static void main(String[] args) {
        Config config = new Config();
        HazelcastInstance h = Hazelcast.newHazelcastInstance(config);
        ITopic<String> topic = h.getTopic("my-distributed-topic");
        topic.addMessageListener(new DistributedTopic());
        topic.publish("Hello to distributed world");
    }
    @Override
    public void onMessage(Message<String> message) {
        System.out.println("Got message " + message.getMessageObject());
    }
}
```

Distributed Events

```
public class DistributedEventListener implements EntryListener<String, String> {
    public static void main(String[] args) {
        HazelcastInstance instance = Hazelcast.newHazelcastInstance();
        DistributedEventListener listener = new DistributedEventListener();
        IMap<String, String> map = instance.getMap ("default");
        map.addEntryListener (listener, true);
        map.addEntryListener (listener, "key", true);
        map.put("key", "value");
    }
    @Override
    public void entryAdded(EntryEvent<String, String> event) {
        System.out.println("Added " + event.getKey() + ":" + event.getValue());
    }
    @Override
    public void entryRemoved(EntryEvent<String, String> event) {
        System.out.println("Removed " + event.getKey() + ":" + event.getValue());
    }
    @Override
    public void entryUpdated(EntryEvent<String, String> event) {
        System.out.println("Updated " + event.getKey() + ":" + event.getValue());
    }
    @Override
    public void entryEvicted(EntryEvent<String, String> event) {
        System.out.println("Evicted " + event.getKey() + ":" + event.getValue());
    }
}
```

Executor Service

Runnable / Callable's can be created and executed on a specific node, on any available node, on a collection of defined nodes, and on a node owning a key. Hazelcast also supports adding Callback and getting notified when execution is finished.

A Simple Callable

```
public class EchoMessage implements Callable<String>, Serializable, HazelcastInstanceAware {
    final String message;
    private HazelcastInstance hazelcast;
    EchoMessage(String message) {
        this.message = message;
    }
    @Override
    public String call() {
        Cluster cluster = hazelcast.getCluster();
        return "Echo " + message + " from " + cluster.getLocalMember();
    }
    @Override
    public void setHazelcastInstance(HazelcastInstance hazelcastInstance) {
        this.hazelcast = hazelcastInstance;
    }
}
```

Execution

```
Config config = new Config();
HazelcastInstance h = Hazelcast.newHazelcastInstance(config);
IExecutorService ex = h.getExecutorService("my-distributed-executor");
// Execute on any member.
ex.submit(new EchoMessage("MESSAGE_TO_ANY_MEMBER"));
// Execute on a specific member.
// In this case the first member in cluster
Member firstMember = h.getCluster().getMembers().iterator().next();
ex.submitToMember(new EchoMessage("MESSAGE_TO_SPECIFIC_MEMBER"), firstMember);
// Execute on all members
ex.submitToAllMembers(new EchoMessage("MESSAGE_TO_ALL_MEMBERS"));
// Execute on any member.
ex.submitToKeyOwner(new EchoMessage("MESSAGE_TO_KEY_OWNER"), "key");

Config config = new Config();
HazelcastInstance h = Hazelcast.newHazelcastInstance(config);
IExecutorService ex = h.getExecutorService("my-distributed-executor");
// Execute on any member.
Future<String> f1 = ex.submit(new EchoMessage("MESSAGE_TO_ANY_MEMBER"));
System.out.println(f1.get());

// Execute on a specific member.
// In this case the first member in cluster
Member firstMember = h.getCluster().getMembers().iterator().next();
Future<String> f2 = ex.submitToMember(new EchoMessage("MESSAGE_TO_SPECIFIC_MEMBER"), firstMember);
System.out.println(f2.get());

// Execute on all members
Map<Member, Future<String>> multiResult = ex.submitToAllMembers(new EchoMessage("MESSAGE_TO_ALL_MEMBERS"));
for(Future<String> f: multiResult.values()){
    System.out.println(f.get());
}

// Execute on a member owning particular key.
Future<String> f3 = ex.submitToKeyOwner(new EchoMessage("MESSAGE_TO_KEY_OWNER"), "key");
System.out.println(f3.get());
```

Executor Service (Continued)

Sample Callback Attaching

```
Config config = new Config();
HazelcastInstance h = Hazelcast.newHazelcastInstance(config);
IExecutorService ex = h.getExecutorService("my-distributed-executor");
ExecutionCallback printCallback = new ExecutionCallback() {
    @Override
    public void onResponse(Object o) {
        System.out.println(o);
    }
    @Override
    public void onFailure(Throwable throwable) {
        System.err.print("Failure: " + throwable.toString());
    }
};
ex.submit(new EchoMessage("MESSAGE_TO_ANY_MEMBER"), printCallback);
```

About Hazelcast

Hazelcast is an open-source in-memory data grid providing Java developers with an easy-to-use and powerful solution for creating highly available and scalable applications. Hazelcast can be used in the areas like clustering, in-memory NoSQL, application scaling, database caching and as an Oracle Coherence replacement and Software AG Terracotta alternative.

Resources

Getting Started Video: **hazelcast.org/start**

Docs: **hazelcast.org/docs**

Download: **hazelcast.org/download**

Group: **hazelcast.org/group**

Love: **hazelcast.org/love**

Blog: **hazelcast.org/blog**

Use Cases: **hazelcast.org/use-cases**

Support: **hazelcast.com/support**