

JBossCache

Bela Ban

Lead JGroups, Manager Clustering Team

JBoss Inc

Agenda

TreeCache

- **Architecture, API**
- **Aspects**
 - Local vs replicated cache**
 - Locking**
 - Transactional support**
 - Eviction and cache loading**

PojoCache

- **API**
- **Demo**

What is JBossCache ?

2 implementations

- **TreeCache**

Stores and replicates values in a tree (hence the name)

Each value is associated with a path and key

- **TreeCacheAOP (aka PojoCache)**

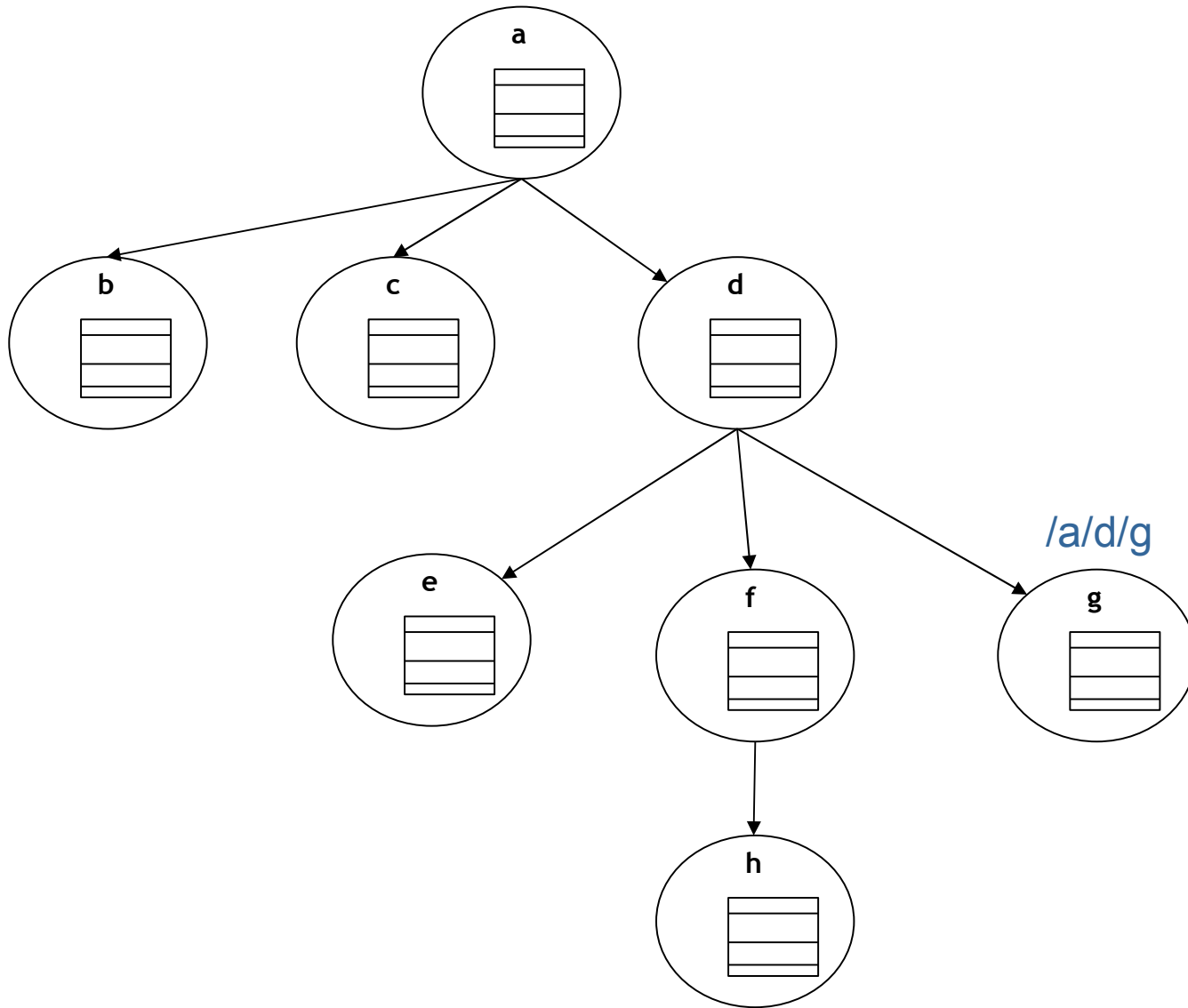
Manages caching, replication and persistence of Java objects (Pojos)

Object Oriented cache

Standalone or embeddable (MBean)

- **TreeCacheAOP requires JBossAOP**
- **Can be used with other app servers as well**

Architecture



API

put(FQN node, Object key, Object val)

- Adds a key and value to a given node.
- If the node doesn't exist, it will be created

put(FQN node, Map data)

- Adds a new node to the tree and sets its data. If the node already has data, then the new data will override the old one.

Object get(FQN node, Object key)

- Finds a node given its name and returns the value associated with a given key in its data map

API

remove(FQN node)

- Removes node and its children from the tree

remove(FQN node, Object key)

- Removes key from the node's hashmap

Sample code

```
try {
    tx=(UserTransaction)new
InitialContext(p).lookup("UserTransaction");
    TreeCache c=new TreeCache("test", null, 10000);
    c.setMode(TreeCache.REPL_ASYNC);
    c.start();
    tx.begin();
    c.put("/a/b/c", "age", new Integer(38));
    c.put("/a/b/c", "age", new Integer(39));
    tx.commit();
    assertEquals(new Integer(39), c.get("/a/b/c", "age"));
}
catch(Throwable t) {
    if(tx != null) tx.rollback();
}
```

Architecture

Based on interceptors

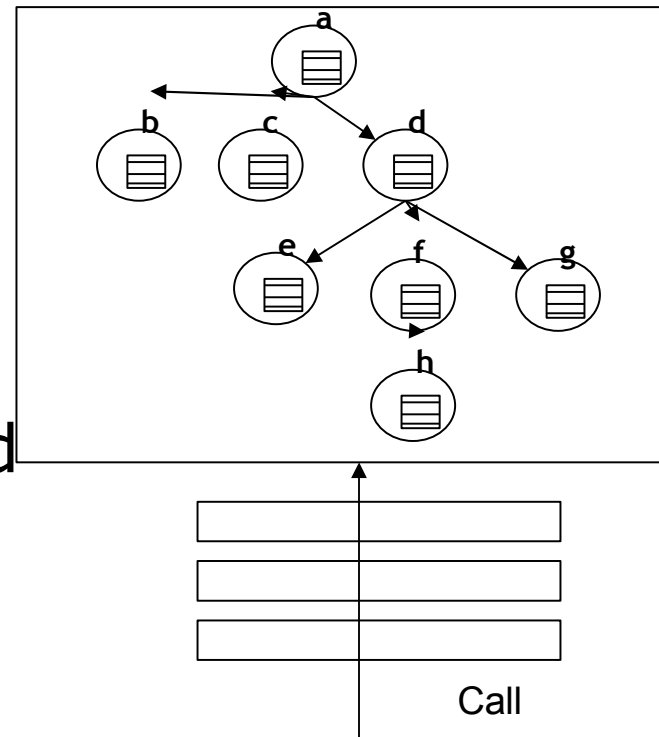
- Locking
- Replication
- Cache loading / cache storing
- Invoking method calls

Interceptor chain is configured at startup

- Based on configuration of cache

Add your own interceptors

- Metrics: cache hit/miss ratio
- Auditing



Aspects of TreeCache

Local or replicated

Locking

Eviction

Cache loading

Transactions

Configuration

Local versus replicated

LOCAL

- **Modifications are not replicated**

REPL_ASYNC

- **Asynchronous replication, done on a separate thread**

REPL_SYNC

- **Synchronous replication, caller blocks until modifications have been applied at every node in the cluster**

INVALIDATE_ASYNC

INVALIDATE_SYNC

Replication

If transaction is present, replication occurs at transaction commit time

- **Multiple changes can be made to the cache within a TX**
- **Single replication message is sent as part of the commit handling**
- **If there is a rollback, we have zero replication cost**

Changes without a transaction are replicated immediately

- **If there are 1000 changes, there will be 1000 replication messages**
- **Compare to 1000 changes in a TX, 1 replication message**

Buddy Replication

Instead of replicating everything to everyone, we replicate only to N backups

Example

- **10 nodes, every node has 100MB data on avg**
Default repl: every node has 1GB of data
BR (N=1): every node has 200MB of data

BR allows us to scale (data-wise), data is *not* a function of cluster size

Enabled via simple XML config attribute

Locking

Concurrent access is guarded

Optimistic locking

- Workspaces, commit fails if data modified by other TX

Pessimistic locking

- Locks at node level
- Isolation levels define locking policy

NONE

READ_UNCOMMITTED

READ_COMMITTED

REPEATABLE_READ

SERIALIZABLE

Eviction

Elements are evicted from cache

- **Keeps cache size bounded**

Time- or size-based

Eviction policy pluggable

- **Implement your own**

Eviction policies apply to a cache region

- **/myshop: evict after 10 minutes inactivity**
- **/myshop/products/pricelist: no eviction**
- **/myshop/shoppingCarts: evict after 30 mins inactivity**

Cache loaders

Opposite of eviction

Also pluggable

Loads elements from store into cache

Stores elements from cache into store

- **On put(), or on eviction (passivation/activation)**

Implementations

- **FileCacheLoader: file system**
- **JDBCCacheLoader: DB (Oracle, MySql, MSSQL, Postgres tested)**
- **BdbjeCacheLoader: Berkeley DB**

Cache loaders

Hierarchical CacheLoader

- **Uses TCP/JGroups/RMI to access remote cache**

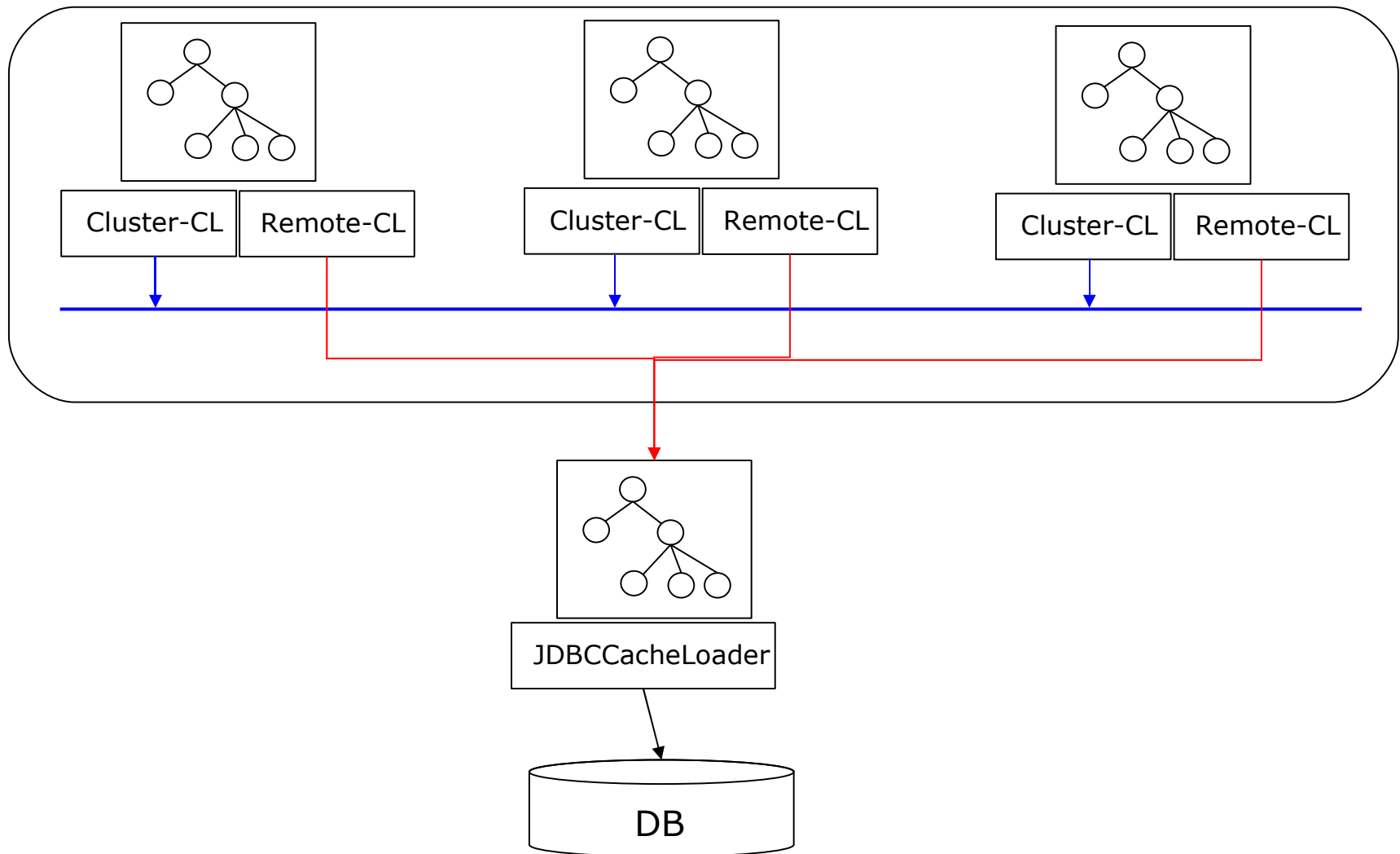
Clustered CacheLoader

- **Uses lookup across cluster to find element**

Chaining CacheLoader

- **Allows for CacheLoader chaining**
- **Clustered CacheLoader, followed by Remote CacheLoader, followed by JDBCClassLoader**

Example of hierarchical setup



Transactions

Transaction support pluggable

- **Just let JBossCache know the TransactionManager**
- **JTA interfaces**

If not defined, JBossCache runs without TXs

Implementations

- **JBoss**
- **Generic (Sun, WL, WS)**
- **Standalone**

Configuration

Via XML file

Same syntax as for JBoss MBeans

- **But also works standalone, or in other appservers**

Programmatic access

- **Setters**

What is PojoCache ?

Extends TreeCache

Manages Pojos rather than keys/values

Pojos are inserted to the cache, cache

- **keeps track of modifications to Pojos**
- **replicates modifications across cluster (possibly atomically with TXs)**
- **persists modifications to DB (via CacheLoader, if configured), possibly also at TX commit**

We use JBossAOP to instrument Pojos and keep track of state changes

API

putObject(FQN node, Object pojo)

- Adds a Pojo, manages it from now on
- All modifications are replicated/persisted, with ACIDity if TXs are used

Object getObject(FQN node)

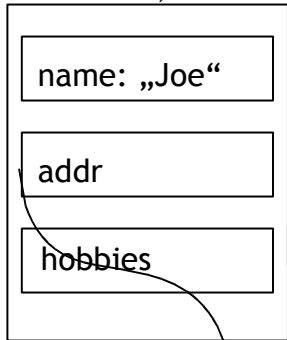
void removeObject(FQN node)

- Removes an object, PojoCache doesn't manage it anymore
- All modifications are written directly into the Pojo, no replication/persistence

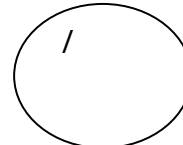
After adding Pojo to cache, Pojos are accessed directly

putObject() Mapping

Person p
(key=/husband)

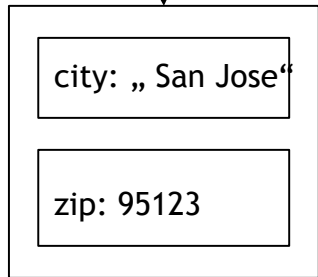
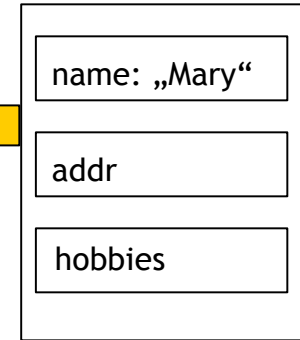


name	Joe

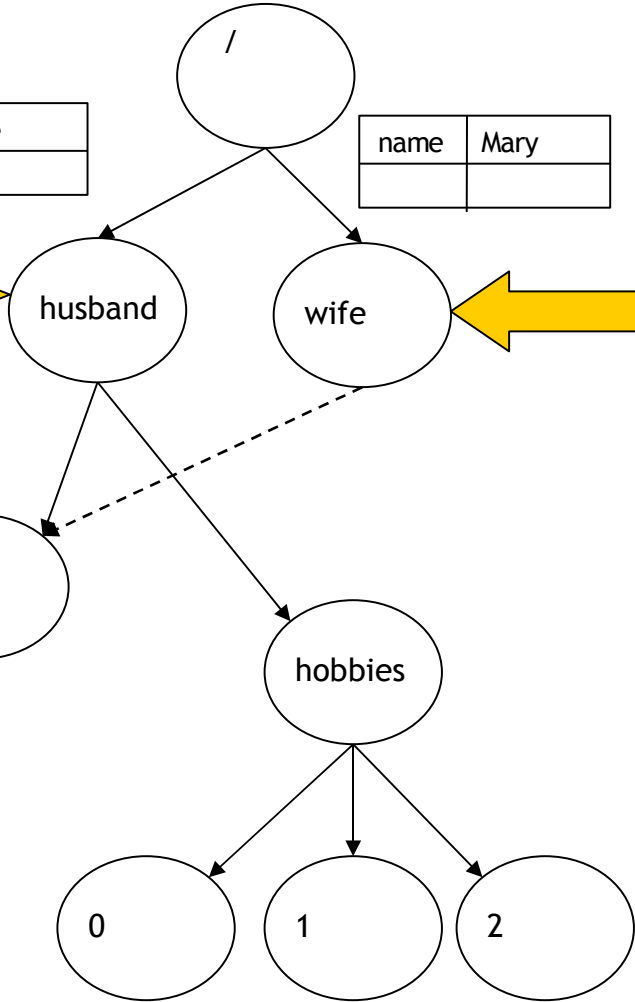


name	Mary

Person p
(key=/wife)



city	San Jose
zip	95123



How do we detect changes to a Pojo?

Tag Pojos with a marker annotation

Byte code for tagged Pojos is instrumented

- **Offline: aopc**
- **Online: loader / java.lang.instrument**

Instrumentation adds an interceptor stack

- **PojoCache adds a CacheInterceptor**
- **All access to Pojo is through the CacheInterceptor**
- **CacheInterceptor redirects reads/writes to Pojos to PojoCache !**
- **Modifications are written back to Pojo at TX commit**

Instrumentation of Pojos

Needed to detect state changes

- Online (load time) or offline (aopc)
- `putObject()` adds interceptor to Person's stack
- `removeObject()` removes it again

JDK 5: use of annotations

JDK 1.4: use of javadoc and annotation compiler

```
@AopMarker // JDK5.0 annotation  
public class Person {  
}
```


Sample code

```
TreeCacheAop cache = new TreeCacheAop();  
PropertyConfigurator config = new PropertyConfigurator();  
config.configure(cache, "META-INF/replySync-service.xml");  
cache.start(); // kick start TreeCacheAop  
  
Person joe = new Person("Joe Black", 30); // Just a Pojo  
cache.putObject("/person/joe", joe); // ask cache to manage joe  
joe.setAge(40); // set to cache (will replicate as well).  
  
Person p=cache.getObject("/person/joe"); // An alias for joe.  
p.getAge(); // Should be 40  
  
cache.removeObject("/person/joe"); // detach from cache  
joe.setAge(50); // not replicated, not persisted (not managed)
```

Demo

TreeCacheGui2: mbr=10.1.16.194:2674

Operations

- /
 - aop
 - joe
 - _JBossInternal_
 - _RefMap_
 - _aop_joe_address
 - p

Name	Value
age	41
__jboss:internal:clas...	class org.jboss.cache.aop.test.Person
AOPInstance	org.jboss.cache.aop.AOPInstance@11381e7
name	Joe Black

BeanShell Desktop 1.1

Bsh Workspace: 0

```
File Font
skills=null, languages=null>
bsh % p.setAge(33);
bsh % p;
<name=null, age=33, hobbies=, address=street=123 Albert Ave, city=Sunnyvale,
zip=94086, skills=null, languages=null>
bsh % tree.printLockInfo();
<
/aop
  /joe
/ _JBossInternal_
  / _RefMap_
    / _aop_joe_address
/p
>
bsh % tree.get("p");
<
name=p
fqcn=/p
data={}
read locked=false
write locked=false>
bsh %
```

Bsh Workspace: 0

Links

JBossCache:

www.jboss.com/products/jboss-cache