



JBOSS CACHE

Giorgio Vinci (Ha lavorato per diversi anni utilizzando tecnologie open source. Consulente K-Tech si occupa dell'analisi e progettazione di applicazioni web e di portali. E' certificato in JBOSS versione 4.0.)

Luca Stancapiano (Ha iniziato come programmatore C/C++ in ambiente Unix. Si è specializzato nella programmazione in java e in c, e collabora con la K-Tech srl dal 2001. Ha tenuto consulenze per importanti progetti in Finsiel, Vitrociset e altre società. E' specializzato sulla piattaforma Jboss, sistemi operativi Solaris, AIX, Unix e windows e su vari progetti Open Source.)



Step del talk

- Introduzione
- Architettura
- TreeCache
- Treecache AOP
- Conclusioni



A chi si rivolge il Talk

A chi lavora con databases e connessioni remote

Obiettivi del Talk

Mostrare come JBossCache possa rendere piu' veloci le nostre applicazioni



Cos'è una cache

- “In computer science, a **cache** (pronounced kăsh) is a collection of data duplicating original values stored elsewhere or computed earlier, where the original data are expensive (usually in terms of access time) to fetch or compute relative to reading the cache. Once the data are stored in the cache, future use can be made by accessing the cached copy rather than refetching or recomputing the original data, so that the average access time is lower.” Da Wikipedia



Perché usare una Cache

- **Problema: La lentezza**

- Lentezza nell'accesso ad uno o più db
- Lentezza nell'esecuzione di query complesse e costruzione di oggetti java a partire da dati relazionali
- Lentezza nell'accesso a repository remoti (content management system, sistemi legacy, eis, etc.)
- Lentezza e impiego costante di risorse (cpu, rete, memorie).

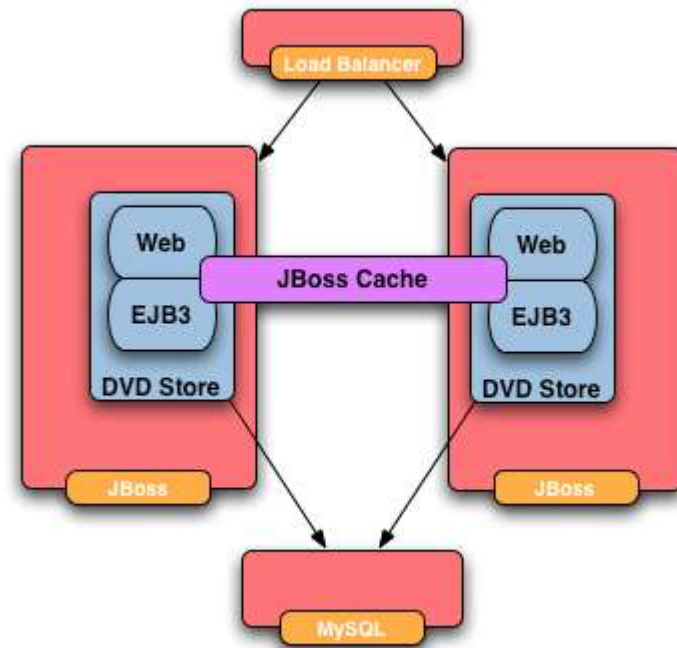
- **Soluzione**

- Mantenere in memoria le informazioni relativa alle ultime richieste.



Un problema reale

- Applicazione clusterizzata che accede in modo massivo ad un db remoto che ha tempi di risposta elevati.
- Immaginiamo un grosso carico di utenze che chiedono le stesse informazioni.
- Problematiche:
 - Lentezza
 - Server remoto totalmente occupato a rispondere a noi (inutilizzabile da parte di altre applicazioni)
 - Rete intasata

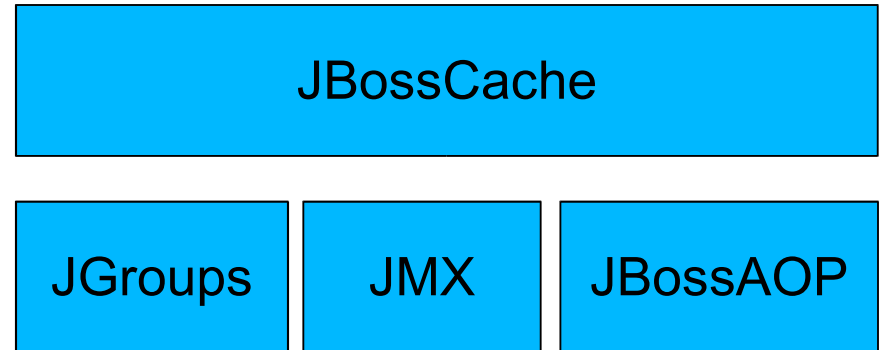


Introduzione



Architettura

- Totalmente integrata all'interno dell'AS.
- Standalone



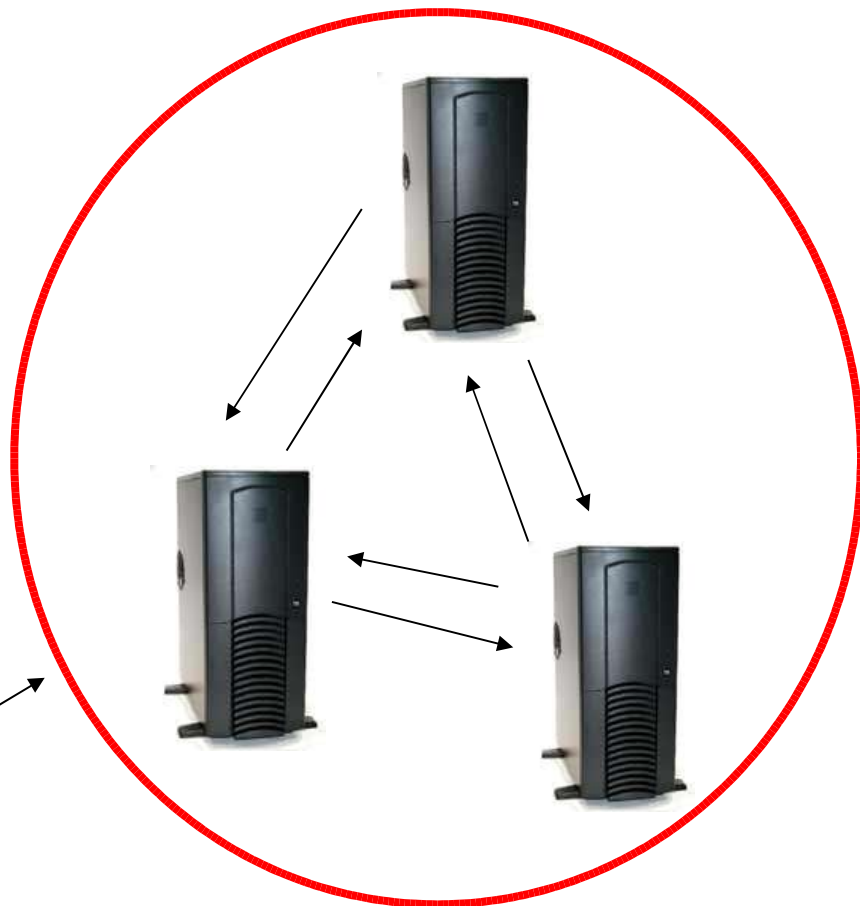
Architettura



Il Cluster (JGroups)

Scalabilità – Load Balancing

Affidabilità – Fault tolerance



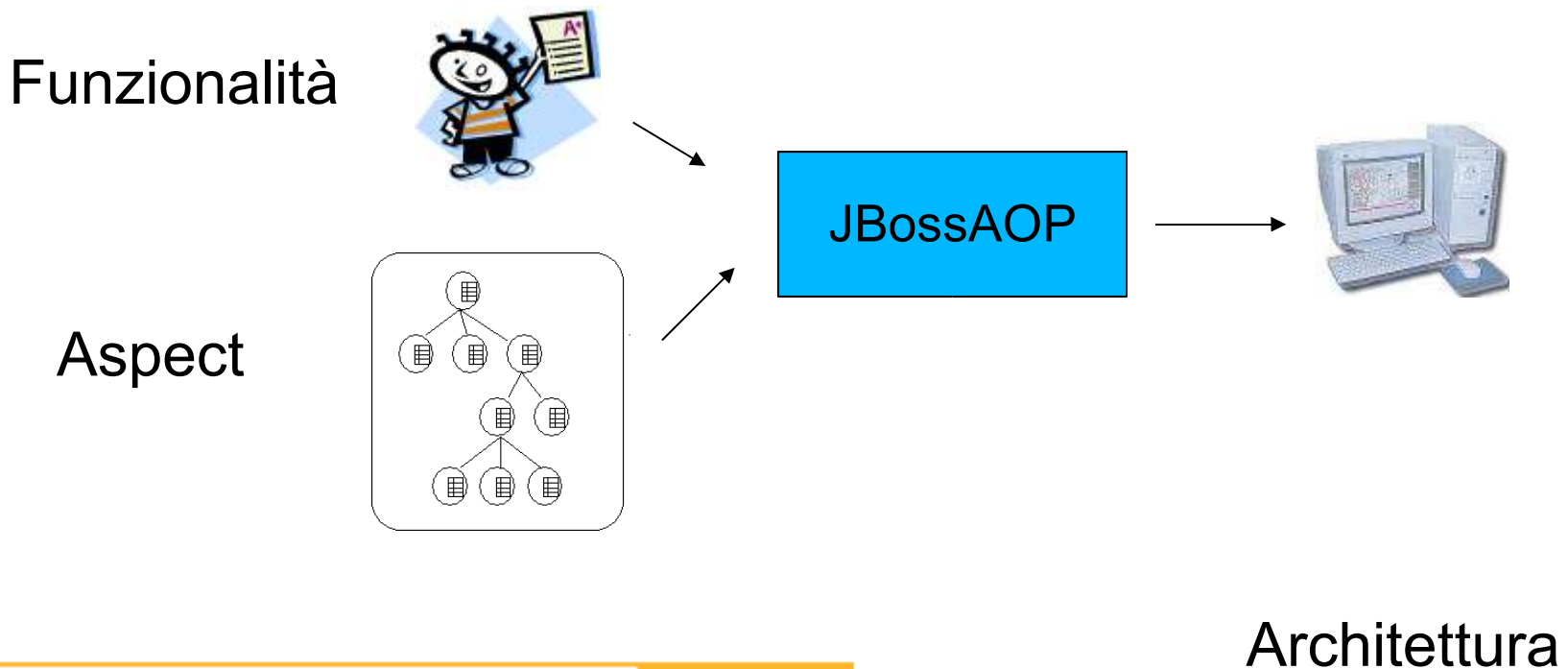
Architettura



AOP

Aspect Oriented Programming

- Separazione delle funzionalità trasversali





JBoss Cache

Due tipologie di Cache simili ma diverse

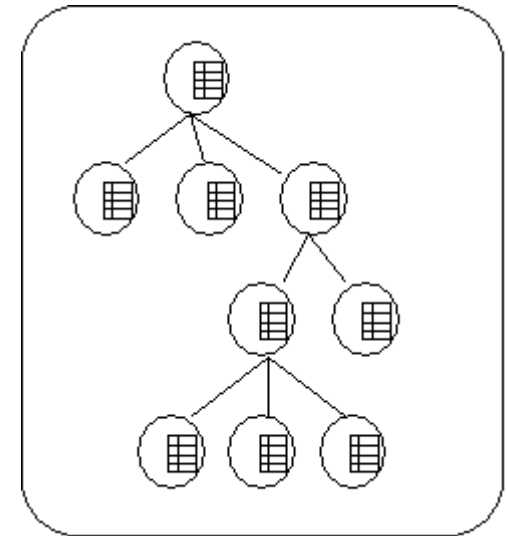
- Tree Cache
- Tree Cache AOP

Architettura



Tree Cache

- Struttura ad albero.
- Ogni nodo contiene una Hashmap e può avere più sottonodi



```
TreeCache tree = new TreeCache();
PropertyConfigurator config = new PropertyConfigurator();
config.configure(tree, "META-INF/replSync-service.xml");
tree.createService(); // not necessary
tree.startService(); // kick start tree cache
```

```
tree.put("/jipday/intervento1", "titolo", "JBossCache");
tree.put("/jipday/intervento1", "teachers", "Luca e Giorgio");
```

TreeCache



Tree Cache

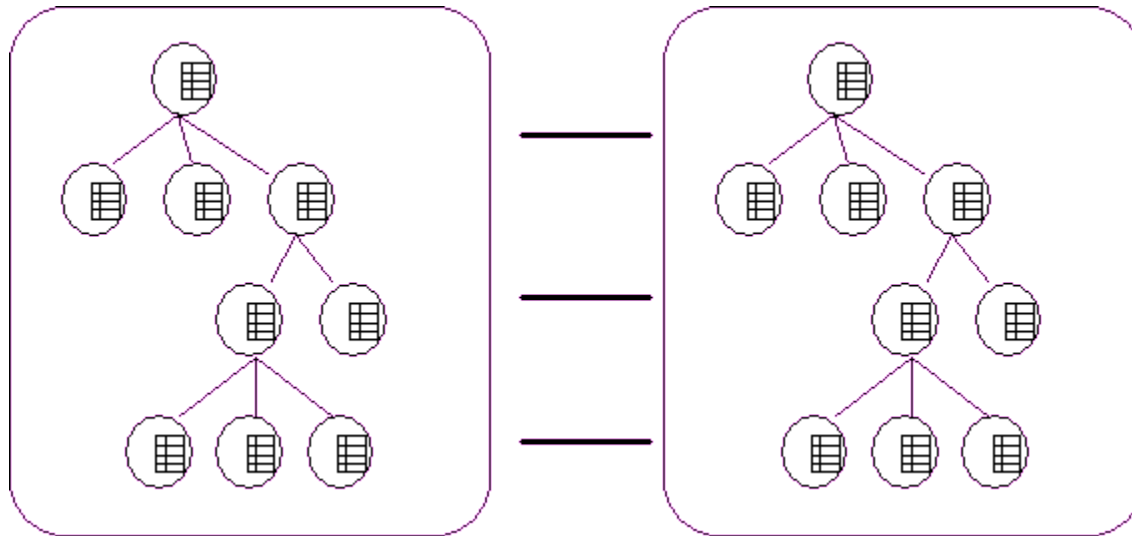
- Replicazione
- Transazionalita'
- Locking
- Persistenza
- Ottimizzazione della memoria

TreeCache



Tree Cache - Replicazione

- Multicast IP e porta
- GroupName
- Replicazione sincrona e asincrona



TreeCache



Tree Cache – Transazionalita'

Atomicità dell'operazione

```
Properties prop = new Properties();
prop.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.cache.transaction.DummyContextFactory");
UserTransaction tx=(UserTransaction)new InitialContext(prop).lookup
("UserTransaction");
TreeCache tree = new TreeCache();
PropertyConfigurator config = new PropertyConfigurator();
config.configure(tree, "META-INF/replSync-service.xml");
...
try {
    tx.begin();
    tree.put("/jipday/intervento1", "titolo", "JBossCache");
    tree.put("/jipday/intervento1", "teachers", "Luca e Giorgio");
    tx.commit();
}
catch(Throwable ex) {
    try { tx.rollback(); } catch(Throwable t) {}
}
```

TreeCache



Tree Cache - Locking

- Pessimistic Locking

- NONE
- READ_UNCOMMITTED
- READ_COMMITTED
- REPEATABLE_READ
- SERIALIZABLE

`<attribute name="IsolationLevel">REPEATABLE_READ</attribute>`

TreeCache



Transaction Isolation: Riepilogo

Isolation Level	Dirty Reads?	Unrepeatable Reads?	Phantom Reads?
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

TreeCache



Transaction Isolation: Riepilogo

- **READ_UNCOMMITTED (fastest)**
 - You can see other transactions' uncommitted updates
 - No data sharing, non mission-critical apps.
- **READ_COMMITTED (fast)**
 - You can't see other transactions' uncommitted updates
 - Rough Reporting tools. Data should be committed. Only need a snapshot.
- **REPEATABLE_READ (medium)**
 - When you read a row, nobody can touch that row until you're done
 - Mission-critical, high data sharing. You have not performed a query, so phantoms are OK, so long as the data you're working with isn't modified.
- **SERIALIZABLE (slow)**
 - After you do a query, nobody can insert new rows that satisfies your query until you're done
 - Mission-critical, high data sharing. You have performed a query, and you'd like to know about any new phantom rows.

TreeCache



Tree Cache - Persistenza

- FileCacheLoader
- JDBCClassLoader
- LocalDelegatingCacheLoader
- RpcDelegatingCacheLoader
- RmiDelegatingCacheLoader

```
<mbean code="org.jboss.cache.TreeCache" name="jboss.cache:service=TreeCache">
  <attribute
name="CacheLoaderClass">org.jboss.cache.loader.bdbje.BdbjeCacheLoader</attribute>
  <!-- attribute
name="CacheLoaderClass">org.jboss.cache.loader.FileCacheLoader</attribute -->
  <attribute name="CacheLoaderConfig" replace="false">
    location=c:\\tmp\\bdbje
  </attribute>
  <attribute name="CacheLoaderShared">true</attribute>
  <attribute name="CacheLoaderPreload">/</attribute>
  <attribute name="CacheLoaderFetchTransientState">false</attribute>
  <attribute name="CacheLoaderFetchPersistentState">true</attribute>
  <attribute name="CacheLoaderAsynchronous">true</attribute>
</mbean>
```

TreeCache



Tree Cache - Memoria

- Pattern LRU
- Non replicabile

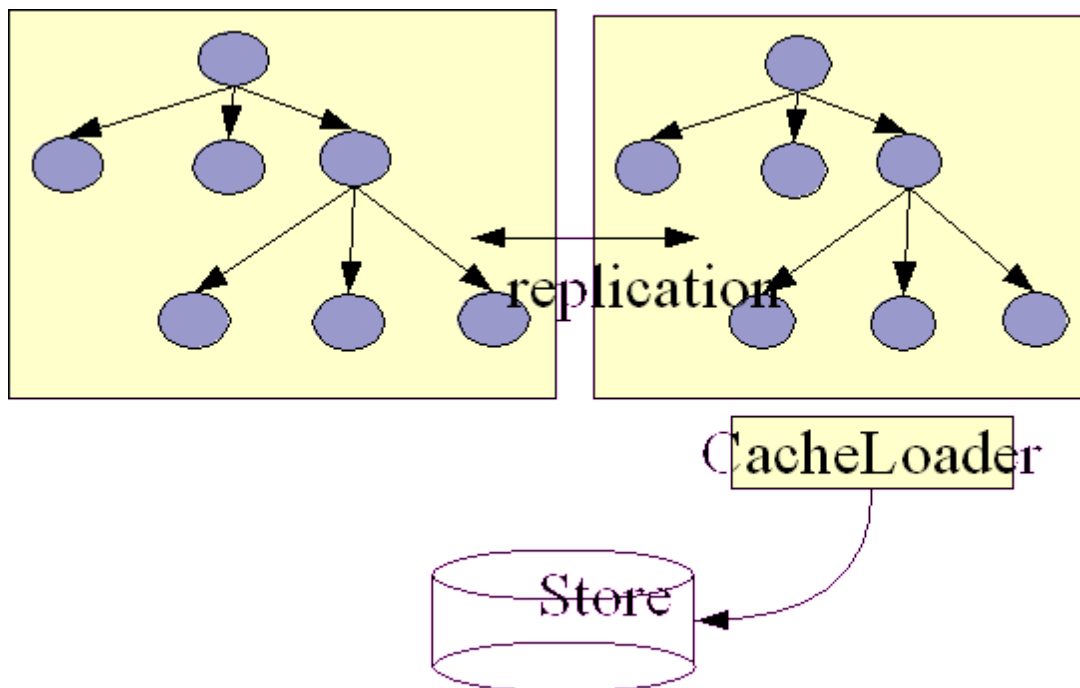
```
<attribute name="EvictionPolicyClass">org.jboss.cache.eviction.LRUPolicy</attribute>
  <!-- Specific eviction policy configurations. This is LRU -->
  <attribute name="EvictionPolicyConfig">
    <config>
      <attribute name="wakeUpIntervalSeconds">5</attribute>
      <region name="/org/jboss/test/data">
        <attribute name="maxNodes">5</attribute>
        <attribute name="timeToLiveSeconds">4</attribute>
      </region>
      <region name="/org/jboss/data">
        <attribute name="maxNodes">5000</attribute>
        <attribute name="timeToLiveSeconds">1000</attribute>
      </region>
    </config>
  </attribute>
```

TreeCache



TreeCache

n Nodi - 1 CacheLoader

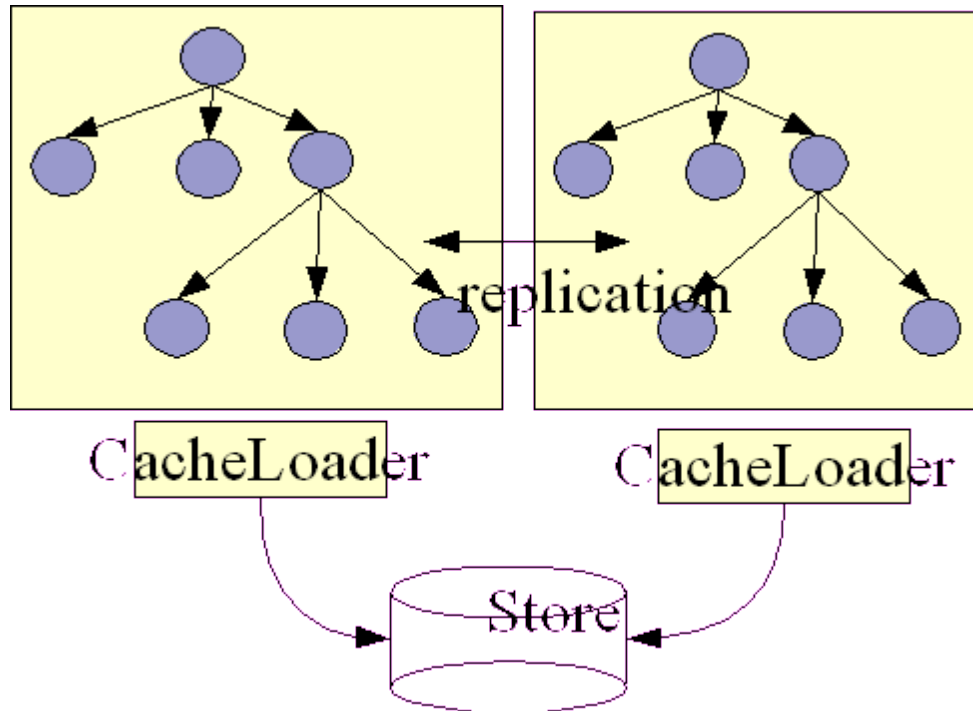


TreeCache



TreeCache

n Nodi – n CacheLoader - 1 store

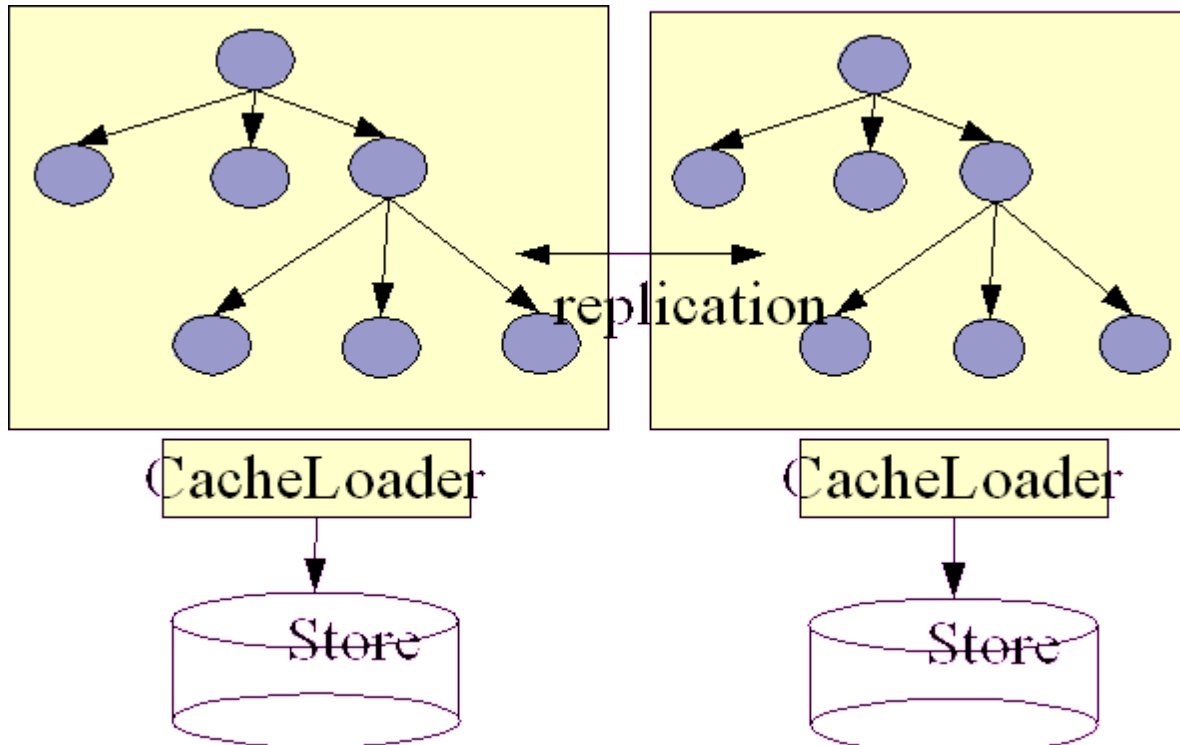


TreeCache



TreeCache

1 Nodo - 1 CacheLoader



TreeCache



TreeCache - Configurazione

1 di 3

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
  <classpath codebase="./lib" archives="jboss-cache.jar, jgroups.jar"/>
  <mbean code="org.jboss.cache.TreeCache"
  name="jboss.cache:service=TreeCache">
    <depends>jboss:service=Naming</depends>
    <depends>jboss:service=TransactionManager</depends>
    <attribute name="TransactionManagerLookupClass">
      org.jboss.cache.DummyTransactionManagerLookup
    </attribute>
    <attribute name="IsolationLevel">REPEATABLE_READ</attribute>
    <attribute name="CacheMode">REPL_ASYNC</attribute>
    <attribute name="UseReplQueue">>false</attribute>
    <attribute name="ReplQueueInterval">0</attribute>
    <attribute name="ReplQueueMaxElements">0</attribute>
    <attribute name="ClusterName">DefaultPartition</attribute>
    <attribute name="ClusterConfig">
```

TreeCache



TreeCache - Configurazione

2 di 3

```
<config>
  <UDP mcast_addr="228.1.2.3" mcast_port="45566"
ip_ttl="8" ip_mcast="true"
mcast_send_buf_size="800000" mcast_rcv_buf_size="150000"
ucast_send_buf_size="800000" ucast_rcv_buf_size="150000"
loopback="false"/>
  <PING timeout="2000" num_initial_members="3"
up_thread="true" down_thread="true"/>
  <MERGE2 min_interval="10000" max_interval="20000"/>
  <FD shun="true" up_thread="true" down_thread="true"
timeout="2500" max_tries="5"/>
  <VERIFY_SUSPECT timeout="3000" num_msgs="3"
up_thread="true" down_thread="true"/>
  <pbcast.NAKACK gc_lag="50" retransmit_timeout="300,600,1200,2400,4800"
max_xmit_size="8192"
up_thread="true" down_thread="true"/>
  <UNICAST timeout="300,600,1200,2400,4800" window_size="100" min_threshold="10"
down_thread="true"/>
  <pbcast.STABLE desired_avg_gossip="20000"
up_thread="true" down_thread="true"/>
  <FRAG frag_size="8192"
down_thread="true" up_thread="true"/>
  <pbcast.GMS join_timeout="5000" join_retry_timeout="2000"
shun="true" print_local_addr="true"/>
  <pbcast.STATE_TRANSFER up_thread="true" down_thread="true"/>
</config>
```

TreeCache



TreeCache - Configurazione

3 di 3

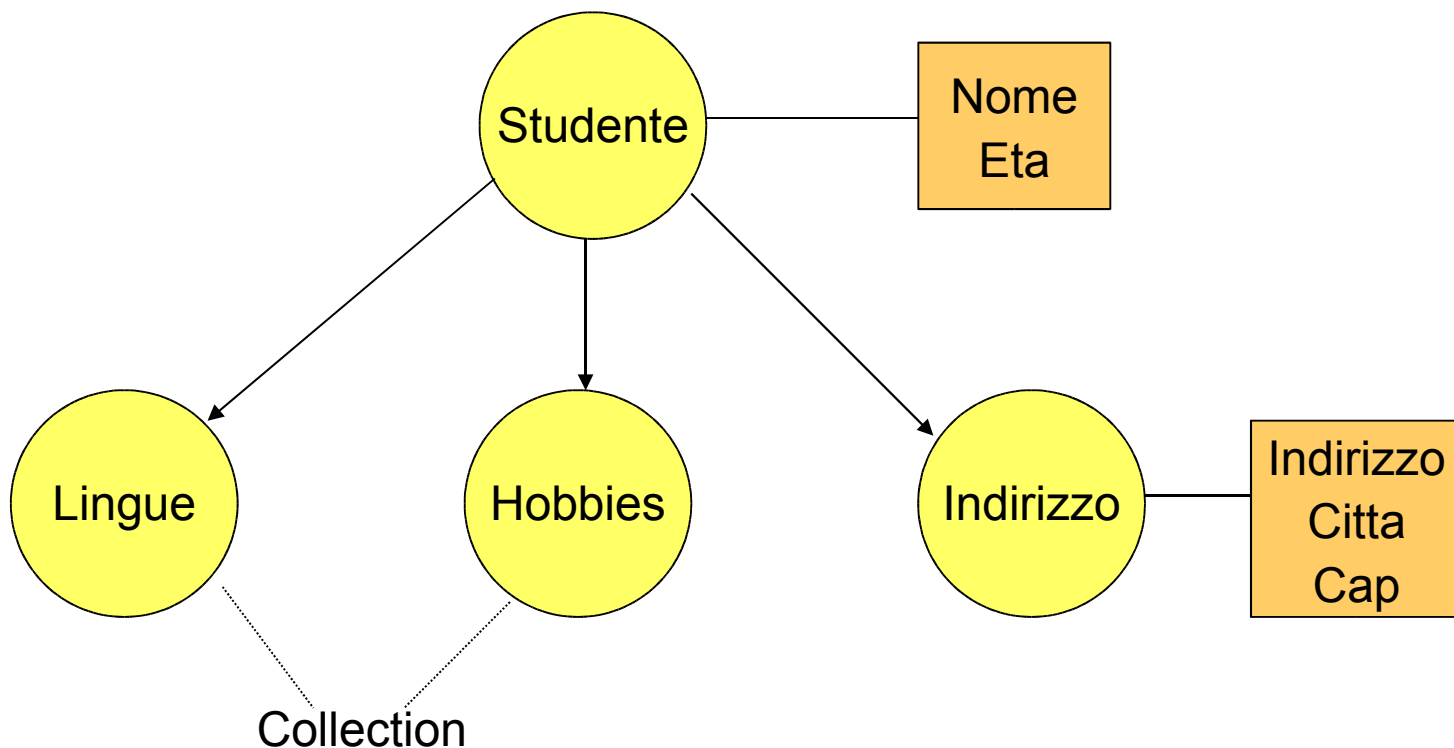
```
</attribute>
<attribute name="InitialStateRetrievalTimeout">5000</attribute>
<attribute name="SyncReplTimeout">10000</attribute>
<attribute name="LockAcquisitionTimeout">15000</attribute>
<attribute name="EvictionPolicyClass">    </attribute>
<!--<attribute name="FetchStateOnStartup">>false</attribute>
<attribute
name="CacheLoaderClass">org.jboss.cache.loader.JDBCCacheLoader</attribute>
<attribute name="CacheLoaderConfig">
    cache.jdbc.datasource=DefaultDS
</attribute>
<attribute name="CacheLoaderShared">>true</attribute>
<attribute name="CacheLoaderPreload">/</attribute>
<attribute name="CacheLoaderFetchTransientState">>false</attribute>
<attribute name="CacheLoaderFetchPersistentState">>true</attribute>-->
</mbean>
</server>
```

TreeCache



TreeCacheAOP - Introduzione

“Object-Oriented” Cache



TreeCacheAOP



TreeCacheAOP - Introduzione

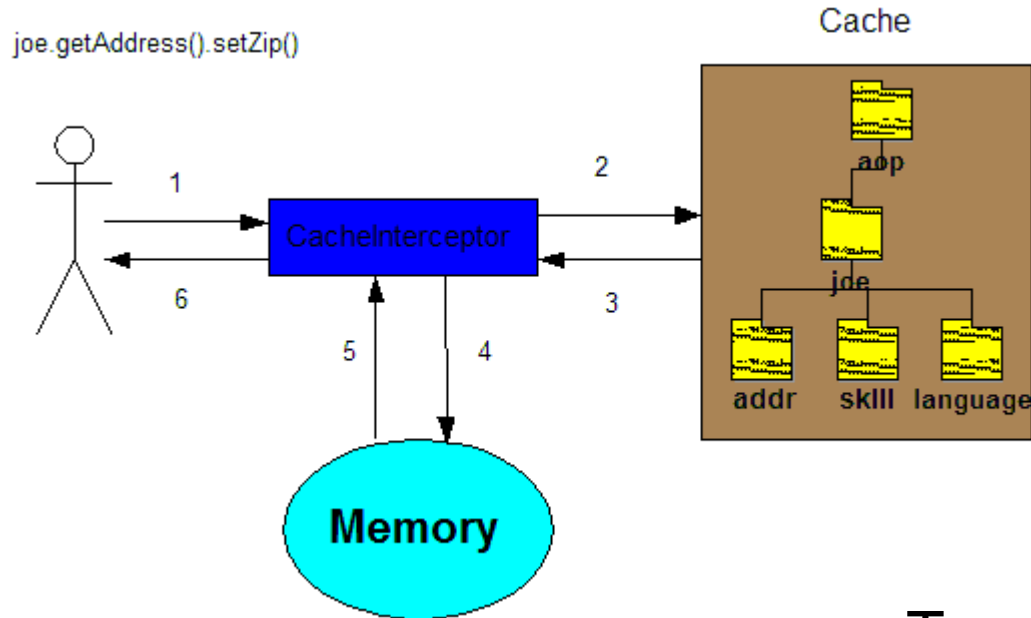
- Mapping automatico di oggetti complessi
- Aggiunge funzionalità e semplicità di utilizzo alla TreeCache.

TreeCacheAOP



TreeCacheAOP - POJO

- Il POJO viene intercettato da AOP (InstanceAdvisor)
- Non necessita di essere serializzabile



TreeCacheAOP



TreeCacheAOP - AOP

java.lang.reflect

```
InvocationHandler handler = new MyInvocationHandler(...);  
Class proxyClass = Proxy.getProxyClass(  
    Person.class.getClassLoader(), new Class[] { Foo.class });  
Person f = (Person) proxyClass.  
    getConstructor(new Class[] { InvocationHandler.class }).  
    newInstance(new Object[] { handler });
```

InvocationHandler

```
invoke(Object proxy, Method method, Object[] args)
```

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<aop>
```

```
<prepare expr="field(* $instanceof{org.jboss.test.cache.test.standAloneAop.Link}->*)" />
```

```
<prepare expr="field(* $instanceof{org.jboss.test.cache.test.standAloneAop.Student}->*)" />
```

```
<prepare expr="field(* $instanceof{org.jboss.test.cache.test.standAloneAop.Address}->*)" />
```

```
<prepare expr="field(* $instanceof{org.jboss.test.cache.test.standAloneAop.Person}->*)" />
```

```
</aop>
```

- Prepare expr - crea il pointcut
- Field – indica che l'intercettazione viene eseguita a livello di campo (->* segnala tutti i campi della classe)
- \$instanceof – indica che tutte le sottoclassi della classe saranno intercettate

TreeCacheAOP



TreeCacheAOP - Componenti

Mantiene automaticamente allineate le properties di un componente del POJO registrato

```
Persona luca = new Persona(); // instantiate a Person object named joe  
luca.setNome("Luca Stancapiano");  
luca.setEta(29);
```

```
Address addr = new Address(); // instantiate a Address object named addr  
addr.setCity("Roma");  
addr.setStreet("Via Pietralata, 13");  
addr.setZip(00100);  
luca.setAddress(addr); // set the address reference
```

```
tree.startService(); // kick start tree cache  
tree.putObject("/jipday/teacher/luca", luca);  
addr.setZip(456);
```

```
<aop>  
  <prepare expr="field(* org.jboss.test.standAloneAop.Persona->*)" />  
  ...  
</aop>
```

TreeCacheAOP



TreeCacheAOP – Ereditarietà

Cache automatica delle property del POJO attraverso superclassi e interfacce

```
Studente marco = new Studente(); // Student extends Person class
marco.setNome("Marco Rossi"); // This is base class attributes
marco.setEta(29); // This is also base class attributes
marco.setClasse("Terza"); // This is Student class attribute
tree.putObject("/jipday/studente/marco", marco);
```

```
//...
```

```
marco = (Studente)tree.getObject("/jipday/studente/marco");
Persona persona = (Persona)marco; // it will be correct here
marco.setClasse("Seconda"); // will be intercepted by the cache
marco.setNome("Marco Rossi II"); // also intercepted by the cache
```

```
<aop>
```

```
  <prepare expr="field(* $instanceof{org.jboss.test.standAloneAop.Persona}->*)" />
```

```
</aop>
```

TreeCacheAOP



TreeCacheAOP - Liste

....e le liste

```
Person joe = new Person();  
joe.setName("Joe Black"); // This is base class attributes
```

```
ArrayList lang = new ArrayList();  
lang.add("English");  
lang.add("Mandarin");
```

```
joe.setLanguages(lang);
```

```
// This will map the languages List automatically and swap it out with the  
proxy reference.
```

```
tree.putObject("/aop/student/joe", joe);
```

```
ArrayList lang = joe.getLanguages(); // Note that lang is a proxy reference  
lang.add("French"); // This will be intercepted by the cache
```

TreeCacheAOP



TreeCacheAOP - InternalTree

```
Person joe = new Person(); //  
instantiate a Person object named joe  
Person mary = new Person(); //  
instantiate a Person object named mary
```

```
Address addr = new Address(); //  
instantiate a Address object named addr  
addr.setCity("Sunnyvale");  
addr.setStreet("123 Albert Ave");  
addr.setZip(94086);
```

```
joe.setAddress(addr); // set the address  
reference
```

```
mary.setAddress(addr); // set the  
address reference
```

```
tree.startService(); // kick start tree
```

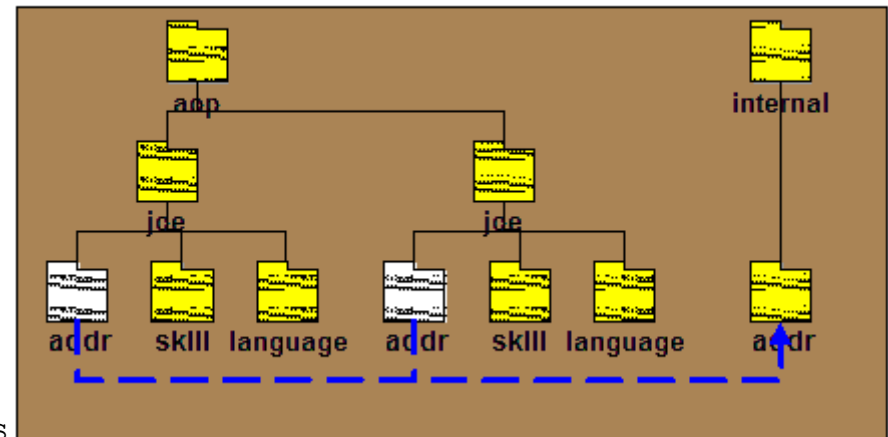
```
tree.putObject("/aop/joe", joe);
```

```
tree.putObject("/aop/mary", mary);
```

```
tree.removeObject("/aop/joe");
```

```
Address maryAddr = mary.getAddress(); // Should still have the address.
```

Cache



TreeCacheAOP



Pro e Contro di JBossCache

- **Pro**
 - Standalone può essere riutilizzata su qualsiasi progetto e piattaforma
 - Distribuita
 - Transazionale
- **Contro**
 - Non supporta totalmente il two phase commit (in caso di CacheLoader su due db distinti).
 - Utilizza solo pessimistic locking.

Conclusioni



TreeCache vs TreeCacheAOP

- La TreeCacheAOP gestisce in modo automatico l'aggiornamento dei componenti di un nodo nella cache.
- TreeCacheAOP ottimizza il lavoro dello sviluppatore (meno codice)
- TreeCache è più rapido perché fa un uso meno pesante della reflection.

Conclusioni



Q&A

TUTTO CHIARO NO?!?!?

Conclusioni



Ringraziamenti

- JBOSS Group per le immagini (www.jboss.org)
- Fabrizio Marini per le slide sulle transazioni (www.javaportal.it)
- Daniele Montagni e Mirko Scognamiglio per l'articolo sull'AOP (www.javaportal.it)
- Il sito wikipedia.org
- Tutti voi che avete partecipato!

Conclusioni