

Ehcache による分散キャッシュ

1. Ehcache とは	2
2. 機能	3
3. パーティション・キャッシュの実現	6
3.1. キャッシュサーバのインストール	6
3.2. キャッシュサーバの設定	7
3.3. キャッシュの設定	9
3.4. REST フル API	13
3.5. クライアントの実装	14
4. まとめ	16

1. Ehcache とは

皆様は、アクセス数やデータ量の増加に伴い、アプリケーションのレスポンスが悪化する体験を少なからずしていると思います。主な原因として「データベース負荷」と「CPU 負荷」が考えられます。

●データベース負荷

ほとんどのアプリケーションは、データベースへデータを格納し、データベースからデータを取得します。そのため、データベースのパフォーマンスの悪化は、アプリケーション全体のパフォーマンスに直結します。

アクセス数の増加は、データベースへの問い合わせを増加させます。さらに、問い合わせが増加するとデータベースの問い合わせを処理するスピードの低下や処理しきれない問い合わせが停滞することになります。また、昨今、データの最終的な保存場所であるディスクの I/O の性能は、容量の増加に比べあまり進化していません。そのため、データ量の増加によって低速なディスクへアクセスしている時間が長くなり、データベースへの問い合わせ時間が増加します。

●CPU 負荷

アプリケーションにアクセスがあるたびに高コストの処理を行っている、アクセス数の増加により CPU を酷使することになります。高コスト処理の例としてアプリケーションサーバでのデータ加工やデータベースでの複雑な問い合わせが挙げられます。

以上のようなデータベース負荷と CPU 負荷を考慮し、データベースへの問い合わせや高コストの処理をできるだけ避けるアーキテクチャにすることによって、レスポンスが悪化しにくいアプリケーションにすることができます。

Ehcache は、O/R マッピングライブラリである Hibernate や軽量コンテナである Spring でもサポートされるなど幅広く使用され、様々な用途に使用できる Java で実装された分散キャッシュです。Apache ライセンスの下で使用することができます。

Ehcache を使用すると、データベースから取得したデータや高コストの処理を行った結果データを高速なメモリにキャッシュし、再度そのデータが必要なときに Ehcache からデータを取得することによって、データベースへの問い合わせや高コストの処理を減少させることができます。

2. 機能

Ehcache は、以下のような特徴的な機能を持っています。

●メモリストア、ディスクストア

高パフォーマンスなメモリストア、および、ディスクストアを提供します。メモリストアにキャッシュできるオブジェクト数を超えた場合は、必要性が最も低いオブジェクトをキャッシュから削除し、新たなオブジェクトをキャッシュする機能が持っています。必要性が最も低いと判断するポリシーには以下の3つあります。

- LRU
最近、使用されていないオブジェクトを必要性が最も低いと判断します。
- LFU
最も使用頻度が少ないオブジェクトを必要性が最も低いと判断します。
- FIFO
最初にキャッシュしたオブジェクトを必要性が最も低いと判断します。

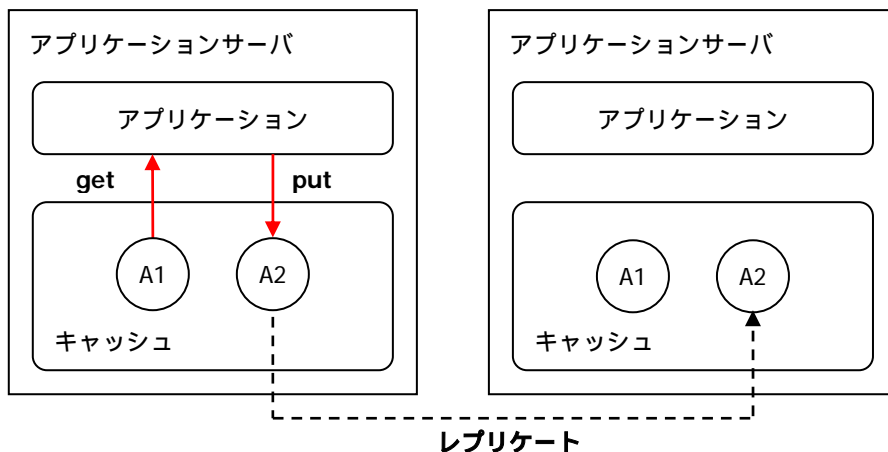
●gzip 圧縮サーブレットフィルタ

リクエスト URI とクエリ文字列に基づいたページ全体、または、一部をキャッシュできるサーブレットフィルタです。gzip 圧縮によるブラウザへの転送をサポートし転送量を削減することができます。

●レプリケーション

Ehcache は、RMI を使用してレプリケーション・キャッシュを実現することができます。レプリケーション・キャッシュとは、あるアプリケーションサーバのキャッシュにデータを格納すると他のアプリケーションサーバのキャッシュへデータをレプリケートするキャッシング方式です。

図1 レプリケーション・キャッシュ



レプリケーション・キャッシュは、アプリケーションがキャッシュからデータを取得するとき、アプリケーションサーバ内のメモリからデータを取得するため高速に動作します。しかし、キャッシュにデータを格納するときに一貫性を考慮しレプリケーションを同期で行った場合は、レプリケーションが終了するまで待機することになります。アプリケーションサーバが多くなると待機時間が長くなるので注意が必要です。また、キャッシュ容量は、1つのプロセスに割り当て可能なメモリ容量が上限となります。キャッシュするデータ量がそれほど多くないアプリケーションに向いています。

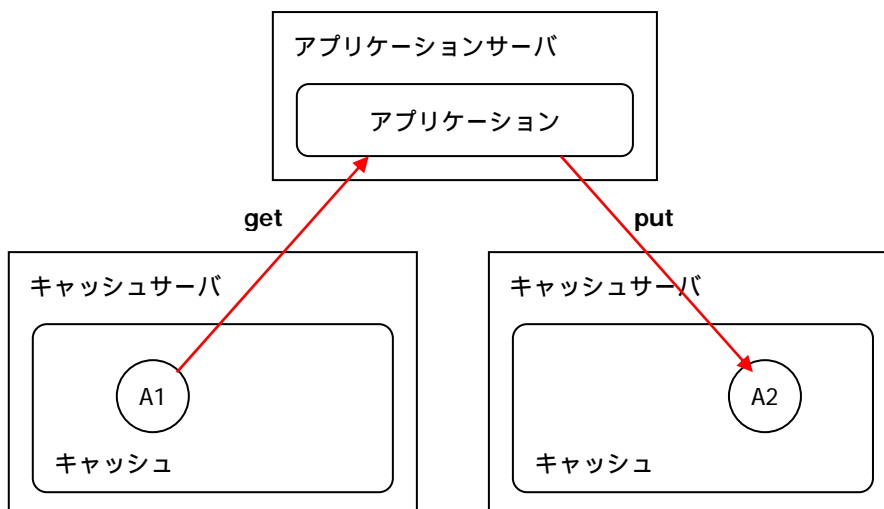
Ehcache では、レプリケート対象となるキャッシュを指定する方法は、対象を固定的に指定する方法とマルチキャストにより自動的に対象を探す方法があります。

●キャッシュサーバ

キャッシュサーバは、REST フルと SOAP の 2 つの API に対応しています。両 API とも様々なプログラミング言語をサポートします。しかし、Ehcache コミュニティではクライアントを提供していないため、利用者がクライアントを実装する必要があります。

キャッシュサーバとどのデータをどのキャッシュサーバにキャッシュさせるかを決定する分散アルゴリズムを組み込んだクライアントを組み合わせることでパーティション・キャッシュを実現することができます。パーティション・キャッシュとは、複数のキャッシュサーバにデータを分散させてキャッシュするキャッシング方式です。

図2 パーティション・キャッシュ

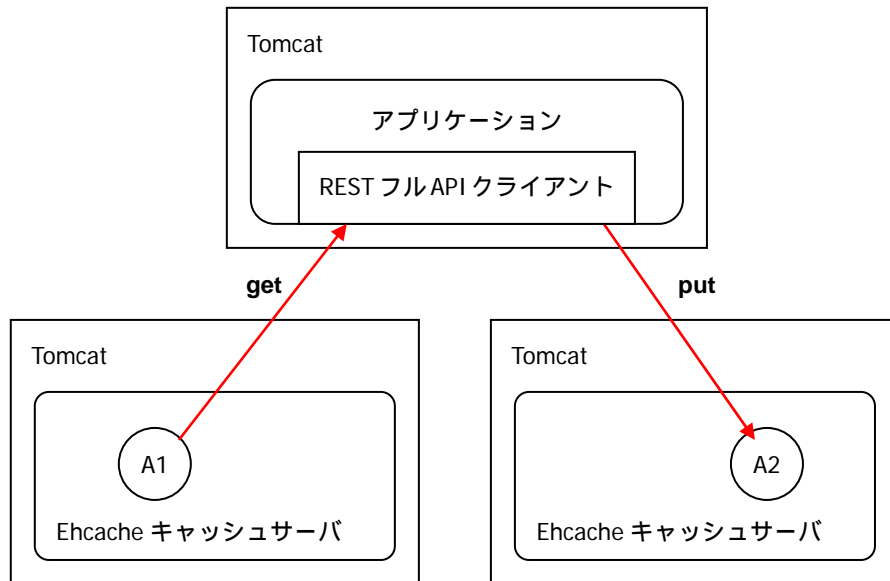


パーティション・キャッシュは、キャッシュサーバを追加することでキャッシュ容量を拡張することができます。大量データをキャッシュする必要があるアプリケーションに向いています。

3. パーティション・キャッシュの実現

この章では、分散アルゴリズムを組み込んだ REST フル API クライアントを実装し、Tomcat にインストールしたキャッシュサーバと組み合わせることで、図 3 のような構成のパーティション・キャッシュを実現する方法について説明します。

図 3 キャッシュサーバと REST フル API クライアントによるパーティション・キャッシュ



なお、サンプルでは 1 台の Tomcat にキャッシュサーバを 2 つインストールしています。

3.1. キャッシュサーバのインストール

キャッシュサーバを動作させるには、以下のソフトウェアが必要です。

- Java
Java SE 5、または、6
- Tomcat 6
Glassfish V2/V3、Jetty 6 のいずれでも動作します。

今回は、Tomcat へのインストールについて説明しますが、他のアプリケーションサーバでも同様に簡単にインストールすることができます。

キャッシュサーバを以下のサイトからダウンロードします。

<http://ehcache.sourceforge.net/>

今回は、ehcache-server-0.5-distribution.tar.gz というパッケージを取得しました。

で取得した ehcache-server-0.5-distribution.tar.gz を解凍します。

で取得した ehcache-server-0.5.war を Tomcat に配備します。

3.2. キャッシュサーバの設定

キャッシュサーバの設定を行うには、ehcache-server-0.5.war を展開してきたディレクトリの WEB-INF/web.xml ファイルを編集します。

もし、REST フル API が必要ない場合は、REST サブレットに関する記述をコメントアウトします。また、SOAP API が必要ない場合は、SOAP サブレットに関する記述をコメントアウトします。

➤ REST フル API の web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5">

  <!--REST Servlet-->
  <!-- Comment out to disable RESTful web services-->
  <servlet>
    <servlet-name>Ehcache RESTful Web Service</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>com.sun.jersey.config.property.resourceConfigClass</param-name>
      <param-value>com.sun.jersey.api.core.PackagesResourceConfig</param-value>
    </init-param>
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>net.sf.ehcache.server.rest.resources</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
```

```
</servlet>
<servlet-mapping>
  <servlet-name>Ehcache RESTful Web Service</servlet-name>
  <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
<listener>
  <listener-class>net.sf.ehcache.server.ServerContext</listener-class>
</listener>
</web-app>
```

➤ SOAP API の web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5">

  <!--
  SOAP Servlet
  Comment out to disable SOAP Web Services
  -->

  <servlet>
    <servlet-name>EhcacheWebServiceEndpoint</servlet-name>
    <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>EhcacheWebServiceEndpoint</servlet-name>
    <url-pattern>/soap/EhcacheWebServiceEndpoint</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
```

```
</session-config>
<listener>

<listener-class>com.sun.xml.ws.transport.http.servlet.WSServletContextListener</listener-class>
</listener>

<listener>
  <listener-class>net.sf.ehcache.server.ServerContext</listener-class>
</listener>
</web-app>
```

3.3. キャッシュの設定

キャッシュの設定を行うには、ehcache-server-0.5.war を展開してきたディレクトリのWEB-INF/classes/ehcache.xml ファイルを編集します。

以下は、インストール直後の ehcache.xml です（コメントは除外しました）。サンプルとして様々なバリエーションに対応した設定が記述されています。

```
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="ehcache.xsd">
  <diskStore path="java.io.tmpdir"/>

  <cacheManagerEventListenerFactory class="" properties=""/>

  <cacheManagerPeerProviderFactory
    class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
    properties="peerDiscovery=automatic,
      multicastGroupAddress=230.0.0.1,
      multicastGroupPort=4446, timeToLive=1"
    propertySeparator=","
  />

  <cacheManagerPeerListenerFactory
    class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"/>
```

```
<defaultCache
    maxElementsInMemory="20000"
    eternal="false"
    timeToIdleSeconds="300"
    timeToLiveSeconds="300"
    overflowToDisk="true"
    diskSpoolBufferSizeMB="50"
    maxElementsOnDisk="10000000"
    diskPersistent="false"
    diskExpiryThreadIntervalSeconds="120"
    memoryStoreEvictionPolicy="LRU"
/>

<cache name="sampleCache1"
    maxElementsInMemory="10000"
    maxElementsOnDisk="1000"
    eternal="false"
    overflowToDisk="true"
    diskSpoolBufferSizeMB="20"
    timeToIdleSeconds="300"
    timeToLiveSeconds="600"
    memoryStoreEvictionPolicy="LFU"
/>

<cache name="sampleCache2"
    maxElementsInMemory="1000"
    eternal="true"
    overflowToDisk="false"
    memoryStoreEvictionPolicy="FIFO"
/>

<cache name="sampleCache3"
    maxElementsInMemory="500"
    eternal="false"
    overflowToDisk="true"
    timeToLiveSeconds="1"
```

```
        diskPersistent="true"
        diskExpiryThreadIntervalSeconds="1"
        memoryStoreEvictionPolicy="LFU"
    />

    <cache name="sampleDistributedCache1"
        maxElementsInMemory="10"
        eternal="false"
        timeToIdleSeconds="100"
        timeToLiveSeconds="100"
        overflowToDisk="false">
        <cacheEventListenerFactory
            class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"/>
        <bootstrapCacheLoaderFactory
            class="net.sf.ehcache.distribution.RMIBootstrapCacheLoaderFactory"/>
    </cache>

    <cache name="sampleDistributedCache2"
        maxElementsInMemory="10"
        eternal="false"
        timeToIdleSeconds="100"
        timeToLiveSeconds="100"
        overflowToDisk="false">
        <cacheEventListenerFactory
            class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
            properties="replicateAsynchronously=false, replicatePuts=false,
                replicateUpdates=true, replicateUpdatesViaCopy=true,
                replicateRemovals=false"/>
    </cache>

    <cache name="sampleDistributedCache3"
        maxElementsInMemory="10"
        eternal="false"
        timeToIdleSeconds="100"
        timeToLiveSeconds="100"
        overflowToDisk="false">
```

```

<cacheEventListenerFactory
    class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
    properties="asynchronousReplicationIntervalMillis=200"/>
</cache>
</ehcache>

```

以下に ehcache.xml で定義する主要なタグについて説明します。

➤ diskStore タグ

ディスクストアを使用する際にデータファイルが保存されるディレクトリを設定します。

path 属性の値	説明
user.home	ユーザのホームディレクトリ
user.dir	ユーザのカレントディレクトリ
java.io.tmpdir	デフォルトの一時ディレクトリ

➤ defaultCache/cache タグ

defaultCache タグ、cache タグともにキャッシュの設定を行います。

defaultCache タグは、cache タグで設定されていないキャッシュを作成したときに適用される設定です。

属性名	必須	説明
name		キャッシュを一意に判別する名前。defaultCache タグにはありません。
maxElementsInMemory		メモリストアにキャッシュ可能なオブジェクト数。
maxElementsOnDisk		ディスクストアにキャッシュ可能なオブジェクト数。デフォルトは0で、無制限を表します。
eternal		true である場合、オブジェクトのタイムアウトの設定に関係なく無制限にキャッシュします。デフォルトは false です。
overflowToDisk		true である場合、キャッシュしたオブジェクトの数が maxElementsInMemory で設定した数を越えた場合にディスクストアにキャッシュします。デフォルトは false です。

timeToIdleSeconds		設定された時間 (秒) にオブジェクトへのアクセスがなかった場合、キャッシュから削除されます。デフォルトは 0 で、無制限を表します。
timeToLiveSeconds		オブジェクトがキャッシュされている時間 (秒)。デフォルトは 0 で、無制限を表します。
diskPersistent		true の場合、VM が再起動している間、キャッシュをディスクに永続化します。デフォルトは false です。
diskExpiryThreadIntervalSeconds		ディスクへのアクセスするスレッドが起動している時間 (秒)。デフォルトは 120 です。
diskSpoolBufferSizeMB		ディスクへのアクセスに使用するスプールバッファのサイズ (MB)。デフォルトは 30 です。
memoryStoreEvictionPolicy		maxElementsInMemory に達した際にキャッシュからオブジェクトを削除するポリシー。設定可能なポリシーは、LRU、FIFO、LFU で、デフォルトは LRU です。

3.4. REST フル API

REST フル API は、URI と HTTP メソッドによりキャッシュサーバを操作します。URI は以下ようになります。

`http://{ドメイン}/{コンテキスト}/rest/{キャッシュ名}/{キー}`

各操作 API は、下表のようになります。(URI は、「`http://{ドメイン}/{コンテキスト}/rest`」を省略しています。)

URI	HTTP メソッド	説明
/	OPTIONS	この URI で使用できる HTTP メソッドの WADL で取得します。
	GET	キャッシュのリストを取得します。
{キャッシュ名}	OPTIONS	この URI で使用できる HTTP メソッドの WADL で取得します。
	HEAD	指定したキャッシュがあるか確認します。
	GET	指定したキャッシュのメタデータを取得します。

	PUT	ehcache.xml の defaultCache タグの設定でキャッシュを作成します。
	DELETE	指定したキャッシュを削除します。
/{キャッシュ名}/{キー}	OPTIONS	この URI で使用できる HTTP メソッドの WADL で取得します。
	HEAD	指定したキーがあるか確認します。
	GET	指定したキーのオブジェクトを取得します。
	PUT	指定したキーに本文にセットしたオブジェクトを格納します。
	DELETE	指定したキーのオブジェクトを削除します。

3.5. クライアントの実装

クライアントのサンプルのソースを以下のリンクより取得してください。

<サンプルダウンロード>

<http://www.bbreak.co.jp/technique/doc/ehcache/EhcacheJavaClient.zip>

EhcacheJavaClient クラスでは、キャッシュサーバへ格納する put メソッドとキャッシュサーバから取得する get メソッドを実装しました。

EhcacheJavaClient クラスの 43～53 行目のように、コネクションの Content-Type リクエストプロパティに「application/x-java-serialized-object」をセットし、キャッシュ対象のオブジェクトをシリアライズすることによって Java オブジェクトをキャッシュすることができます。

```

43 URL url = new URL(server);
44 connection = (URLConnection) url.openConnection();
45 connection.setRequestProperty("Content-Type",
46     "application/x-java-serialized-object");
47 connection.setDoOutput(true);
48 connection.setRequestMethod("PUT");
49 connection.connect();
50
51 out = new ObjectOutputStream(connection.getOutputStream());
52 out.writeObject(value);
53 out.flush();

```

キャッシュ対象の Java オブジェクトは、シリアライズする必要があるため、`java.io.Serializable` をインプリメントしている必要があります。

また、`EhcacheJavaClient` クラスの 122 ~ 126 行目の `selectServer` メソッドに分散アルゴリズムを実装しています。

```
122 private String selectServer(String key) {  
123     int hash = Math.abs(key.hashCode());  
124     int index = hash % urls.length;  
125     return urls[index];  
126 }
```

今回実装した分散アルゴリズムは、キーのハッシュコードをサーバの数で割った剰余によりサーバを決定しています。`EhcacheJavaClient` クラスを利用する `EhcacheJavaClientSample` クラスを実行した結果は以下のようになります。

```
PUT:http://localhost:8080/ehcache1/rest/sampleCache/1  
PUT:http://localhost:8080/ehcache2/rest/sampleCache/2  
GET:http://localhost:8080/ehcache1/rest/sampleCache/1  
GET:http://localhost:8080/ehcache2/rest/sampleCache/2  
result1:[name=name1, value=value1]  
result2:[name=name2, value=value2]
```

1 行目と 2 行目を見ると、キーが 1 の場合は `ehcache1` に、2 の場合は `ehcache2` に格納され、データが分散されていることが確認できます。

この剰余を使った分散アルゴリズムは、キャッシュサーバを追加や削除すると剰余が以前とはまったく異なったものになるためヒット率が低下するという欠点があります。この欠点を補うアルゴリズムに `Consistent Hashing` がありますので、ご興味がある方は実装されてはいかがでしょうか。

4. まとめ

分散キャッシュは、更新頻度が多いデータをキャッシュしてしまうとキャッシュの更新頻度が増加し、パフォーマンスを悪化させる可能性があるので適材適所に使用することが重要です。

分散キャッシュは、Ehcache 以外にも多く存在します。オープンソースでは大規模な Web サイト（国内では mixi の事例が有名）で実績があり様々な派生が登場している「memcached」、Java で実装されているオープンソースでは「Apache Jakarta JCS (Java Caching System)」や「JBoss Cache」、商用製品では「Oracle Coherence」やまだ開発中でリリースされていませんが「Microsoft Velocity」があります。また、米 Amazon.com の「Amazon S3」のようなクラウドストレージの基盤技術として分散キャッシュが使われています。

今後、ますますアプリケーションが扱うデータ量は増加すると予想されますが、データ量が増えてもスピードは落としたいという要求はますます強くなると思います。このような要求に応えるソリューションとして、Ehcache のようなデータ量に合わせてスケールアウトすることができるパーティション・キャッシュを利用する機会が増えていくと考えられます。

開発部

大森 洋行

山野 信行