キャッシングを使って高速な Web アプリケーションを作成する

頻繁に表示するデータを Java Caching System を使ってキャッシュする

Kellen F. Bombardier
Software Engineer
IBM

2008年 12月 02日

キャッシュ・ユーティリティーを使用すると、Java™ 技術を使用する Web アプリケーションのパフォーマンスを即座に改善することができます。Java アプリケーションのための強力な分散型キャッシング・システムである JCS (Java Caching System) は、単純な API を持ち、非常に柔軟な構成が可能なツールです。この記事では、JCS の概要と、JCS を使って Web アプリケーションを高速化する方法について説明します。

多くの Web アプリケーションは、デスクトップ・アプリケーションを作成し直すことで作成されています。理想的には、Web アプリケーションはデスクトップ・アプリケーションと同じように高速でスケーラブルでなければなりません。大幅な高速化をすることで、ほとんどすべての Web アプリケーションにはメリットがもたらされます。例えば、あまり変更されず頻繁に表示されるデータをキャッシングすると、ユーザーの待ち時間を短縮する非常に効果的な方法になります。また、単純で手軽な API を使ってデータのキャッシングを容易に行えるユーティリティーは高速化の実現に役立ちます。Apache Jakarta プロジェクトの 1 つであるオープンソースの JCS は、そうしたツールの 1 つです。この記事では、Web アプリケーション用のデータをキャッシングするための JCS の構成方法と使用方法を説明します。

JCS の概要

JCS は Java 言語で作成されたキャッシング・システムであり、Java によるデスクトップ・アプリケーションや Web アプリケーションの作成に使用することができます。JCS には、キャッシュへのデータの保存、キャッシュからのデータの取得、キャッシュからのデータの削除、等々のための便利なメカニズムが用意されています。

JCS を使用すると、さまざまな指定データ領域にキャッシュ・データを保存することができます。JCS では、メモリー、ディスク、ラテラル、そしてリモートという 4 つのタイプのコア領域が定義されています。これらのコア領域を組み合わせて使用することで、キャッシュ・データを保存するための方法と場所を非常に柔軟に指定することができます。JCS では、どの領域を最初に使用し、どういう場合に別の領域にフェールオーバーするのかを指定できるのです。

メモリー領域

メモリー領域は純粋にメモリーを利用したキャッシュ領域であり、LRU (Least Recently Used) アルゴリズムを使用します。メモリーのキャッシュ領域が一杯になると、LRU アルゴリズムによって、使われた時刻が最も古いキャッシュ・データが最初に削除されます。メモリーを利用したデータ領域のパフォーマンスは高く、JCS を使う人の大部分は最初に使用するデフォルトのキャッシュ領域としてメモリーを指定します。

JCS のプラガブル・コントローラー

JCS でキャッシュの保存先として複数の領域を簡単に使用できるのは、複合キャッシュ (Composite Cache) のおかげです。複合キャッシュには各キャッシュ領域用にプラガブル・コントローラーが用意されています。複合キャッシュは、各キャッシュ領域をいつどのように使用するかを判断する難しい作業を行ってくれます。開発者はキャッシュとのデータの出し入れのみを気にすればよく、面倒な作業の大部分は JCS が行ってくれるのです。

ディスク領域

ディスク領域は Web サーバーのファイル・ディスク上にあるキャッシュ・データ領域です。JCS ではパフォーマンスを改善するために、キャッシュ・データのキーをメモリーに保存し、実際のキャッシュ・データをファイル・ディスクに保存します。JCS の典型的な構成では、最初にメモリー領域を使用してから、メモリー領域に収まらないデータをディスク領域に書き込みます。

ラテラル領域

ラテラル・キャッシュ領域は、複数のサーバーにキャッシュ・データを分散することができる柔軟な構成になっています。キャッシュ・データ・サーバーはリッスン用のポートを開いておく必要があり、またソケット接続を作成する必要があります。この領域ではキャッシュ間でのデータの一貫性が保証されないため、問題が起こる可能性があります。しかしこの領域が計画的に使われる場合には、その問題が起こる可能性は低くなります。

リモート領域

リモート領域は RMI (Remote Method Invocation) API を使ってキャッシュ領域を提供します。この領域ではリモート・サーバーを使ってキャッシュ・データが処理されます。また、キャッシュ・データを保存するために、複数の JCS クライアント・アプリケーションでリモートのキャッシュ・サーバーを使用することができます。リスナーはクライアントとサーバーから、リクエストを収集するように指定されます。このキャッシュ領域は、シリアライズによるオーバーヘッドや接続点が複数あることによるオーバーヘッドの軽減に役立ちます。

JCS の構成

JCS の構成を行うために必要なことは、cache.ccf ファイルを作成してそこにデータを追加することだけです。このファイルは、キャッシュに使用される領域と、それらのキャッシュ領域の属性やオプションを指定します。アプリケーションの要求に合わせてこのファイルを調整すれば、キャッシュのスケールを簡単に変更することができて便利です。以下のサンプルは構成の主要なポイントを示すために、できるだけ簡単にしてあります。いくつかのオプションや属性を指定することで、皆さんの要求に最も適した構成にすることができます。

リスト 1 は、純粋にメモリーを利用してキャッシュを構成した、最も基本的な cache.ccf ファイルを示しています。

リスト 1. JCS の基本的な構成

```
jcs.default=jcs.default.cacheattributes=org.apache.jcs.engine.CompositeCacheAttributes
jcs.default.cacheattributes.MaxObjects=1000
jcs.default.cacheattributes.MemoryCacheName=
 org.apache.jcs.engine.memory.lru.LRUMemoryCache
```

リスト 1 の最後の行を見ると、この構成ファイルはメモリー・キャッシュを `LRUMemoryCache` と指定していることがわかります。また、メモリーの中に保持されるオブジェクトの最大数が `1000` に設定されていることもわかります。

大部分のアプリケーションのキャッシング・システムの構成は、リスト 1 よりも複雑なはずです。リスト 2 の構成では、メモリー領域とディスク領域を使用する一方、独自の領域 (`OUR_REGION`) を定義しています。

リスト 2. JCS の構成の中で定義される領域

```
jcs.default=DISK_REGION
jcs.default.cacheattributes=org.apache.jcs.engine.CompositeCacheAttributes
jcs.default.cacheattributes.MaxObjects=1000
jcs.default.cacheattributes.MemoryCacheName=
  org.apache.jcs.engine.memory.lru.LRUMemoryCache

jcs.region.OUR_REGION=DISK_REGION
jcs.region.OUR_REGION.cacheattributes=org.apache.jcs.engine.CompositeCacheAttributes
jcs.region.OUR_REGION.cacheattributes.MaxObjects=1000
jcs.region.OUR_REGION.cacheattributes.MemoryCacheName=
  org.apache.jcs.engine.memory.lru.LRUMemoryCache
jcs.region.OUR_REGION.cacheattributes.UseMemoryShrinker=true
jcs.region.OUR_REGION.cacheattributes.MaxMemoryIdleTimeSeconds=3600
jcs.region.OUR_REGION.cacheattributes.ShrinkerIntervalSeconds=60
jcs.region.OUR_REGION.cacheattributes.MaxSpoolPerRun=500
jcs.region.OUR_REGION.elementattributes=org.apache.jcs.engine.ElementAttributes
jcs.region.OUR_REGION.elementattributes.IsEternal=false

jcs.auxiliary.DISK_REGION=org.apache.jcs.auxiliary.disk.indexed.IndexedDiskCacheFactory
jcs.auxiliary.DISK_REGION.attributes=
  org.apache.jcs.auxiliary.disk.indexed.IndexedDiskCacheAttributes
jcs.auxiliary.DISK_REGION.attributes.DiskPath=c:/jcs/disk_region
jcs.auxiliary.DISK_REGION.attributes.maxKeySize=100000
```

JCS の補助領域

JCS には、中心となる 4 つのキャッシュ実装に加えて、補助領域が用意されています。補助領域は、この 4 つのキャッシュ領域が使用できるオプションのプラグインで、インデックス付きディスク・キャッシュ、TCP ラテラル・キャッシュ、そしてリモート・キャッシュ・サーバーの 3 種類があります。例えばディスク・キャッシュを使うと、メモリー領域がしきい値に達した場合に、キャッシュ·データをディスク上にスワップすることができます。このように各キャッシュ領域ではキャッシングを柔軟にコントロールできるようになるため、大部分のオペレーティング・システムで使用されている仮想メモリーと似たような動作をさせることができます。各補助領域をアプリケーションの要求に合わせて調整するには、cache.ccf 構成ファイルを使用します。

リスト 2 の最初の行を見ると、この構成ではデフォルト領域を `DISK_REGION` に設定していることがわかります。`DISK_REGION` は `IndexedDiskCacheFactory` 型であり、ディスク上では c:\jcs\disk_region というファイルとして存在します。リスト 2 の 2 番目の構成グループは、私の独自領域を定義しており、オプションをいくつか追加しています。このタイプの構成 (つまりメモリー

領域とディスク領域の両方を使用する一方で、ユーザー定義領域を指定する構成) は一般的です。リスト 2 の 3 番目の構成グループは「補助領域」を定義しています。

JCS には、`concurrent` と `commons-logging` という 2 つの依存関係があります (1.2.7.0 より前のバージョンの JCS には、`commons-collections` と `commons-lang` という、さらに 2 つの依存関係がありました)。

JCS の基本的な使い方

JCS の基本を学ぶ良い方法は、JCS の API の中で最もよく使われるメソッドを調べてみることです。そのための出発点としては、キャッシュ領域の初期化動作を調べてみるのが最適です。JCS キャッシュ領域オブジェクトの初期化動作では、必要となる一般的なメソッドのほとんどすべてを使用します。リスト 3 は JCS オブジェクトを初期化し、そしてデフォルトのキャッシュ領域のインスタンスを取得するコードです。

リスト 3. デフォルトのキャッシュ領域を取得する

```
// Initialize the JCS object and get an instance of the default cache region
JCS cache = JCS.getInstance("default");
```

JCS のインスタンスを取得できると、必要なメソッドの大部分を呼び出すことができます。`put` メソッドは新しいオブジェクトをキャッシュの中に配置します。そのために必要なものは `key` (1 番目のパラメーター) と `value` (2 番目のパラメーター) のみです。リスト 4 はこの基本的な例を示しています。

リスト 4. キャッシュ対象項目を設定する

```
// Set up
String key = "key0";
String value = "value0";

// Place a new object in the cache
cache.put(key, value);
```

リスト 4 の例ではストリング値をパラメーターに使っていますが、どんなオブジェクトを使用することも可能です。

キャッシュされたオブジェクト取得するためには、JCS に用意されている `get` メソッドを使用すればよいだけです。リスト 5 はその簡単な例を示しています。この例の場合もストリング・パラメーターを使っていますが、やはりどんなオブジェクトを使用することも可能です。

リスト 5. キャッシュされたオブジェクト取得する

```
// Retrieve a cached object
String cachedData = (String)cache.get(key);
```

キャッシング・システムを扱う際には、キャッシュ・データが有効かどうかをテストするためのメソッドも必要です。JCS では、キャッシュされたデータが存在しているかどうかをテストするためだけのメソッドは定義されていませんが、代わりに `get` メソッドの戻り値が役立ちます。リスト 6 はこの必要な機能を実現するためのコードを示しています。

リスト 6. キャッシュされた項目が有効かどうかをテストする

```
// Retrieve a cached object
String cachedData = (String)cache.get(key);

// Check if the retrieval worked
if (cachedData != null) {
  // The cachedData is valid and can be used
  System.out.println("Valid cached Data: " + cachedData);
}
```

一般的なキャッシュ・ユーティリティーとして必要なものの最後は、JCS やキャッシュされた項目、そしてキャッシュ領域を使い終わった後にクリーンアップするためのユーティリティーです。JCS には、呼び出されたキャッシュ領域からキャッシュ・データをすべて削除するための `clear` メソッドと、特定のキャッシュ項目を削除する `remove` メソッドが用意されています。また、初期化された JCS 領域を破棄する `dispose` メソッドもあります。リスト 7 はこれらのメソッドの使い方を示しています。

リスト 7. キャッシュ領域をクリーンアップする

```
// Dispose of a specific cached item
cache.remove(key);

// Dispose of all cache data
cache.clear();

// Dispose of the cache region
cache.dispose();
```

JCS と Java オブジェクト

他のキャッシング・システム (「参考文献」を参照) よりも JCS が優れている点の 1 つが、JCS ではオブジェクトをうまく扱える点です。Java 技術を使って作成された Web アプリケーションの大部分ではオブジェクト指向の方法が使われています。例えばオブジェクトをキャッシングすると、データベースからオブジェクトの各部分を連続的に取得する方法よりもアプリケーションのパフォーマンスはずっと良くなります。

JCS を利用して単純なオブジェクト指向サイトを設計するための最初のステップとして、保存が必要なオブジェクトから作業を始めます。次の例では基本的なブログ・サイトを作成するため、ブログ・オブジェクトそのものを保存します。リスト 8 は、ここで使用する `BlogObject` クラスを示しています。

リスト 8. `BlogObject` クラス

```
package com.ibm.developerWorks.objects;

import java.io.Serializable;
import java.util.Date;

public class BlogObject implements Serializable {
  private static final long serialVersionUID = 6392376146163510046L;
  private int blogId;
  private String author;
  private Date date;
```

```
  private String title;
  private String content;

  public BlogObject(int blogId, String author, Date date, String title, String content) {
    this.blogId = blogId;
    this.author = author;
    this.date = date;
    this.title = title;
    this.content = content;
  }

  public int getBlogId() {
    return this.blogId;
  }

  public String getAuthor() {
    return this.author;
  }

  public Date getDate() {
    return this.date;
  }

  public String getTitle() {
    return this.title;
  }

  public String getContent() {
    return this.content;
  }
}
```

クラスの中でオブジェクトを表現できると、次にそのオブジェクトを扱うクラスが必要です。ブログ・マネージャーはブログ・オブジェクトに関するすべての管理側面とキャッシュ機能を処理します。この例では、ブログ・マネージャーは次の 3 つの主要なタスクを処理します。

- ブログ・オブジェクトを取得する
- キャッシュにブログ・オブジェクトを入れる
- キャッシュからブログ・オブジェクトを削除する

リスト 9 に示す `getBlog` メソッドはブログ・オブジェクトを取得します。このメソッドはまず、キャッシュからブログ・オブジェクトを取得しようと試みます。キャッシュの中にブログ・オブジェクトがない場合には、次のように別のメカニズムからブログ・オブジェクトを取得します。

リスト 9. ブログ・マネージャーからブログ・オブジェクトを取得する

```
public BlogObject getBlog(int id) {
  BlogObject blog = null;

  try {
    blogCache = JCS.getInstance(blogCacheRegion);
    blog = (BlogObject)blogCache.get(id);
  } catch (CacheException ce) {
    blog = null;
  }

  if (blog == null) {
    blog = DatabaseManager.getBlog(id);
    this.setBlog(
      blog.getBlogId(),
      blog.getAuthor(),
```

```
      blog.getDate(),
      blog.getTitle(),
      blog.getContent()
    );
  }

  return blog;
}
```

リスト 9 では、オブジェクトを取得するための別の方法としてデータベースを使用しています。別のメカニズムからオブジェクトを取得する場合には、そのオブジェクトをキャッシュに入れ、次の取得動作でキャッシュから直接取得できるようにする必要があります。

リスト 10 に示す `setBlog` メソッドはブログ・オブジェクトをキャッシュに配置します。このメソッドは渡された情報を持つ新しいブログ・オブジェクトを作成してキャッシュに配置するだけなので、単純です。

リスト 10. ブログ・マネージャーを使ってブログ・オブジェクトをキャッシュに配置する

```
public boolean setBlog(int bId, String author, Date date, String title, String content) {
  BlogObject blog = new BlogObject(bId, author, date, title, content);

  try {
    blogCache = JCS.getInstance(blogCacheRegion);
    blogCache.put(bId, blog);
    return true;
  } catch (CacheException ce) {
    return false;
  }
}
```

リスト 11 に示す `cleanBlog` メソッドは、指定された 1 つのブログ、またはすべてのブログをキャッシュから削除します。このメソッドは JCS の `remove` メソッドと `clear` メソッドを使ってキャッシュ・オブジェクトをクリーンアップします。

リスト 11. ブログ・マネージャーを使ってキャッシュからブログ・オブジェクトを削除する

```
public boolean cleanBlog(int blogId) {
  try {
    blogCache = JCS.getInstance(blogCacheRegion);
    blogCache.remove(blogId);
  } catch (CacheException ce) {
    return false;
  }
  return true;
}

public boolean cleanBlog() {
  try {
    blogCache = JCS.getInstance(blogCacheRegion);
    blogCache.clear();
  } catch (CacheException ce) {
    return false;
  }
  return true;
}
```

上記のさまざまなクラスを見ると、JCS を使うことで非常に簡単にオブジェクトをキャッシングできることがわかります。オブジェクトと、そのオブジェクトのマネージャーを単純に表現するだけで、Web アプリケーションのオブジェクトを強力かつ簡単な方法で処理することができるのです。

メタデータのキャッシュ

JCS には、アプリケーションにキャッシュ機能を追加するための方法として、ここまでに説明してきた基本的なもの以外にも、はるかに多くのものが用意されています。例えば JCS には、キャッシュ・オブジェクトやキャッシュ領域のメタデータを収集するためのユーティリティーが用意されています。このユーティリティーを使用すると、以下のものを容易に取得することができます。

- キャッシュ・キーの名前
- キャッシュ項目が作成された時刻
- キャッシュの最大存続時間
- キャッシュ項目が失効するまでの時間

リスト 12 の例は、キャッシュ項目のメタデータを取得する方法を示しています。

リスト 12. キャッシュ項目のメタデータを取得する

```
try {
  JCSAdminBean admin = new JCSAdminBean();
  LinkedList linkedList = admin.buildElementInfo(regionName);
  ListIterator iterator = linkedList.listIterator();

  while (iterator.hasNext()) {
    CacheElementInfo info = (CacheElementInfo)iterator.next();
    System.out.println("Key: " + info.getKey());
    System.out.println("Creation Time: " + info.getCreateTime());
    System.out.println("Maximum Life (seconds): " + info.getMaxLifeSeconds());
    System.out.println("Expires in (seconds): " + info.getExpiresInSeconds());
  }
} catch (Exception e) {
}
```

キャッシュされた項目のメタデータは有用なものですが、このメタデータは各キャッシュ領域のメタデータを取得する際にも役立ちます。この情報を利用することで、どの領域にどれほどの量のキャッシュ・データが配置されるのかを (キャッシュ・ミス、キャッシュ・ヒット、キャッシュの更新などを含めて) 知ることができます。リスト 13 の例は、この情報を収集するための方法を示しています。

リスト 13. キャッシュ領域のメタデータを取得する

```
try {
  JCSAdminBean admin = new JCSAdminBean();
  LinkedList linkedList = admin.buildCacheInfo();
  ListIterator iterator = linkedList.listIterator();

  while (iterator.hasNext()) {
    CacheRegionInfo info = (CacheRegionInfo)iterator.next();
    CompositeCache compCache = info.getCache();
    System.out.println("Cache Name: " + compCache.getCacheName());
    System.out.println("Cache Type: " + compCache.getCacheType());
    System.out.println("Cache Misses (not found): " + compCache.getMissCountNotFound());
    System.out.println("Cache Misses (expired): " + compCache.getMissCountExpired());
    System.out.println("Cache Hits (memory): " + compCache.getHitCountRam());
    System.out.println("Cache Updates: " + compCache.getUpdateCount());
  }
} catch (Exception e) {
}
```

キャッシュ領域やキャッシュ項目のメタデータを収集すると、Web サイトのどの領域やどの項目に最適化が必要かを分析する上で役立ちます。またメタデータは、正確な時間管理が必要なキャッシュ・データを管理する上でも役立ちます。例えば各キャッシュ項目の最大存続時間と失効時間を利用すると、更新されたデータを必要とする特定のユーザーに対してキャッシュ・データを最新のものに更新することができます。

まとめ

JCS は Java 開発者のための、強力で簡単に使用できるキャッシング・システムです。JCS はデスクトップ・アプリケーションにも Web アプリケーションにも同様にデータ・キャッシング機能を提供することができます。デスクトップ風の Web アプリケーションが増えるにつれ、Web アプリケーションにはより一層速さとアジリティーが求められるようになっています。そのためにはデータをキャッシングすることが役立ちます。この記事では、JCS の構成方法と使い方の概要を説明しました。また、基本的なキャッシング方法に必要な構文や、一般的な Web アプリケーションに使われるオブジェクトのキャッシング方法、そしてキャッシュのメタデータの取得方法も説明しました。これで皆さんも JCS の基本知識を身に付けたことになり、データ・キャッシング機能の強力さを利用した次期 Web サイトの開発を即座に開始できるはずです。また、JCS の持つさらなる側面として、例えば HTTP サーブレットへのアクセスや JCS ユーティリティー、基本的な HTTP 認証、その他の補助領域といった、高度な機能を提供できる側面についても調べてみるとよいかもしれません。

著者について

Kellen F. Bombardier

Kellen Bombardier は IBM のソフトウェア・エンジニアであり、また情報管理チームの開発者です。