memcached と Grails: 第 1 回 memcached をインストールして使用する

memcached コマンドを学び、キャッシュのパフォーマンスを評価する

James G. Goodwill

2009年 9月 15日

memcached と Grails に焦点を当てるこの 2 回連載の第 1 回目では、著者の James Goodwill がオープンソースのキャッシング・ソリューション、memcached を紹介します。今回の記事で取り上げる内容は、memcached のインストール、構成、memcached クライアント・コマンド、そしてキャッシュの有効性を評価する方法です。この記事では、memcached を各言語に固有のクライアントで使用する方法について説明するのではなく、memcached サーバーとの間で行われる直接のやりとりに焦点を絞ります。今回目標とするのは、memcached のインスタンスをモニターするために必要なツールについて説明すること、そして第 2 回の記事で memcached をGrails アプリケーションに統合するための下準備をすることです。

このシリーズの他の記事を見る

memcached は、もともとは Danga Interactive 社によって開発された汎用の分散型メモリー・キャッシング・システムですが、現在は BSD ライセンスの下で配布されています。

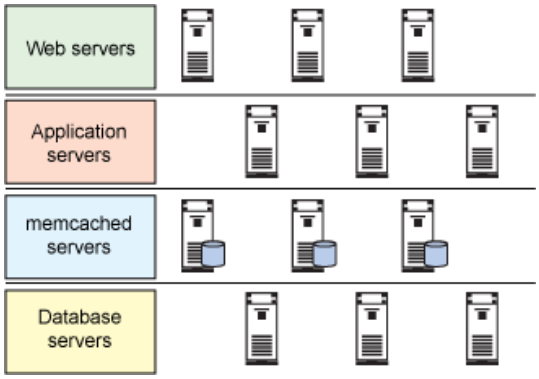
Danga Interactive 社が memcached を開発した理由は、同社が運営する Web サイト LiveJournal.com には大量のトラフィックがあるため、これに対処可能なメモリー・キャッシング・システムがどうしても必要だったからです。1 日 2000 万を超えるページ・ビューにより、LiveJournal のデータベースには莫大な負荷がかかっていました。そこで Danga Interactive 社の Brad Fitzpatrick によって考案されたのが、memcached です。memcached はこのサイトのデータベース負荷を削減しただけでなく、今では大量のトラフィックを扱う世界中の Web サイトの多くで使用されているキャッシング・ソリューションとなっています。

この記事では最初に memcached の概要を説明した後、memcached をインストールするプロセスと、開発者それぞれの環境でこのキャッシング・システムをソースからビルドするプロセスを手順に沿って紹介します。また、memcached クライアント・コマンド (全部で 9 つ) を紹介し、memcached の基本操作、および高度な操作でそれぞれのクライアント・コマンドを使用する方法を説明します。そして最後は memcached コマンドを使用してキャッシュのパフォーマンスと有効性を測定する上での秘訣をいくつか紹介して今回は締めくくりとします。

環境のなかで memcached をインストールするのに適した場所とは？

memcached をインストールして使用する手順に取り掛かる前に、お使いの環境のなかで memcached をインストールするのに適した場所について簡単に説明しておきます。memcached を使用する場所は何箇所でも構いませんが、私が思うに、memcached はデータベース層で長時間実行されている複数のクエリーがある場合に最も効果を発揮します。そのため私は大抵、一連の memcached インスタンスをデータベース・サーバーとアプリケーション・サーバーの間にセットアップし、単純なパターンに従ってサーバーのそれぞれに対して読み取りと書き込みを行うようにしています。私がこのようなアプリケーション・アーキテクチャーをどのようにセットアップしているかは、図 1 を見ればわかるはずです。

図 1. memcached を使用したアプリケーション・アーキテクチャーの例



このアーキテクチャーは簡単に理解できるはずです。まず、Apache インスタンスが含まれる Web 層があります。その下にある層はアプリケーションそのものです。アプリケーションは大抵の場合、Apache Tomcat やその他のオープンソースのアプリケーション・サーバー上で実行されることになります。memcached インスタンスを構成する場所はその下の層で、つまりアプリケーション・サーバーとデータベース・サーバーの間に位置します。このような構成を使用する場合、データベースに対する読み取り操作と書き込み操作は多少方法が異なってきます。

読み取り操作

読み取り操作を行う場合、まずは Web 層からリクエストを取得し (それにはデータベース・クエリーが必要です)、そのクエリーの結果がすでにキャッシュに保存されていないかどうかを調べます。探している値が見つかった場合には、その値を返します。値が見つからなければクエリーを実行し、その結果をキャッシュに保存してから Web 層に返します。

書き込み操作

データベースに書き込む際には、最初にデータベースへの書き込み操作を行い、続いてキャッシュに格納されている結果のなかに、この書き込み操作によって影響を受ける結果がある場合、その結果を無効にします。この手順により、キャッシュとデータベースの間でデータに矛盾が生じることがなくなります。

memcached のインストール方法

memcached は、Linux®、Windows®、Mac OS、Solaris をはじめとする、いくつかのオペレーティング・システムをサポートしますが、この記事ではソースから memcached をビルドしてインス

トールする手順を説明します。私がこの方法を選択する第 1 の理由は、ソースからビルドすれば、疑問が生じた場合にはソースを調べられるためです。

libevent

memcached をインストールするための前提条件は、memcached が依存する非同期イベント通知ライブラリー libevent だけです。libevent のソースは monkey.org で入手できるので、このサイトにアクセスして最新のソース・ディストリビューションを選択してください。この記事で使用するのは、安定版の 1.4.11 です。アーカイブ・ファイルを入手したら、適当な場所に解凍し、リスト 1 のコマンドを実行します。

リスト 1. libevent をビルドしてインストールする

```
cd libevent-1.4.11-stable/

./configure

make

make install
```

memcached

memcached のソースは Danga Interactive にあります。この場合も同じく、最新のディストリビューションを選択してください。この記事を執筆している時点での最新バージョンは 1.4.0 です。tar.gz を適当な場所に解凍したら、リスト 2 のコマンドを実行します。

リスト 2. memcached をビルドしてインストールする

```
cd memcached-1.4.0/

./configure

make

make install
```

以上の手順が完了すると、実際に動作する memcached がインストールされて使用できる状態になります。それでは早速、memcached を使ってみましょう。

memcached の使用方法

memcached を使用するには、まず memcached サーバーを起動し、それから telnet クライアントを使用して起動したサーバーに接続します。

リスト 3 のコマンドを実行すると、memcached が起動します。

リスト 3. memcached を起動する

```
./memcached -d -m 2048 -l 10.0.0.40 -p 11211
```

上記のコマンドによって、memcached はデーモン (`-d`) として起動されます。この場合、割り当てられるメモリーは 2GB (`-m 2048`) で、リッスンするのはローカル・ホスト (ポート 11211) で

す。これらの値は必要に応じて変更しますが、この演習ではこの設定で正常に動作します。次のステップは、memcached に接続することです。ここでは単純な telnet クライアントを使用して memcached サーバーに接続します。

ほとんどのオペレーティング・システムには telnet クライアントが組み込まれていますが、Windows ベースの OS を使用しているとしたら、サード・パーティーのクライアントをダウンロードする必要があります。私はクライアントに PuTTy を使用することをお勧めします。

telnet クライアントがインストールされたら、リスト 4 のコマンドを実行します。

リスト 4. memcached に接続する

```
telnet localhost 11211
```

すべて問題なければ、**ローカル・ホストに接続された**ことを示す telnet の応答が返ってきます。このレスポンスが表示されない場合は、前のステップに戻って libevent と memcached のソースがどちらも正常にビルドされていることを確認してください。

memcached サーバーにログインしたので、この時点から一連の単純なコマンドを使って memcached と通信できるようになります。クライアント・サイドの 9 つの memcached コマンドは、次の 3 つのグループに分類することができます。

- 基本コマンド
- 拡張コマンド
- 管理用コマンド

memcached の基本クライアント・コマンド

単純な操作には、5 つの基本的な memcached コマンドを使用します。これらのコマンドおよび操作は以下のとおりです。

- `set`
- `add`
- `replace`
- `get`
- `delete`

上記のコマンドのうち、最初の 3 つのコマンドは、memcached に保存されたキーと値のペアを操作するための標準的な変更コマンドです。いずれも至って単純なコマンドで、リスト 5 に記載する構文を使用します。

リスト 5. 変更コマンドの構文

```
command <key> <flags> <expiration time> <bytes>
<value>
```

表 1 に、memcached 変更コマンドのパラメーターとその用途を示します。

表 1. memcached 変更コマンドのパラメーター

パラメーター	用途
key	キャッシュに入れられた値を検索するためのキー
flags	クライアントがキーと値のペアに関連付けた数値情報として、キーと値のペアと一緒に保存できる整数パラメーター
expiration time	キーと値のペアをキャッシュに保持する秒単位の期間 (0 は永続的に保持されることを意味)
bytes	キャッシュに保存するバイト数
value	保存される値 (常に 2 行目に位置)

ここからは、それぞれのコマンドの動作を説明します。

set

`set`コマンドは、新しいキーと値のペアをキャッシュに追加します。そのキーがすでにキャッシュにある場合は、前の値が新しい値に置き換えられます。

以下の `set` コマンドによるクライアント・サーバー間の対話を見てください。

```
set userId 0 0 5
12345
STORED
```

`set` コマンドによってキーと値のペアが正しく設定された場合、サーバーからは STORED というレスポンスが返されます。上記の例では、`userId` というキーと値 `12345` のペアをキャッシュに追加しました。有効期限は 0 に設定されています。つまり memcached には、この値をユーザーが削除するまでキャッシュに保持するよう指示しているということです。

add

`add` コマンドは、該当するキーがキャッシュにない場合にのみ、新しいキーと値のペアをキャッシュに追加します。既に該当するキーがある場合は、前の値がそのまま残され、NOT_STORED というレスポンスが返されます。

以下は、`add` コマンドによるクライアント・サーバー間の典型的な対話です。

```
set userId 0 0 5
12345
STORED

add userId 0 0 5
55555
NOT_STORED

add companyId 0 0 3
564
STORED
```

replace

`replace` コマンドは、該当するキーがキャッシュ内にある場合にのみ、キーと値のペアを置き換えます。キャッシュ内に該当するキーがない場合は、memcached サーバーからは NOT_STORED というレスポンスが返されます。

以下は、replace コマンドによるクライアント・サーバー間の典型的な対話です。

```
replace accountId 0 0 5
67890
NOT_STORED

set accountId 0 0 5
67890
STORED

replace accountId 0 0 5
55555
STORED
```

最後に説明する基本コマンドは get と delete です。この 2 つのコマンドの実行内容はかなり明白で、どちらも以下に示すような構文を使用します。

```
command <key>
```

これらのコマンドを実際にどのように使用するかを見てみましょう。

get

get コマンドは、追加済みのキーと値のペアに関連付けられた値を取得するために使用します。ほとんどのデータ取得操作には、get を使用することになります。

以下は、get コマンドによるクライアント・サーバー間の典型的な対話です。

```
set userId 0 0 5
12345
STORED

get userId
VALUE userId 0 5
12345
END

get bob
END
```

ご覧のように、get コマンドは至って単純です。キーを設定して get を呼び出すと、そのキーがキャッシュ内に存在する場合には値が返されます。存在しない場合には、何も返されません。

delete

最後の基本コマンドは delete です。delete コマンドは、memcached に保存されている既存の値を削除するために使用します。キーを設定して delete を呼び出すと、そのキーがキャッシュ内に存在する場合には値が削除されます。存在しない場合には、NOT_FOUND というメッセージが返されます。

以下は、delete コマンドによるクライアント・サーバー間の対話です。

```
set userId 0 0 5
98765
STORED

delete bob
NOT_FOUND

delete userId
DELETED

get userId
END
```

memcached の拡張クライアント・コマンド

memcached で使用できる拡張コマンドには、`gets` と `cas` の 2 つがあります。`gets` コマンドと `cas` コマンドは併せて使用するように意図されています。既存の名前と値のペアを新しい値に設定する際にこの 2 つのコマンドを併せて使用することで、値が更新済みであれば、新しい値が設定されないようにすることができます。以下に、それぞれのコマンドについて順に説明します。

gets

`gets` コマンドの機能は、基本的な `get` コマンドとかなり似ています。この 2 つのコマンドの違いは、`gets` が返す情報には `get` が返す情報にさらなる情報が追加されている点です。この追加の情報である 64 ビットの整数は、いわば名前と値のペアの「バージョン」ID のような役割を持ちます。

以下は、`gets` コマンドによるクライアント・サーバー間の対話です。

```
set userId 0 0 5
12345
STORED

get userId
VALUE userId 0 5
12345
END

gets userId
VALUE userId 0 5 4
12345
END
```

ここで、`get` コマンドと `gets` コマンドの違いについて検討してみましょう。`gets` コマンドが返す追加情報の値 (上記の例では、整数値 4) は、名前と値のペアを識別します。この名前と値のペアに対してさらに `set` コマンドを実行すると、今度は `gets` によって返される追加情報の値が変更されています。つまり、名前と値のペアは更新されたということです。リスト 6 に、一例を記載します。

リスト 6. set によるバージョン指定子の更新

```
set userId 0 0 5
33333
STORED

gets userId
VALUE userId 0 5 5
33333
END
```

gets によって返された末尾の値に注目してください。値は 5 に更新されています。この値は、名前と値のペアに変更を加えるたびに変更されます。

cas

cas (check and set) は、gets コマンドを最後に実行してから名前と値のペアが更新されていなければ、名前と値のペアの値を設定するという重宝なコマンドです。このコマンドは set コマンドと同様の構文を使用しますが、値がもう 1 つ追加されます。それは、gets によって返された追加情報の値です。

以下の cas コマンドによるクライアント・サーバー間の対話を見てください。

```
set userId 0 0 5
55555
STORED

gets userId
VALUE userId 0 5 6
55555
END

cas userId 0 0 5 6
33333
STORED
```

上記を見るとわかるように、整数 6 が追加された gets コマンドを使用したこの操作は、問題なく完了しました。今度はリスト 7 の一連のコマンドを見てください。

リスト 7. 古いバージョン指定子を設定した cas コマンド

```
set userId 0 0 5
55555
STORED

gets userId
VALUE userId 0 5 8
55555
END

cas userId 0 0 5 6
33333
EXISTS
```

上記では、gets によって返された最新の整数を使用していません。そのため、cas コマンドは失敗し、EXISTS という値が返されています。要するに、gets コマンドと cas コマンドを併せて使用することで、最後に読み取った後に更新された名前と値のペアを「侵害」しないようにできるというわけです。

キャッシュ管理用コマンド

最後に紹介する 2 つの memcached コマンドは、memcached のインスタンスをモニターする場合、そしてクリーンアップする場合に使用します。この 2 つのコマンドとは、`stats` と `flush_all` です。

stats

`stats` コマンドが実行する内容はまさにその名前のとおり、接続している memcached インスタンスの現行の統計をダンプすることです。一例として、`stats` コマンドの実行結果として表示された現行の memcached インスタンスに関する情報を以下に記載します。

```
stats
STAT pid 63
STAT uptime 101758
STAT time 1248643186
STAT version 1.4.11
STAT pointer_size 32
STAT rusage_user 1.177192
STAT rusage_system 2.365370
STAT curr_items 2
STAT total_items 8
STAT bytes 119
STAT curr_connections 6
STAT total_connections 7
STAT connection_structures 7
STAT cmd_get 12
STAT cmd_set 12
STAT get_hits 12
STAT get_misses 0
STAT evictions 0
STAT bytes_read 471
STAT bytes_written 535
STAT limit_maxbytes 67108864
STAT threads 4
END
```

上記の出力のほとんどは一目瞭然の内容です。これらの値の意味は、次回の記事でキャッシュのパフォーマンスを取り上げる際に改めて詳しく説明します。今回は出力にざっと目をとおした後、新しいキーを設定した `set` コマンドをいくつか実行し、それから `stats` コマンドをもう一度実行して内容がどのように変更されるかを確認してください。

flush_all

`flush_all` は、今回の記事で学ぶ最後の memcached コマンドです。今まで紹介したなかで最も単純なこのコマンドは、すべての名前と値のペアをキャッシュからクリアするだけにすぎません。キャッシュをクリーンな状態にリセットする必要があるときには、この `flush_all` が大いに役立ちます。以下は、`flush_all` の使用例です。

```
set userId 0 0 5
55555
STORED

get userId
VALUE userId 0 5
55555
END

flush_all
OK

get userId
END
```

キャッシュのパフォーマンス

この記事の締めくくりとして、これから memcached の拡張コマンドを使用してキャッシュのパフォーマンスを調べる方法を説明します。キャッシュの使用率を調整するために欠かせないのは、`stats` コマンドです。注意していなければならない最も重要な統計には、get_hits と get_misses の 2 つがあります。これらの値によって、名前と値のペアが検出された回数 (get_hits) に対し、名前と値のペアが検出されなかった回数 (get_misses) を比較することができます。

この 2 つの値の組み合わせは、キャッシュがどれだけ有効に利用されているかを判断する手段となります。キャッシュを初めて起動した後に get_misses が増加するのは当然ですが、しばらく使用した後には get_misses の値が安定していなければなりません。この値が安定しているということは、キャッシュに最もよく使われる読み取り操作の結果がキャッシュに格納されていることを意味します。get_misses の値が急上昇している一方、get_hits が安定しているとしたら、キャッシュの内容を調べる必要があります。誤ったものをキャッシュしている可能性があるからです。

キャッシュの有効性を判断するもう 1 つの手段は、キャッシュのヒット率を調べることです。キャッシュのヒット率によって、`get` を実行した回数のうち、`get` が失敗した回数のパーセンテージがわかります。このパーセンテージを判断するには、`stats` コマンドをもう一度実行してください (リスト 8 を参照)。

リスト 8. キャッシュのヒット率を計算する

```
stats
STAT pid 6825
STAT uptime 540692
STAT time 1249252262
STAT version 1.2.6
STAT pointer_size 32
STAT rusage_user 0.056003
STAT rusage_system 0.180011
STAT curr_items 595
STAT total_items 961
STAT bytes 4587415
STAT curr_connections 3
STAT total_connections 22
STAT connection_structures 4
STAT cmd_get 2688
STAT cmd_set 961
STAT get_hits 1908
STAT get_misses 780
STAT evictions 0
STAT bytes_read 5770762
```

```
STAT bytes_written 7421373
STAT limit_maxbytes 536870912
STAT threads 1
END
```

上記の結果による get_hits の値を cmd_gets の値で割ってください。この例では、ヒット率は約 71 パーセントということになります。理想的には、このパーセンテージはもっと高くなければなりません。パーセンテージが高ければ高いほど、ヒット率が良好ということになります。このように統計を長期的に観察して測定することで、キャッシング・ストラテジーの有効性を十分に把握することができます。

第 1 回のまとめ

キャッシングは大量のデータを扱う Web アプリケーションに不可欠な部分ですが、そのキャッシングを行う上で memcached は最適な選択肢です。個人的な経験として、私は memcached を利用して大きな成功を収めてきました。memcached をキャッシング・ソリューションとして使用してみれば、このソリューションがいかに有効であるかがわかるはずです。

連載の第 2 回では、memcached を Grails アプリケーションに統合する方法を説明します。この統合は、スケーラブルな Web アプリケーション開発の刺激的かつ有望なスタックを詳しく探る機会となるだけでなく、実に素晴しい結果をもたらします。次回の記事を待つ間、memcached で他の操作も試してみる上では、今回学んだ内容が十分な足掛かりとなるはずです。ぜひmemcached をインストールして、いろいろと試してみてください。

関連トピック

- 「Distributed Caching with Memcached」(Brad Fitzpatrick 著、Linux Journal、2004年8月): Danga Interactive 社の Brad Fitzpatrick 自らが memcached を紹介しています。
- 「Server load-balancing architectures: Transport-level architectures」(Gregor Roth 著、JavaWorld、2008年10月): この記事では、複数マシンでの HttpResponse メッセージのキャッシングに基づく memcached 負荷分散ソリューションを紹介しています。
- 「Performance tuning considerations in your application server environment」(Sean Walberg 著、developerWorks、2009年1月): Web アプリケーションの各種コンポーネントが対話する方法について、よくあるパフォーマンスのボトルネットとキャッシングなどのソリューションを含め、概要を説明しています。
- 「Is Memcached a Good or Bad Sign for MySQL?」(Gary Orenstein 著、Gigaom.com、2009年5月): アプリケーション・スタックのなかで memcached が適切な場所、そして軽量のキャッシング手段が RDBMS に代わる候補となっている理由について、概要を読んでください。
- developerWorks Java technology ゾーン: Java プログラミングのあらゆる側面を網羅した記事が豊富に用意されています。
- memcached をダウンロード: 分散型メモリー・キャッシング・システムです。
- libevent をダウンロード: 非同期イベント通知ライブラリーです。
- PuTTy をダウンロード: Windows 対応 telnet クライアントです。