AOPによるキャッシュのすすめ

2008.11.1

TANIGUCHI Hikaru (id:tanigon)

アジェンダ

- 自己紹介
- キャッシュ概要
- キャッシュを「使う」
- S2CachingとehCacheのすること
- キャッシュ導入Tips
- まとめ＋その他

自己紹介

- 谷口　光　(id: tanigon)
- コミッタとか
  - S2Velocity
  - S2Caching
- 携帯電話向けWebサービス作りが仕事
- 音楽、車、手品、ルービックキューブ
- 連絡先?
  - http://iddy.jp/profile/tanigon
  - tanigon2001@gmail.com　とか Twitter

キャッシュ概要

- キャッシュとは
  - あるレイヤー(層)間でのリクエストと、そのレスポンスの対を記憶しておき、同様のレスポンスが予想されるリクエストに対して、記憶しておいたレスポンスを返却すること
  - 応答速度が向上する
  - 処理負荷が軽減される
  - リクエストが同じなら　レスポンスも同じであるという状況で、レスポンスの生成の時間がかかる場合に有効

キャッシュ概要

- キャッシュの例
  - Webブラウザのキャッシュ
    - あるURLにリクエストしたときのレスポンス(HTML、画像など)をブラウザがディスクなどに記録しておいて、次回以降の応答が速くなる
  - CPUに搭載されているキャッシュ
    - メインメモリへの読み出しがあるとき、メインメモリよりさらに高速なメモリにその内容を記録しておいて、同じ番地に対する読み込みを高速化する
  - (微妙な例) arpエントリのキャッシュ
    - IPアドレスから EthernetのMACアドレスを取得するという通信手順(arp)はSW-HUBやPCでキャッシュされている

キャッシュ概要

- キャッシュの例 (続き)
  - データベースに搭載されているキャッシュ
    - テーブルなどディスクに格納されているデータへのアクセス時、同様の内容をメモリに記録しておいて、二回目以降の読み出しを高速化できる
    - SQLのコンパイル結果を記録しておいて二回目以降同じSQLについてはコンパイルを省略し、高速化できる

キャッシュ概要

- 今日お話するキャッシュの使い方
  - メソッド呼び出しのキャッシュ
    - ある引数に対して応答が一定である(と思われる)場合に、応答を記録しておいて、同じ呼び出しに対しては記録した値を即座に返却して処理を省略する
    - **「どのオブジェクトの」「どのメソッドを」「どんな引数で呼び出したか」**が リクエスト (キャッシュのキー)
    - そのときの**「戻り値」**がレスポンス
    - この対を記録しておき、キャッシュとして活用する
    - 本日は S2Cachingの紹介ということで、Java で、Seasar2 の AOPを使う前提で紹介

キャッシュ概要

- キャッシュのExpireポリシー
  - キャッシュは「記憶する」ため、リクエストのバリエーションが豊富な場合は、もちろん記憶域をどんどん消費していってしまう
  - そのため、なんらかのポリシーで一度記憶したものを削除する必要がある (記憶域の解放)

キャッシュ概要

- キャッシュのExpireポリシーの例
  - 期限切れ
    - 追加されたエントリは一定時間がたつと期限切れとして削除される（賞味期限切れ?)
  - LRU付き容量制限
    - 記憶域が一杯になってくると、最後に参照された時期がもっとも古いものを削除する
  - 意図的なパージ
    - キャッシュエントリを利用者の意思で削除するというもの

キャッシュを「使う」

- キャッシュという機構は実は簡単に作成し、使うことができる
  - キーと値を記憶しておく仕組みがあればいい
  - HashMapなど
  - Mapのキーを「リクエスト」、値を「レスポンス」としておけばよい

キャッシュを「使う」

- キャッシュの実装例
  - 記憶域(Map)に存在すればそれをそのまま使う
  - 存在しなければメソッドを呼び出して、戻り値を格納しておく(次回以降に活用する)

```
Object value;
if (cache.containsKey(arg)) {
    value = cache.get(arg);
} else {
    value = service.method(arg);
    cache.put(arg, value);
}
```

キャッシュを「使う」

- キャッシュを仕掛ける場所、という問題
  - 呼び出し元(クライアント)に仕掛けるか?

<クライアント側>

```
Object value;
if (cache.containsKey(arg)) {
    value = cache.get(arg);
} else {
    value = service.method(arg);
    cache.put(arg, value);
}
```

<サービス側>

```
public String method(Date arg) {
  .
  .
  return 何か;
}
```

キャッシュを「使う」

- キャッシュを仕掛ける場所、という問題
  - 呼び出され側(サービス)に仕掛けるか?

<クライアント側>

```
value = service.method(arg);
```

<サービス側>

```
public String method(Date arg) {
  String value;
  if (cache.containsKey(arg)) {
    value = cache.get(arg);
  } else {
    value = 計算処理いろいろ;
    cache.put(arg, value);
  }
  return value;
}
```

キャッシュを「使う」

- 呼び出し側に仕掛ける場合
  - 良い点
    - サービス側のコードを汚染しない
    - サービス側はビジネスロジックやデータアクセスの本質的な部分の記述だけで良い
  - 悪い点
    - 呼び出し対象が同じ引数に対して同じ値を返すという条件を知っている必要がある
      - サービスのメソッドが副作用を持つときに、その副作用が処理されなくても大丈夫！とわかる場合でないと処理を省略してはいけない

キャッシュを「使う」

- 呼び出され側に仕掛ける場合
  - 良い点
    - 呼び出し元が複数ある場合に、単一のコードで全てのクライアントに統一してキャッシュの機構を提供できる
    - 引数に対して　戻り値が一定なのかどうか、は自分自身のことなのでよく知っている(キャッシュの制御ができる)
    - 重要な副作用がある場合には処理を省略しない、など柔軟なキャッシュポリシーを実現できる
  - 悪い点
    - 本来、チューニングの扱いであるキャッシュ処理がビジネスロジックを汚染する (本質的ではない処理だから)
      - テストが難しくなったりいろいろと危険!

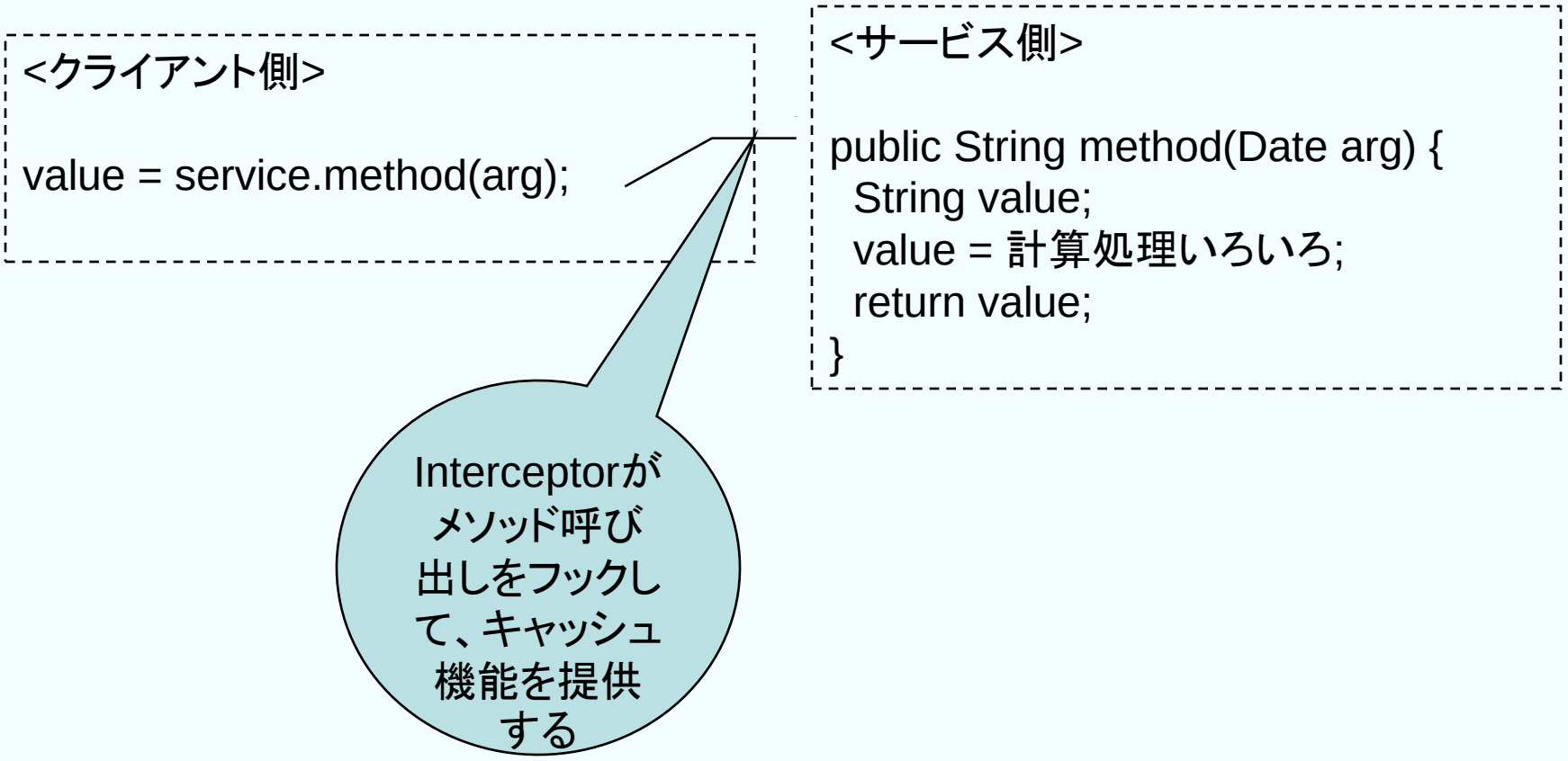
キャッシュを「使う」

- 両者の問題をほどよく解決する答え

  ＝『**AOP**によるキャッシュ機能の実現』

キャッシュを「使う」

- AOPによるキャッシュなら…



キャッシュを「使う」

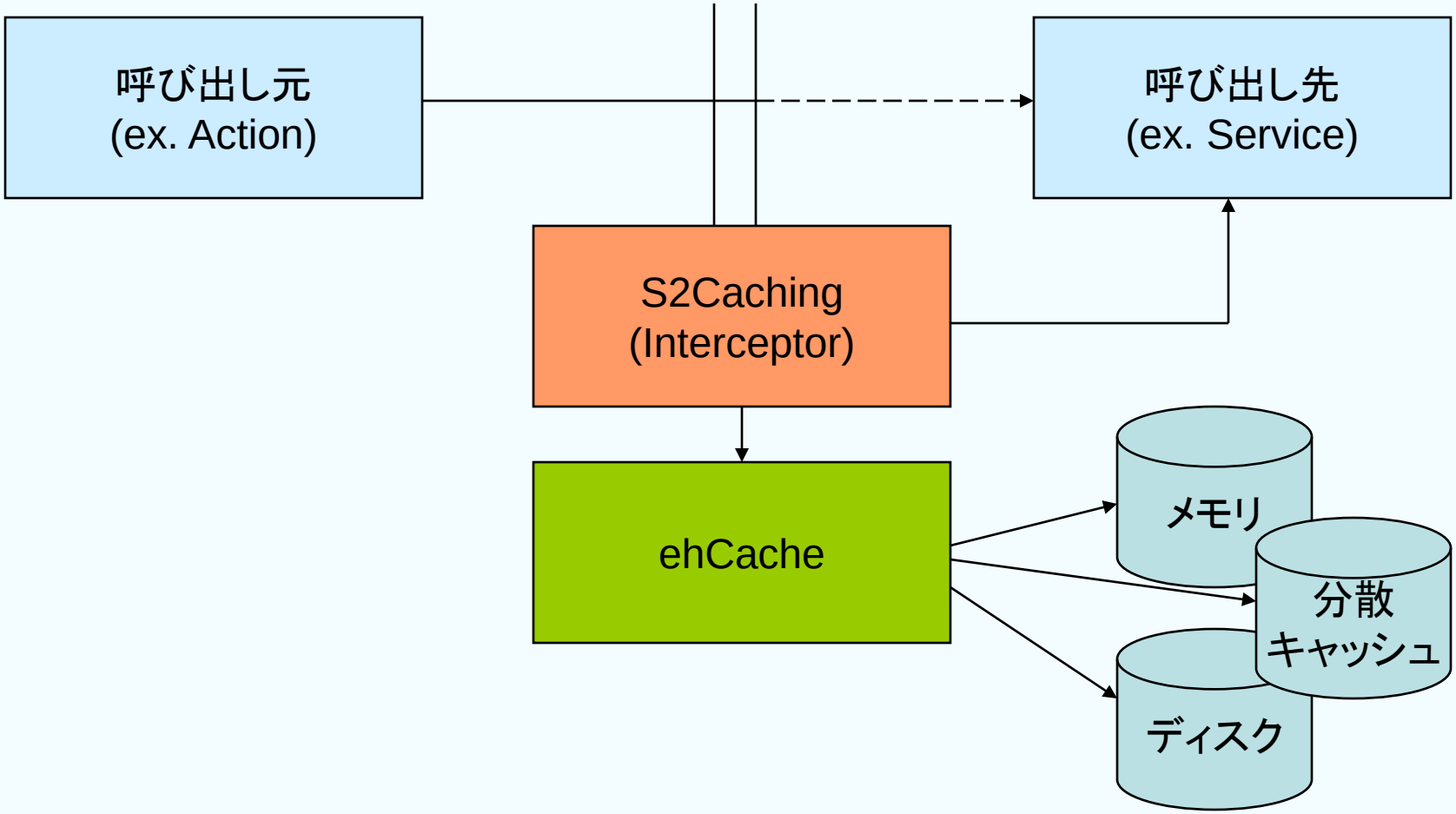
- AOPによるキャッシュのポイント
  - 呼び出し側のコードも呼び出され側のコードも書き換えなくてよい (汚染なし)
  - ソースコード以外の設定ファイル(たとえばXML)でポイントカットを定義できるので、再コンパイルなしで柔軟にキャッシュを仕掛ける場所を変更できる
  - キャッシュの振る舞いを共通で書くことができる
    - 逆に柔軟さはなくなる
    - Don't Repeat Yourselfの原則に準拠しやすくなる

S2CachingとehCache

- というわけで、AOPによるキャッシュはオススメ
- 私が開発中の “S2Caching” がオススメ
  - ひどい宣伝
- S2Cachingのいいところ
  - 秒間400PV程度の多数のWebサイトで導入実績
  - Seasar2.3/2.4 + S2Struts + S2Velocity …
  - Action/Service/Dao のレイヤー分割で Serviceのメソッドや Daoのメソッドにキャッシュを「後天的」に設置する
  - 1行もソースを修正せずにパフォーマンスチューニング

- S2Cachingの特徴
  - キャッシュそのものの振る舞いは ehCacheに任せている
    - 将来的にmemcachedや他のキャッシュライブラリ/フレームワーク/実装を使えるようにするかも
  - 非常にシンプルなコード
    - Seasar2でしかテストしていないが、aopalliance準拠のinterceptorを受け入れるフレームワーク(/環境)であれば変更なしか、ちょっとした変更で動くかも

- イメージ



S2CachingとehCache

- S2Cachingのすること
  - メソッドの呼び出しをフック
  - 以下の情報からキャッシュキーを生成
    - 呼び出し先のオブジェクトのアドレス（的なもの）
    - 呼び出し先のメソッドシグネチャ
    - すべての引数
  - キャッシュに存在するかどうかを確認
    - 期限切れなどはehcacheが管理している
    - S2Cachingは「ありますか？」と聞くだけ

S2CachingとehCache

- S2Cachingのすること
  - キャッシュに存在していれば
    - その値を取得して、**シリアライズによって複製を取り**返却する
  - キャッシュに存在していなければ
    - 実際のメソッド呼び出しを発行し、戻り値を得る
    - キャッシュに登録する
    - 値を、**シリアライズによって複製を取り**返却する

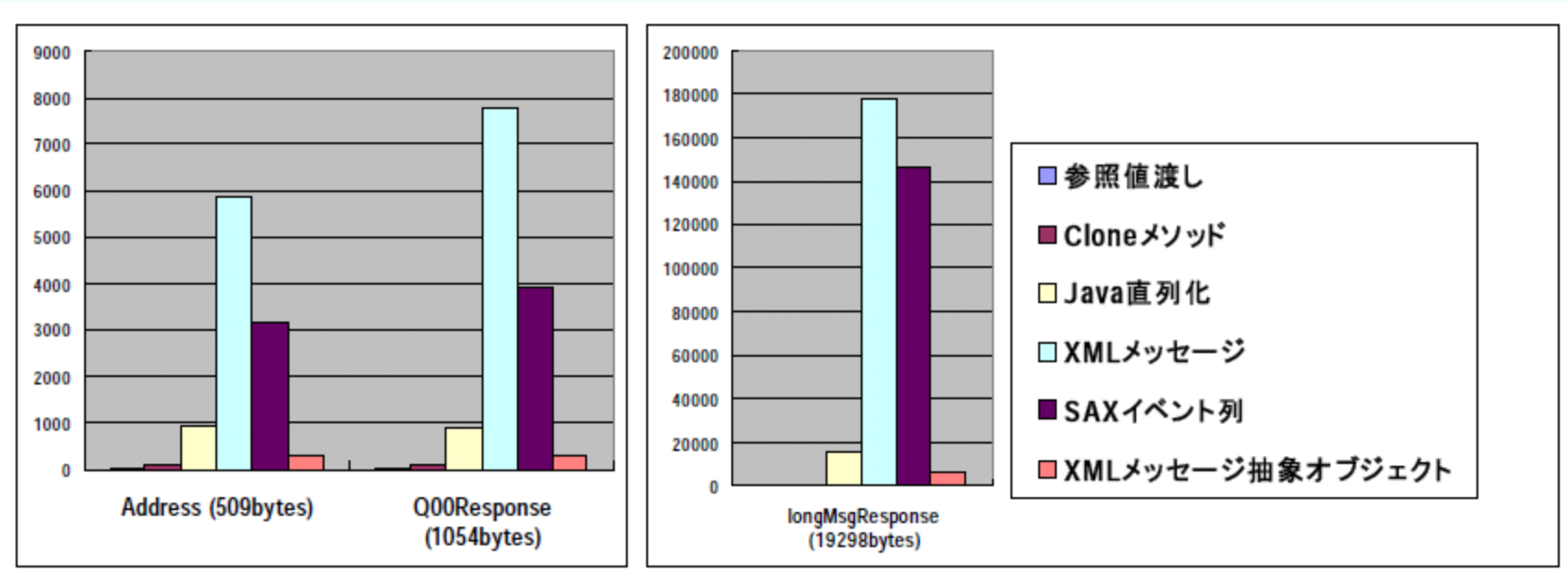
S2CachingとehCache

- なぜ複製なのか
  - 戻り値がオブジェクトである場合、単純にキャッシュにputすると「参照」が格納されてしまう
  - 呼び出し元がそのオブジェクトに対して破壊操作を行わない保証がない
  - ステートフルなオブジェクトであれば状態そのものが変化してしまい、二回目以降正しい状態のオブジェクトが帰らない（ように見える）こともある

S2CachingとehCache

- なぜシリアライズなのか
  - 浅いコピーと深いコピー問題
    - 戻り値の型が中にさらにオブジェクト(の参照)を持つ場合
    - フィールドにArrayListを持つケースなどよくあること
  - 深いコピーが必要！手段は？
    - XML化 (xstreamとか) → パフォーマンス
    - Clonable と clone()
      - Clonableは流行っていない （実装していないクラスが多い）
    - Serializable
      - StringとかArrayListなどはSerializable。使える。
      - たいていのPOJOなDto/Beanは”Serializable”をつければ手間はいらない(利用者の負担だが、あきらめてもらう)

- パフォーマンスについて
  - 出典: Webサービス･クライアントにおける効果的な応答キャッシュにむけて (高瀬 俊郎, 立堀 道昭;日本アイ･ビー･エム東京基礎研究所)



S2CachingとehCache

- ehCacheのすること
  - キャッシュ機構の提供
    - キーと、値のペアを記憶する、という機能
  - キャパシティ管理
    - メモリ内に指定された件数を上限として保持する
    - 上限を上回った場合、ディスクに書き出すということも出来る
  - 分散キャッシュ
    - 複数のノード間でキャッシュの内容をシェアする機能
    - RMIによる全ノード間でのキャッシュエントリの通知
      - memcachedとは違うよ、速いよというアピール記事があるが、ノード数が増えると通信量が爆発するはず

- ehCacheのすること
  - エントリの期限切れの管理
    - エントリの追加から一定期間が過ぎると強制的に賞味期限切れになる設定
    - エントリが一定期間参照されないと賞味期限切れになる設定
    - 容量が一杯になったときに、最も参照されていないエントリから削除する、という設定
    - 容量が一杯になったときに、最後に参照された時期がもっとも古いものから削除する、という設定
    - 上記の組み合わせをキャッシュのポリシーとして設定できる

キャッシュ導入Tips

- **キャッシュは「チューニング」と思うべし**
  - 本質的に不要な処理を盛り込んで性能を稼ぐ
  - パフォーマンスが問題になってから初めて取り組むべき
- **プロファイリングで「回数×時間」の大きいポイントにしかけるべし**
  - おすすめはやはり Dao などリモートへのアクセスが発生する部分（どうしても処理時間が長い）
  - Daoはコール回数も多い
  - マスタなど「変わらない」データへの問い合わせ

まとめ

- キャッシュは本質的にはMapと同じ
- キャッシュは「チューニング」 焦って使わない
- キャッシュを仕掛けるには、対象の副作用や、関数としての特性を理解しておく必要がある
- なるべく本質的な処理とは切り離すべし
  - ビジネスロジックを汚染するべきではない
- S2Caching(ehCache)を使うとキャッシュ導入コストが低くなる
  - 自前で実装するより機能も豊富
  - まだ SNAPSHOTですが商用実績もあります