Javaの理論と実践: Web層での状態複製

Webアプリケーションをクラスター化する方法は複数ある

Brian Goetz
Principal Consultant
Quiotix

2004年 7月 29日

重要なWebアプリケーションでは、例えばユーザーの買い物かごの内容など、何らかの形のセッション状態を保持する必要があります。クラスター化されたサーバー・アプリケーションの中で、状態をどのように管理、複製するかは、そのアプリケーションのスケーラビリティに大きな影響を与えます。多くのJ2SEやJ2EEアプリケーションでは、Servlet APIが提供するHttpSessionに状態を保存します。今月はコラムニストのBrian Goetzが、状態複製に関わる幾つかのオプションと、良好なスケーラビリティやパフォーマンスを得るためにはHttpSessionをどのように使えば最も効果的かを説明します。

このシリーズの他の記事を見る

J2EEアプリケーションを構築しているにせよJ2SEアプリケーションを構築しているにせよ、皆さんはJSP技術やVelocity、それにWebMacroなどのようなプレゼンテーション層を通して直接に、あるいはAxisやGlueといったサーブレット・ベースのWebサービス実装を通して、何らかの形でJavaサーブレットを使用しているでしょう。サーブレットAPIが提供する最も重要な機能の一つがセッション管理、つまり`HttpSession`インターフェースを通した、ユーザー毎のセッション状態の認証、期限満了、それに維持管理などといったセッション管理です。

セッション状態

Webアプリケーションであればほとんど必ず、何らかのセッション状態があります。このセッション状態は、あなたがログインしているかどうかを記憶するというような単純なものの場合もあれば、例えば買い物かごの中身であるとか、前に行ったクエリーのキャッシュ結果、あるいは20ページにも渡る動的アンケートの完全な応答履歴などといった、より詳細なセッション履歴の場合もあります。HTTPプロトコル自体はステートレスなので、セッション状態はどこかしらに保存する必要があります。またブラウザ・セッションと関連付けられている必要があり、次回に同じWebアプリケーションに対してページを要求する時には簡単に呼び出せるようになっている必要があります。幸いJ2EEには、セッション状態を管理するために幾つかの方法が用意されています。データ層に保存することもできれば、サーブレットAPIの`HttpSession`インターフェースを使ってWeb層に保存することもでき、ステートフル・セッション・ビーンズを使って

EJB（Enterprise JavaBeans）層に、あるいはクッキーや隠しフォーム・フィールドを使ってクライアント層に保存することさえできるのです。残念ながら軽率なセッション状態管理をすると、深刻なパフォーマンス問題を引き起こす可能性があります。

そのアプリケーションがユーザーのセッション状態を保存する上で `HttpSession` が適しているのであれば、他の方法ではなくこの方法を使用した方が良い場合が多いのです。HTTPクッキーや隠しフォーム・フィールドを使ってセッション状態をクライアントに保存する方法は、重大なセキュリティ・リスクがあります。この方法では、アプリケーションの内部を信頼できないクライアント層にさらすことになります。（初期の電子商取引サイトの一つでは、（価格を含めて）買い物かごの中身を隠しフォーム・フィールドに保存していました。ですからHTMLやHTTPが分かるユーザーであれば、どんな買い物でも$0.01でできてしまったのです。あらあら・・）それに、クッキーや隠しフォーム・フィールドを使う方法はぐちゃぐちゃでエラーが起こりやすく、壊れやすいものです（しかもクッキー・ベースの手法は、ユーザーがブラウザのクッキーを使用不可にしてしまうと全く動作しません）。

J2EEアプリケーションでサーバー側の状態を保存するための方法としては、他にステートフル・セッション・ビーンズを使用する方法、またはデータベースに会話状態を保存する方法があります。ステートフル・セッション・ビーンズを使えばセッション状態管理がより柔軟になりますが、現実的な選択をするとセッション状態をWeb層に保存する方が有利、という場合が相変わらずあるのです。ビジネス・オブジェクトがステートレスであれば、WebサーバーとEJBコンテナの両方を追加するのではなく、単純にWebサーバーを追加するだけでアプリケーションをスケール・アップすることができる場合がよくあります。一般的にはその方が安くすみ、また簡単でもあります。会話状態を保存するために`HttpSession`を使う方法のもう一つの利点は、セッション期限が切れた時の簡単な通知方法がサーブレットAPIに用意されている、という点です。会話状態をデータベースに保存する方法は、とても使用できないほど高くつきます。

サーブレットの仕様は、サーブレット・コンテナに対して何らかのセッション複製やパーシスタンスを行うことを義務づけてはいませんが、状態複製はサーブレットのそもそもの存在理由の重要な一部だと言っています。そして、セッション複製を行う選択をしたコンテナに対しては、幾つかの要求を課しています。セッション複製を使うことによって、負荷分散やスケーラビリティ、耐障害性、高可用性など、様々な点で有利になります。従って大部分のサーブレット・コンテナでは何らかの形の`HttpSession`複製をサポートしていますが、複製の機構や構成、それにタイミングなどは実装依存となっています。

HttpSession API

簡単に言うと、`HttpSession`インターフェースは複数のHTTPリクエストに渡ってセッション複製を保持するために、サーブレットやJSPページ等のプレゼンテーション層コンポーネントを使用するためのメソッドを幾つかサポートしています。このセッションはある特定なユーザーに結びついていますが、Webアプリケーションの全てのサーブレットに渡って共有されており、ある一つのサーブレットに特有なものではありません。セッションとは、セッションの継続期間中にオブジェクトを保存する`Map`のようなものだ、と考えると便利です。`setAttribute`を使ってセッション属性を名前で保存し、`getAttribute`を使ってそのセッション属性を取り出すことができます。`HttpSession`インターフェースには、（コンテナに対してセッションを捨てるべきだと通知する）`invalidate()`のようなセッション・ライフサイクル・メソッドも含まれています。リスト1は`HttpSession`インターフェースのうち最も一般的に使用される要素を示しています。

リスト1. HttpSession API

```
public interface HttpSession {
    Object getAttribute(String s);
    Enumeration getAttributeNames();
    void setAttribute(String s, Object o);
    void removeAttribute(String s);
    boolean isNew();
    void invalidate();
    void setMaxInactiveInterval(int i);
    int getMaxInactiveInterval();
    ...
}
```

理論的には、セッション状態を均一にクラスター全体に渡って完全に複製することは可能です。そうすればクラスター中のノードがどれも、どのリクエストでもサービスでき、フェールしたホストを避けて、ダム・ロードバランサーがラウンドロビン方式で単純にリクエストをルーティングします。ただし、そのようにきっちりとした複製はパフォーマンスのコストがかなり高く、実装も複雑なものになります。しかもクラスターが一定のサイズに近づくと、スケーラビリティの問題も起きてきます。

より一般的な手法としては、負荷分散とセッションの類似性（session affinity）を組み合わせることです。ロードバランサーは接続とセッションを関連付け、セッション内でのその後のリクエストを同じサーバーにルーティングすることができます。この機能をサポートしているハードウェアやソフトウェアのロードバランサーは無数あります。ということは、複製されたセッション情報がアクセスされるのは、プライマリー・コネクション・ホスト(primary connection host)がフェール(fails)し、そのセッションを別のサーバーにフェールオーバーする必要がある時のみ、ということになります。

複製の手法

可用性や耐障害性、それにスケーラビリティを含め、複製には幾つかの潜在的な利点があります。また、セッション複製には無数の方法がありますが、どの方法を選択するかはアプリケーション・クラスターのサイズや複製の目標、それにサーブレット・コンテナがサポートする複製機構に依存します。複製にはパフォーマンスのコストがかかり、CPUサイクル（セッションに保存されたオブジェクトをシリアル化する）、ネットワーク帯域（更新を伝搬する）、そしてディスクベースの方式であればディスクやデータベースに書き込むコストなどを考慮する必要があります。

ほとんど全てのサーブレット・コンテナは、`HttpSession`に保存されたオブジェクトをシリアル化することによって`HttpSession`複製を実行します。ですから配布可能なアプリケーションを作りたいのであれば、セッション中にあるのは確実にシリアル化可能なオブジェクトのみ、となるようにしておく必要があります。（一部のコンテナでは、EJB参照やトランザクション・コンテキスト、それにシリアル化できない他のJ2EEオブジェクト・タイプのような実体に対して特別な扱いをします。）

JDBCベースの複製

セッション複製の一つの手法として、セッションの内容を単純にシリアル化し、それをデータベースに書き込むという方法があります。この手法は単純至極であり、他のどんなホストにも

セッションをフェールオーバーできるだけでなく、クラスター全体のフェールでもセッション・データが生き残れるという利点もあります。ただしデータベースを後ろ盾とする複製の欠点は、パフォーマンスのコストです。つまりデータベース・トランザクションは高価なのです。この方式はWeb層ではうまくスケールアップできますが、データ層でスケーリングの問題を引き起こす可能性があります。クラスターが大きくなりすぎると、データ層に大量のセッション・データを収容することが困難、またはコスト的に不可能になり得ます。

ファイル・ベースの複製

ファイル・ベースの複製は、セッション・データの保存にデータベースではなく、共有ファイル・サーバーを使用することを除けば、シリアル化したセッションの保存にデータベースを使用する方式と似ています。この方式の方が一般的にデータベースを使用するよりも（ハードウェア・コスト、ソフトウェア・ライセンス、それに計算オーバーヘッドなどの）コストが低くなりますが、信頼性の面では若干劣ります（データベースの方が、ファイルシステムよりも強力なパーシスタンス保証があります）。

メモリー・ベースの複製

複製の手法としてもう一つは、シリアル化したセッション・データのコピーを、クラスター中の複数のサーバーで共有する方法です。全てのセッションを全てのホストに対して複製すれば最大の可用性が得られ、ロードバランサーにとっては最も楽になりますが、やがて各ノードでのメモリー消費要求と複製メッセージが消費するネットワーク帯域のために、クラスター・サイズに上限が出てきます。一部のアプリケーション・サーバーではメモリー・ベースの複製として、「buddy」ノードに複製する方式をサポートしており、各セッションが一つのプライマリー・サーバーと、一つ（またはそれ以上）のバックアップ・サーバーに存在するようになっています。この方式は全てのセッションを全てのサーバーに複製するよりもスケーラブルですが、セッションを他のサーバーにフェールオーバーする必要がある時には（どのサーバーにセッションがあるのかを判別する必要があるので、）ロードバランサーの仕事が複雑になります。

タイミングに関する考慮

複製したデータをどのように保存するかを決めることに加えて、いつデータを複製するかという問題もあります。最も信頼性が高く、しかし最も高価な手法は、（例えばサーブレット呼び出しの終了時の都度など）データが変化する度に複製する方法です。それよりも安価ですが、しかしフェールオーバーの際に一部データが失われる危険性のある方法は、毎N秒以内ごとに複製する方法です。

タイミングに関連した問題として、全セッションを複製するのか、それともセッション中で変化した属性（非常に少ないデータしかない可能性があります）のみを複製しようとするのか、という問題があります。これらは全て信頼性とパフォーマンスのトレードオフであり、どこでトレードオフするかはアプリケーションに依存します。サーブレット開発者は、フェールオーバーの際にはセッション状態が（幾つか前のリクエストの複製から判断すると）「古い」場合があり得ることを意識して、セッション内容が最新ではない場合に対応できるようにしておく必要があります。（例えばインタビューのステップ3がセッション属性を生成するようになっていてユーザーがステップ4にいる時、リクエストは、セッション状態の複製がリクエスト2つ分古いシステムにフェールオーバーされます。ステップ4に対するサーブレット・コードはセッションにその属性

が無いことを想定している必要があり、（その属性がそこにあると想定して、その属性がない時に`NullPointerException`を投げるのではなく）例えば適切にリダイレクトするなどの対応動作をとる必要があります。）

コンテナのサポート

`HttpSession`複製に対するサーブレット・コンテナのオプションや、そうしたオプションをどのように構成するかは、それぞれによって様々です。IBM WebSphere®の複製オプションは最も豊富であり、メモリー内複製かデータベース・ベースの複製か、サーブレット終了時に複製するかタイムベースに基づく複製か、セッション全体のスナップショットを伝搬するか、変化した属性だけか、等が選択できるようになっています。メモリー・ベースの複製は JMS publish-subscribe に基づいているので、全てのクローンに複製することもでき、単一の「buddy」にも、あるいは専用の複製サーバーにも複製することができるようになっています。

WebLogicにも（単一のbuddy複製を使用した）メモリー内複製やファイル・ベース、またデータベース・ベース複製など様々な選択肢が用意されています。JBossは、TomcatまたはJettyサーブレット・コンテナを使用する時にはメモリー・ベースの複製を行いますが、複製タイミングの選択肢にはサーブレット終了時またはタイムベースのいずれかがあり、また変化した属性のみのスナップショットをとるというオプション（JBoss 3.2とそれ以降）が用意されています。Tomcat 5.0では、全クラスター・ノードに対するメモリー・ベースの複製を用意しています。さらに、WADIのようなプロジェクトのおかげで、TomcatやJettyなどのようなサーブレット・コンテナに対して、サーブレット・フィルタリング機構を通してセッション複製を追加することもできます。

分散Webアプリケーションでのパフォーマンスを改善する

どのようなセッション複製機構を選択するにせよ、Webアプリケーションのパフォーマンスやスケーラビリティを幾つかの方法で向上することができます。第一に、セッション複製の効果を得るためには、Webアプリケーションのデプロイメント・デスクリプターに、配布可能としてマークを付けておく必要があることを忘れないで下さい。また、セッションに置かれるものは全てシリアル化可能である必要があることもよく確かめて下さい。

セッションを最小限に保つ

セッションの複製はコストを伴い、コストはセッション中に保存されるオブジェクト・グラフのサイズと共に増大するので、セッション中に置くデータは、現実的に考えられる限り少なくするように心がけるべきです。そうすることによってシリアル化のオーバーヘッドやネットワーク帯域幅の要求、それに複製のためのディスク要求を減らすことができます。特にセッション中の共有オブジェクトを保存すると、その共有オブジェクトが属する全てのセッションで複製されてしまうので、共有オブジェクトの保存は一般的には良いことではありません。

setAttributeを省略しない

セッション中の属性を変化させる時に、サーブレット・コンテナが何らかの形の最小更新（minimal updating: 変化した属性のみを伝搬する）をしているようであれば注意して下さい。`setAttribute`を呼ばないと、コンテナは属性が変わったかどうかに気が付かない可能性があります。（買い物かごの中身を表現する`Vector`がセッション中にある場合を考えてみて下さい。

単純に`getAttribute()`を呼んで`Vector`を取り出し、何かを追加した後で`setAttribute`を呼ばないでおくと、コンテナは`Vector`が変化したことに気が付かない可能性があります。）

きめ細かいセッション属性を使用する

最小更新をサポートするコンテナでは、巨大な単一オブジェクトではなく、きめ細かな複数のオブジェクトをセッションに置くことによってセッション複製のコストを引き下げることができます。そうすればデータが高速で変化しても、それによってコンテナが遅いデータもシリアル化して伝搬させてしまうのを防ぐことができます。

終了したら無効にする

ユーザーがセッションを終了したことが分かっているのであれば（例えばユーザーがログアウトを選択した、など）、必ず`HttpSession.invalidate()`を呼ぶようにします。そうしないとセッションは期限が切れるまで持続し、セッションの有効期限切れタイムアウトによっては長期間メモリーを消費する可能性があります。多くのサーブレット・コンテナでは全セッションに渡って使用できるメモリーの量に制限を設けており、最も以前に使用されたセッションをシリアル化してディスクに書き込みます。ユーザーがセッションを終了したことが分かっているのであれば、コンテナを節約するためにセッションを無効にします。

セッションをきれいに保つ

セッション中になにか大きな項目があって、それがセッションの一部でしか使われないような場合には、その項目を使い終わったら削除するようにします。大きな項目を削除することで、セッション複製のコストを引き下げることができます。（この方法は、ガーベジ・コレクターを補助するために明示的なnull(explicit nulling)を使用するのと似ています。私がこれを一般的に勧めるわけではないことは読者の皆さんが知っていると思います。ただし、複製によってガーベジをセッション中に保持するコストがあまりにも高い時には、この方法でコンテナを助けるだけの価値はあります。）

まとめ

サーブレット・コンテナは`HttpSession`複製を使用することによって、複製が作られた高可用性Webアプリケーションを構築する上での大仕事をこなしてくれます。ただし、コンテナによって異なりますが複製のための構成オプションが幾つかあり、複製の方針によってアプリケーションの耐障害性やパフォーマンス、それにスケーラビリティなどに差が現れます。複製の方針は後から考えるべきものではなく、Webアプリケーションを構築する時に考慮すべきものです。そして当然のことですが、アプリケーションのスケーラビリティを判断するための負荷テストを（顧客があなたに代わってテストする前に）行うことを忘れないで下さい。

著者について

Brian Goetz

Brian Goetz は18 年間以上に渡って、専門的ソフトウェア開発者として働いています。彼はカリフォルニア州ロスアルトスにあるソフトウェア開発コンサルティング会社、Quiotixの主席コンサルタントであり、またいくつかのJCP Expert Groupの一員でもあります。2005年の末にはAddison-Wesleyから、Brianによる著、Java Concurrency In Practiceが出版される予定です。Brian著による有力業界紙に掲載済みおよび掲載予定の記事のリストを参照してください。