WAS 虎の巻第 7 回

WebSphere Application Server（WAS）のセッション管理

目次

1. セッション管理

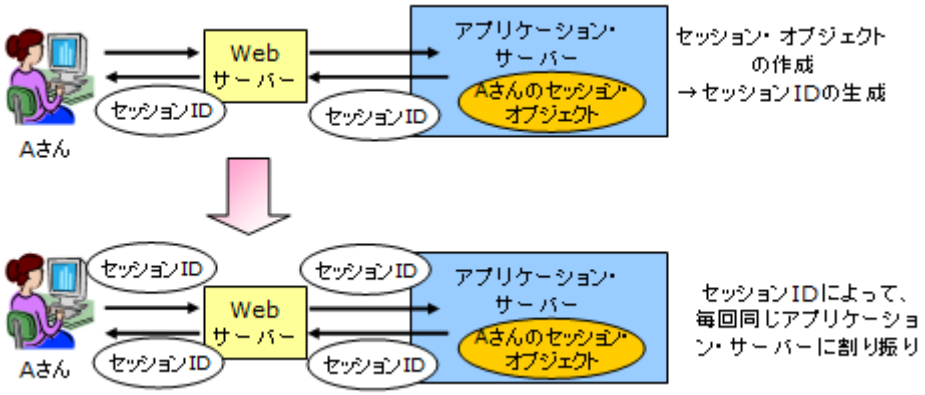
セッション管理は、インタラクティブな Web サイトを構築するための重要な機能です。WAS 虎の巻第 1 回ではセッション管理の必要性を、第 2 回ではセッション・アフィニティー機能とセッション・パーシスタンス機能を紹介しました。
WAS を設計・構築する上では、セッション管理を理解しておくことが重要な項目のひとつとなりますので、WAS 虎の巻第 7 回では、セッション管理機能の詳細をもう少し詳しくご紹介します。
この記事を読む前に、WAS 虎の巻第 1 回と第 2 回に目を通して、セッション管理の概要について復習して頂くことをお勧めします。

1.1. セッション ID とセッション・オブジェクト

まず、セッションに関する基本的な単語を復習します。

Java サーブレットの仕様では、特定のブラウザーやユーザーからの一連のリクエストを関連付けてインタラクティブな処理を実現する機能が定義されています。この仕様を「**セッション**（HTTP Session API）」といいます。WAS は、このサーブレット仕様に準拠してセッション管理機能を提供しています。

アプリケーション・サーバーは、メモリー上にユーザーのセッション情報を保持しています。これを**セッション・オブジェクト**と呼びます。また、ユーザーのセッション情報を保持するために、**セッション ID** と呼ばれる識別子を使用します。このセッション ID をブラウザーとやり取りすることで、個々のユーザーからのリクエストと、アプリケーション・サーバー上のセッション・オブジェクトを紐付けて、セッション管理が実現できます。



2. セッション管理の設定

では、WAS でのセッション管理に関する設定項目について確認していきましょう。
図は、WAS のセッション管理に関する設定画面の例です。この画面で以下のような項目を設定します。

① セッション・トラッキング・メカニズム
② メモリー内の最大セッション・カウント
③ セッション・タイムアウト
④ セッション・アクセスのシリアライズ
⑤ 分散環境設定

これらのセッション管理の設定は、以下のように、アプリケーションの要件に合わせてきめ細かく設定できます。それと同時に、ご自身の環境ではどのレベルで設定されているかを注意してください。

・ アプリケーション・サーバー・レベル

デフォルトのレベルです。この場合、サーバー内の全ての Web モジュールに対して設定が反映されます。

- アプリケーション・レベル
  アプリケーション内の全ての Web モジュールに対して設定が反映されます。
  アプリケーション・サーバー・レベルの設定をオーバーライドできます。
- Web モジュール・レベル
  特定の Web モジュールのみに有効な設定となります。
  アプリケーション・サーバー・レベル、アプリケーション・レベルの設定をオーバーライドできます。

アプリケーション・レベル、Web モジュール・レベルでのセッション管理設定方法は、以下のマニュアルを参考にしてください。

レベルごとのセッション管理の構成
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Ftprs_cnfs.html

それでは、これらの基本項目の内容をみていきます。

2.1. セッション・トラッキング・メカニズム

セッション ID を WAS⇔ブラウザー間で運ぶ方法を指定する項目で、Cookie、SSL ID、URL 再書き込み（URL リライティング）が選択できます。デフォルトでは Cookie が選択され、多くの場合 Cookie が使用されています。
SSL ID は、Web サーバーが IBM HTTP Server、iPlanet Web サーバーの場合のみサポートされ、WAS V7 から非推奨となっています。
URL 再書き込みを使用する場合は、アプリケーション・ロジックをプログラミングする必要があります。

Cookie を使用する場合は以下のようなパラメーターを定義します。

- **Cookie 名**

  Cookie の名前を指定します。セッション ID を運ぶ Cookie の名前は、サーブレット仕様で JSESSIONID であると決められていましたが、サーブレット 3.0 から変更可能となりました。

  WAS では、柔軟性を考慮して、サーブレット 3.0 準拠より前から Cookie 名を変更することができます。ただし、WAS 上で稼動するアプリケーションやパッケージ製品のサポートポリシーは異なる可能性がありますので、各々のサポートポリシーをご確認ください。

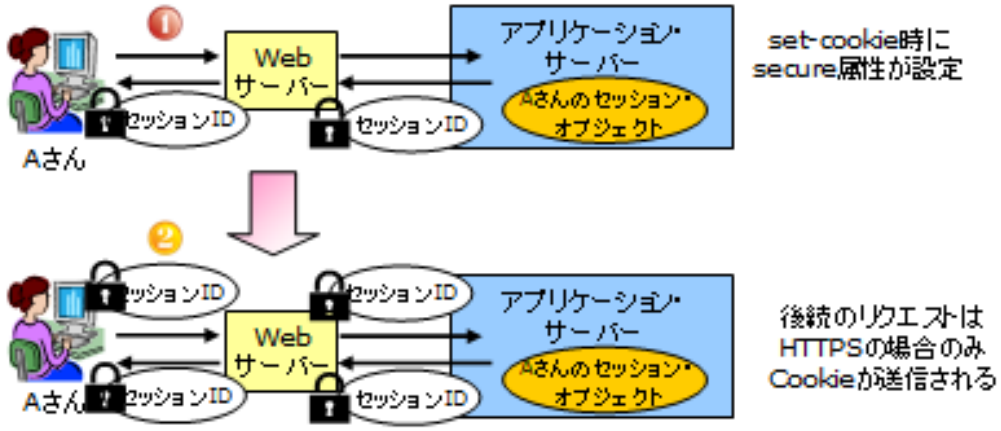
- **Cookie を HTTPS セッションに制限します**

  アプリケーション・サーバーで Cookie 生成時（set-cookie 時）に Secure 属性を設定するか否かを指定します。

  Secure 属性が指定された Cookie は HTTPS 通信時のみ送信されます。

  このオプションを指定した場合の動作は以下をご覧ください。このオプションを指定すると、下図①の通信が HTTP でも HTTPS でも Cookie に Secure 属性が付与されま

す。Secure 属性が付与されると、後続のリクエストとなる下図②では HTTPS 通信時のみ Cookie が送信され、HTTP 通信では Cookie が送信されません。



- **セッション Cookie を HTTPOnly に設定して、クロスサイト・スクリプティング・アタックを阻止します。**

  Cookie に HTTPOnly 属性を設定するか否かを指定するパラメーターで V8 からデフォルトで On になっています。このパラメーターの説明と注意点については下記リンクをご参照ください。

  アプレットから WAS V8 にリクエストする場合の注意点(WAS-12-002)
  http://www.ibm.com/support/docview.wss?uid=jpn1J1009240

- **Cookie ドメイン**

  この値によって、ブラウザーがどのサーバーに Cookie を送信するかが決まります。デフォルトはドメイン指定なしで、Cookie を発行したサーバーのみに Cookie が送信されます。例えば AAA.ibm.com で発行された Cookie は、デフォルトでは AAA.ibm.com のみに送信されます。Cookie ドメインに ibm.com を指定すると、AAA.ibm.com と BBB.ibm.com にも送信されます。ドメインを指定すると Cookie が送信される範囲が広がる点に注意してください。

- **Cookie 最大経過時間**

  Cookie がブラウザーで存続する有効期限を指定します。デフォルトでは有効期限の指定なしで、Cookie は現行のブラウザー・セッションのみ使用され、ブラウザーを閉じると消去されます。一時的な Cookie という意味で、セッション Cookie と呼ばれることがあります。

  有効期限を指定すると、指定した時間まではブラウザーを閉じても Cookie は消去されずブラウザー内部に保管されます。これをパーシステント Cookie と呼ぶことがあ

ります。

- **Cookie パス**

  Cookie を送信する範囲をパスで指定します。デフォルトではルート・ディレクトリーが指定されており、どの URL にアクセスする場合でも Cookie は送信されます。例えば、コンテキスト・ルートが XXX と YYY のアプリケーションが同一ホストで稼動している場合、デフォルトでは http://AAA.ibm.com/XXX にも http://AAA.ibm.com/YYY にも Cookie が送信されます。コンテキスト・ルートについては WAS 虎の巻第 5 回を参照してください。

  「コンテキスト・ルートを使用」を選択すると、同一ホストで複数のアプリケーションが稼動している場合に､他のアプリケーションへの Cookie 送信が制限されます。「4.5. 複数のアプリケーションで JSESSIONID を使用する場合」もご参照ください。

2.2. メモリー内の最大セッション・カウント

Web モジュールごとにメモリーに保持する最大セッション数を指定します。Web モジュールごとに最大セッション数を指定したい場合は、アプリケーション・レベルや Web モジュール・レベルでこの値を設定してください。

このパラメーターは、セッションをメモリーのみに保持する場合（ローカル・セッションと呼ばれることもあります）と、セッション・パーシスタンスを構成する場合で意味が異なります。

ローカル・セッションの場合は、アプリケーション・サーバーで保持できるセッションの最大数を指定します。最大セッション数を超えるとセッションが生成できなくなります。アクセス数、セッション・タイムアウト、セッション・サイズなどを考慮して、値を設定してください。最低限、サーバーの最大スレッド・プール・サイズよりは大きく設定する必要があります。

あわせて「オーバーフローの許可」を選択すると、最大セッション数を超えても、メモリーに余裕がある限り制限なくセッションを生成できます。しかし、想定よりも多くトラフィックが発生した場合や悪意のある攻撃が起きた場合にメモリーを消費してしまう可能性があります。また、オーバーフローで使用されるメモリー領域は、「メモリー内の最大セッション・カウント」で確保された領域ほど最適化されていない点にも注意してください。

セッション・パーシスタンスを使用する場合は、この値を上回るセッションが生成できます。セッション数がこの値を上回ると、メモリー上のセッション・オブジェクトが古い順に削除されますが、セッション･オブジェクトは DB などの外部データストアにも保存されているため、セッションは継続されます。

「オーバーフローの許可」は、セッション・パーシスタンスの場合は使用されません。

2.3. セッション・タイムアウト

セッション・オブジェクトは使用しなくなった段階で、アプリケーション内で明示的に破棄する（invalidate コマンド）ことが推奨されています。アプリケーションでセッションを破棄すると、サーバー上のセッションも無効化されます。
しかし、アプリケーションからログアウトせずにブラウザーのウィンドウを閉じてしまうと、サーバー上にセッション・オブジェクトが残ってしまうことがあります。
このようなセッション・オブジェクトは、「セッション・タイムアウト」で指定した時間を超えてアクセスが無いとメモリー上から破棄されます。セッション・パーシスタンスを使用する場合は外部データストアからも破棄されます。
セッション・タイムアウトはアプリケーション内にも指定でき、アプリケーションの web.xml に session-timeout を指定している場合は、web.xml の値が有効となります。
「タイムアウトなし」を選択すると、アプリケーション内でセッションを明示的に無効化しない限り、セッションはメモリー上に存在し続けてしまいますのでご注意ください。

また、セッション・タイムアウトが発生した場合、即座にセッションがメモリー上から破棄されるわけではありません。セッション・タイムアウトとなったセッションは、無効化の候補となり、その後一定間隔で作動するリーパー・スレッドにて実際に破棄が行われます。そのため、実際に破棄されるまでの時間は、最大で「セッション・タイムアウト+リーパー・スレッドの実行間隔」 ということになります。リーパー・スレッドの実行間隔は HttpSessionReaperPollInterval カスタム・プロパティーで調整することも可能です。

2.4. セッション・アクセスのシリアライズ

サーブレット仕様では、サーバー内のセッションの同時アクセスをサポートしています。WAS の拡張機能として、サーバー内のセッションの同時変更を防ぐために、複数スレッドからの同時セッション・アクセスを許可せず、スレッドセーフなアクセスを確保することができます。
このオプションを指定すると、セッション・オブジェクトが取得された段階でロックされ、同じセッション・オブジェクトに対する並行アクセスは「最大待機時間」の値まで待機します。最大待機時間を超えると「タイムアウト時のアクセスを許可する」が選択されている場合は、通常通り同時アクセスが許可されます。選択されていない場合は、Exception がスローされ、SystemOut.log には SESN0189E のエラーメッセージが出力されます。

2.5. 分散環境設定

セッション・パーシスタンスを構成します。セッション・パーシスタンスについては次章

で説明します。

3. セッション・パーシスタンス

虎の巻第 2 回でご紹介したように､セッション情報を外部データストアに書き込むことで、複数サーバーを跨ってセッション情報を使用することができます。これをセッション・パーシスタンスとよびます。
セッション・パーシスタンスは、以下のような場合に使用します。

- あるクラスター・メンバーの障害時や停止時に、別のクラスター・メンバーにセッション情報をフェイルオーバーさせたい場合。
- クラスター構成でなくても、セッション情報が重要であり、サーバー障害時でもセッション情報を消失させたくない場合。
- メモリー上に保持するセッション数を制御したい場合。「メモリー内の最大セッション・カウント」を超えたセッションに対し､「オーバーフローの許可」をさせずに外部データストアに格納することで､メモリー上のセッション数を制御できます。

3.1. セッション・パーシスタンスの構成

WAS でセッション・パーシスタンスを設定する前に、セッションを書き込む外部データストアを用意します。
セッション・オブジェクトを格納する外部データストアとして以下のようなコンポーネントを選択できます。

- **データベース**

  セッション情報を保管するデータベースは、セッション DB と呼ばれることがあります。

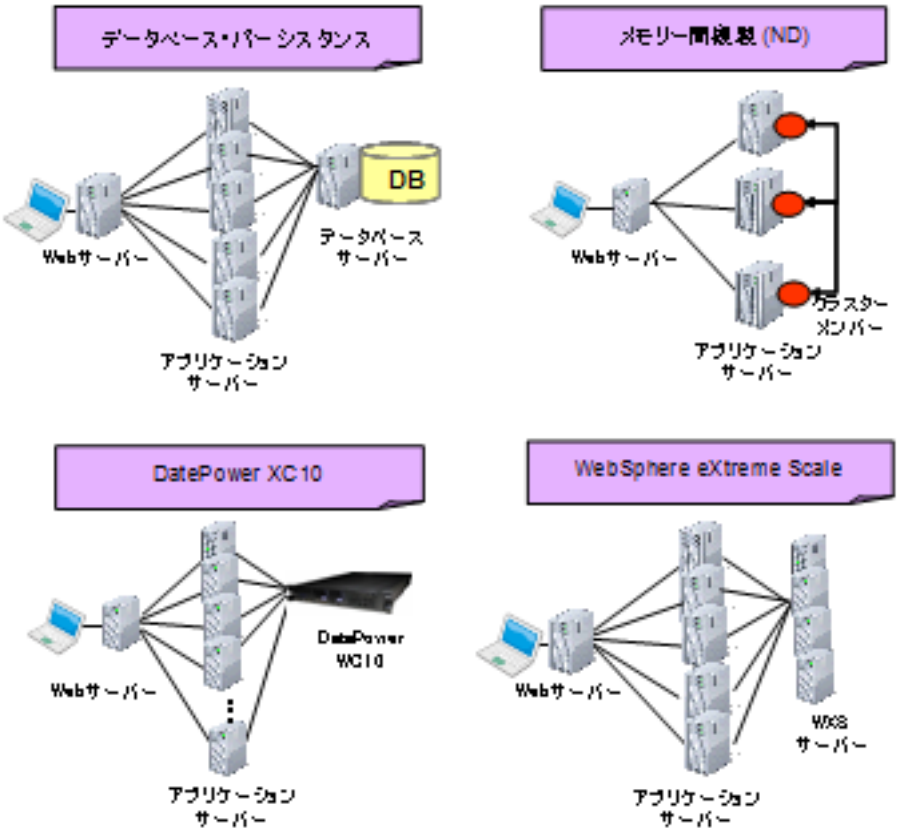
  WAS Base はアプリケーション・サーバーが 5 台まで、Express は 2 台までという制限があります。WAS ND にはアプリケーション・サーバー台数の制限はありません。
- **メモリー間複製**

  WAS ND で使用できる機能です。他のクラスター・メンバーのメモリー上にセッション・オブジェクトのコピーを作成します。
- **WebSphere DataPower XC10 Appliance**

  DataPower XC10 というアプライアンス製品にセッション情報を保管します。
- **WebSphere eXtreme scale**

  WebSphere eXtreme scale（WXS）はメモリー上のデータへの高速アクセスを実現する製品です。WXS の JVM 上にセッション・オブジェクトを格納します。WAS

V8.5.5 からは WAS に WXS のライセンスが含まれています。



これらの選択指針については、以下のガイドの P.42 をご参照ください。
WAS V8.0 による Web システム基盤設計ワークショップ資料「トポロジー設計」P.42
http://public.dhe.ibm.com/software/dw/jp/websphere/was/was8_guide/WASV80Design_01Topology.pdf

3.2. セッション・パーシスタンスの前提条件

外部ストアに保存されるセッション・データは全てシリアライズ可能である必要があります。セッション・パーシスタンスを使用する場合は、アプリケーションの中で、HTTP セッションに格納するデータは全て java.io.Serializable インターフェースを実装してください。ローカル・セッションの場合はこの処理は必須ではありません。しかしセッション・データがシリアライズ化されていないと、ローカル・セッションからセッション・パーシスタンスへ変更する際にアプリケーション・コードの変更が発生します。
アプリケーション開発時にはセッション・パーシスタンスを実装する計画がなかったとしても、その後 Web サイトのアクセス数が伸びサーバー台数が増え、セッション・パーシスタンスが必要になるかもしれません。将来性も考慮して、セッション・オブジェクトのシリアライズ化を検討してください。

3.3. セッション DB の設定

前述のとおりセッション・パーシスタンスのデータストアにはいくつか選択肢がありますが、最もポピュラーなセッション DB の設定方法をご紹介します。

セッション DB を使用するためには、まずセッション格納用のデータベースを用意します。データベースが WAS のサポート対象であれば、セッション専用のデータベースを用意することも業務アプリケーションのデータベースと共存させることも可能です。データベースの運用方針などを考慮して選択してください。
セッション DB が DB2 の場合、表スペースとページ・サイズは WAS の管理コンソールから設定できます。Oracle の場合には、ユーザー作成時に指定したものが使用されます。
テーブルと索引は、アプリケーション・サーバー始動時に自動で作成されます。テーブル名はデフォルトで SESSIONS ですが、SessionTableName カスタム・プロパティーで変更可能です。索引も、SessionTableSkipIndexCreation カスタム・プロパティーを使用して手動作成も可能ですが、自動作成させることが推奨されています。

データベースを作成したら、WAS 上に、セッション DB へアクセスするための XA 非対応データ・ソースを設定してください。アプリケーションのデータ・ソースと同様に JDBC の最大接続数やステートメント・キャッシュ・サイズなどを、必要に応じて調整してください。詳細は虎の巻第 3 回を参考にしてください。

データ・ソースを設定したら、第 1 章でご紹介した「分散セッション」のパラメーターを設定します。「分散セッション」で「データベース」をクリックします。

するとデータベース設定画面が表示されます。ここでセッション DB に関する設定をします。

あらかじめ設定したデータ・ソースの JNDI 名、セッション DB へアクセスするユーザーIDとパスワードを指定してください。

- **DB2 行サイズ、テーブルスペース名**

  前述のとおりセッション DB が DB2 の場合に使用できるパラメーターです。

- **複数行スキーマを使用する**

  セッション・オブジェクトのサイズに関連する重要なパラメーターです。デフォルトでは、セッション・オブジェクトはデータベースの 1 行に格納されます。セッション・テーブルは、1 行に対して 2MB までのデータが格納できるように作成されるため、セッション・オブジェクトのサイズは 2MB までという制限があります。
  このオプションを選択すると、セッションが 1 行に格納されるのではなく、セッションの各属性が 1 行に格納されます。セッション・オブジェクト・サイズの制限はなくなりますが、データベースの使用サイズが増える点に注意してください。
  下記リンクに考慮点がまとめられています。

  複数行スキーマへの切り替え
  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Ftprs_swsk.html

3.4. セッション・パーシスタンスの調整

セッション・パーシスタンスを使用する場合は、以下の 3 点を調整します。
これらの設定は、分散環境設定のカスタム・チューニング・パラメーターで指定します。

- **書き込み頻度**

  どのような頻度でセッション・データを外部データストアへ書き込むか、を指定します。

  ・サーブレット・サービスの終了

  サーブレット処理の完了後に、セッション・データが書き込まれます。End of servlet service を略して EOS と呼ばれることがあります。

  ・手動更新

  サーブレットが IBMSession.sync メソッドを呼び出すときのみ、セッション・データを書き込みます。

  ・時間基準

指定された時間間隔で、セッション・データが書き込まれます。Time-based write で TBW と略されることがあります。
TBW を選択した場合、セッション・データの整合性を保つには、セッション・アフィニティー機能が必要です。また、「メモリー内の最大セッション・カウント」もアクティブなセッションを格納できるように十分に確保する必要があります。この値が十分でないと、新しいセッションが生成される度に、古いがアクティブであるセッション・オブジェクトを外部データストアへ書き出す処理が発生してしまうからです。

セッションの「last access time」属性は、そのほかのセッション・データの変更有無に関わらず、サーブレットや JSP からアクセス（getSession()）されると必ず更新されます。セッションをタイムアウトさせないためにこのような動作をします。
従って、EOS を選択した場合は、サーブレットや JSP がアクセスされる度に必ず更新処理が発生します。TBW の場合は、データ変更部分が累積されて更新されるため、書き込み頻度が抑えられパフォーマンス向上が見込めます。しかし、TBW や手動更新では、アプリケーション・サーバーがフェイルオーバーした場合にセッション･データが失われる可能性があり可用性は低下します。

- **書き込みの内容**

どのデータを外部へ書き込むか、を指定します。
・更新された属性のみ
setAttribute() と removeAttribute()で更新された HTTPSession の属性のみを外部データストアへ書き出します。「複数行スキーマを使用する」オプションを選択している場合は、そのセッション属性の行のみが更新されるので有用です。
setAttribute()を Call しない場合コンテナが属性の変更を認識できないため、アプリケーション･ロジックでは setAttribute()処理をバイパスしないように注意してください。
setAttribute()を使用せずにセッション情報が更新されている可能性がある場合は、「すべてのセッション属性」を選択してください。

・すべてのセッション属性
すべての HTTPSession の属性を外部データストアへ書き出します。

アプリケーション･コーディングの柔軟性の観点からは、「すべてのセッション属性」を選択したほうがよいですが、更新データ量が多くなるためパフォーマンスが劣化する可能性があります。「書き込み頻度」とあわせてご検討ください。

• セッション・クリーンアップのスケジュール

データベースまたは別のアプリケーション・サーバー・インスタンスの無効なセッションをクリーンアップする設定です。

セッション無効化には、invalidate()メソッドを使用する方法と、タイムアウト後にリーパー・スレッドで無効化される方法をすでにご紹介しました。セッション・パーシスタンスを使用する場合は、このパラメーターを使用してクリーンアップのタイミングを制御することができます。

4. セッション管理の考慮点

この章では、パフォーマンス上の一般的な考慮点をご紹介します。

4.1. セッション・サイズ

セッション・オブジェクトのサイズが大きいと、それだけメモリーを消費します。
たとえば、30 分に 100 ユーザーがサイトを訪れ、1 ユーザーが 1MB のセッション・オブジェクトを必要とし、セッション・タイムアウトが 30 分であれば、最初の 30 分で 100MB のメモリーが必要となります。使用メモリーは、1 ユーザーのセッションがタイムアウトせずに継続するほど、新規ユーザーがアクセスするほど増えていきます。
セッションの使用メモリーが増えれば、アプリケーション実行に関わる他のタスクが使用するメモリー領域が減ってしまいます。これを防ぐには、いくつかの選択肢が考えられます。

- セッション・オブジェクトのサイズを削減する
- メモリー上のセッション数を削減する
- アプリケーション・サーバーを追加する
- 不要なセッションを適切なタイミングで無効化する
- 使用できるメモリーを増やす
- セッション・タイムアウトの時間を短くする

セッション・オブジェクトのサイズは、セッション・パーシスタンス使用時には特に重要です。メモリー使用量への影響だけではなく、大きなデータをシリアライズして外部データストアに書きこむ処理にはオーバーヘッドがかかるからです。
一般的には、容易に取得できるデータや不要なデータはセッション・オブジェクトから削除して、セッションで管理するものは極力限定することが望まれます。

セッション・オブジェクトのサイズや数は、WAS の管理コンソールから確認できます。
WAS の Tivoli Performance Viewer というモニタリング・ツールで、「サーブレット・セッ

ション・マネージャー」の「LiveCount」（セッション数）と「SessionObjectSize」（セッション・オブジェクト・サイズ）を確認してみてください。デフォルトでは使用不可になっているため、使用可能に設定してください。

Tivoli Performance Viewer を使用したパフォーマンスのモニター
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Ftprf_tpvmonitor.html

サーブレット・セッションのカウンター
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Frprf_datacounter6.html

また、セッションの中身を確認する JSP ファイルもあわせてご紹介します。

WAS 小ワザ集: 第 13 回：HttpSession に格納されているオブジェクトを調査する
http://www.ibm.com/developerworks/jp/websphere/library/was/was_tips/13.html

4.2. セッションの共有

サーブレット 2.3 では、セッション・オブジェクトにアクセスできるのは同一 Web モジュールからと定義されています。従って、Web モジュールを跨ったセッションの共有はできません。
WAS では、共有セッション・コンテキストという拡張機能を使用すると、アプリケーション間でセッション・データを共有することができます。この機能を使用するには、IBMApplicationSession という WAS 固有の API を使用する必要があります。

詳細は、下記リンクをご参照ください。

セッション・データを共有可能にするアセンブル
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Frprf_datacounter6.html

4.3. 障害復旧後のリクエスト・ディスパッチ動作の指定

WAS 虎の巻第 2 回でご紹介したように、サーバー障害が発生した場合、クライアントからのリクエストがフェイルオーバーして処理が継続します。サーバー障害復旧後には元のサーバーにフェイルバックします。

NoAffinitySwitchBack カスタム・プロパティーを true にすると、リクエストは、復旧したサーバーへフェイルバックせずに、フェイルオーバーしたサーバーへ転送され続けます。

4.4. JSESSIONID のフォーマット変更

WAS 虎の巻第 2 回では、セッション ID のフォーマット例を紹介しました。

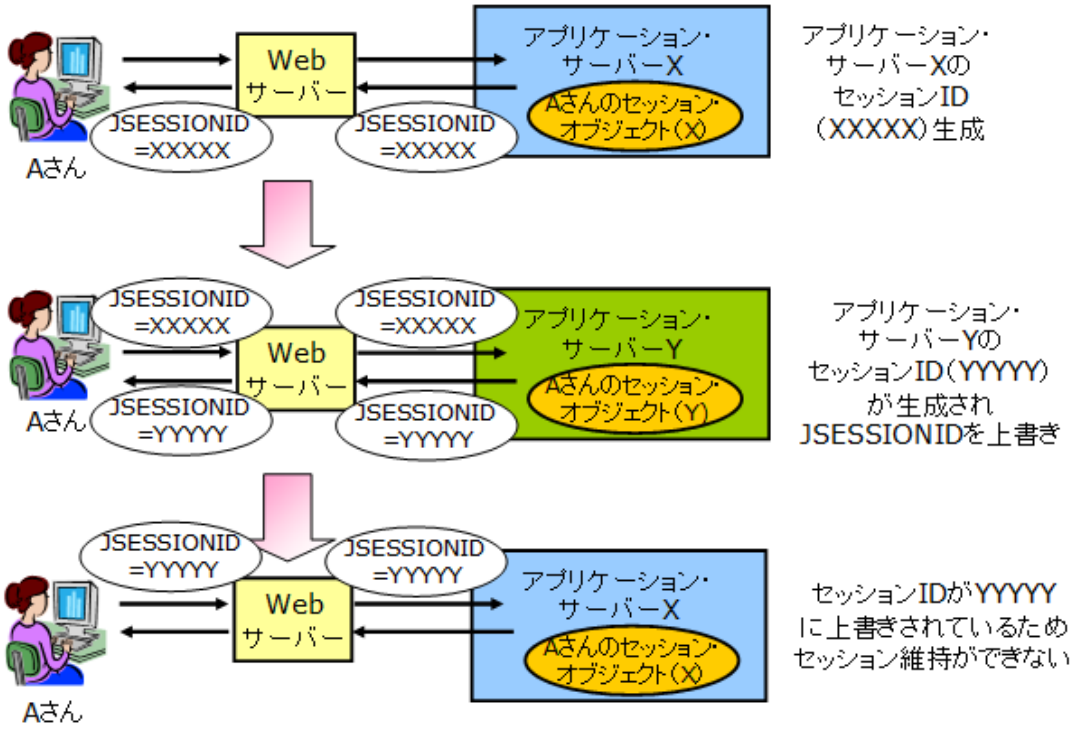
セッション ID : CacheID + HTTP セッション ID + クローン ID
例：JSESSIONID=0000A2MB4IJozU_VM8IffsMNfdR:v544d0o0

これらのフォーマットはアプリケーションの要件に応じてカスタマイズすることが可能です。

- HttpSessionCloneId カスタム・プロパティー：
  クローン ID は自動生成されますが、このカスタム・プロパティーで指定した文字列で上書きできます。
- HttpSessionIdLength カスタム・プロパティー：
  セッション ID の長さを指定できます。デフォルトは 23 文字です。
- CloneSeparator と CloneSeparatorChange カスタム・プロパティー：
  セッション ID とクローン ID の間はコロン (:)で分離されますが、これを変更できます。

4.5. 複数のアプリケーションで JSESSIONID を使用する場合

同一ドメインに WAS 上のアプリケーションが複数稼動している場合、Cookie が同一ドメイン、同一パス、同一 Cookie 名の場合は、JSESSIONID が上書きされてしまい、セッション管理ができないことがあります。



この解決策のひとつは、Cookie パスをアプリケーション毎に指定することです。

また、セッション・パーシスタンスが構成されていない環境であれば、HttpSessionIdReuse カスタム・プロパティーを true に設定すると、ブラウザーから送信されたセッション ID を使用してセッション情報が保管でき、複数のアプリケーションでもセッションを維持させることができます。

4.6. その他の注意点

HTTP セッションを使用すると、フレームを使用する Web ページのセッション管理が問題になったり、セッション・データ・クロスオーバー（あるクライアントに別のセッション・データが表示される）が問題になったりすることがあります。
このような HTTP セッションに関わる様々な問題については、以下のリンクにガイドがまとめられていますので、参考にしてください。

HTTP セッション使用のベスト・プラクティス
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Fcprs_best_practice.html

HTTP セッションの問題
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Frtrb_httpsessprobs.html

セッション・パーシスタンスのパフォーマンスについては下記ガイドをご参照ください。
セッション・データのサイズによる性能比較、EOS と TBW の性能比較、セッション DB とメモリー間複製の性能比較などが紹介されています。
WebSphere Application Server V8 Session Replication Performance
ftp://public.dhe.ibm.com/software/webservers/appserv/was/WASV8_SessionReplication_Performance_v1.0.pdf