Javaの理論と実践: JTSを理解する -- 見えない魔法

J2EEコンテナーは複雑なトランザクション管理をどのように隠しているか

Brian Goetz
Principal Consultant
Quiotix

2002年 4月 01日

トランザクションに関するこのシリーズの第1回では、トランザクションとは何か、また、信頼性の高い分散アプリケーションを構築する上でなぜそれが重要になるのか、という点について基本となるポイントを説明しました。今回の記事では、J2EEアプリケーションがどのようにトランザクションとして構造化されるか、またJTSとJ2EEコンテナーが、トランザクション境界の設定やリソースの使用、トランザクションの伝搬などのトランザクション・サービスを、コンポーネント・プログラマーにほとんど意識させずにどのようにして管理するのかについて説明します。

このシリーズの他の記事を見る

このシリーズの第1回では、トランザクションについて考え、その基本特性である原子性、一貫性、独立性、持続性について説明しました。トランザクションは、企業用アプリケーションの基本となるビルディング・ブロック（構成要素）です。トランザクションがなければ、耐障害性をもつ企業用アプリケーションの作成はほとんど不可能です。幸いJava Transaction Service (JTS) とJ2EEコンテナーがトランザクション管理作業の大半を自動的に行ってくれるので、トランザクションに対する意識をコンポーネント・コードに直接組み込む必要はありません。その結果はまるで魔法のようです。いくつかの簡単なルールに従うことで、J2EEアプリケーションは、ほとんど、または、まったくコンポーネント・コードへの追加をすることなく、トランザクション・セマンティクスを自動的に手に入れることができます。この記事では、トランザクション管理がどこでどのように発生するかを示し、この魔法のなぞを解いていきます。

JTSとは

JTSは、コンポーネント・トランザクション・モニターです。これはどういう意味でしょうか。第1回では、アプリケーションのために分散トランザクションの実行を調整するプログラムであるトランザクション処理モニター (TPM) の概念を紹介しました。TPMにはデータベースと同じくらい長い歴史があります。最初にIBMが1960年代後半にCICSを開発しました。これは今日も使用されています。従来の (または手続き型の) TPMは、(データベースなどの) トランザクション・リソース上でオペレーションのシーケンスとして手続き型で定義されるトランザクションを管理します。CORBAやDCOM、RMIなどの分散オブジェクト・プロトコルが現れ、よりオブジェクト指

© Copyright IBM Corporation 2002

向的なトランザクションに対する見方が望まれるようになりました。トランザクションのセマンティクスをオブジェクト指向のコンポーネントに与えるために、TPMモデルの拡張が必要になりました。このモデルでは、トランザクションは、従来の方法とは異なり、トランザクション型のオブジェクトに対するメソッド呼び出しという観点から定義されています。まさにこれがJTSです。つまりコンポーネント・トランザクション・モニター (オブジェクト・トランザクション・モニターとも呼ばれます) 、すなわちCTMです。

JTSとJ2EEのトランザクション・サポートの設計は、CORBA Object Transaction Service (OTS) に大きく影響されました。実際、JTSはOTSを実装し、トランザクション境界を定義する下位レベルAPIのJava Transaction APIとOTSの間のインターフェースとして機能します。新しいオブジェクト・トランザクション・プロトコルを発明する代わりにOTSを使用することによって、既存の標準を基礎とし、J2EEコンポーネントとCORBAコンポーネントの間の互換性への道が開かれます。

一見したところ、手続き型トランザクション・モニターからCTMへの移行は、用語上の変更にすぎないように思えます。しかし、その違いはもっと重要なものです。CTMのトランザクションがコミットまたはロールバックを行う場合、そのトランザクションにかかわるオブジェクトによって行われたすべての変更は、グループとしてコミットされるか、または取り消されます。しかし、そのトランザクションの間にオブジェクトが何を行ったかをCTMはどのようにして知るのでしょうか。EJBコンポーネントのようなトランザクション・コンポーネントは、`commit()` や`rollback()` のメソッドを持っていませんし、何を行ったかをトランザクション・モニターに登録することもしません。では、J2EEコンポーネントによって実行されたアクションは、どのようにしてトランザクションの一部になるのでしょうか。

透過的なリソースの徴集

アプリケーションの状態はコンポーネントによって操作されますが、この状態は、やはりトランザクション・リソース・マネージャー (たとえばデータベースやメッセージ・キュー・サーバー) に格納されています。そして、このトランザクション・リソース・マネージャーは、分散トランザクションの中でのリソース・マネージャーとして登録可能です。第1回では、複数のリソース・マネージャーが単一のトランザクションでどのように使用され、トランザクション・マネージャーによって調整されるかを説明しました。リソース・マネージャーは、アプリケーションの状態の変更を特定のトランザクションに関連付ける方法を認識しています。

しかしこれでは、トランザクションにかかわっているリソースをコンテナーがどのように認識し、使用することができるのか、という疑問の焦点をコンポーネントからリソース・マネージャーに移したにすぎません。典型的なEJBセッションBeanに見られる次のようなコードについて考えてみましょう。

リスト1. Bean管理のトランザクションによる透過的なリソースの使用

```
InitialContext ic = new InitialContext();
UserTransaction ut = ejbContext.getUserTransaction();
ut.begin();
DataSource db1 = (DataSource) ic.lookup("java:comp/env/OrdersDB");
DataSource db2 = (DataSource) ic.lookup("java:comp/env/InventoryDB");
Connection con1 = db1.getConnection();
Connection con2 = db2.getConnection();
// perform updates to OrdersDB using connection con1
// perform updates to InventoryDB using connection con2
ut.commit();
```

この例には、現在のトランザクションでのJDBC接続を集め挙げるようなコードがありません。コンテナーがそれを行ってくれます。それがどのように行われるかを見てみましょう。

3種類のリソース・マネージャー

EJBコンポーネントは、データベース、メッセージ・キュー・サーバー、あるいはそれ以外のトランザクション・リソースにアクセスする場合に、(通常はJNDIを使用して) リソース・マネージャーへの接続を獲得します。さらに、J2EE仕様は、JDBCデータベース、JMSメッセージ・キュー・サーバー、および「JCAを介してアクセスするその他のトランザクション・サービス」の3種類のトランザクション・リソースのみを認識します。このなかの最後のクラスのサービス (ERPシステムなど) は、JCA (J2EE Connector Architecture) を介してアクセスされるものでなければなりません。これらの種類の各リソースに対して、コンテナーまたはプロバイダーは、トランザクションでリソースを集め挙げるのに役立ちます。

リスト1では、`con1` と`con2` は、`DriverManager.getConnection()` から戻された普通のJDBC接続のように見えます。これらの接続は、JNDIでデータ・ソース名を検索して得られたJDBC`DataSource` から得られます。データ・ソース (`java:comp/env/OrdersDB`) を見つけるためにEJBコンポーネントのなかで使用される名前は、そのコンポーネントに固有のものです。コンポーネントの配備記述子(deployment descriptor)の`resource-ref` セクションが、その名前を、コンテナーによって管理されアプリケーション全体の範囲に対して定義される`DataSource` のJNDI名に対応付けます。

隠れたJDBCドライバー

すべてのJ2EEコンテナーは、トランザクションを意識できプールされた`DataSource` オブジェクトを作成することができますが、それがどのように行われるかについては、J2EE仕様外であるため示されていません。J2EEの文書を見ても、どのようにJDBCデータ・ソースが作成されるかはまったく説明されていません。そのため、あなたが使おうとするコンテナーに関する文書を調べなければならないでしょう。どのコンテナーを使用するかによって、データ・ソースの作成は、プロパティーまたは構成ファイルへのデータ・ソース定義の追加を含むかもしれないし、あるいはGUI管理ツールを使用して行われるのかもしれません。

各コンテナー (またはPoolManのような接続プール・マネージャー) は、`DataSource` を作成するための独自のメカニズムを持っています。JTAの魔法は、このメカニズム内に隠されているのです。接続プール・マネージャーは、指定されたJDBCドライバーから`Connection` を獲得しますが、それをアプリケーションに戻す前に、同じ`Connection` (のインターフェース) を実装するファサードでラップすることにより、アプリケーションと元になる接続の間に割り込みます。接続が確立するか、またはJDBCオペレーションが実行されると、ラッパーは、現在のスレッドがトランザクションのコンテキストで実行中かどうかをトランザクション・マネージャーに尋ね、それが存在する場合は、トランザクションに対してその`Connection` を自動的に登録します。

他のトランザクション・リソース、すなわちJMSメッセージ・キューとJCAコネクターも、リソースの使用をユーザーに隠すために、同様のメカニズムを利用しています。配備時にJ2EEアプリケーションがJMSキューを利用できるようにする場合は、プロバイダー固有のメカニズムを再び使用して、管理対象JMSオブジェクト (キュー接続ファクトリーと宛先) を作成し、JNDIネームスペースで公開します。プロバイダーによって作成された管理対象オブジェクトには、コンテナーが提供している接続プール・マネージャーにより追加されたJDBCラッパーと似た自動登録コードが含まれています。

透過的なトランザクション制御

コンテナー管理(container-managed)とBean管理(bean-managed)という2種類のJ2EEトランザクションは、トランザクションをどのように開始/終了するかにおいて異なります。トランザクションが開始/終了する場所は、トランザクション境界と呼ばれます。リスト1のコード例は、Bean管理によるトランザクション (プログラマチック・トランザクションとも呼ばれる) を示しています。Bean管理によるトランザクションは、`UserTransaction` クラスを使用して、コンポーネントによって明示的に開始/終了されます。EJBコンポーネントは`ejbContext` を介して`UserTransaction` を利用することができ、他のJ2EEコンポーネントはJNDIを介して利用することができます。

コンテナー管理によるトランザクション (または宣言トランザクション) は、コンポーネントの配備記述子のトランザクション属性に基づいて、コンテナーによってアプリケーションに意識させずに開始/終了されます。EJBコンポーネントが、Bean管理によるトランザクション・サポートを使用するか、あるいはコンテナー管理によるトランザクション・サポートを使用するかは、`transaction-type` 属性を`Container` または`Bean` のいずれかに設定することによって指定します。

コンテナー管理によるトランザクションによって、EJBクラスまたはEJBメソッドのレベルでトランザクション属性を割り当てることができます。また、EJBクラスのトランザクション属性のデフォルト・セットを指定したり、さらには、異なるメソッドが異なるトランザクション・セマンティクスを持つ場合に、それぞれのメソッドの属性を指定することもできます。これらのトランザクション属性は、アセンブリー記述子(assembly descriptor)の`container-transaction` セクションで指定されます。リスト2は、アセンブリー記述子の例です。`trans-attribute` のサポート値を以下に示します。

- `Supports`
- `Required`
- `RequiresNew`
- `Mandatory`
- `NotSupported`
- `Never`

`trans-attribute` は、メソッドがトランザクション内での実行をサポートするかどうか、メソッドがトランザクション内で呼び出される場合のコンテナーのアクション、また、メソッドがトランザクション外で呼び出される場合のコンテナーのアクションを決定します。最も一般的なコンテナー管理によるトランザクション属性は、`Required` です。`Required` が設定されると、処理中のトランザクションが、そのトランザクション内でBeanを登録しますが、トランザクションがすでに実行中でない場合は、コンテナーがそれをひとつ開始します。さまざまなトランザクション属性の違いと、それぞれを使用するケースについては、第3回で説明します。

リスト2. EJBアセンブリー記述子の例

```
<assembly-descriptor>
  ...
  <container-transaction>
    <method>
      <ejb-name>MyBean</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
  <container-transaction>
    <method>
      <ejb-name>MyBean</ejb-name>
      <method-name>updateName</method-name>
      </method>
   <trans-attribute>RequiresNew</trans-attribute>
  </container-transaction>
  ...
</assembly-descriptor>
```

強力だが危険でもある

リスト1の例とは異なり、宣言トランザクション境界では、コンポーネント・メソッドにトランザクション管理コードがありません。このことによって (雑然としたトランザクション管理コードがないため)、結果として生じるコンポーネント・コードが読みやすくなるだけでなく、さらに重要なもう1つのメリットがあります。それは、コンポーネントのソース・コードを修正せずに、またそれにアクセスさえせずに、コンポーネントのトランザクション・セマンティクスをアプリケーション・アセンブリー時に変更できるという点です。

コードから離れてトランザクション境界を指定できることは、非常に強力な機能ですが、アセンブリー時に判断を誤ると、アプリケーションが不安定になり、パフォーマンスが大きく低下します。コンテナー管理によるトランザクションの境界を設定する際の正しさは、コンポーネント開発担当者とアプリケーション組立担当者の両者にゆだねられます。コンポーネント開発担当者は、コンポーネントが何を行うかについて十分な文書を作成すべきであり、そうすることによって、アプリケーション配備担当者は、アプリケーションのトランザクションを構造化する方法について優れた決定を行うことができます。またアプリケーション組立担当者は、コンポーネントがアプリケーションでどのようなインタラクションを行うかを理解する必要があります。そうすることによって、トランザクションはアプリケーションの一貫性を遵守し、パフォーマンスを低下させずに境界を設定することができるのです。これらの問題については、このシリーズの第3回で取り上げます。

透過的なトランザクションの伝搬

どちらの種類のトランザクションでも、意識せずにリソースを登録できます。コンテナーは、トランザクションの過程で使用されるトランザクション・リソースを現在のトランザクションで自動的に登録します。このプロセスは、リスト1で得られたデータベース接続などのトランザクション・メソッドによって使用されるリソースに拡張されるだけでなく、それが呼び出すメソッドによって使用されるリソースへも (リモート・メソッドでも) 拡張されます。それがどのように行われるかを見てみましょう。

コンテナーがトランザクションとスレッドを関連付ける

オブジェクトA の`methodA()` がトランザクションを開始し、オブジェクトB の`methodB()` を呼び出したとします。オブジェクトB の`methodB()` は、JDBC接続を獲得し、データベースを更新するものとします。B によって得られた接続は、A によって作成されたトランザクションで自動的に登録されます。コンテナーはこうする方法をどのようにして知ったのでしょうか。

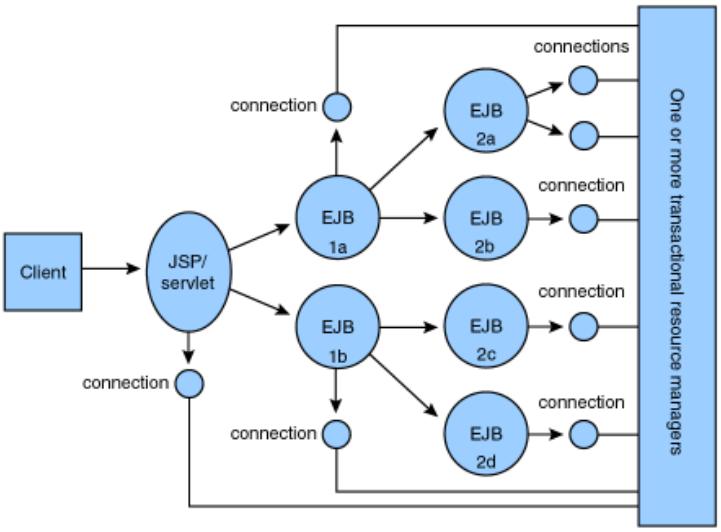
トランザクションが開始されると、トランザクション・コンテキストが、実行中のスレッドに関連付けられます。A がトランザクションを作成した場合、A が実行されているスレッドが、そのトランザクションに関連付けられます。ローカル・メソッドの呼び出しは、呼び出し側と同じスレッドで実行されるので、A によって呼び出されたメソッドは、どれもそのトランザクションのコンテキスト内にあります。

クローゼットの中の秘密

オブジェクトB が、別のスレッドや、あるいは別のJVMで実行中のEJBコンポーネントのスタブである場合はどうでしょうか。驚くべきことに、リモート・オブジェクトB によってアクセスされたリソースが、現在のトランザクションに登録されます。EJBオブジェクト・スタブ (呼び出し側のコンテキストで実行する部分)、EJBプロトコル (RMI over IIOP)、およびリモート側のスケルトン・オブジェクトが協力して、これを意識させることなく行います。スタブは、呼び出し側がトランザクションを実行中であるかどうかを判定します。トランザクションを実行中である場合、トランザクションIDまたはXid が、IIOPコールの一部としてメソッド・パラメーターとともにリモート・オブジェクトに伝えられます (IIOPはCORBAリモート呼び出しプロトコルです。これは、トランザクション・コンテキストやセキュリティー・コンテキストなどの実行コンテキストのさまざまな要素を伝えます。RMI over IIOPに関しては、参考文献を参照してください)。呼び出しがトランザクションの一部である場合、リモート・システム上のスケルトン・オブジェクトは、リモート・スレッドのトランザクション・コンテキストを自動的に設定するので、実際のリモート・メソッドが呼び出された時点で、それはすでにトランザクションの一部です (スタブとスケルトン・オブジェクトも、コンテナー管理によるトランザクションの開始とコミットを管理します)。

トランザクションは、EJBコンポーネント、サーブレット、JSPページ (あるいはコンテナーがサポートする場合は、アプリケーション・クライアント) のどのJ2EEコンポーネントによっても開始することができます。つまり、要求が着くと、アプリケーションは、サーブレットまたはJSPページでトランザクションを開始し、サーブレットまたはJSPページ内で処理を行い、ページのロジックの一部として複数のサーバー上のエンティティーBeanやセッションBeanにアクセスし、この作業をすべて1つのトランザクションの一部としますが、これを意識させないように行うことができます。図1は、トランザクション・コンテキストが、サーブレットからEJBそして次のEJBへとどのような実行のパスをたどるのかを示しています。

図1. 単一のトランザクション内の複数のコンポーネント



最適化

コンテナーがトランザクションを管理するということで、コンテナーは、ある程度の最適化の決定をすることができます。図1では、サーブレットと複数のEJBコンポーネントは、単一のトランザクションのコンテキスト内でデータベースにアクセスします。それぞれがデータベースへの`Connection`を獲得し、まさに同じデータベースにアクセスします。異なるコンポーネントのから同じリソースに対して複数の接続が行われている場合でも、JTSは、複数のリソースがトランザクションにかかわっているのか否かを検出し、トランザクションの実行を最適化することができます。第1回で説明したように、複数のリソース・マネージャーが単一のトランザクションにかかわるには、2フェーズ・コミット・プロトコルの使用が必要でした。これは、単一のリソース・マネージャーによって使用される1フェーズ・コミットよりも高価です。JTSは、トランザクションで単一のリソース・マネージャーのみが使用されているかどうかを判定することができます。JTSは、トランザクションにかかわるすべてのリソースが同じであることを検出した場合、2フェーズ・コミットをスキップし、リソース・マネージャーが自らトランザクションを処理できるようにします。

結論

トランザクション制御、リソースの使用、およびトランザクションの伝搬を意識させずに行える魔法は、JTSの一部ではなく、J2EEコンテナーが、J2EEアプリケーションのために陰でどのようにJTAサービスとJTSサービスを使用するかという点に含まれます。EJBスタブとスケルトン、コンテナー・ベンダーによって提供されるJDBCドライバー・ラッパー、データベース・ベンダーによって提供されるJDBCドライバー、JMSプロバイダー、およびJCAコネクターなどの多くのエンティティーが、この魔法を意識させずに行えるように陰で協力しています。これらのエンティティーはすべてトランザクション・マネージャーとインタラクションを行うので、アプリケーション・コードがそれを行う必要はありません。

第3回では、J2EEコンテキストにおけるトランザクションの管理に関連する実際の問題としてトランザクション境界および独立性について、そしてアプリケーションの一貫性、安定性、およびパフォーマンスに対するその影響について取り上げます。

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。

© Copyright IBM Corporation 2002
(www.ibm.com/legal/copytrade.shtml)
商標
(www.ibm.com/developerworks/jp/ibm/trademarks/)