Designing Web System Infrastructure with WAS V8.0

WASV8.0によるWebシステム基盤設計Workshop

トランザクション設計

Designing Web System Infrastructure with WAS V8.0

当セッションの目的

- トランザクションの基本概念を理解する。
  - トランザクションの考え方
  - グローバル・トランザクションとローカル・トランザクション
- WAS V8.0で提供されるトランザクション機能について理解する。
  - JTA1.1に準拠した機能とWASの独自実装について
- WAS V8.0でトランザクション機能利用時に考慮すべき設計項目を理解する。
  - アプリケーション、障害対策、運用管理面から

2

当セッションの目的です。

Designing Web System Infrastructure with WAS V8.0

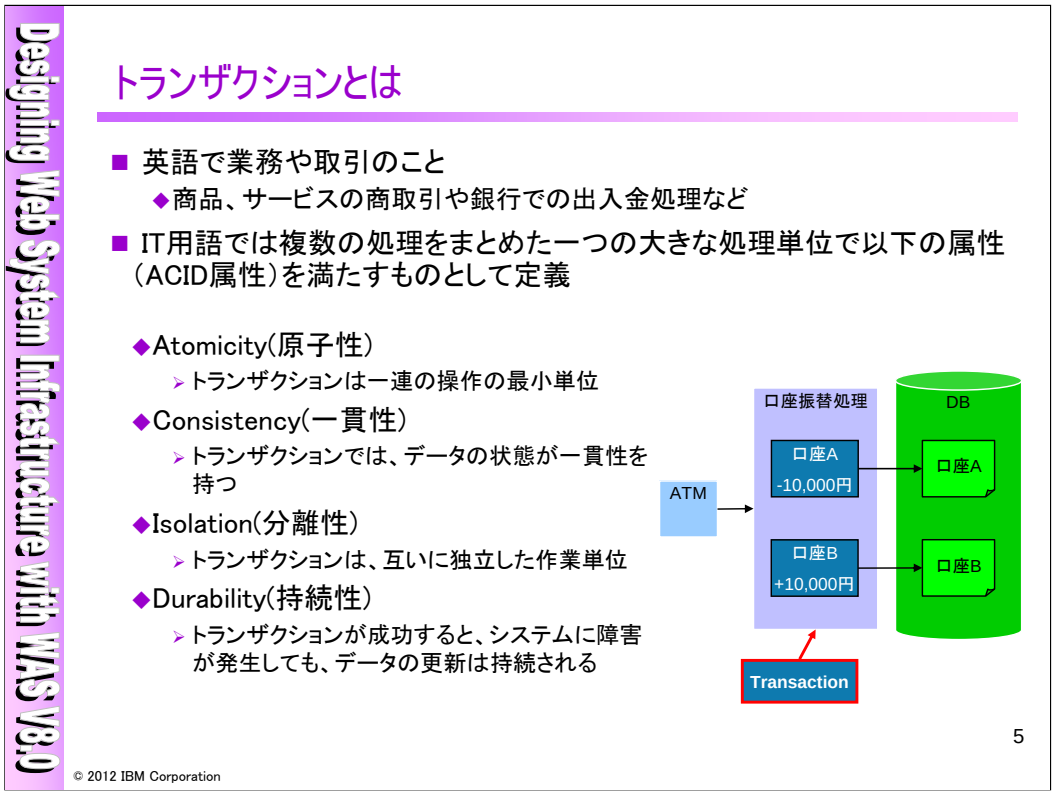
Agenda

3

Designing Web System Infrastructure with WAS V8.0

1. トランザクションの基礎

4



トランザクションとは英語で業務や取引という意味になり、店頭での商品売買取引、銀行での口座出入金処理などをさします。これらの取引、業務は複数の細かい処理（振込みの際の口座Aからの出金処理と口座Bへの入金処理など）などから成り立っており、IT用語としてはこれら複数の細かい処理をまとめた一つの処理単位をさし、以下のACID属性と呼ばれる性質を持つものとして定義されます。

・Atomicity(原子性)

トランザクションは一連の操作の最小単位になります。トランザクションの一部分だけが実行されても意味がありません。（口座Aからお金が引かれても口座Bにお金が加算されなければ処理が成り立ちません。）

・Consistency(一貫性)

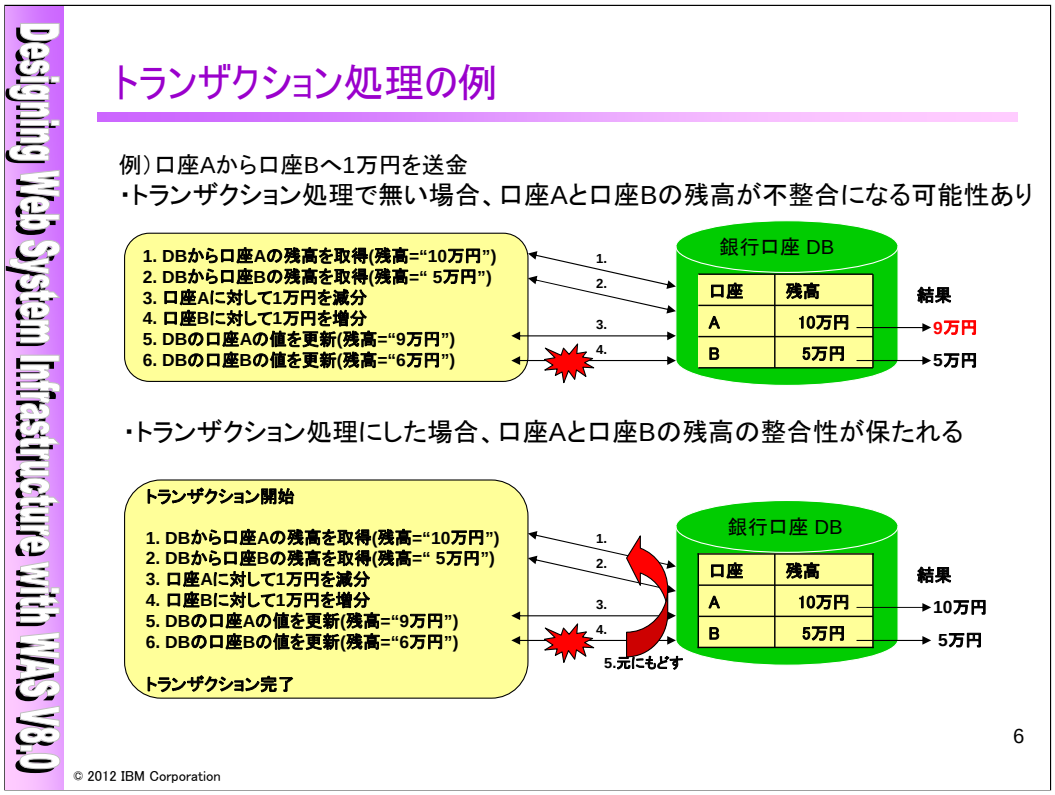
トランザクションでは、処理の前後でデータの一貫性が保たれなければいけません。（口座Aと口座Bの合計が、トランザクション処理の前後で異なっていては、処理が成り立ちません。）

・Isolation(分離性)

トランザクションは、独立した作業単位になり、複数のアプリケーションにより並行してトランザクション処理が実施されていても、その処理がお互いに影響を及ぼさない必要があります。

・Durability(持続性)

トランザクションは回復の処理単位になり、トランザクション処理が正常に終了したら、システムに障害が発生してもそのデータの更新は持続される必要があります。

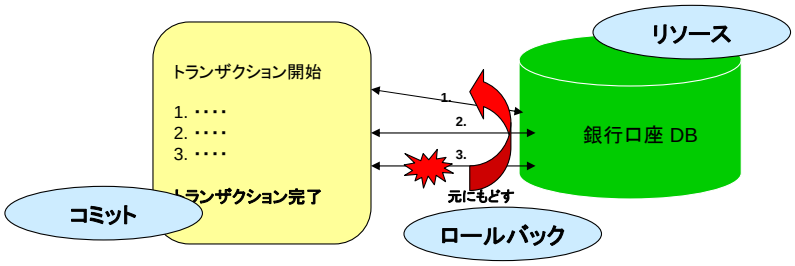


上図は、トランザクション処理でなければ、原子性を保つことが出来ないという例になります。トランザクション処理であれば実行結果は、口座A=10万円/口座B=5万円か、口座A=9万円/口座B=6万円のどちらかになります。

Designing Web System Infrastructure with WAS V8.0

トランザクション基本用語

- リソース
  - トランザクション処理中にデータ更新の対象となるもの
  - データベースやMQなどのメッセージング基盤など
- コミット（commit）
  - トランザクション処理が成功したときにリソースの更新内容を確定すること
- ロールバック(rollback)
  - トランザクション処理が失敗したときに途中のリソースに対する更新内容を破棄してもとのトランザクション処理前の状態のもどすこと



7

このセッションで使用するトランザクション用語を説明します。

・リソース

リソースとはトランザクション処理中にデータ更新の対象となるものをいいます。具体的にはDB2、OracleなどのデータベースやMQやME　といったメッセージング基盤製品などです。リソース・マネージャーと呼ばれることもあります。

・コミット

トランザクション確定のことをコミットといいます。トランザクション処理中の更新内容はコミット処理をすることによって確定され、各リソースに更新内容が反映されます。

・ロールバック

トランザクション処理中に障害による不整合の発生などでトランザクションを中断させたいときには、ACID属性の制約から一部の更新内容だけを反映させることはできません。このとき、更新内容を処理前の整合性のとれた状態に戻す必要があり、この巻き戻し処理のことをロールバックと呼びます。

Designing Web System Infrastructure with WAS V8.0

トランザクションの種類

- ローカル・トランザクション
  - 1つのリソースを対象としたトランザクション
  - 1フェーズ・コミット(1PC)
- グローバル・トランザクション
  - 複数のリソースにまたがるトランザクション
  - トランザクションをコントロールするトランザクション・マネージャーが存在
  - 2フェーズ・コミット(2PC)

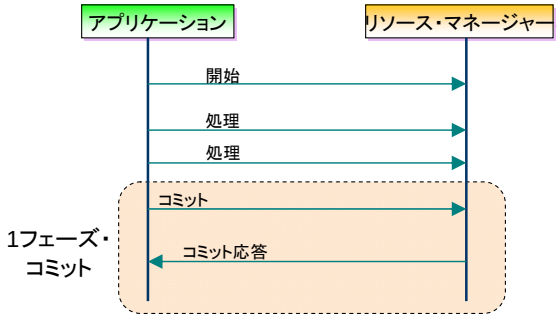
8

トランザクションには大きく分けて2種類あります。ひとつめは1つのリソースを対象としたものでローカル・トランザクションと呼ばれます。もうひとつは複数のリソースを対象としたトランザクション処理でグローバル・トランザクションと呼ばれます。

Designing Web System Infrastructure with WAS V8.0

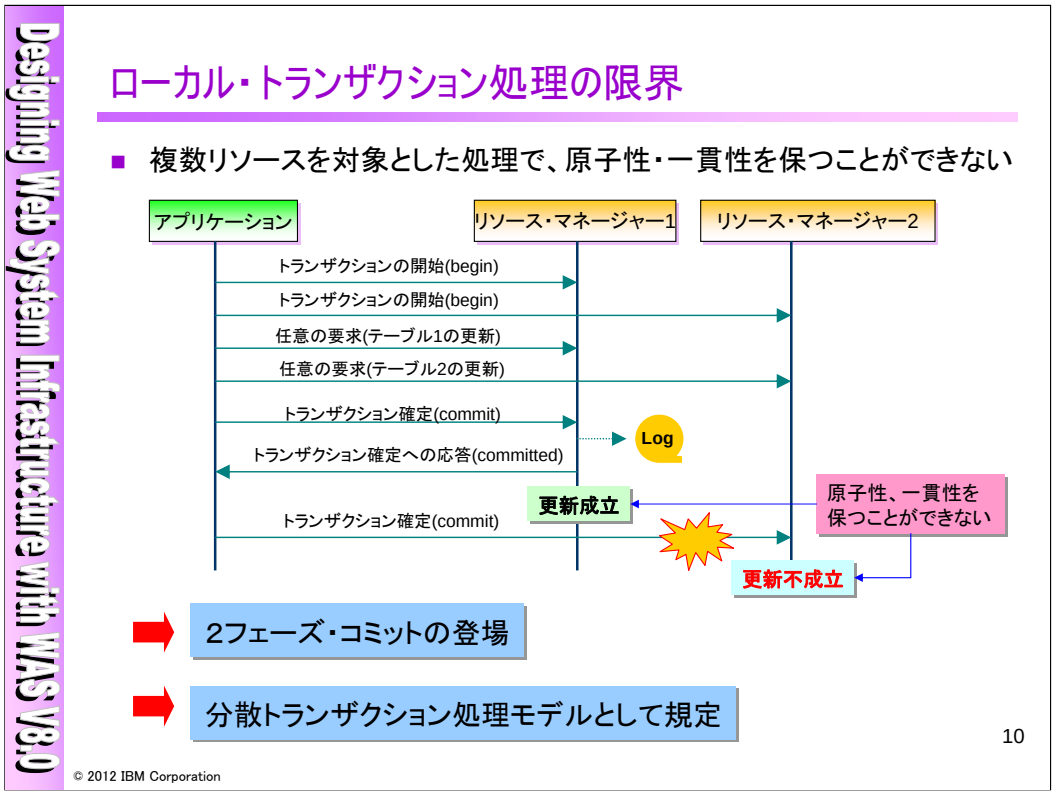
ローカル・トランザクション

- 1つのリソース内で完結
  - DB2、MQなど単一のリソース内でトランザクション処理は完結
  - リソースは**リソース・マネージャー**(RM)とも呼ばれる
- トランザクションの開始やコミットはアプリケーションから指示
  - コミット処理は簡潔 (1フェーズ・コミット)
- データの整合性はリソース・マネージャの責任
  - リソース・マネージャーがトランザクション処理のログを出力
  - Atomicity/Durability属性の保証
  - 更新ログ
    - Beforeイメージ
    - Afterイメージ
  - 同期点制御ログ
    - Beginログ
    - Syncpointログ



9

ローカル・トランザクションは1つのリソースを更新対象とするトランザクション処理です。この場合には各リソースが主体となってトランザクションを管理します。リソースはリソース・マネージャーとも呼ばれます。リソース・マネージャーはアプリケーションからトランザクションの開始やコミット要求を受付け、各処理ごとにログ出力をして状態を管理します。

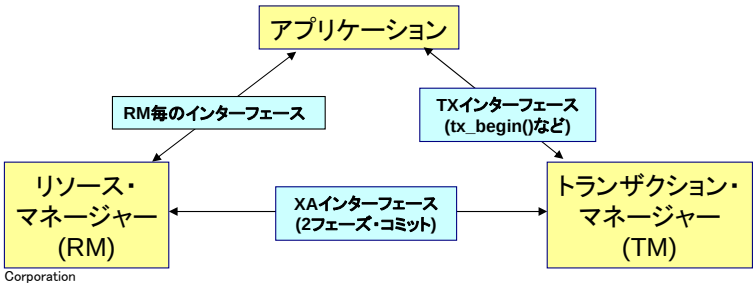


ローカル・トランザクション処理の場合は、複数リソースを対象にすると、原子性・一貫性を保つことが出来ないという問題が発生します。この問題を解決する為に、2フェーズ・コミット(2PC)が登場し、この処理モデルが分散トランザクション処理モデルとして規定されました。

Designing Web System Infrastructure with WAS V8.0

分散トランザクション処理モデルの定義

- The Open Groupが仕様を規定
  - Distributed Transaction Processing:The XA Specification
    - https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?catalogno=c193
  - オープン環境における2フェーズ・コミット・プロトコルを標準化した規格
- 定義していること
  - サブ・システムの役割の明確化
    - トランザクション・マネージャー(TM) ・・・調停者(製品例: WAS, TXSeries, WebSphere MQ…)
    - リソース・マネージャー(RM) ・・・被調停者(製品例:DB2, WebSphere MQ, Oracle,…)
  - サブ・システム間のインターフェース(XAインターフェース)
    - 分散トランザクション処理モデル
    - アプリケーションからは透過的(不可視)のやりとり



11

分散トランザクション処理モデルとは、The Open Groupが規定した処理モデルになり、Java EE(JTA)は分散トランザクション・モデルに基づいたトランザクション処理をサポートします。各コーディネーターは、それぞれ以下の役割があります。

＜アプリケーション＞

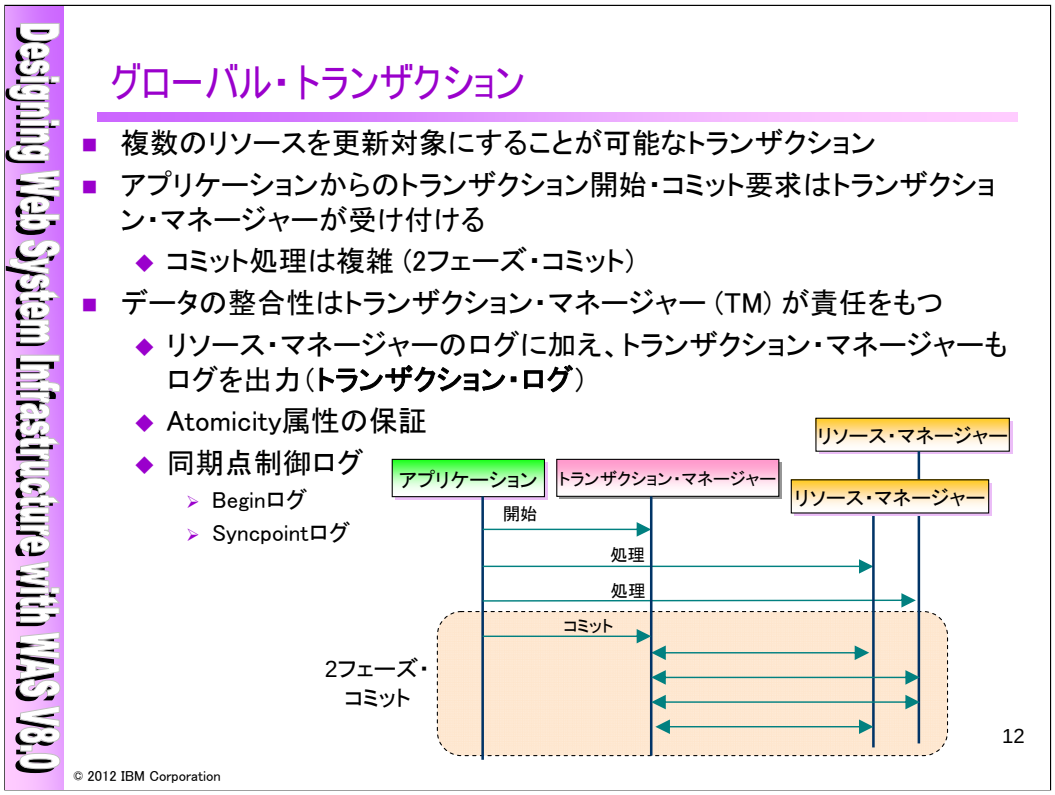
TMに対して、TXインターフェースを使用して、トランザクションの開始/終了/commit/rollback等の指示を行います。

RMに対して、RM毎のインターフェースを使用して、アプリケーションによるDB参照/更新等のビジネス・ロジックを実施します。

＜トランザクション・マネージャー/リソース・マネージャー＞

アプリケーションからのトランザクションの開始/終了/commit/rollback等の指示に対して、TMとRMが協業し、実際のトランザクション処理をコーディネートします。

TMとRM間の遣り取りにおいて、XAインターフェースが規定されており、その中で2フェーズ・コミット・プロトコルが規定されています。



グローバル・トランザクションは複数のリソースを更新対象とすることが可能なトランザクション処理です。この場合にはトランザクション全体を制御するトランザクション・マネージャーが存在します。トランザクション・マネージャーがアプリケーションからトランザクションの開始やコミット要求を受付け、リソース・マネージャーを指示しつつトランザクション全体の調停をおこないます。グローバル・トランザクションの場合は、トランザクション・マネージャーもログを出力します。また、コミット時には2フェース・コミットと呼ばれる方法が使用されます。これは前頁のローカル・トランザクション時の複数リソース更新での問題点を解決します。ちなみに更新対象リソースが1つの場合はパフォーマンスの理由からコミット処理は1フェーズに最適化されます。

Designing Web System Infrastructure with WAS V8.0

2フェーズ・コミット

- 2フェーズ・コミット・プロトコルとは
  - トランザクション・マネージャーとリソース・マネージャー間のトランザクションを調整するプロトコル
  - 複数リソースにわたる更新を一つのトランザクション・スコープ (UOW) として整合性を保って行うためのプロトコル
- 特徴
  - 2つのフェーズから成るトランザクション調停
    - 準備(Prepare)・フェーズ
      - トランザクション・マネージャー・・・変更をコミットすることをすべてのリソース・マネージャーに対して通知し、同意を取り付ける
      - リソース・マネージャー・・・変更を永続的に保持できるように準備する
    - コミット(commit)・フェーズ
      - トランザクション・マネージャー・・・リソース・マネージャーの同意に基づいて、変更のコミットを指示する
      - リソース・マネージャー・・・変更が確定したことをログに書き出し、保持していたロックを開放する

13

2フェーズ・コミット・プロトコルとは、トランザクション・マネージャーとリソース・マネージャー間のトランザクションを調整するプロトコルであり、複数のリソース・マネージャーにまたがる更新を一つのUOW(UnitOfWork)として、整合性を保って行うためのプロトコルになります。特徴として、以下の2つのフェーズから成るトランザクションをコーディネートします。

・準備(prepare)フェーズ

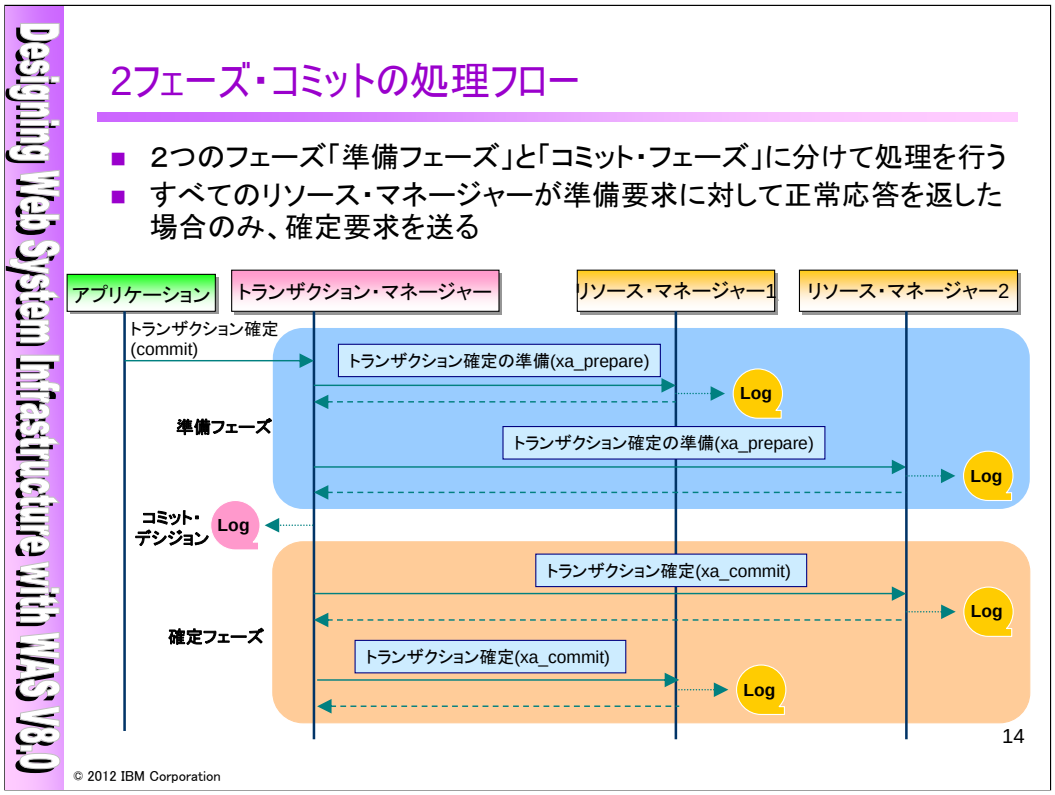
トランザクション・マネージャー・・・変更をコミットすることをすべてのリソース・マネージャーに対して通知し、同意を取り付ける

リソース・マネージャー・・・変更を永続的に保持できる状態にする

・コミット(commit)・フェーズ

トランザクション・マネージャー・・・リソース・マネージャーの同意に基づいて変更をコミットする

リソース・マネージャー・・・変更が確定したことをログに書き出し、保持していたロックを開放する



上図は、2フェーズ・コミットの処理フローです。

・トランザクション確定(commit)

アプリケーションからトランザクション・マネージャーに対して、"トランザクションの確定(commit)"処理が発行され、ここから2フェーズ・コミットの処理フローが開始されます。

＜準備フェーズ＞

・トランザクションの確定の準備(リソース・マネージャー1)

トランザクション・マネージャーからリソース・マネージャー1に対して、"トランザクションの確定の準備要求(xa_prepare)"処理が発行されます。この時、リソース・マネージャーは、要求に対して準備されているかをログに書込み、処理結果をトランザクション・マネージャーに返答します。

・トランザクションの確定の準備(リソース・マネージャー2)

同様に、トランザクション・マネージャーからリソース・マネージャー2に対して、"トランザクションの確定の準備要求(xa_prepare)"処理が発行されます。この時、リソース・マネージャーは、要求に対して準備されているかをログに書込み、処理結果をトランザクション・マネージャーに返答します。また、トランザクション・マネージャーは、両方のリソース・マネージャーから準備されているかの返答をログに書き込みます。両方のリソース・マネージャーが準備されている場合は、"トランザクションの確定(commit)"処理にてcommitされます。両方のリソース・マネージャーが準備されていない場合には、"トランザクションの確定(xa_commit)"処理にてrollbackされます。更にその時の情報がログに書き込まれます。

＜確定フェーズ＞

・トランザクション確定(リソース・マネージャー2) / トランザクション確定(リソース・マネージャー1)

"トランザクション確定の準備(xa_prepare)"処理にて確定した判断が、それぞれのリソース・マネージャーに対して実行され、一連のトランザクション処理が完了する流れとなります。

Designing Web System Infrastructure with WAS V8.0

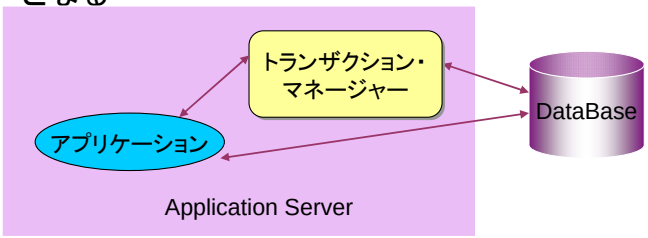
2. WAS V8.0のトランザクション機能概要

15

Designing Web System Infrastructure with WAS V8.0

トランザクション処理におけるWASの役割

- JTA（Java Transaction API）に準拠したトランザクション・サービスを提供する
  - WAS V8.0ではJTA1.1に準拠
- Java EEコンポーネントのアプリケーションの実行環境となる
  - トランザクション機能を利用するアプリケーションの実行環境を提供する
- グローバル・トランザクションのトランザクション・マネージャーとなる



16

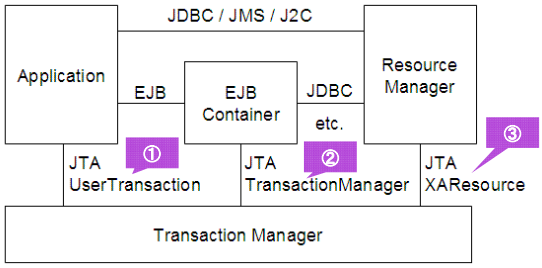
WAS V8.0ではJTA1.1に準拠したトランザクション・サービスを提供します。具体的にはJTA1.1の仕様に沿って作成されたユーザー・アプリケーションからトランザクション機能を使用する場合の実行環境を提供し、グローバル・トランザクションの際にはトランザクション・マネージャーとして機能します。

WASV8.0 Information Center「WebSphere Application Server でのトランザクション・サポート」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/cjta_trans.html

Designing Web System Infrastructure with WAS V8.0

JTAで提供されているトランザクション機能



1. UserTransactionインターフェース
   - ユーザー・アプリケーションが直接トランザクションを制御する際に利用
   - トランザクションの開始、Commit/Rollbackなどを行うことができる
2. TransactionManagerインターフェース
   - EJBコンテナーなどがトランザクションの制御を行うために利用するAPI
   - 直接アプリケーション開発者が利用することを意図してデザインされていない
3. XAResourceインターフェース
   - トランザクション・マネージャーがデータベースやメッセージング・システムなどのリソース・マネージャーを操作するために利用

17

JTAはThe Open Groupが規定した分散トランザクション処理モデルに基づいています。この処理モデルに基づき、JTAではアプリケーション、リソース・マネージャー、トランザクション・マネージャーの各役割とその各コンポーネント間のインターフェースが定められています。

Designing Web System Infrastructure with WAS V8.0

WAS V8.0の独自機能

- トランザクションのピア・リカバリー
  - トランザクション・マネージャー障害時、未解決トランザクションのリカバリー機能
  - HAマネージャーによるトランザクション・マネージャーのフェールオーバー
- タイマー機能
  - トランザクション・タイムアウトの設定が可能
    - ある一定時間を経過したトランザクションをロールバックする
- モニター機能
  - 実行トランザクションのスナップショット
  - Tivoli Performance Viewerでトランザクション状態をモニター
    - 実行中のトランザクション数
    - ロールバックされたトランザクション数 etc…
- トランザクション・リソースのコミット優先順位設定 V7～
  - 2 フェーズ・コミットの処理時に、トランザクション・リソースの処理順序を指定可能

18

WASが実装している独自の機能としては、以下があります。

＜トランザクションのピア・リカバリー＞

WAS V6から実装されている機能で、HAマネージャーによるトランザクション・マネージャーの冗長化をサポートします。トランザクション・マネージャー障害時に、HAマネージャーにより障害を回復するためのサービスが起動し、その起動したサービスがトランザクションを最後まで完了させるという機能になります。

＜タイマー機能＞

WASでは、トランザクション・タイムアウトを設定する事が出来ます。ある一定の時間を経過したトランザクションに対してロールバックを行うという機能になります。

＜モニター機能＞

実行トランザクションのスナップショットを、管理コンソールやwsadminコマンドを利用して確認することが出来ます。また、Tivoli Performance Viewerで、実行中のトランザクション数やロールバックされたトランザクション数等のトランザクション状態をモニターすることが出来ます。

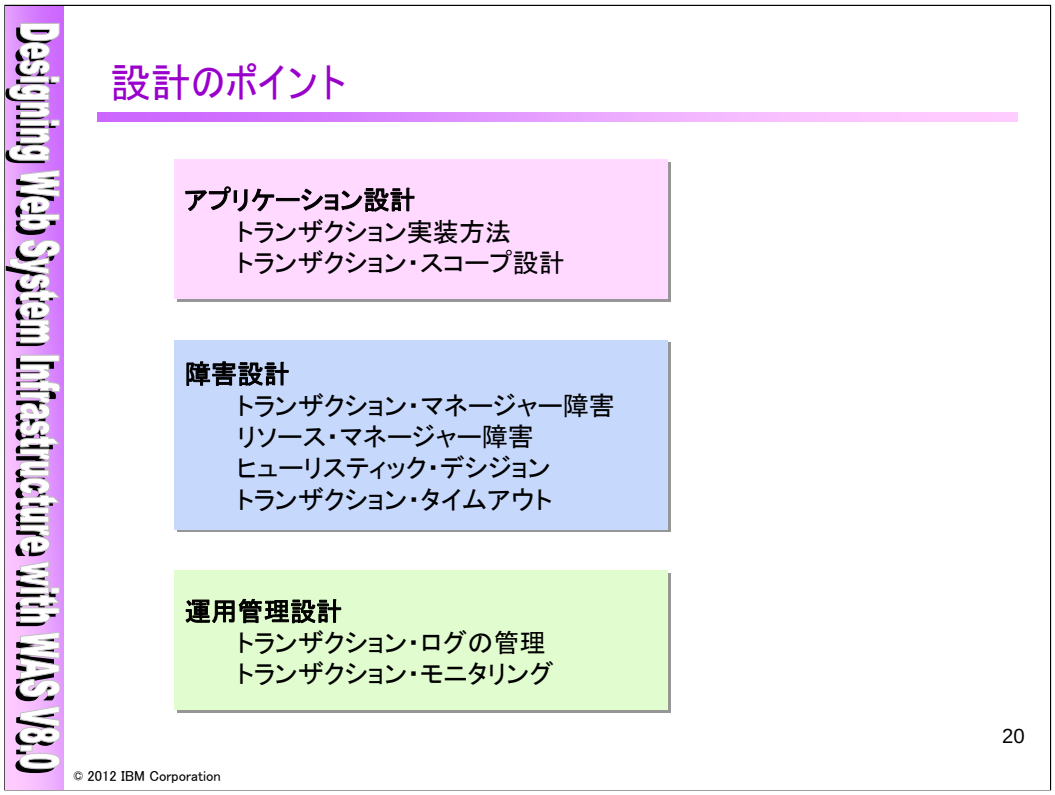
＜コミット優先順位設定機能＞

コミットフェーズでのコミット順序は実装依存であり、デフォルトではリソース・マネージャーの種別によってコミットの順序が異なります。V7からはコミットの処理順序を指定することが可能になりました。

Designing Web System Infrastructure with WAS V8.0

3. トランザクション設計のポイント

19



ここではトランザクション設計として、以下の3つが観点から考えていきます。

＜アプリケーション設計＞

アプリケーションによるトランザクションの実装方法と、1つのトランザクション処理範囲（トランザクション・スコープ）の考え方を中心に説明します。

＜障害設計＞

グローバル・トランザクションにおける障害設計、リカバリー方法および障害時に備えてタイマー設定を中心に説明します。

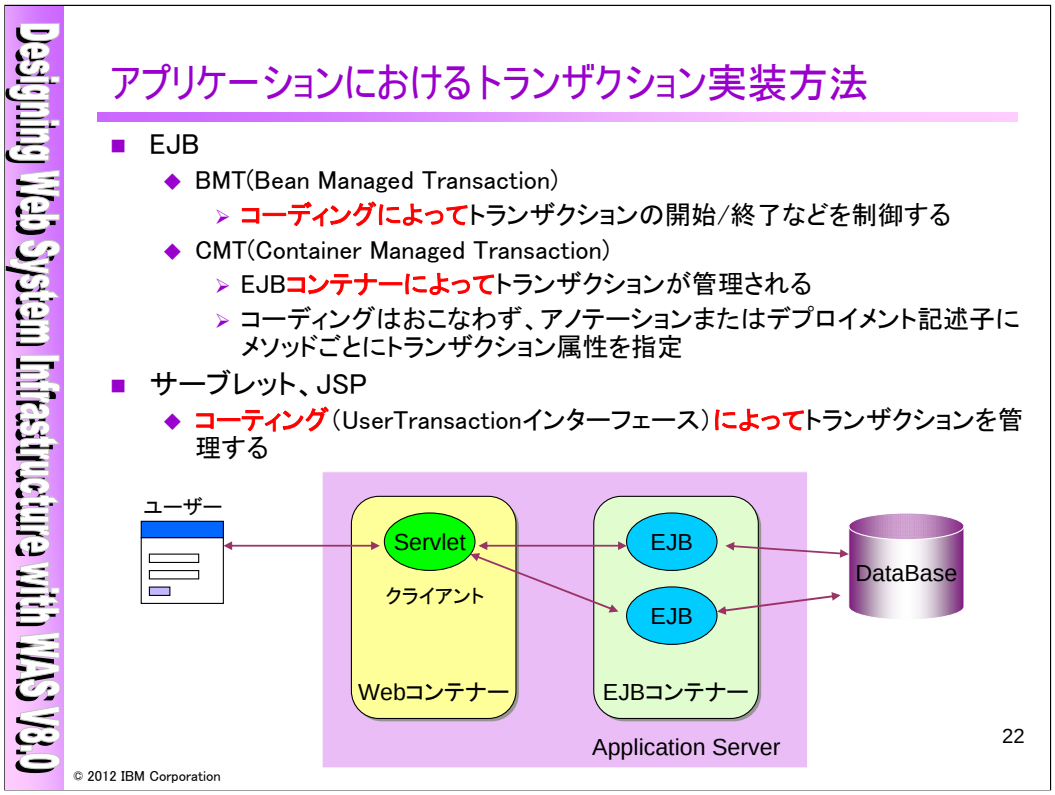
＜運用管理設計＞

トランザクション・ログの管理およびTPVを用いたトランザクションのモニターについてお話します。

Designing Web System Infrastructure with WAS V8.0

3. トランザクション設計のポイント

21



アプリケーションにおけるトランザクションの実装方法として、EJBで実装する方法とEJB以外(サーブレット、JSPなど)で実装する方法があります。

＜EJB＞

EJBコンテナーによって管理されるCMT(Container Managed Transaction)と、トランザクション開始/終了/commit/rollbackなどをコーディングで制御するBMT(Bean Managed Transaction)の2種類があります。CMTの場合はトランザクション処理をコーディングではなく、アノテーションやデプロイメント記述子を用いて各メソッドごとにトランザクション属性を指定します。

＜EJB以外＞

EJB以外のアプリケーションでは、UserTransactionインターフェースを使用してトランザクション開始/終了/commit/rollbackなどを制御し、トランザクションを実装します。

Designing Web System Infrastructure with WAS V8.0

アノテーションの使用

- アノテーションとは、ソースコード中にコードの属性情報を付加する機能
  - アノテーション・サポートによる開発効率・保守効率の向上
    - ベンダー依存のデプロイメント記述子の排除
    - 仕様書：JSR175 Metadata Facility for Java / JSR250 Common Annotation
- アノテーションを用いてメソッドなどに追加情報を付与したり、DI（Dependency Injection）機能が使える
  - DIとはコーディング時は変数に内容を指定せず、実行時にコンテナに注入させる機能
  - アノテーション(@Resource)を使用したDIの例

```
@Resource
private UserTransaction ut;
```

この変数utにはコンテナが実行時にインスタンスを注入する

  - getCustomerメソッドのトランザクション属性を指定した例

```
@TransactionAttribute(TransactionAttributeType.REQUIRED)
public Customer getCustomer(String ssn) { }
```

23

WASV8では、EJB3.1 / Servlet3.0 をサポートしており、アノテーションとよばれる機能を使用することが可能です。アノテーションとは、ソースコード中に属性情報を付加する機能で、Java SE5.0から導入されました。アノテーションを使用することによってクラスやメソッドに、今までデプロイメント記述子に指定していた内容を指定したり、DI（Dependency Injection）の機能を使用することができます。DIとは日本語では依存性注入と訳され、クラス実行時に変数にコンテナによって実体を注入させる機能です。

Designing Web System Infrastructure with WAS V8.0

EJBでの実装方法 - BMT-

- コーディングによってトランザクションの開始/終了を制御する
  - BMTを使用することを宣言する必要がある。
    - EJB3.0以降ではアノテーションを用いて指定可能
  - javax.transaction.UserTransactionインターフェースを利用
  - プログラム型トランザクション(programmatic transaction)

```
@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class EJB3BankBean implements EJB3BankService {
        @Resource
        private UserTransaction ut;
...
        public Customer getCustomer(String ssn)  throws ITSOBankException {
                try {
                        ut.begin();
                        // execute ejb business logic..
                        ...
                        ut.commit();
                } catch(Throwable t) {
                        if(ut!=null) ut.rollback();
                }
    }
}
```

BMTを指定

UserTransactionを取得

トランザクション開始

トランザクション終了

例外発生時のロールバック処理

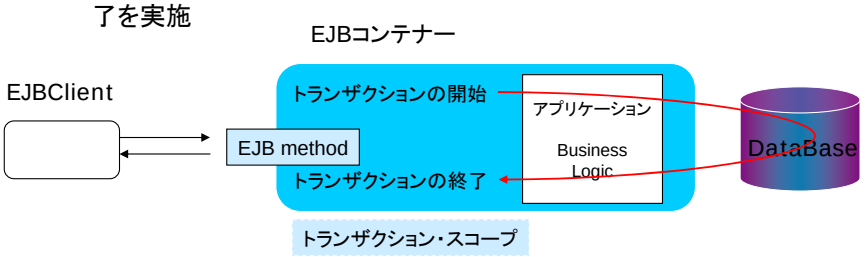
24

BMT(SessionBean内のコーディングによってトランザクションの開始/終了を制御する)の場合、javax.transaction.UserTransactionインターフェースを利用して、ユーザーがコーディングする必要があります。ユーザーがコーディングによって制御するトランザクションを、プログラム型トランザクション(programmatic transaction)と呼びます。上記ではアノテーションを用いてBMTであることを指定し、トランザクションの開始、終了、ロールバック処理をコーディングしている例になります。

Designing Web System Infrastructure with WAS V8.0

EJBでの実装方法 - CMT- (1/3)

- コンテナーによってトランザクションが管理される
  - アプリケーション内のコーディングで開始、終了などの制御をしない
    - ビジネス・ロジックとトランザクション制御の分離
  - アノテーション(EJB3.0以降)またはデプロイメント記述子(EJB2.1以前)に記述
    - 上記方法で各メソッドごとにトランザクション属性を指定
  - Beanを呼び出すとEJBコンテナーが自動的にトランザクションの開始/終了を実施



25

CMTでは、EJBコンテナーがトランザクションを管理します。

EJBClientが、EJBのmethodを呼び出すと、EJBコンテナーが自動的にトランザクションの開始/終了/commit/rollbackを実施します。この時、アノテーションまたはデプロイメント記述子でメソッドごとに適切なトランザクション属性を与えて、トランザクションを制御します。また、EJBコンテナーが管理するトランザクションの事を、宣言型トランザクション(declarative transaction)とも呼びます。

Designing Web System Infrastructure with WAS V8.0

EJBでの実装方法 - CMT- (2/3)

- EJB3.0以降での実装方法
  - アノテーションを使用可能
  - CMTの使用と各メソッドごとのトランザクション属性を指定

CMTを指定

クラスのデフォルト属性はNOT_SUPPORTED

```
@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public class EJB3BankBean implements EJB3BankService {
        ･･･省略･･･
        @TransactionAttribute(TransactionAttributeType.REQUIRED)
        public Customer getCustomer(String ssn) throws TSOBankException {
        ･･･省略･･･
        }
}
```

getCustomerメソッドのトランザクション属性はREQUIRED

26

上記はアノテーションを用いた場合のCMTの実装例になります。CMTの使用宣言と各メソッドごとのトランザクション属性の指定をアノテーションを用いておこなっています。

Designing Web System Infrastructure with WAS V8.0

EJBでの実装方法 - CMT- (3/3)

- EJBの場合、トランザクション属性によってスコープが決定
  - CMTにEJBがどのように参加するかを決定
  - メソッドごとにアノテーションまたはデプロイメント記述子に指定
- トランザクション属性の種類 詳細は参考資料参照
  - Required
    - クライアントがトランザクションを開始していればその中でEJBのメソッドが実行される。
    - トランザクションを開始していなければEJBコンテナーが開始する
  - RequiresNew
    - 新しいトランザクションを開始・終了する
  - Supports
    - クライアントがトランザクションを開始していればその中でEJBのメソッドが実行される。
    - トランザクションが開始していなければEJBコンテナーは何もしない
  - Mandatory
    - クライアントが必ずトランザクションを開始している必要がある
  - NotSupported
    - クライアントがトランザクションを開始しているかどうかに関わらず、EJBのメソッド実行はその中に含まれない
  - Never
    - EJBメソッドは、トランザクションに参加しない

27

CMTで指定できるトランザクション属性は以上の6つになります。メソッド単位にアノテーションまたはデプロイメント記述子にトランザクション属性を指定します。

詳細は参考資料をご参照ください。

Designing Web System Infrastructure with WAS V8.0

サーブレット・JSPでの実装方法

- UserTransactionインターフェースを使用してトランザクションの開始/終了のコードを記述
  - サーブレット、JSPなどEJB以外のコンポーネント
  - javax.transaction.UserTransactionインターフェースを利用
  - Webコンテナーではネーミング・サービスを使用してUserTransactionインスタンスを取得
    - Servlet2.5ではアノテーションも使用可能

```
public class TranTestServlet extends HttpServlet {
    @Resource
    UserTransaction ut;
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        try {
            ut.begin();
            //ビジネス・ロジック
            ut.commit();
        } catch(Throwable t) {
            if(ut!=null) ut.rollback();
        }
```

UserTransactionを取得

トランザクション開始

トランザクション終了

例外発生時のロールバック処理

28

EJB以外のアプリケーションの場合、BMTと同様ですが、UserTransactionを使用してトランザクションの開始/終了をユーザーがコーディングする必要があります。EJBと異なる点は、Webコンテナーでは、SessionContextをWebコンテナーでは使用することが出来ませんので、Contextによるネーミング・サービスを使用してUserTransactionインスタンスを取得する必要があります。またServlet2.5以上ではネーミング・サービスへのlookupの代わりにアノテーションを使用することも可能です。

Designing Web System Infrastructure with WAS V8.0

トランザクション実装方法の選択

問題発生リスクが低い
迅速な開発が可能

- CMT
  - トランザクション・スコープはBeanメソッド単位で制御
  - Beanクラスにトランザクションのロジックを書かなくてもよい
    - コーディング時間の短縮
    - アノテーションを用いて指定可能（デプロイメント記述子も使用可能）
    - 管理をコンテナーが実施することにより、間違ったトランザクションの使い方がない
- BMT
  - Beanがトランザクション・スコープを完全に制御
  - 1リモート・メソッドの中で複数の小さなトランザクションも実行可能
- サーブレット/JSP
  - UserTransactionインターフェースを使用
    - サーブレットなどWebモジュールからトランザクションを開始

29

CMTとBMT、UserTransactionについて、そのトランザクション実装方法についてまとめます。

＜CMT＞

・トランザクション・スコープはBeanメソッド単位で制御

・Beanクラスにトランザクションのロジックを書かなくてもよい

上記よりCMTを使用して開発する方が、問題発生リスクが低く、迅速な開発が可能であると言えます。ただし、CMTではメソッド毎にトランザクション・スコープが定義されますので、Beanメソッドの中で更に細かい制御を実施したい場合には、BMTを使用する必要があります。

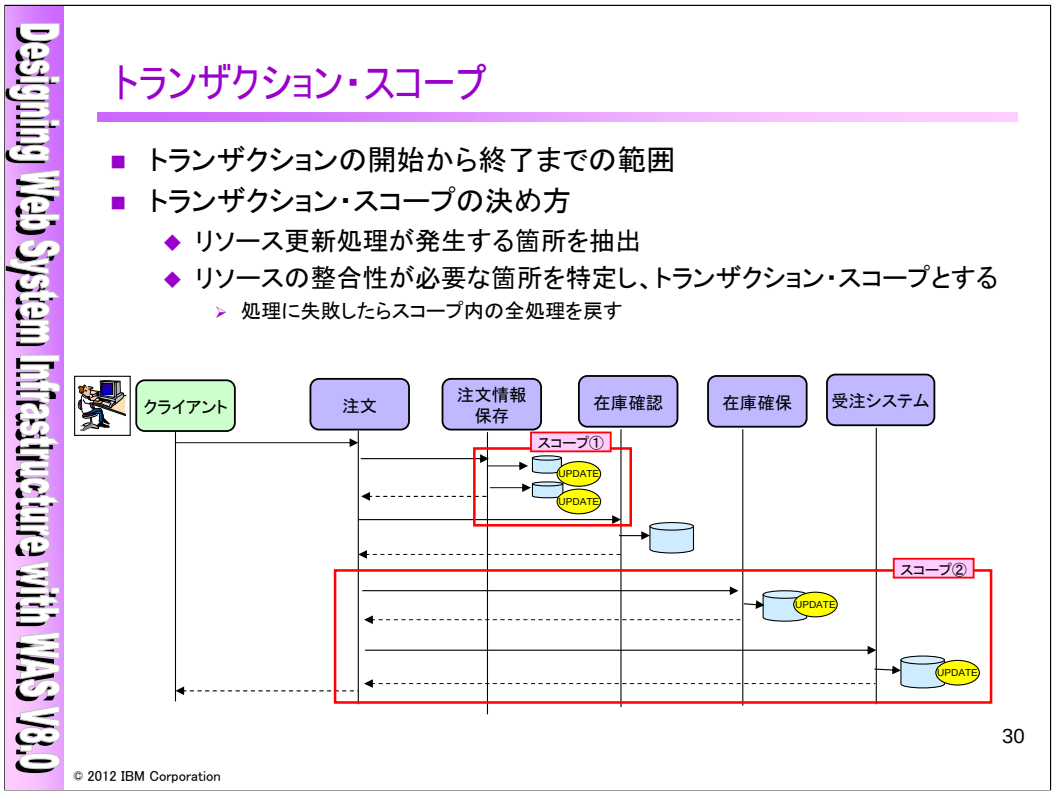
＜BMT＞

・Beanがトランザクション・スコープを完全に制御可能

・1リモート・メソッドの中で複数の小さなトランザクションも実行可能

＜サーブレット・JSPなどのEJB以外のコンポーネント＞

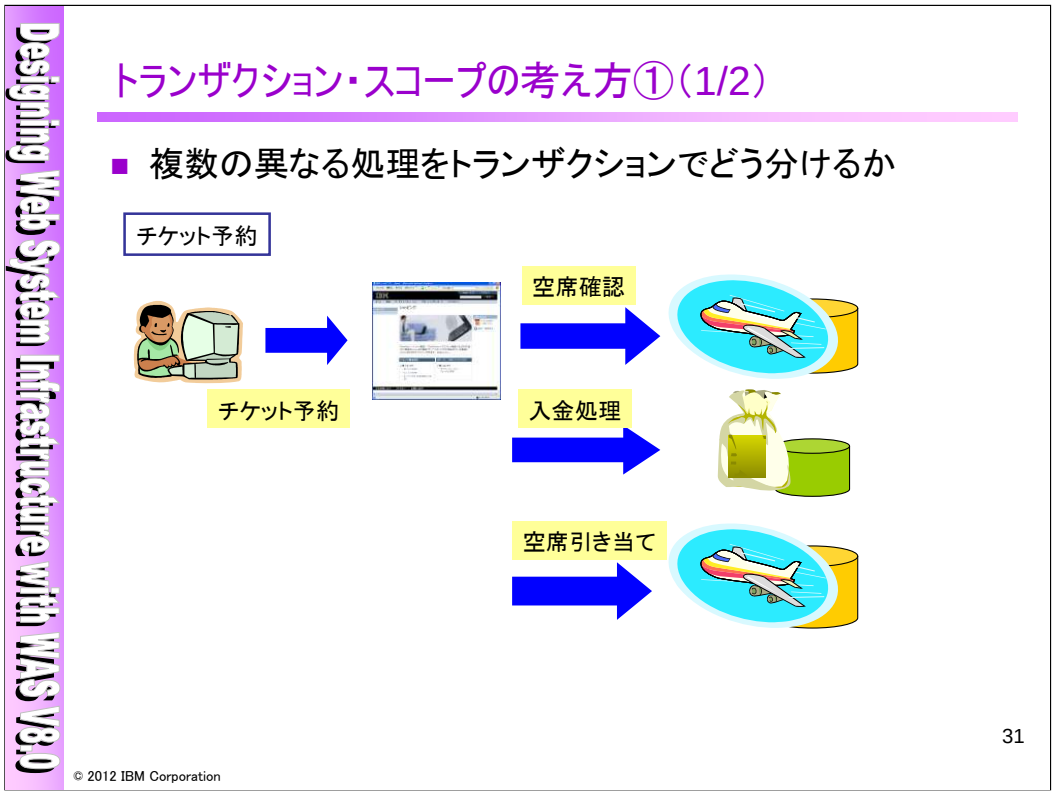
・EJBを使用しない場合、UserTransactionインターフェースを用いる



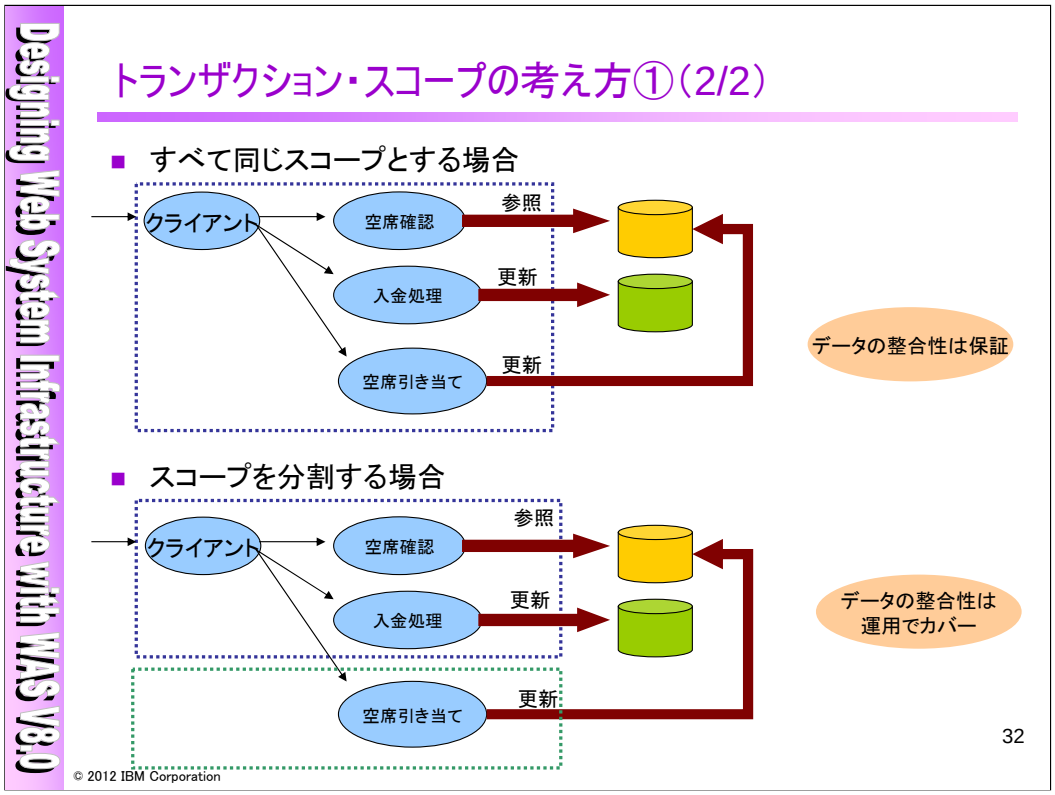
トランザクション・スコープとは、トランザクションの開始から終了までの範囲になります。

スコープを決める際には、まずリソースの更新処理が発生する箇所を洗い出します。

さらに、そのリソースの更新処理に整合性をとる必要がある場合（ACID属性を適用）に、トランザクション・スコープとします。

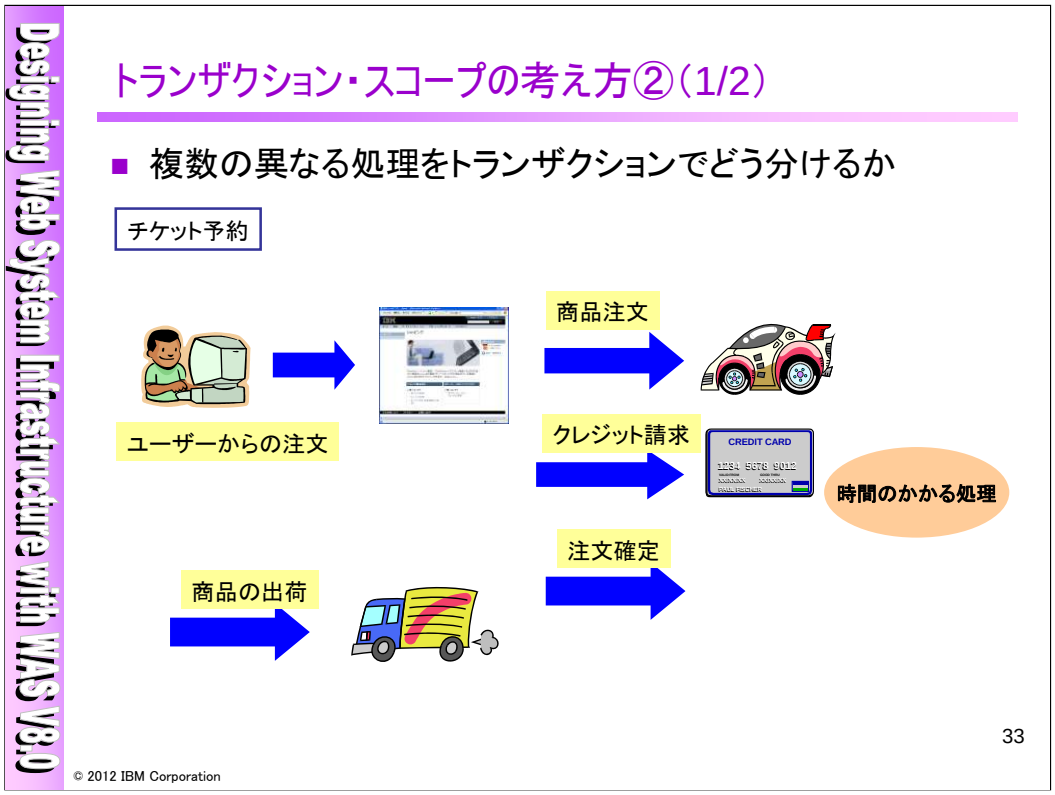


複数の異なる処理をトランザクションでどのように分けるかといった場合に、トランザクション・スコープを検討する必要があります。例えば、飛行機のチケットを予約する際に、空席確認→入金処理→空席引き当てという処理の流れになりますが、どのようにトランザクションを分けるかよって、トランザクション・スコープを選択する必要があります。

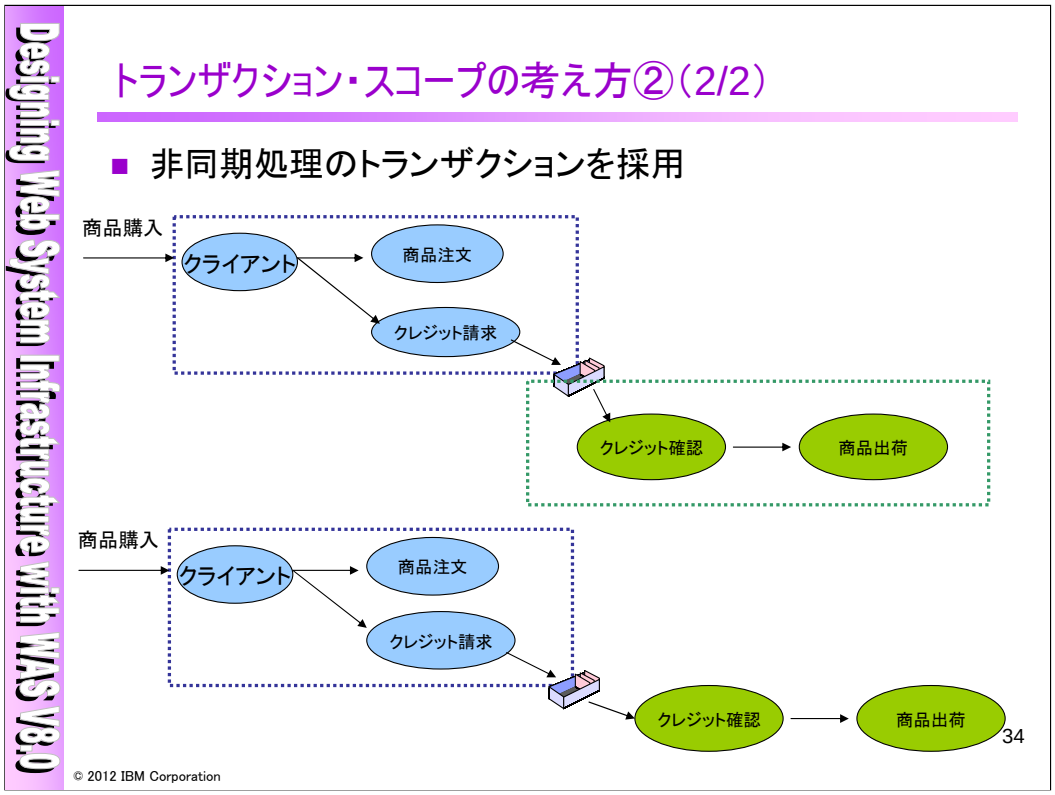


すべての処理を同一のスコープにする場合、2フェーズ・コミットとする必要があります。その場合には、処理が一連の流れとなり、必ず空席を割り当てる事が出来ます。この時、データの整合性が保たれます。

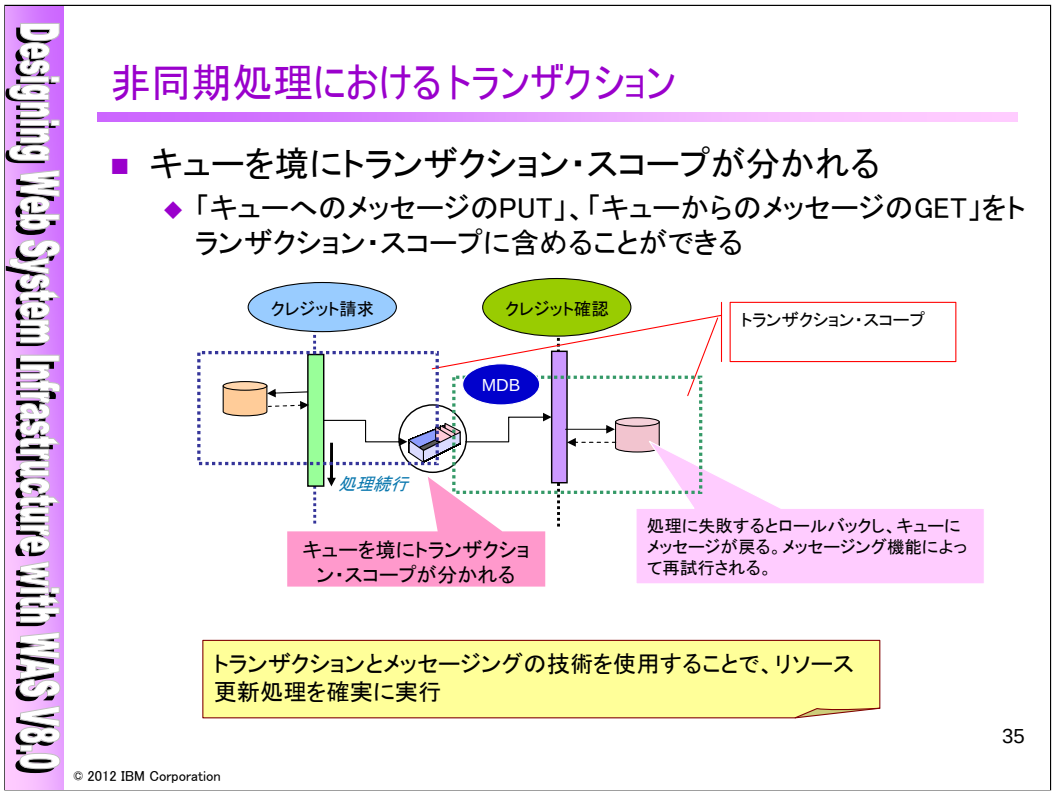
パフォーマンスを考慮する場合には、2フェーズ・コミットではなく1フェーズ・コミットを採用するケースもあるかと思います。その場合には、空席確認と入金処理/空席引き当て処理を別のスコープに定義します。ただし、必ずしも空席を引き当てられるわけではなく、席を余分に用意しておくなど、データの整合性を運用等で補填する必要があります。



前回と同様にチケット予約処理になりますが、今回のケースではクレジット請求処理が含まれており、ある程度の時間を要する処理になります。この場合に、どのようにトランザクションを分けて、トランザクション・スコープを選択するかについて説明します。



クレジット請求のような時間を要する処理の場合、MQなどを使用して、非同期のトランザクションを採用するという考え方もあります。クレジットの請求処理はキューを介した処理となるため、スコープがわかれます。また、クレジットの確認や商品出荷処理は、ひとつのスコープに纏めてトランザクション管理を実装することも、トランザクション管理を実装しないことも可能です。



非同期処理におけるトランザクションではトランザクション・スコープが分かれますが、キューへのメッセージのPUT処理、キューからのメッセージのGET処理をそれぞれのリソース更新処理と同じトランザクション・スコープに含めることができます。

キューのPUTが完了した段階でクライアントの処理は完了します。クレジット確認処理は、処理が成功するまで再試行が行われます。

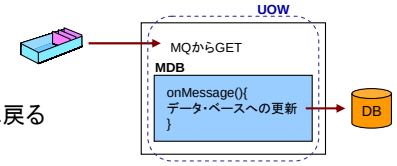
この場合のアプリケーションは、キューからEJB経由でメッセージを取り出す必要があり、MDB(Message Driven Bean)を使用することになります。

MDBはEJBの1つで、メッセージ受信をトリガーに非同期に起動することが可能です。

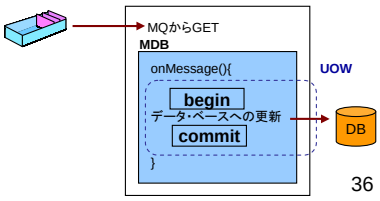
Designing Web System Infrastructure with WAS V8.0

MDBのトランザクション管理

- MDBはEJBの一種であるため、CMT・BMTを使用することが可能
  - 設定可能なCMTトランザクション属性はRequiredとNotSupportedのみ
- スコープの分け方によって、障害時のメッセージの行方やサーバー側の対応が異なる
  - CMT Required:
    コンテナーの受信処理とonMessage()内の処理が同一トランザクション・スコープ
    - 処理が異常終了した場合は全ての処理がロールバックされる
    - ロールバックした時のメッセージはキューに戻る
    - EJBコンテナーによる2フェーズ・コミット



  - BMT:
    コーディングによってトランザクションの開始/終了を制御する
    - commit()での例外はonMessage()内で発生
    - 必要な例外処理を記述する必要がある



36

MDBで設定可能なトランザクション属性は、"Required"と"NotSupported"になります。

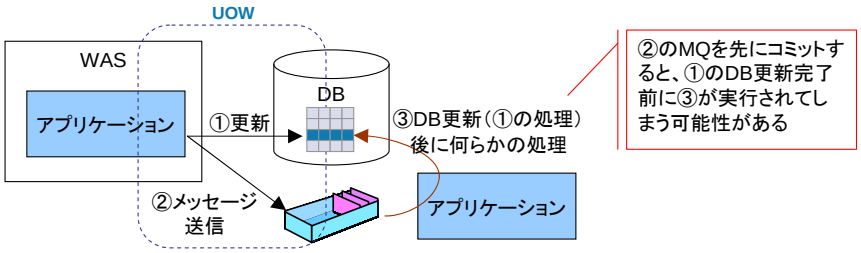
"Required"はコンテナーの受信処理とonMessage()内の処理が同一トランザクション・スコープとなり、"NotSupported"はコンテナーの受信処理もonMessage()メソッド内の処理もトランザクションを管理しないという設定です。

また、トランザクション・スコープの分け方によって、障害時のメッセージの行方やサーバー側の対応が異なります。

Designing Web System Infrastructure with WAS V8.0

コミット優先順位の指定

- 1UOWで必ず先にコミットさせたい処理がある場合



- 設定方法

RADのWeb拡張エディターで指定
デプロイメント記述子の設定が必要なためアノテーションは使用不可

37

コミットフェーズでのコミット順序は実装依存であり、リソース・マネージャーの種別によってコミットの順序が異なります。WAS V7からはコミットの処理順序を指定することが可能になりました。

この設定にはRADが必要です。RADのWeb拡張エディターから、リソース参照で指定をします。そのためリソース参照にデプロイメント記述子が必要になるため、アノテーションが使用された場合、この機能は使用できません。

Designing Web System Infrastructure with WAS V8.0

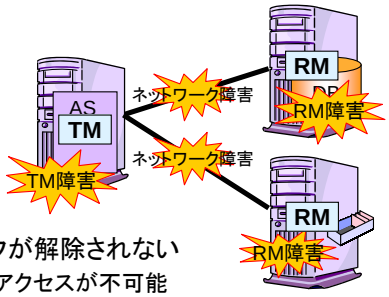
3. トランザクション設計のポイント

38

Designing Web System Infrastructure with WAS V8.0

グローバル・トランザクションにおける障害

- 障害発生箇所
  - トランザクション・マネージャー
  - リソース・マネージャー
  - ネットワーク
- 障害によって発生する問題
  - リソース(DBの行、MQのキュー)のロックが解除されない
    - 他のアプリケーションからリソースへのアクセスが不可能
    - 他のアプリケーションまで処理がWait
  - データの不整合



- 仕掛かり中のトランザクションに対して適切なリカバリーが処理が必要
  - 障害時のトランザクション・リカバリーはトランザクション・マネージャーがコーディネート

39

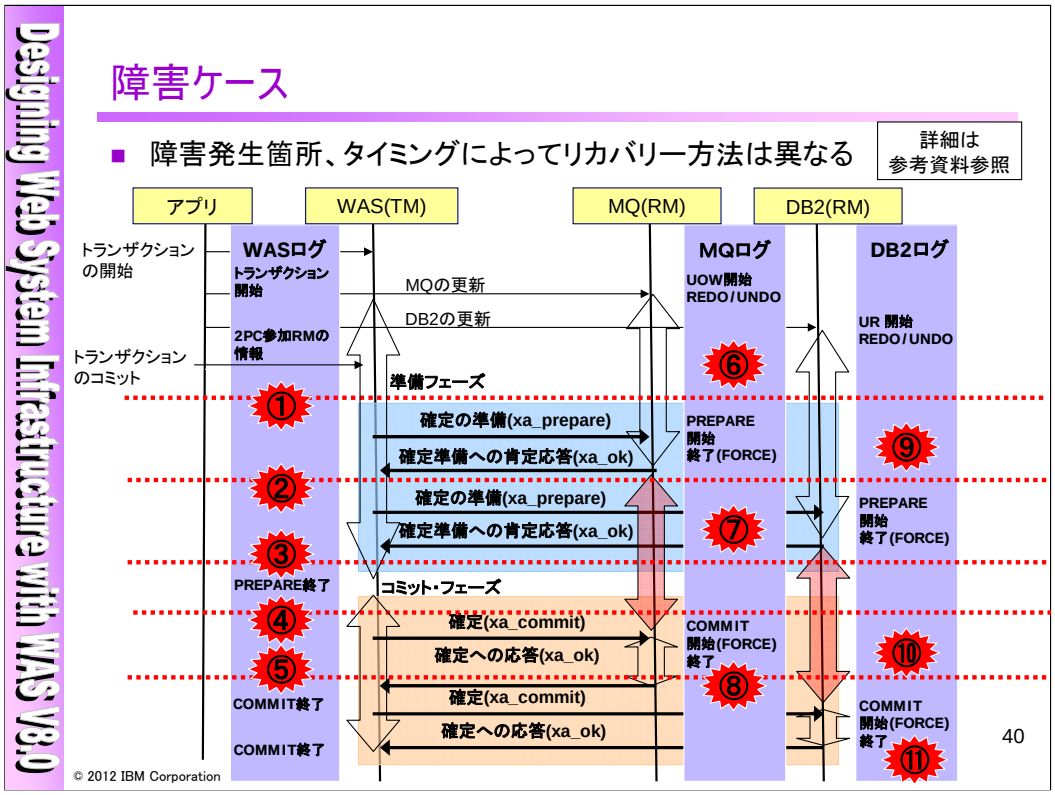
トランザクション処理の障害設計のポイントについて説明します。ここではWASがトランザクション・マネージャーとなり複数のリソース・マネージャーへの更新処理をおこなうグローバル・トランザクションのパターンを考えます。

障害発生時には以下のような問題が生じる可能性があります。

・リソース(DBの行、MQのキュー)にロックがかかり、(他のアプリケーションからリソースへのアクセスが不可能、他のアプリケーションまで処理がWait)

・データの不整合

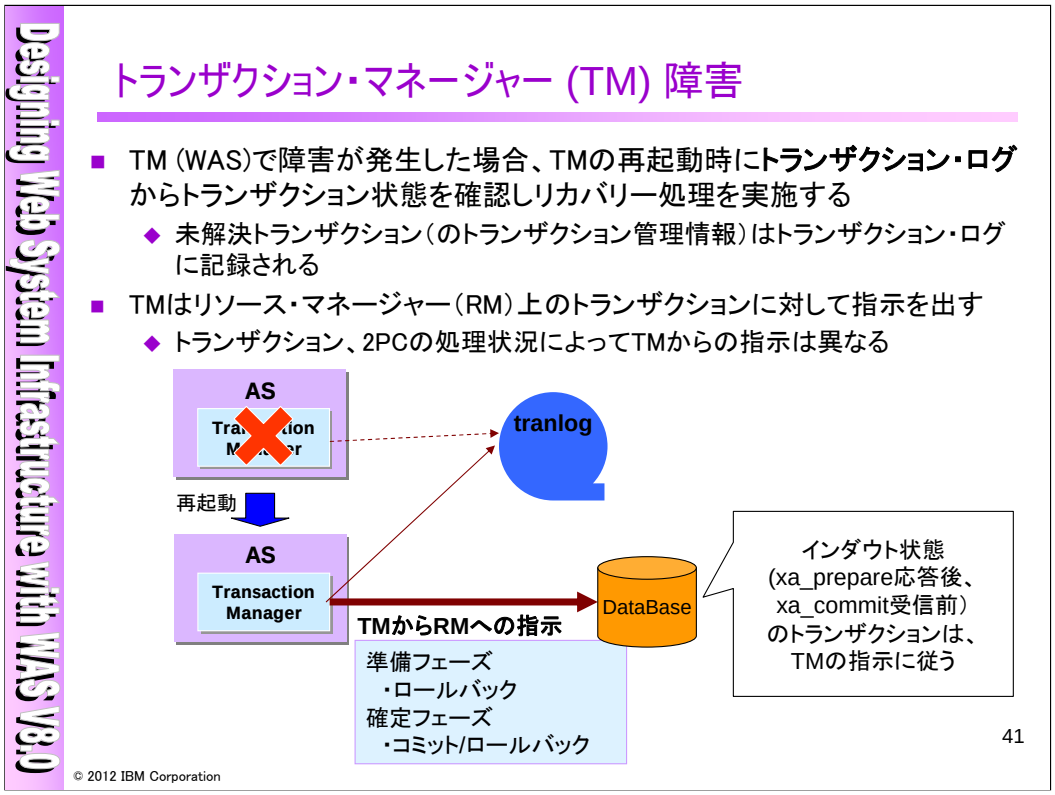
従いまして、仕掛かり中のトランザクションに対して、適切なリカバリーが処理が必要になります。障害時のトランザクション・リカバリー処理は、トランザクション・マネージャーがコーディネートします。障害発生箇所としては、トランザクション・マネージャーとリソース・マネージャーが考えられます。トランザクション・マネージャーとリソース・マネージャーが別ノードにある場合は、ネットワーク障害も考えられます。



2フェーズ・コミットの障害は、どこで障害が起こるのかによってそれぞれ発生する問題は変わってきます。上記はWASをトランザクション・マネージャー、MQ、DB2をリソース・マネージャーとした場合の障害パターンになります。

また、参照ページ以降にて各発生箇所毎の対応方法を載せています。

次ページからは、TMに障害が起きた場合のリカバリー方法と、RMに障害が起きた場合のTMの動作について説明します。



TM(WAS)で障害が発生した場合、TMの再起動時にトランザクション・ログからトランザクション状態を確認し、リカバリー処理を実施します。TM障害ですので、RM上のトランザクションは仕掛かり中であり、TMの再起動によってTMがトランザクションの指示を行います。また、その指示内容は、各フェーズによって異なります。

＜TMからRMへの指示＞

・準備フェーズ

必ずロールバックが行われます。

・確定フェーズ

TMが保持しているトランザクション・ログを参照して、コミット/ロールバックを行います。

＜RM側の動作＞

・in-flight(インフライト)状態（=xa_prepareに対して応答を返すまで）

TMが全トランザクションのロールバックを行います。

・in-doubt(インダウト)（=xa_prepare応答後、xa_commit受信前）

TMの指示に従います。

・in-commit(インコミット)(=commit処理完了）

TMは何もしません。

Designing Web System Infrastructure with WAS V8.0

WAS上のトランザクション・リカバリー

- リカバリー方法
  - トランザクション・ログを使ったリカバリー
    - ①アプリケーション・サーバー再起動によるリカバリー
    - ②代替アプリケーション・サーバーによるリカバリー
    - ③HAマネージャーによるピア・リカバリー
      （NDのクラスター構成のみ）
  - トランザクション・ログを使用せず手動でリカバリー
    （ヒューリスティック・デシジョン）

42

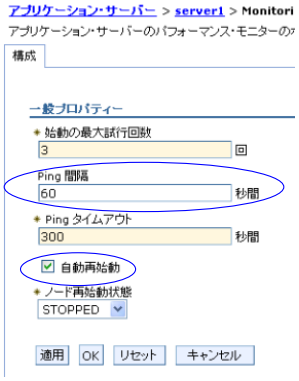
トランザクションのリカバリーには、トランザクション・ログを使ったリカバリー方法と、トランザクション・ログを使用せずに手動でリカバリーする方法があります。

通常はトランザクション・ログを使用してリカバリーを行います。トランザクション・ログを使用したリカバリーはおおきく3つの方法があります。

Designing Web System Infrastructure with WAS V8.0

TM障害時のトランザクション・リカバリー手法 ①

- アプリケーション・サーバー再起動によるリカバリー
  - BASEでは、手動で再起動
  - NDでは、ノード・エージェントが自動再起動
    - デフォルトで自動再始動はON
    - アプリケーション・サーバーの設定で、監視間隔の設定可能
    - サーバー＞サーバー・タイプ＞WebSphere Applicatin Server＞[アプリケーション・サーバー名]＞サーバー・インフラストラクチャー＞Javaおよびプロセス管理 ＞モニター・ポリシー



- リカバリー・モードでのアプリケーション・サーバー再始動によるリカバリー
  - リカバリー処理のみ実行し、新規処理は受け付けない
  - 処理完了後はシャットダウン
  - startServer.sh(bat) –recovery を実行
  - NDでは、ノード・エージェントの自動再起動をOFFにする必要がある

43

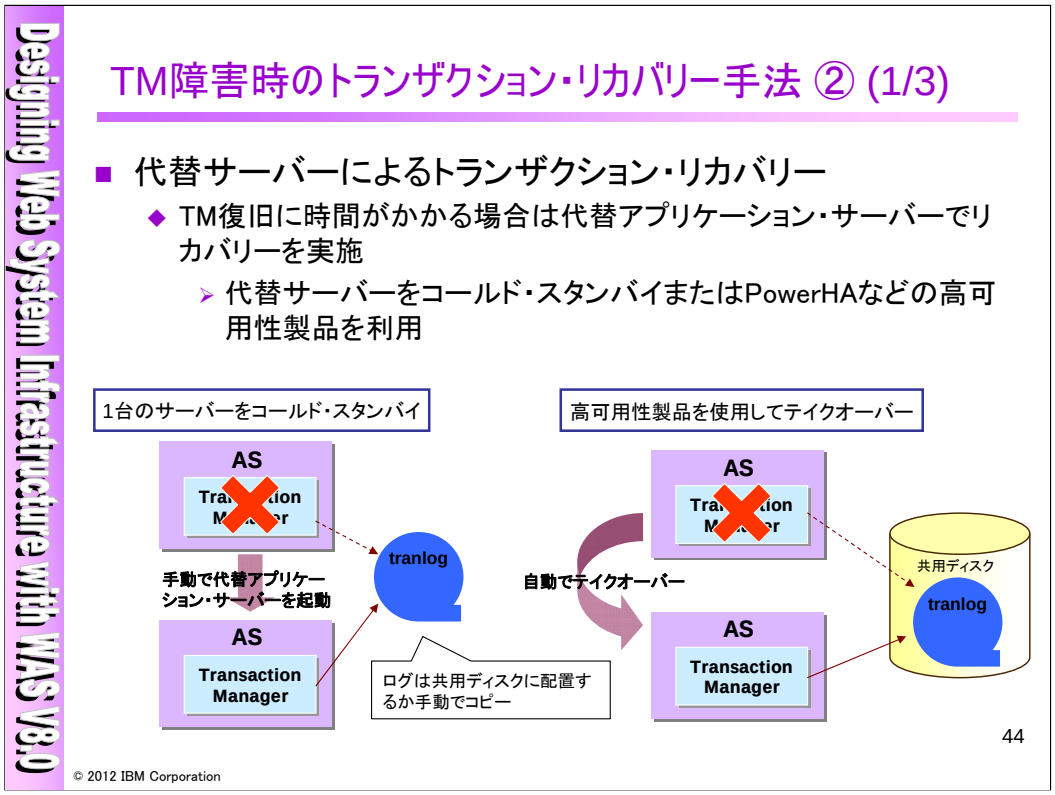
方法①:

アプリケーション・サーバー再起動

BASE構成では、手動でアプリケーション・サーバーを再起動し、トランザクション・ログを読み込んでリカバリーする必要があります。管理エージェント構成では管理エージェントがアプリケーション・サーバーを再起動し、ND構成では、ノード・エージェントがアプリケーション・サーバーを自動で再起動し、トランザクション・ログを読み込んでリカバリーするという仕組みがあります。デフォルトでノード・エージェントによる再起動の仕組みはONになっております。また、上図のように管理コンソールにて自動再始動の設定する事が可能です。

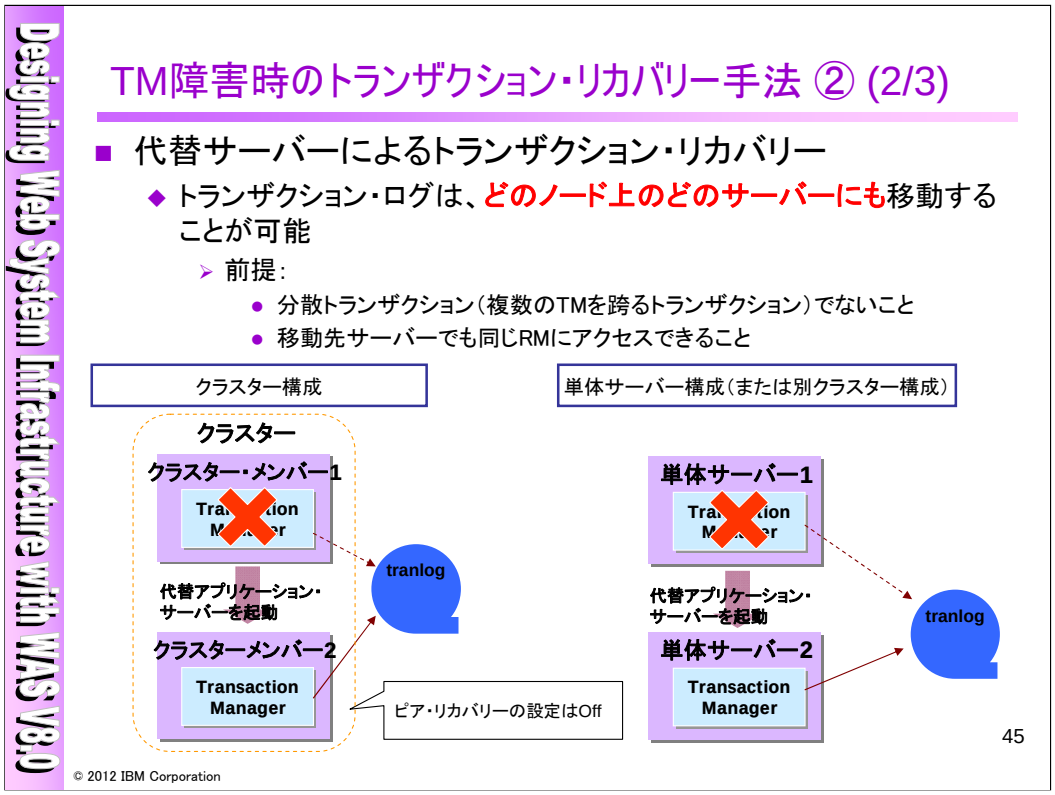
リカバリー・モードでのアプリケーション・サーバー再起動

通常の起動方法の他に、リカバリー・モードでアプリケーション・サーバーを起動することができます。この場合、新規処理を受け付けずにトランザクションのリカバリーのみを行い、リカバリー処理後はシャットダウンされます。新規処理を受け付ける前に未解決トランザクションを完了させたい場合に有効です。



方法②:代替サーバーでのリカバリー

TM復旧に時間がかかる場合は、代替アプリケーション・サーバーでリカバリーを実施します。この方法の場合、プロセス障害だけではなくノード障害にも対応することが出来ますが、コールド・スタンバイの場合にはリカバリーに時間がかかることと、自動化する場合には別途、PowerHAなどの高可用性製品の購入が必要になることが注意点になります。



複数のTMを跨った分散トランザクションでなければ、トランザクション・ログはどのノードのどのサーバーにも移動することができます。

つまり、代替サーバーは、同一クラスター内のクラスター・メンバーでもよいですし、別の単体サーバーでもよいということです。同一クラスター内のクラスター・メンバーを代替サーバーとして使用する場合は、次頁で説明するピア・リカバリーの設定をオフにする必要があります。

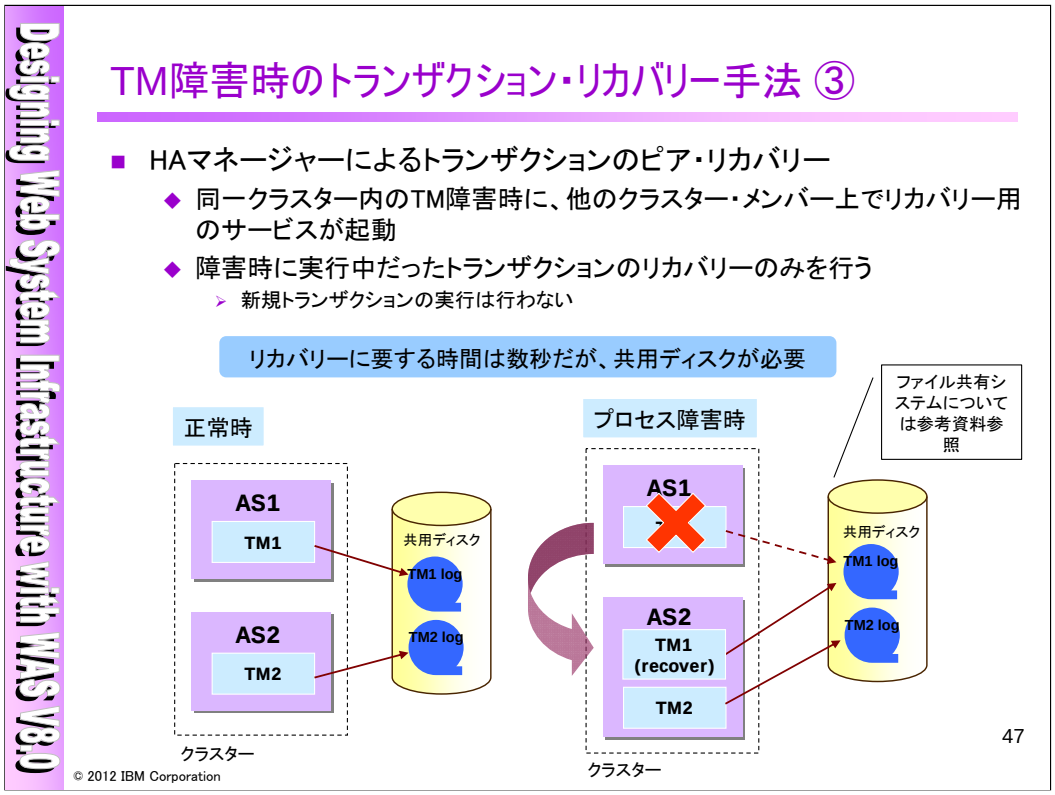
Designing Web System Infrastructure with WAS V8.0

TM障害時のトランザクション・リカバリー手法② (3/3)

- 代替サーバーによるトランザクション・リカバリー
  - トランザクション・ログ移動の制約
    - トランザクション・ログ・ディレクトリー内の全ファイル/ディレクトリーを代替サーバーがアクセス可能な状態で移動すること(状態にすること)
    - 移動先のアプリケーション・サーバーは、「トランザクション・ログ・リカバリーのフェールオーバー」が使用不可に設定されていること(ピア・リカバリーが構成されていないこと)
    - 別セルのアプリケーション・サーバーに移動する場合は、移動元と同じ JAAS (Java Authentication and Authorization Service) 別名が存在すること (J2C 認証データ別名は、デプロイメント・マネージャーのノード名が別名に含まれないように定義されていること)
    - 対象のリソース・マネージャー(DBやMQなど)にアクセス可能なアプリケーション・サーバーに移動すること

46

さらに上記のような制約がありますのでご確認ください。



方法③: HAマネージャーによるトランザクションのピア・リカバリー

同一クラスター内のTM障害時に、他のクラスター・メンバー上でリカバリー用のサービスが起動します。(AS1とAS2が同一のクラスターとして定義されています。)また、TM1のリカバリー・プロセス(TM1 recover)は、新規のトランザクションの実行を行わず、障害時に実行中であったトランザクションのリカバリーのみを行います。

この方法の場合、リカバリーに要する時間は数秒ですが、共有ディスクが必要になります。

Designing Web System Infrastructure with WAS V8.0

トランザクション・リカバリー方法の選択

リカバリー方法	アプリケーション・サーバー再起動によるリカバリー		代替アプリケーション・サーバーによるリカバリー		HAマネージャーによるピア・リカバリー
	サーバー再起動	リカバリー・モードでの再起動		サーバー再起動	自動
WASノード障害	対応不可		代替サーバを別ノードに構成することで可能		クラスターの複数ノード構成で対応可能
復旧時間	中		大		小
リカバリー・プロセス起動の自動化	ND、管理エージェント構成はWASの機能で実施	作りこみが必要（作りこみにより復旧時間も変化）			WASの機能で実施
WASの構成	Base / ND 同一サーバー		Base / ND 同じRMにアクセス可能なサーバー		NDのみ 同一クラスターのメンバー間
共用ディスク(*)	不要		代替サーバを別ノードに構成する場合は必要		必要

(*)共用ディスクがSPOF となったり、負荷が集中することにも考慮が必要

48

ここまでご説明してきたTM復旧方法をまとめました。
共用ディスクの要件については参考セクションをご参照ください。

Designing Web System Infrastructure with WAS V8.0

ヒューリスティック・デシジョン

- ヒューリスティック・デシジョンとは
  - ヒューリスティックな結果（精度が保障されない結果）が伴う処理
  - システム運用者が独自に行う確定処理
- 必要となるのはインダウト・ウィンドウでのTM障害のケース
  - ディスク装置の破損などトランザクション・ログを救えないケース
  - TM復旧を待たずにインダウト・トランザクション回復を優先したいケース
    - リソース(データベースの行・MQのキュー)のロックを早く解除したい
- リスク
  - 他のRMの確定結果と一致する保証はなく、データの不整合もあり得る

確定結果は運用者の責任

ヒューリスティック・デシジョンはデータ不整合の要因となる

49

TM/RM障害が発生した場合、TM/RMを早期回復させる事が重要ですが、それが難しい場合にはヒューリスティック・デシジョン（システム運用者が確定処理を独自に行う）を行う必要があります。ヒューリスティック・デシジョンの実行が必要となるのは2PC処理時のインダウト状態でのTM障害のケースとなり、以下が挙げられます。

・トランザクション・ログを救えないケース(ディスク装置が破損等)

・TM復旧を待たずにインダウト・トランザクション回復を優先したい(リソース(データベースの行・MQのキュー)のロックを早く解除したいなど)

しかしながら、トランザクションは運用者の判断でもってcommit/rollbackが実行されますので、データの不整合が発生してしまう可能性がある事をご注意下さい。

・heuristicとは

(学習者の)発見を助ける、自発研究をうながす、発見的な。

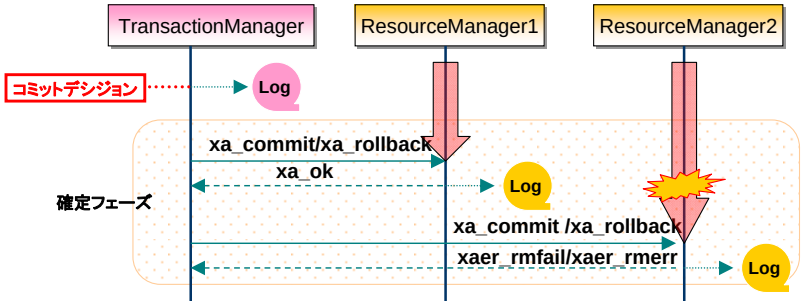
必ず正しい答えがとは限らないが、ある程度のレベルで正解に近い解を得ることが出来る方法。答えの精度は保障されないが、回答に至るまでの時間が少なくて済む。

Designing Web System Infrastructure with WAS V8.0

RM障害時のTMの動作

- TMの動作
  - 2PCの準備フェーズで決定したコミット/ロールバックは必ず全RMに指示する必要がある
  - TMはRMからの応答により、RM復旧まで指示のリトライを行う

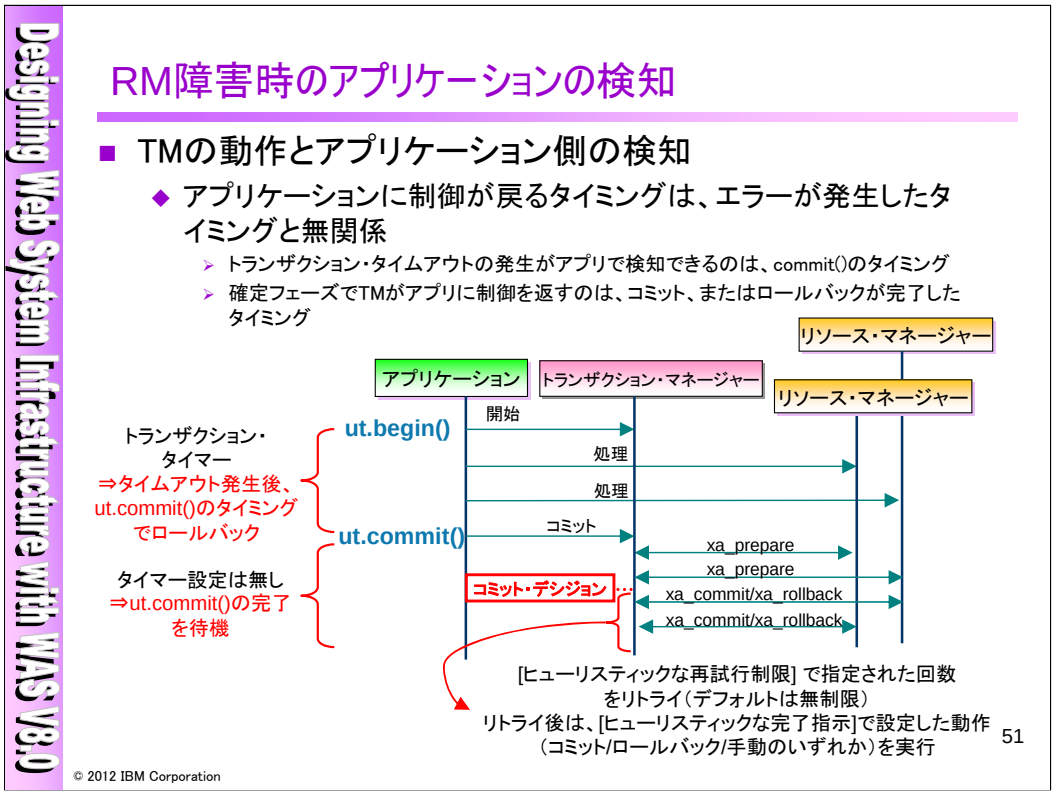
リカバリーには確実なRMの障害回復のしくみが重要



50

TMはデータの整合性を保つ為、2PCの準備フェーズで決定したコミット/ロールバックは必ずすべてのRMに対して指示する必要があります。確定フェーズにおいてRMに障害が発生した場合、TMはRMの復旧まで指示のリトライを行います。

従いまして、RMが障害から復旧しないとトランザクションを完了させる事が出来ませんので、確実にRMを障害回復させる仕組みが重要になります。

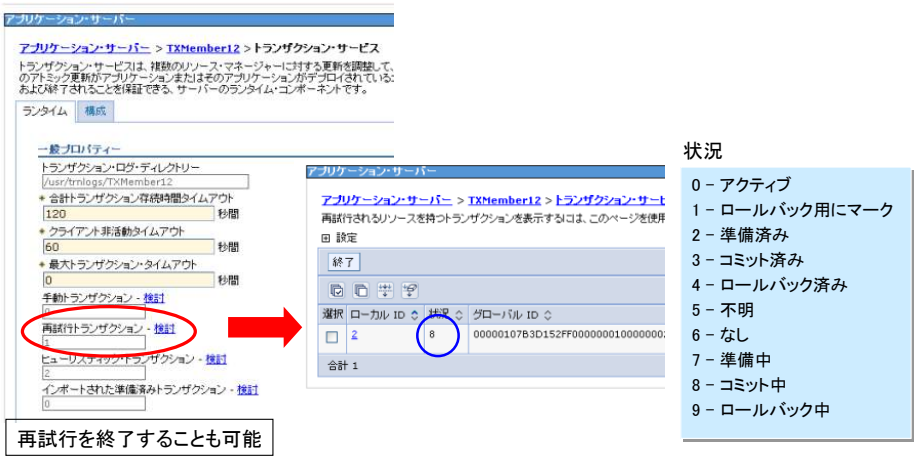


RM障害時に、エラーが発生したタイミングとアプリケーションがそれを検知するタイミングは異なりますので注意が必要です。

Designing Web System Infrastructure with WAS V8.0

再試行中のトランザクションの表示

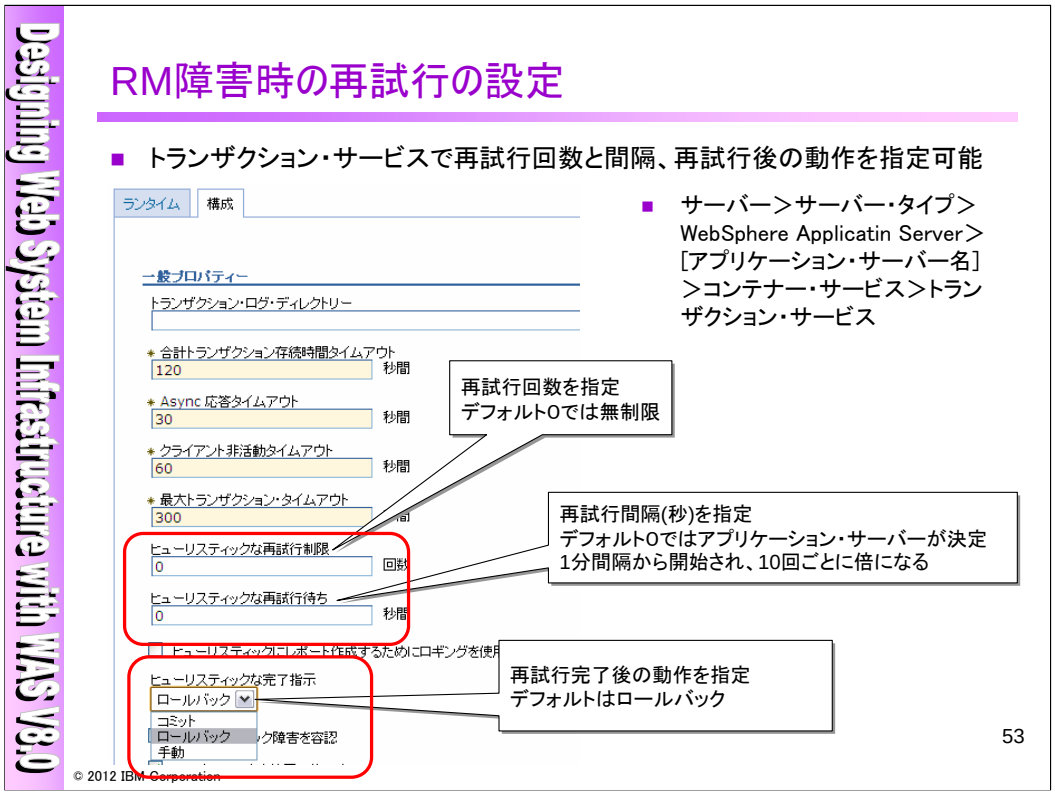
- サーバー＞サーバー・タイプ＞WebSphere Applicatin Server＞[アプリケーション・サーバー名]＞コンテナー・サービス＞トランザクション・サービス＞ランタイム



再試行を終了することも可能

52

WASがリトライを行っている事を管理コンソールにてモニタリングする事が可能です。管理コンソールより、サーバー＞サーバー・タイプ＞WebSphere Applicatin Server＞[アプリケーション・サーバー名]＞コンテナー・サービス＞トランザクション・サービス＞ランタイム>再試行トランザクション – 検討をクリックすることで、トランザクションIDと状況を確認することが出来ます。また、チェックして「終了」ボタンをクリックすることで、再試行(=WASのリトライ)を終了させる事も可能です。

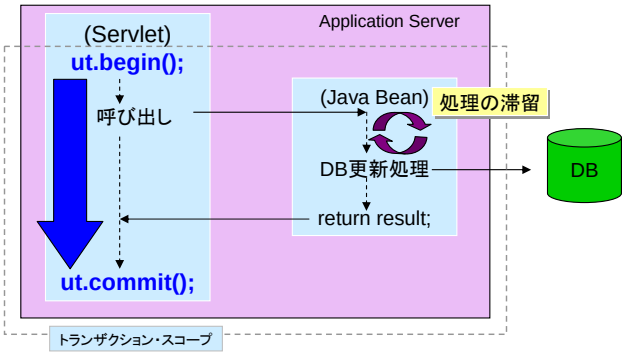


RM障害時の再試行の設定は、再試行回数と再試行間隔、再試行完了後の動作を指定する事が出来ます。管理コンソールより、サーバー＞サーバー・タイプ＞WebSphere Applicatin Server＞[アプリケーション・サーバー名]＞コンテナー・サービス＞トランザクション・サービスを選択し、設定して下さい。

Designing Web System Infrastructure with WAS V8.0

トランザクション・タイマー

- 想定以上に長いトランザクション処理を終了させたい場合に有効
  - クライアント側にタイムアウトを返す
  - トランザクションをロールバックさせる
- 設定可能なタイマー値
  - 合計トランザクション存続時間タイムアウト
  - クライアント非活動タイムアウト
  - 最大トランザクション・タイムアウト



54

トランザクション・タイマーは、想定以上に長いトランザクション処理を中止し、クライアント側にエラーを返したりリソースのロック解除などのためにトランザクションをrollbackしたい場合に有効です。

設定可能なタイマー値として以下のものがあります。

・合計トランザクション存続時間タイムアウト

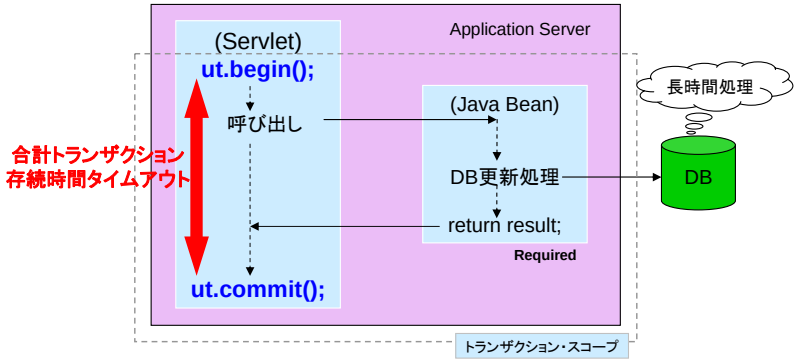
・クライアント非活動タイムアウト

・最大トランザクション・タイムアウト

Designing Web System Infrastructure with WAS V8.0

合計トランザクション存続時間タイムアウト

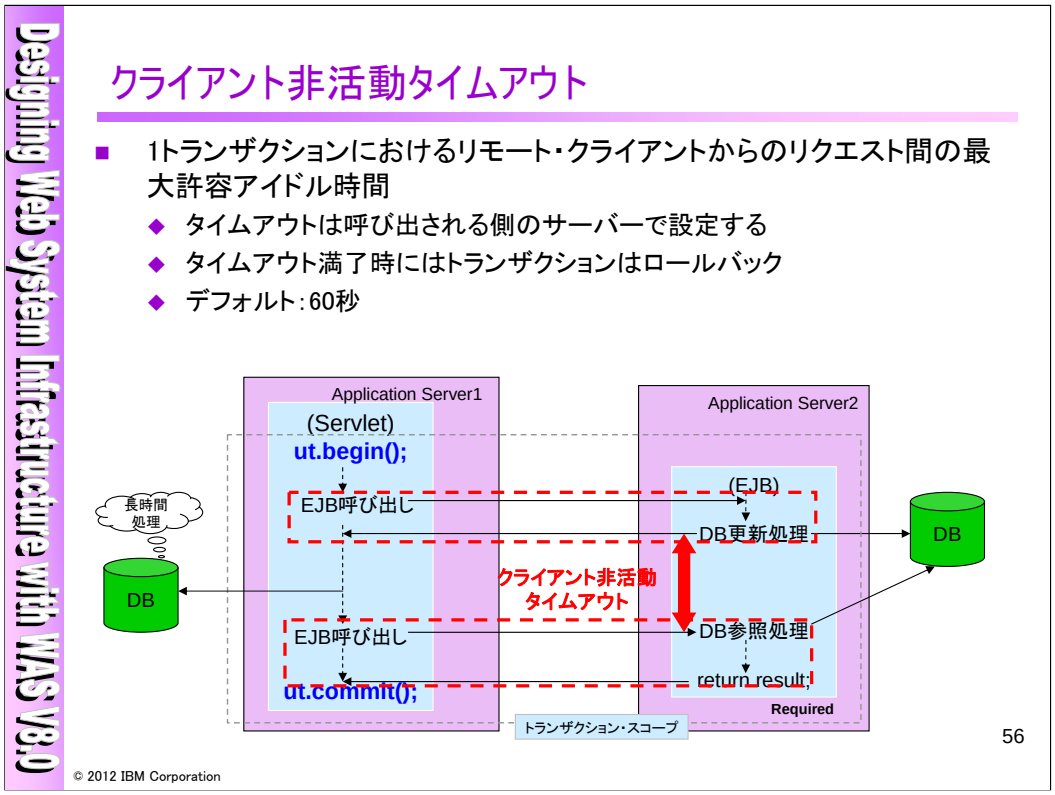
- サーバー上で新規に開始されたトランザクションのタイムアウト値
  - トランザクション開始からコミットまでの時間
  - タイムアウト満了時にコミット処理が開始されていないトランザクションはロールバックされる
  - デフォルト：120秒



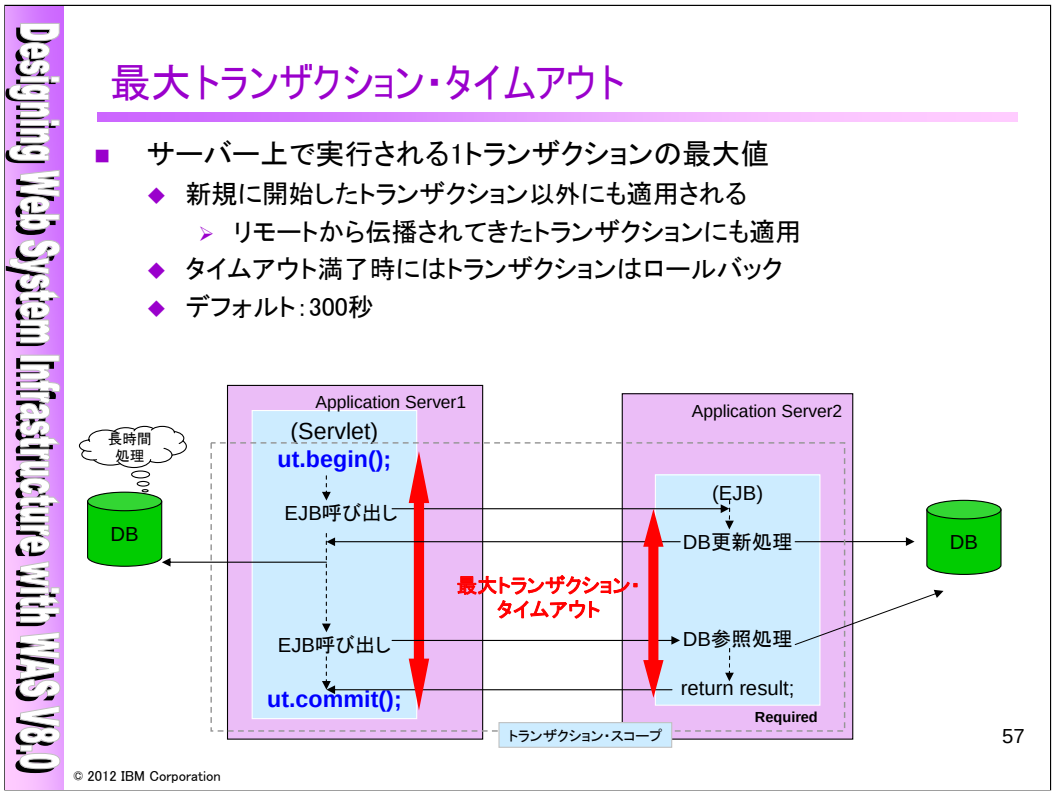
55

合計トランザクション存続時間タイムアウトについて説明します。このタイムアウトはこの値が設定されたアプリケーションサーバーで新規に開始されたトランザクションに対して有効になります。トランザクションの開始からコミットまでの時間がこのタイムアウト値を上回るとトランザクションはロールバックされます。デフォルト値は120秒です。

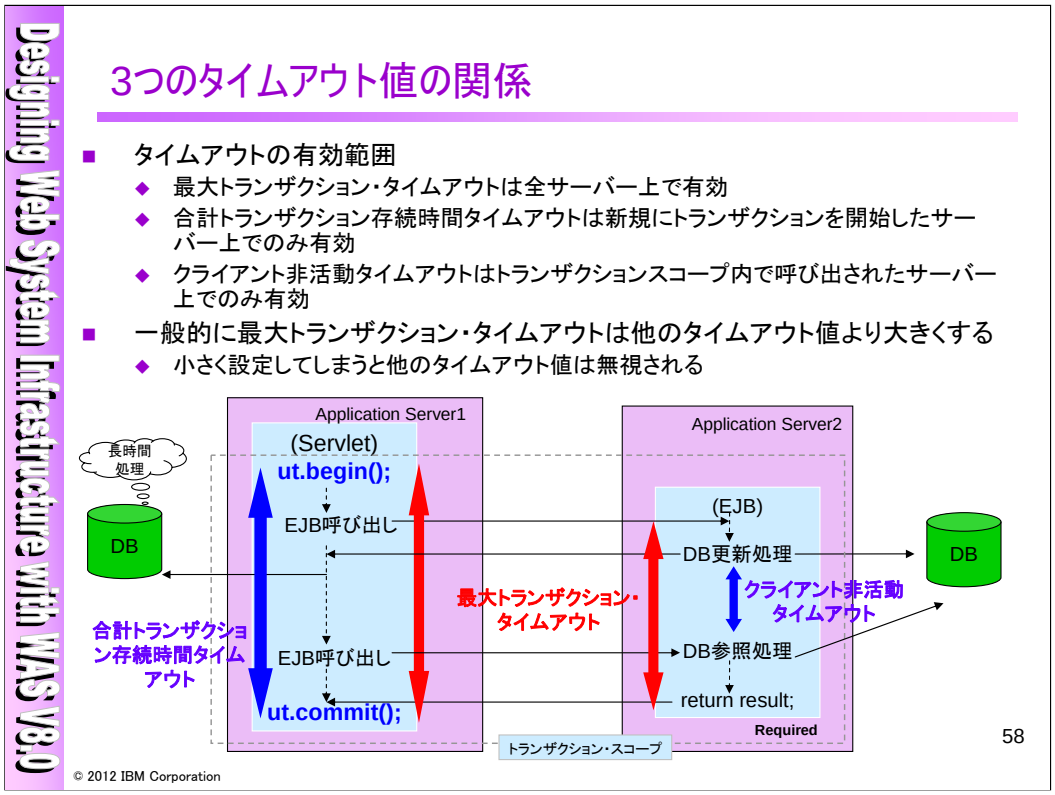
※注意：ロールバック実行のタイミングについてですが、タイムアウト時点ではトランザクションはロールバックされるようにマークされるだけです。実際にロールバックが実行されるのはコミット処理時になります。以降、クライアント非活動タイムアウト、最大トランザクション・タイムアウトについても同様です。



クライアント非活動タイムアウトはトランザクション処理中にリモート呼び出しがあるような場合に有効になります。この値は呼び出されるサーバー側に設定し、クライアントからの呼び出しアイドル時間がこのタイムアウト値を過ぎた場合にトランザクションをロールバックします。デフォルト値は60秒です。



最大トランザクション・タイムアウトは全てのトランザクション処理の各サーバー上でのタイムアウト値です。このタイムアウトはサーバー上で新規に開始されたトランザクションだけではなく、リモートから伝播されてきたトランザクションについても有効となります。デフォルト値は300秒です。



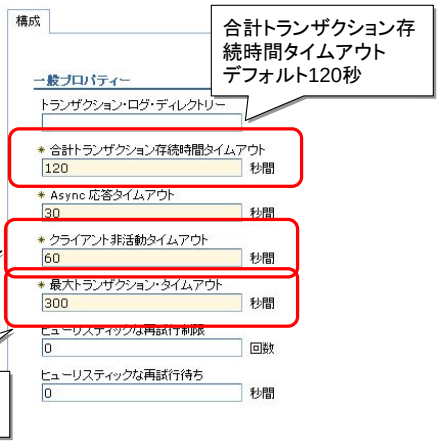
3つのタイムアウトの関係をまとめてみました。最大トランザクション・タイムアウトはサーバーの全トランザクションに適用される最大値で、この値が他のタイムアウト値より短い場合は合計トランザクション存続時間タイムアウトなどの設定は無視されます。一般的には3つのタイムアウト設定を有効にするため、最大トランザクション・タイムアウトは他のタイムアウト値よりも大きな値にします。

WASV8.0 Information Center「アプリケーション・サーバーのトランザクション・プロパティーの構成」
http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tjta_settlog.html

Designing Web System Infrastructure with WAS V8.0

タイムアウト値設定方法

- 管理コンソールでの設定
  - 3つタイムアウト値ともに管理コンソールで設定可能
  - サーバー＞サーバー・タイプ＞WebSphere Applicatin Server＞［アプリケーション・サーバー名］＞トランザクション・サービス



- 合計トランザクション存続時間タイムアウトはアプリケーションでも設定可能
  - 管理コンソールで設定した値を上書きする
  - EJB拡張デプロイメント記述子（ibm-ejb-jar-ext.xmi）
  - javax.transaction.UserTransactionインターフェースのsetTransactionTimeoutメソッド

59

3つのタイムアウト値は管理コンソールから設定可能です。合計トランザクション存続時間タイムアウトはアプリケーション側でも設定することができます。この場合、管理コンソールの値は上書きされます。

WASV8.0 Information Center「トランザクション・サービス設定」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/udat_contranserv.html

WASV8.0 Information Center「トランザクション・デプロイメント属性の構成」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tjta_entra2.html

Designing Web System Infrastructure with WAS V8.0

トランザクション障害設計ポイント

- グローバル・トランザクションにおける障害設計
  - TM障害
    - トランザクション・リカバリーにはトランザクション・ログが必要
    - クラスター構成にすることにより、TMを冗長化することが可能
    - ヒューリスティック・ディシジョンの使用
      - トランザクション・ログの損失など、やむを得ない状況以外では避けるべき
      - データの整合性は保証されない
  - RM障害
    - TMはRMの復旧まで、確定指示のリトライを行う
    - WASの管理コンソールからリトライに関する設定が可能
- タイマー
  - トランザクションに関連するタイマー値
    - 合計存続時間タイムアウト
    - クライアント非活動タイムアウト
    - 最大トランザクション・タイムアウト
  - 一般的に最大トランザクション・タイムアウト値は他タイマー値よりも大きく設定

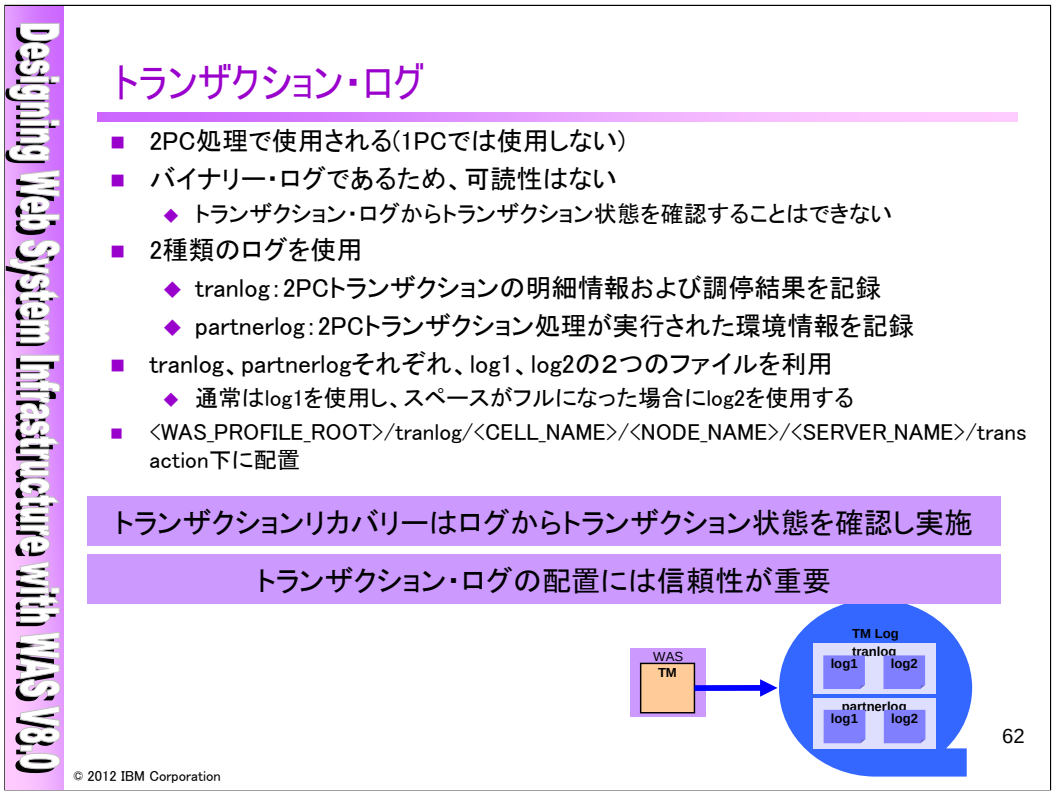
60

トランザクションの障害設計ポイントのまとめです。

Designing Web System Infrastructure with WAS V8.0

3. トランザクション設計のポイント

61



トランザクションに関する運用項目としては、トランザクション・ログの管理があげられます。WASでは、次の2種類のログ使用します。

tranlog：2PCトランザクションの明細情報および調停結果を記録

partnerlog：2PCトランザクション処理が実行された環境情報を記録

tranlog/partnerlogは、それぞれlog1、log2の2つのファイルを利用し、通常はlog1を使用し、log1のスペースがフルになった場合のみlog2を使用します。ログは< *WAS_PROFILE_ROOT* >/tranlog/<*cell_name*>/<*node_name*>/<*server_name*>/transaction下に配置されます。トランザクションのリカバリー時は、ログからトランザクション状態を確認しますので、トランザクション・ログの配置には信頼性が重要になります。

またこれらのログはバイナリー・ログとなっており、可読性はありません。

Designing Web System Infrastructure with WAS V8.0

トランザクション・ログの配置場所

- トランザクション・ログの可用性
  - WAS V8.0にはトランザクション・ログを2重化する機能はない

ディスク障害対策として、RAID装置やミラーリングといった、ハードウェアによる高可用性の実現が必要

- ログ書き込みのパフォーマンス
  - ログへの書き込みにはDisk I/Oが発生するため、ディスクのパフォーマンスがトランザクションのパフォーマンスに直接影響する

パフォーマンスを考慮する場合は、高速ディスクが必要

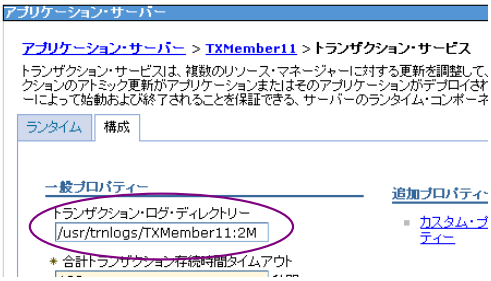
63

信頼性を保つ為に、トランザクション・ログの可用性を考慮する必要がありますが、WAS V8.0にはトランザクション・ログを2重化する機能はありません。ハードウェアなどによる可用性が必要となります。また、2フェーズ・コミット処理では複数のログに書き込む必要があり、ログ書き込みの際にはDisk I/Oが発生する為、パフォーマンスを考慮する場合には高速ディスクが必要となります。

Designing Web System Infrastructure with WAS V8.0

トランザクション・ログのサイジング

- ログ・スペースがフルになるとトランザクションはロールバック
  - ログが使用可能になるまで、トランザクションがコミットできない状態
  - SystemErr.logに、メッセージ「WTRN0083W: トランザクション・ログがフルです。トランザクションはロールバックされました。」が出力
- トランザクション・ログのサイジングが必要なケース
  - 同時に相当量のトランザクションが発生
  - 1UOWに時間のかかるトランザクション
- ログ・ファイルはサーバー始動時に固定サイズで作成(デフォルト:1024KB)
  - 終了したトランザクションのログ記述が上書きされていく循環ログ
  - サイズ指定可能: *directory_name;file_size*



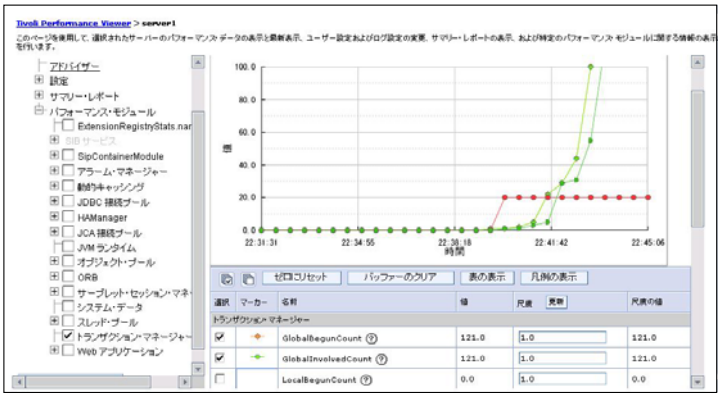
64

ログ・スペースがフルになるとトランザクションはロールバックされますので、同時に相当量のトランザクションが発生する場合や1UOWに時間のかかるトランザクションの場合は、トランザクション・ログのサイジングをご検討下さい。トランザクション・ログは、デフォルトでは1MBになり、上書きされる循環ログ(サイズ指定可能)になります。

Designing Web System Infrastructure with WAS V8.0

Tivoli Performance Viewer (TPV) によるモニタリング

- TPVを用いてトランザクションのモニタリングが可能
  - TPVのパフォーマンス・モジュールで「トランザクション・マネージャー」を選択
  - 「基本」、「拡張」、「すべて」の各レベルで取得できる内容が異なる
    - デフォルトの基本レベルではActiveCount、CommitedCount、RollebackCountの3つの値が取得される



65

Tivoli Performance Viewer (TPV) により、トランザクション処理に関する情報をモニタリングできます。モニタリングする場合はTPVのパフォーマン・スモジュールで「トランザクション・マネージャー」を選択します。取得できる内容は「基本」、「拡張」、「すべて」のレベルで異なります。TPVの使用方法など詳細についてはパフォーマンス設計のセッションを参照ください。

Designing Web System Infrastructure with WAS V8.0

TPVで取得できるカウンター一覧

データ名	説明
GlobalBegunCount	サーバー上で開始されたグローバル・トランザクションの合計数
GlobalInvolvedCount	呼び出し元のグローバル・トランザクションに組み込まれて処理されたトランザクションの合計数
LocalBegunCount	サーバー上で開始されたローカル・トランザクションの合計数
ActiveCount	実行中のグローバル・トランザクションの合計数
LocalActiveCount	実行中のローカル・トランザクションの合計数
OptimizationCount	最適化のために1フェーズ・コミットに変換されたグローバル・トランザクションの合計数
CommittedCount	コミットしたグローバル・トランザクションの合計数
LocalCommittedCount	コミットしたローカル・トランザクションの合計数
RolledbackCount	ロールバックされたグローバル・トランザクションの合計数
LocalRolledbackCount	ロールバックされたローカル・トランザクションの合計数
GlobalTimeoutCount	タイムアウトになったグローバル・トランザクションの合計数
LocalTimeoutCount	タイムアウトになったローカル・トランザクションの合計数
GlobalTranTime	グローバル・トランザクションの平均所要時間(ミリ秒)
LocalTranTime	ローカル・トランザクションの平均所要時間(ミリ秒)
GlobalBeforeCompletionTime	グローバル・トランザクション完了前の平均所要時間(ミリ秒)
LocalBeforeCompletionTime	ローカル・トランザクション完了前の平均所要時間(ミリ秒)
GlobalPrepareTime	グローバル・トランザクション作成の平均所要時間(ミリ秒)
GlobalCommitTime	グローバル・トランザクションのコミットの平均所要時間(ミリ秒)
LocalCommitTime	ローカル・トランザクションのコミットの平均所要時間(ミリ秒)

66

TPVで取得できるカウンターの一覧です。各カウンターの詳細については以下の資料を参照ください。

WASV8.0 Information Center「トランザクションのカウンター」
http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Finfo%2Fae%2Fae%2Ftjta_cfgpeer.html

Designing Web System Infrastructure with WAS V8.0

まとめ・参考文献

67

Designing Web System Infrastructure with WAS V8.0

まとめ

- トランザクションの基本概念
  - トランザクションとは複数処理の集まった1つの意味をもつ処理単位
  - ローカル・トランザクションとグローバル・トランザクションが存在
- WAS V8.0のトランザクション機能
  - JTA1.1に準拠
  - WAS独自の拡張機能を実装
- トランザクション設計
  - アプリケーション設計
    - トランザクション実装方法とトランザクション・スコープ
  - 障害対策設計
    - トランザクション障害時のリカバリー
    - タイマー設定
  - 運用管理設計
    - ログ運用
    - TPVを用いたモニタリング

68

Designing Web System Infrastructure with WAS V8.0

参考文献

- Information Center
  - WebSphere Application Server V 8.0 インフォメーション・センター
  http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp
- ワークショップ資料
  - WebSphere Application Server V8 アナウンスメント・ワークショップ
  http://www.ibm.com/developerworks/jp/websphere/library/was/was8_ws/
- その他
  - Java Transaction API (JTA)
  http://java.sun.com/javaee/technologies/jta/
  - Distributed Transaction Processing:The XA Specification
  https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?catalogno=c193

69

Designing Web System Infrastructure with WAS V8.0

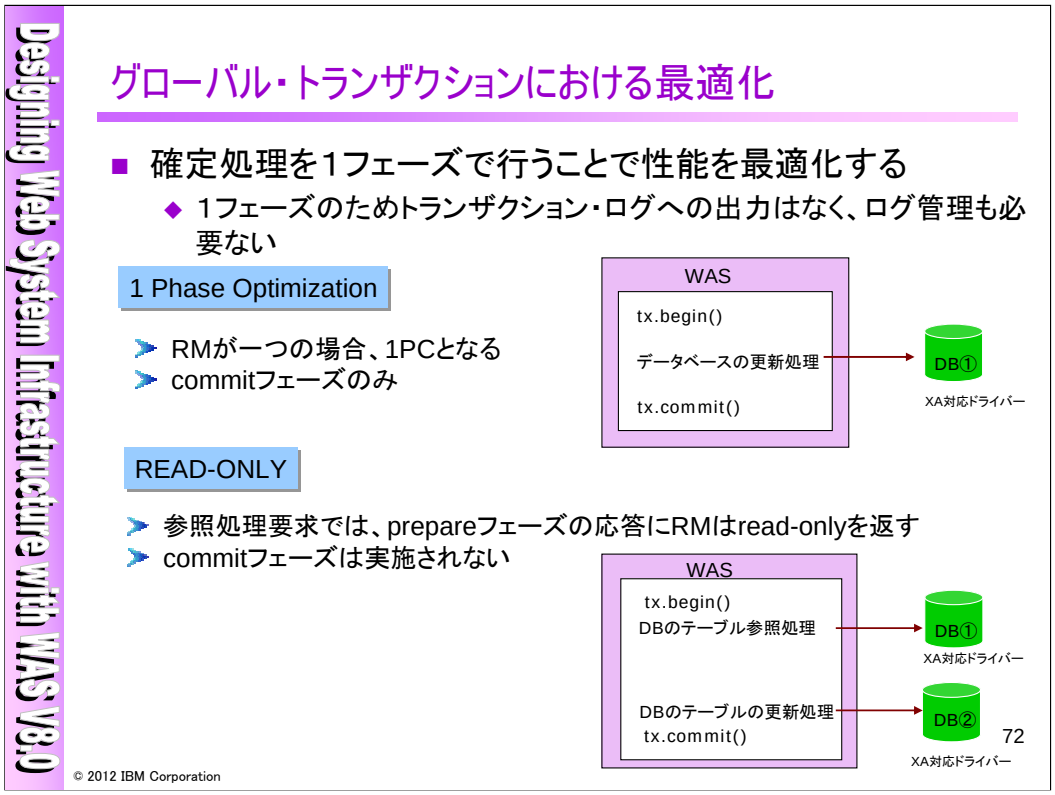
WASV8.0によるWebシステム基盤設計Workshop

トランザクション設計
ー 参考 ー

Designing Web System Infrastructure with WAS V8.0

1. トランザクションの基礎

71



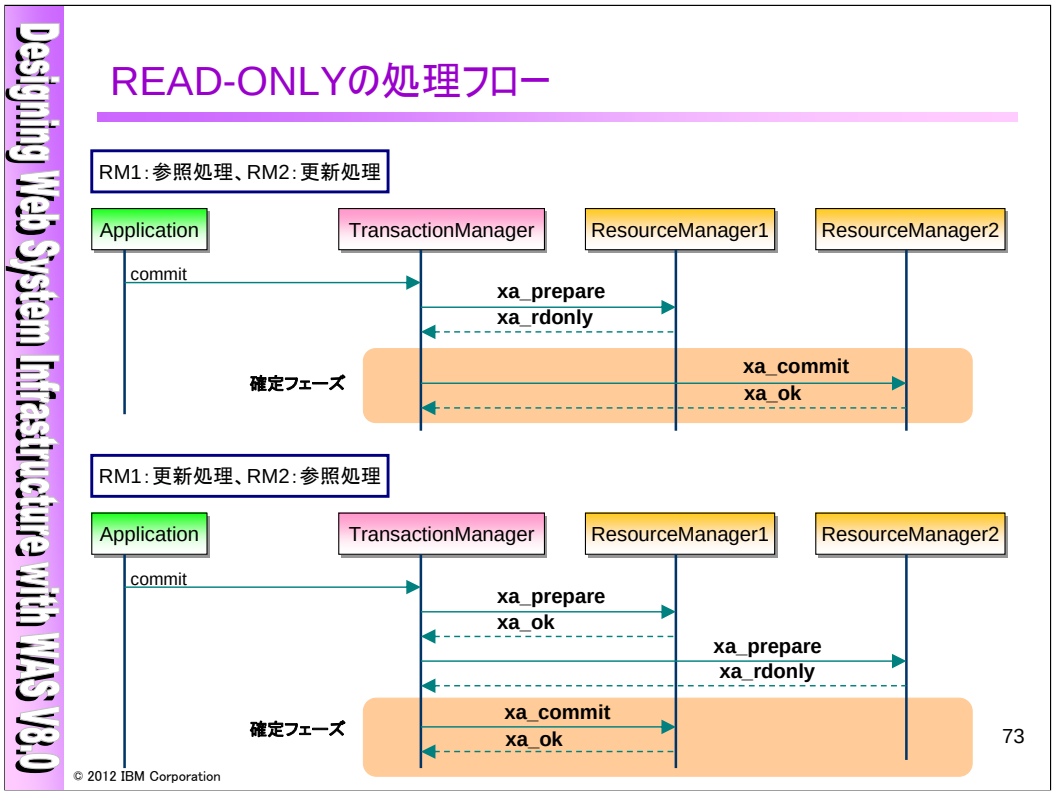
TMが管理するトランザクションをグローバル・トランザクションとも呼びます。通常、2フェーズ・コミットであればprepare /commitフェーズがあるのですが、グローバル・トランザクションにおける最適化はその確定(commit)フェーズを1フェーズで行います。1フェーズですので、トランザクション・ログへの出力がなく、ログを管理する必要がありません。最適化には、以下の2種類があります。

・ 1 Phase Optimization

TMが起動時にアクセス先のリソースがXA対応であるかを確認し、XA対応でありRMがひとつの場合は、2フェーズ・コミットではなく1フェーズ・コミットで対応できると判断します。

・READ-ONLY

2フェーズ・コミット処理を行う際に、参照処理要求では、prepareフェーズの応答にRMがread-onlyを返す事により、commitフェースの実施は必要はないと判断します。



グローバルトランザクションにおける最適化のREAD-ONLYの処理フローについて説明します。RMがTMからのprepare処理に対して、rd_onlyを返すことにより2フェーズ・コミットではなく1フェーズコミットが実行されるという最適化が行われております。

・RM1:参照処理、RM2:更新処理の場合

ApplicationがTMに対してcommit処理を実行し、TMはRM1に対してprepare処理を実行します。RM1はxa_rdonlyを返すことにより、TMはRM1に対してcommitフェーズは必要ないと判断し、即座にRM2に対してのみcommitフェーズを実行します。

・RM1:更新処理、RM2:参照処理の場合

ApplicationがTMに対してcommit処理を実行し、TMはRM1/RM2に対してprepare処理を実行します。この時、RM2は参照処理ですのでxa_rdonlyを返すことにより、TMはRM2に対してはcommitフェーズを実行せず、RM1のみに対してcommitフェーズを実行します。

Designing Web System Infrastructure with WAS V8.0

3. トランザクション設計のポイント

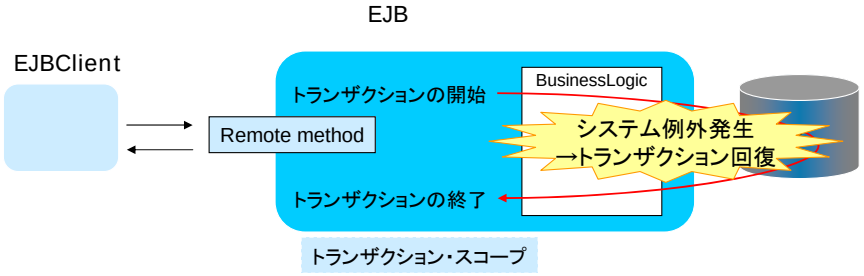
74

Designing Web System Infrastructure with WAS V8.0

CMTの例外処理 (1/2)

- システム例外
  - 予測できない、または予測できてもアプリケーションがリカバリーできない障害によって発生した例外
    - DB接続取得の失敗
    - JNDI APIの例外
    - JVMのエラー

  ⇒ EJBコンテナーが自動的にトランザクションの回復を行う



75

CMT利用時の例外発生について説明します。CMT利用時にはスローされた例外がシステム例外かアプリケーション例外かによってコンテナーの動きが変わります。

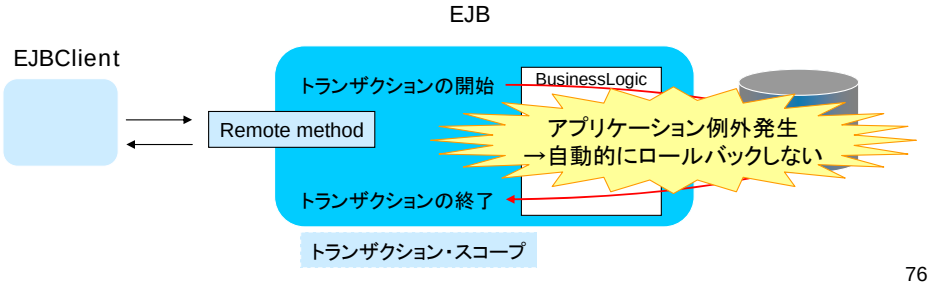
システム例外とは、予測できない、または予測できてもアプリケーションがリカバリーできない障害によって発生した例外になります。この場合、EJBコンテナーがトランザクションの回復(ロールバック)を自動的に実施してくれます。

Designing Web System Infrastructure with WAS V8.0

CMTの例外処理 (2/2)

- アプリケーション例外
  - ビジネス･ロジックの中で対応すべき例外
    - 残高不足、メッセージ･フォーマットの不正など

⇒ アプリケーション例外が発生時、トランザクションをロールバックさせたい場合、アプリケーション内で明示的に記述する必要がある
  - sessionCtx.setRollbackOnly();



76

アプリケーション例外とは、ビジネス･ロジックの中で対応すべき例外(例：残高不足やメッセージフォーマットの不正)になります。この場合、トランザクションをロールバックさせる為には、アプリケーション内で明示的にsessionCtx.setRollbackOnly()メソッドを記述する必要があります。アプリケーション例外が発生しても自動的にロールバックされませんので、ご注意下さい。

Designing Web System Infrastructure with WAS V8.0

EJBのトランザクション属性

- EJBの場合、トランザクション属性によってスコープが決定
  - CMTにEJBがどのように参加するかを決定
  - Beanごとにデプロイメント・ディスクリプター(DD)に指定
- トランザクション属性
  - Required
    - クライアントがトランザクションを開始していればその中でEJBのメソッドが実行される。
    - トランザクションを開始していなければEJBコンテナーが開始する
  - RequiresNew
    - 新しいトランザクションを開始・終了する
  - Supports
    - クライアントがトランザクションを開始していればその中でEJBのメソッドが実行される。
    - トランザクションが開始していなければEJBコンテナーは何もしない
  - Mandatory
    - クライアントが必ずトランザクションを開始している必要がある
  - NotSupported
    - クライアントがトランザクションを開始しているかどうかに関わらず、EJBのメソッド実行はその中に含まれない
  - Never
    - EJBメソッドは、トランザクションに参加しない

77

EJBの場合には、トランザクション属性によってスコープが決定されます。BMTではコーディングにて設定する必要があり、CMTではデプロイメント・ディスクリプター(DD)に、以下のトランザクション属性を設定する必要があります。

・Required

・RequiresNew

・Supports

・Mandatory

・NotSupported

・Never

Designing Web System Infrastructure with WAS V8.0

EJB使用時のトランザクション属性の設定

- トランザクション内でEJBを実行したい
  - Required
  - RequiresNew
- 既存のトランザクション内でEJBを実行したい
  - Required
  - Supports
    - トランザクション内で実行されない可能性があるので注意
  - Mandatory
    - 必ず既存のトランザクション内で実行される
- トランザクション外でEJBを実行したい
  - NotSupported
    - EJBクライアントで既にトランザクションが開始されている場合は、トランザクションを中断
  - Never
    - EJBクライアントで既にトランザクションが開始されている場合は、RemoteExceptionをthrowする

78

トランザクション属性の設定方針についてまとめます。

＜トランザクション内でEJBを実行したい場合＞

・Required

・RequiresNew

＜既存のトランザクション内でEJBを実行したい場合＞（クライアントがトランザクションを実行している場合）

・Required

・Supports（※1）

・Mandatory（※2）

(※1)クライアントがトランザクションを実行していない場合は、Beanのトランザクションが実行されない可能性があります。

(※2)必ず既存のトランザクション内で実行されます。既存のトランザクションでトランザクション属性を指定していない場合には、エラーとなります。

＜トランザクション外でEJBを実行したい場合＞

・NotSupported(※3)

・Never(※4)

(※3) EJBクライアントで既にトランザクションが開始されている場合は、トランザクションを中断します。

(※4) EJBクライアントで既にトランザクションが開始されている場合は、RemoteExceptionをthrowします。

Designing Web System Infrastructure with WAS V8.0

トランザクション属性

トランザクション属性	クライアントのトランザクション（T1）	Beanのトランザクション（T2）
Required	なし	T2
	T1	T1
RequiresNew	なし	T2
	T1	T2
Supports	なし	なし
	T1	T1
Mandatory	なし	エラー
	T1	T1
NotSupported	なし	なし
	T1	なし
Never	なし	なし
	T1	エラー

79

各トランザクション属性の詳細につきましては、以下をご参照下さい。

・InfoCenter －「EJB モジュールのコンテナー・トランザクションの定義」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tejb_addcontainertran.html

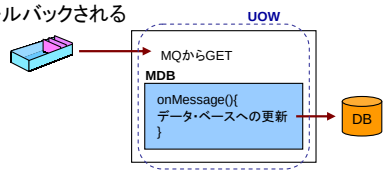
(例)RequiresNew – クライアントのトランザクション:T1、Beanのトランザクション:T2

クライアントがトランザクションが開始している場合、Beanのトランザクションは、クライアントのトランザクション・スコープではなく、Beanで定義されたトランザクション・スコープになります。

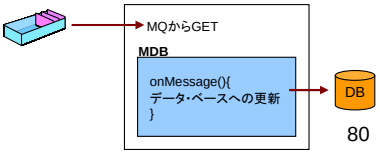
Designing Web System Infrastructure with WAS V8.0

MDBのトランザクション・スコープ

- MDBで設定可能なトランザクション属性はRequiredとNotSupportedのみ
- スコープの分け方によって、障害時のメッセージの行方やサーバー側の対応が異なる
- onMessage()メソッドのトランザクション属性
  - Required：コンテナーの受信処理とonMessage()内の処理が同一トランザクション・スコープ
    - 1処理が異常終了した場合は全ての処理がロールバックされる
    - ロールバックした時のメッセージはキューに戻る
    - EJBコンテナーによる2フェーズ・コミット



  - NotSupported：コンテナーの受信処理もonMessage()メソッド内の処理もトランザクションを管理しない
    - トランザクション管理が行われない



80

MDBで設定可能なトランザクション属性は、"Required"と"NotSupported"になります。トランザクション・スコープの分け方によって、障害時のメッセージの行方やサーバー側の対応が異なります。

【onMessage()メソッドのトランザクション属性】

＜Required：コンテナーの受信処理とonMessage()内の処理が同一トランザクション・スコープ＞

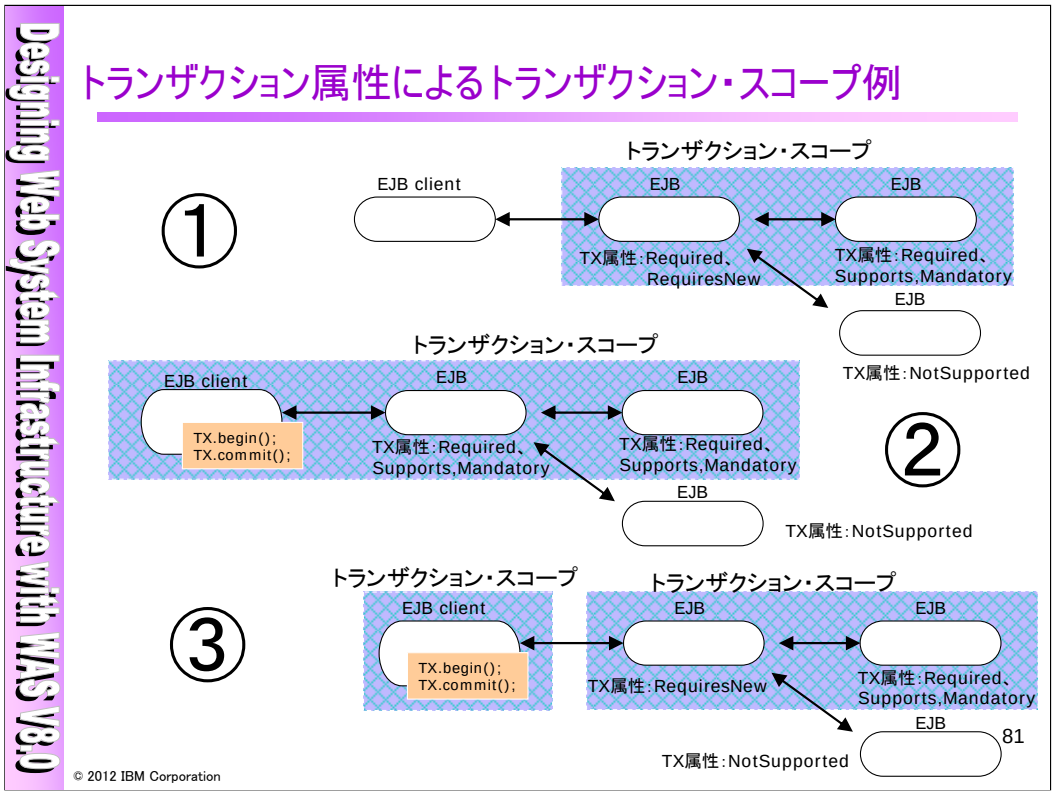
・1処理が異常終了した場合は全ての処理がロールバック

・ロールバックした時のメッセージはキューに戻る

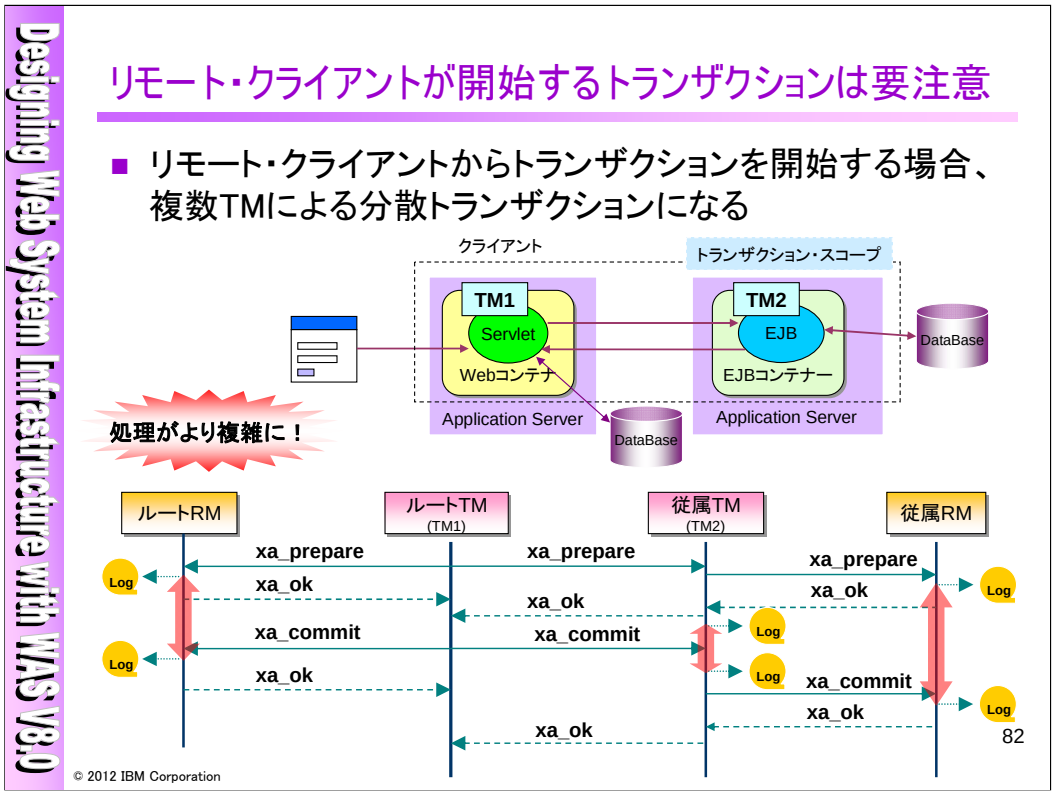
・EJBコンテナーによる2フェーズ・コミット

＜NonSupported：コンテナーの受信処理もonMessage()メソッド内の処理もトランザクションを管理しない＞

・トランザクション管理が行われない



上記は、トランザクション属性によりトランザクション・スコープを分けた例になりますので、ご確認下さい。



リモート・クライアントからトランザクションを開始し、複数のアプリケーションでトランザクション・スコープを割り当てた場合、複数TMによる分散トランザクションとなります。上記の処理フローの通り、インダウト状態が3箇所になり、書き出されるログ数の増加に伴いログ管理が煩雑になります。

従いまして、リモート・クライアントから開始されるトランザクションを実施する場合には、上記考慮点を踏まえたうえ、どうしても使用しなければいけない場合のみご利用下さい。

Designing Web System Infrastructure with WAS V8.0

3. トランザクション設計のポイント

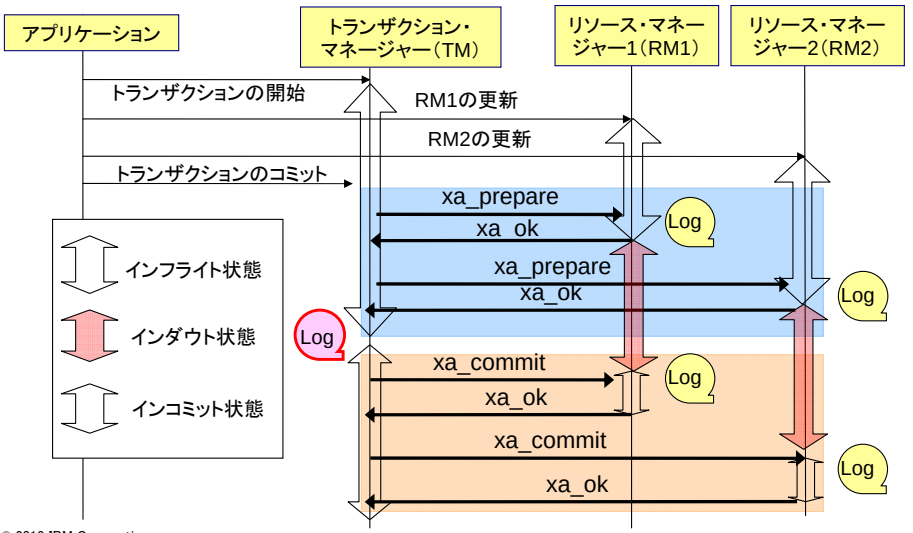
3-1 アプリケーション設計
3-2 障害設計
3-3 運用管理設計

83

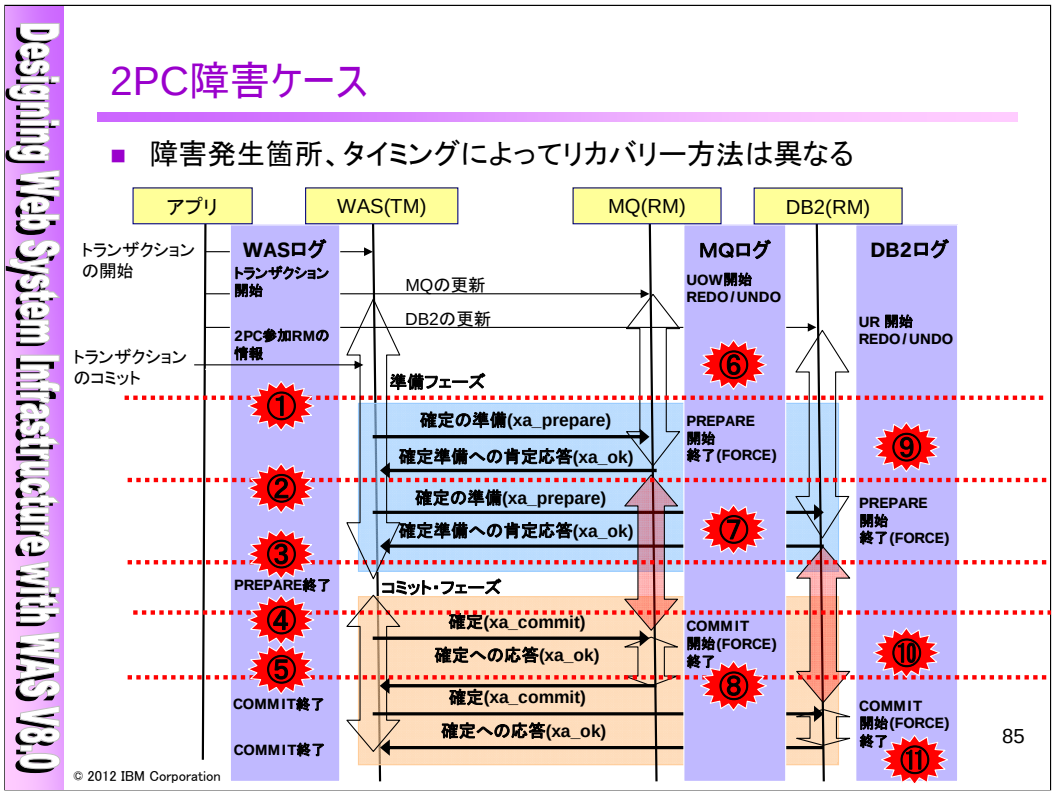
Designing Web System Infrastructure with WAS V8.0

トランザクションの状態

- 2フェーズコミットを使用したトランザクションの場合、処理状況によってトランザクション・マネージャー、リソース・マネージャーは3つの状態をとる。
  - インフライト状態 / インダウト状態 / インコミット状態



84



2フェーズ・コミットの障害は、どこで障害が起こるのかによってそれぞれ発生する問題は変わってきます。上記はWASをトランザクション・マネージャー、MQ、DB2をリソース・マネージャーとした場合の障害パターンになります。

また、次ページ以降にて各発生箇所毎の対応方法を載せています。

・UNDOとは

直前にユーザが行った操作を取り消し、元に戻すこと。

・ REDOとは

直前の操作を取り消すUNDO機能を行使して取り消したユーザの操作を、もう一度やり直すこと。

Designing Web System Infrastructure with WAS V8.0

2PC障害発生時の動き (1/2)

- TM(WAS)に障害が発生したケース

TMの状態	各RMの状態		障害発生コンポーネントと対応	
WAS	MQ	DB2	WASに障害発生	
インフライト:コミットを表すPREPARE終了ログがない	インフライト:PREPARE終了ログがない	インフライト:PREPARE終了ログがない	①	MQ、DB2ともにPREPARE終了ログが無いことから、トランザクションはインフライトと判断され、UNDOログを使ってロールバックされる。WASとのRESYNC処理は行われない。
	インダウト:PREPARE終了ログがある	インフライト:PREPARE終了ログがない	②	WASのRESTART時、パートナーログからMQ、DB2がRMであることを認識。xa_recoverの実行によりMQにインダウトがあることを発見。WASのトランログ内にPREPARE終了ログが無いため、MQへロールバックを指示。
	インダウト:PREPARE終了ログがある	インダウト:PREPARE終了ログがある	③	WASのRESTART時、パートナーログからMQ、DB2がRMであることを認識。xa_recoverの実行によりMQ、DB2にインダウトがあることを発見。WASのトランログにPREPARE終了ログが無いため、MQ,DB2へロールバックを指示。
インコミット:コミットを表すPREPARE終了ログがある			④	WASのRESTART時、パートナーログからMQ、DB2がRMであることを認識。xa_recoverの実行によりMQ、DB2にインダウトがあることを発見。WASのトランログ内にPREPARE終了ログがあるためMQ,DB2へコミットを指示。
	インダウト:PREPARE終了ログがある	インコミット:COMMIT開始ログがある	⑤	WASのRESTART時、パートナーログからMQ、DB2がRMであることを認識。xa_recoverの実行によりMQにインダウトがあることを発見。WASのトランログ内にPREPARE終了ログがあるためMQにコミットを指示。

86

Designing Web System Infrastructure with WAS V8.0

2PC障害発生時の動き (2/2)

- RM(MQ,DB2)に障害が発生したケース

MQの状態		MQに障害発生	DB2の状態		DB2に障害発生
インフライト: PREPARE終了ログがない	⑥	WASはMQからの応答が無いことを察知してDB2にロールバックを指示。MQはRESTART時、UNDOログを使用してインフライトトランザクションをロールバック。	インフライト: PREPARE終了ログがない	⑨	WASはDB2からの応答が無いことを察知してMQにロールバックを指示する。DB2はRESTART時、UNDOログを使用してインフライトトランザクションをロールバックし、保持していたロックを開放する。
インダウト: PREPARE終了ログがある	⑦	MQはRESTART時、MQログ内のPREPARE終了ログからインダウトがあることを認識。WASとのRESYNCが行われる。WASはトランログ内でのPREPARE終了ログの有無から、MQへコミット/ロールバックを指示。	インダウト: PREPARE終了ログがある	⑩	DB2はRESTART時、DB2ログ内のPREPARE終了ログからインダウトがあることを認識。WASとのRESYNCが行われる。WASはトランログ内でのPREPARE終了ログの有無から、DB2へコミット/ロールバックを指示。
インコミット: COMMIT開始ログがある	⑧	MQはRESTART時、MQログ内のCOMMIT開始ログからインコミットがあることを認識し、コミット処理を終了させる。	インコミット: COMMIT開始ログがある	⑪	DB2はRESTART時、DB2ログ内のCOMMIT開始ログからインコミットがあることを認識し、コミット処理を終了させる。

87

Designing Web System Infrastructure with WAS V8.0

ピア・リカバリーについて

88

Designing Web System Infrastructure with WAS V8.0

トランザクションのピア・リカバリーが有効な場合

- アプリケーション・サーバーのプロセス障害やネットワーク障害が発生した場合です。
  - トランザクション・ログの破損や消失には対応できません。
    - トランザクション・ログが破損していた場合
      - リカバリー時に、トランザクション・ログを処理できず、リカバリー・プロセスが失敗します。
      - プロセスが再起動しても、トランザクション・ログが読み込めないために、プロセスの始動が行えません。
    - トランザクション・ログが消失していた場合、リカバリー・プロセスが、新規にトランザクション・ログを作成します。
  - トランザクション・ログは、耐障害性の高いディスク装置に配置してください。

トランザクションは、ログが命です！

89

トランザクションのピア・リカバリーが有効な場合は、アプリケーション・サーバーのプロセス障害やネットワーク障害が発生した場合になります。トランザクション・ログが破損した場合は、リカバリープロセスが失敗したりプロセスが起動しないといった状況が発生します。トランザクション・ログが消失した場合は、新規にトランザクション・ログを作成するという挙動になります。

従いまして、トランザクション・ログは、耐障害性の高いディスク装置に配置して頂くことが必要になります。

Designing Web System Infrastructure with WAS V8.0

ファイル共有システムについて

90

Designing Web System Infrastructure with WAS V8.0

ファイル共有システムとして何を使用するのか？

- トランザクション・ログを配置できる、ファイル共有システムの条件は、下記の点です。
  - 強制書き込みが可能なこと
  - 排他的ロックが使用可能なこと

91

トランザクション・ログを配置できる、ファイル共有システムの条件は、以下の2点になります。

・強制書き込みが可能なこと

・排他的ロックが使用可能なこと

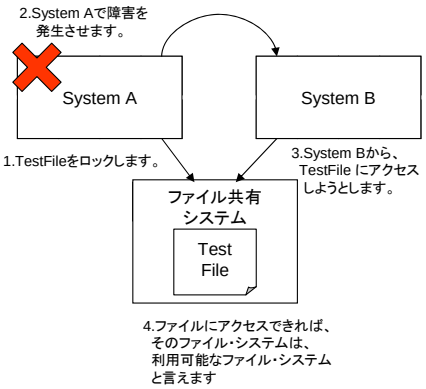
・排他的ロックとは

ロックされたファイルを処理出来るのは1つのリクエストのみになります。リクエストに排他ロックがある場合、他のリクエストはそのファイルを処理する事が出来ません。

Designing Web System Infrastructure with WAS V8.0

ファイル共有システムの確認方法

- 利用するファイルシステムで、障害時にロックが解除されるかを確認する為のコードが利用できます。
  - この方法で確認することにより、障害時にトランザクション・ログが解放され、フェールオーバー先のトランザクション・マネージャーがファイルを使用できるかを確認できます。



利用手順

1. 下記リンクより取得したプログラムを、[System A] [System B]に展開します。
2. アクセス対象となるファイル名を、[properties.txt]に記述します。
   ※ [properties.txt]内の[filename=]を設定します。
3. System Aから[fsLock.class]を実行し、ファイル共有システム上のファイルにアクセスします。
   ※この時点で、ファイルがロックされます。(図[1.])
4. System Aで、プロセス障害やネットワーク障害を発生させます。(図[2.])
5. System Bから[fsVerify.class]を実行し、確認を行います。(図[3.])
6. アクセスできたかどうかが表示されます。(図[4.])
   Lease based lock test FAILED: Lock not freed
   Lease based lock test PASSED: Lock freed

92

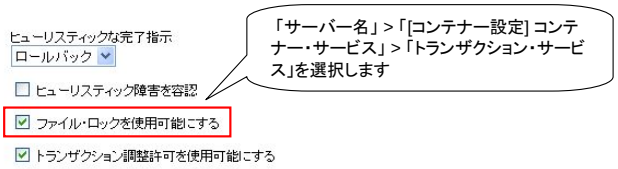
利用するファイル・システムで、障害時にロックが解除されるかを確認する為のコードがあります。これにより、障害時にトランザクション・ログが解放され、フェールオーバー先のトランザクション・マネージャーがファイルを使用できるかを確認できますので、必要に応じてご利用下さい。

「IBM File System Locking Protocol Test for WebSphere Application Server」
http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg24010222

Designing Web System Infrastructure with WAS V8.0

NFSにおける排他的ファイル・ロックの対応

- HA機能は、デフォルトで排他的ファイル・ロックを使用
  - NFSV4の場合　lease-based 排他的ロックが提供されている
    - 障害時、NetworkClientがファイル・ロックをはずす
  - NFSV3の場合　lease-based 排他的ロックが提供されていない
    - WASの排他的ファイル・ロックの機能を使用不可にする設定が必要



- ファイル・ロック機能をOFFにする際の考慮点
  - サーバーが稼動中にも拘らずHAのハートビートの応答が停止する可能性がある
  - 複数サーバーがTranlogにアクセスした場合、データの保全性が失われる可能性がある

93

HA機能は、トランザクション・データを保証するために、デフォルトで排他的ファイル・ロックを使用します。そのためFileSystemのFileLocking機能を使用しますが、FileSystemにより動作が異なります。

・NFSV4は、lease-based 排他的ロックが提供されているため、障害時にNetworkClientがファイル・ロックをはずします。

・NFSV3は、lease-based 排他的ロックが提供されていないため、WASの排他的ファイル・ロックの機能を使用不可にする必要があります。

ただし、WASのファイル・ロック機能をOffにする場合は、以下の点に考慮が必要です。

・システム過負荷またはネットワーク障害などで、サーバが稼動中にも拘らずHAのハートビートの応答が停止する場合があります。

・WASはファイル・ロックを使用してTranlogへの並行アクセスが発生しないようにしていますが、ファイル・ロックをOffにすることで複数のサーバがTranlogにアクセスする可能性が生じ、データの保全性が失われる可能性があります。

ハートビートの待機時間を変更することもできますが、ハートビートによりサーバーの障害を間違って診断するリスクを減らすことと、自動フェールオーバーが発生するまでの時間を長くすることはトレードオフになります。従いまして、アプリケーションの特性、ネットワーク、およびピーク・ワークロードを考慮して調整する必要があります。

ファイル・ロックをOffにする方法は下記をご参照下さい。

WASV8.0 Information Center「ファイル・ロックの使用不可化」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Finfo%2Fae%2Fae%2Ftjta_disable_lock.html

Designing Web System Infrastructure with WAS V8.0

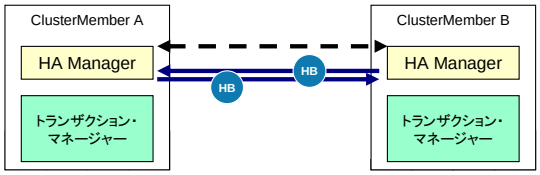
検知のタイミング
フェールオーバー先の設定

94

Designing Web System Infrastructure with WAS V8.0

HAマネージャーによる障害検知

- HAマネージャーが障害として検知するケース
  - 1. プロセス・ダウンにより、ソケットがクローズされた場合
    - プロセスに障害が発生したことにより、ソケットがクローズされるとHAマネジャーにより即時検知され、対象ノードがダウンしたと判断します。
  - 2. HAマネージャー間のHeartBeatが断絶した場合
    - ネットワーク障害やサーバーのハングにより、HeartBeatの送信が正常に行えない回数が一定回数を超えると、HAマネージャーは対象ノードがダウンしたと判断します。
  - 3. OSのTCP/IP KeepAliveのタイムアウトにより、ソケットがクローズされた場合
    - OS側のTCPKeepAlive時間が経過した場合、コネクションがクローズされます。このコネクションのクローズを検知し、対象サーバーがダウンしたと判断します。



95

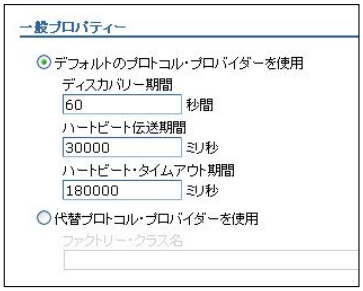
HAマネージャーが障害として検知するケースは以下の3ケースです。

1. プロセス・ダウンにより、ソケットがクローズされた場合
2. HAマネージャー間のHeartBeatが断絶した場合
3. OSのTCP/IP KeepAliveのタイムアウトにより、ソケットがクローズされた場合

Designing Web System Infrastructure with WAS V8.0

ネットワーク障害の検知

- ネットワーク障害は、HAマネージャーのHeartBeatにより検知されます。
- 「サーバー」>「[コア・グループ] >「コア・グループ設定」>「 <コア・グループ名>」>「ディスカバリーおよび障害検出」
  - ディスカバリー時間
    - ディスカバリー・プロトコルが実行される時間間隔 (デフォルト:60秒間)
  - ハートビート伝送期間
    - ハートビートの送信間隔 (デフォルト:30秒)
  - ハートビート・タイムアウト期間
    - 障害検出時間。この時間内にパケットを受信しないと、障害が宣言される (デフォルト180秒)



96

ネットワーク障害はHeartBeatにより検知され、この値を調整することで、障害検知までの時間を設定出来ます。設定は、管理コンソールのコア・グループの追加プロパティにて行います。また、V7.0とV6.1以前のバージョンが同一コア・グループに含まれるような混合セル環境ではV6.1の場合と同様にカスタムプロパティーで設定する必要があります。

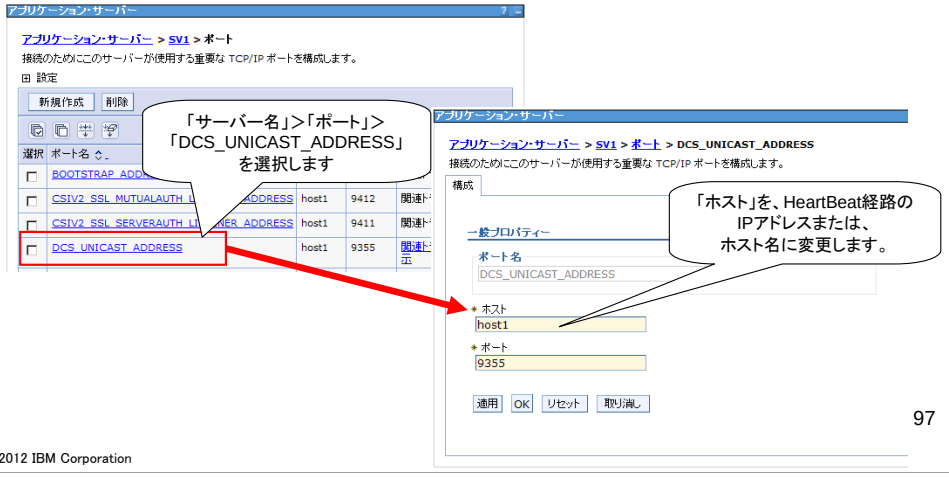
WAS V8.0 Information Center「コア・グループに対するデフォルトの障害検出プロトコルの構成」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=%2Fcom.ibm.websphere.nd.multiplatform.doc%2Finfo%2Fae%2Fae%2Ftrun_ha_cfg_faildetect.html

Designing Web System Infrastructure with WAS V8.0

HeartBeat経路の設定

- HeartBeatとして、デフォルトのトランスポート・タイプである「DCS」を選択している場合のHeartBeat経路は、DCS_UNICAST_ADDRESSを使用して指定します。
  - クラスター・メンバー毎に設定が必要です。



97

HeartBeat経路は、管理コンソールより設定可能ですので、ご確認下さい。

Designing Web System Infrastructure with WAS V8.0

フェールオーバー先の設定

- フェールオーバー先を設定する場合は、ポリシーを作成して定義を行う必要があります。
  - クラスターメンバーが多数あり、フェールオーバー先をグルーピングしたい。
    - 全クラスター・メンバーに対するポリシーを作成します。
      - ポリシー
        - 「Nポリシー中の1つ」
      - 一致基準
        - TYPE=WAS_TRANSACTIONS
        - GN_PS=[セル名]/[ノード名]/[クラスター・メンバー名]
      - 「優先サーバーのみ」をチェック
      - 優先サーバー
        - グルーピング対象のクラスターメンバーを選択します。
  - 処理能力の高いサーバーで、フェールオーバー処理をさせたい。
    - クラスター単位のポリシーを作成します。
      - ポリシー
        - 「Nポリシー中の1つ」
      - 一致基準
        - TYPE=WAS_TRANSACTIONS
        - IBM_hc=[クラスター名]
      - 優先サーバー
        - 処理能力の高いノード上で稼働するクラスター・メンバーを選択します。

98

フェールオーバー先を設定する場合は、ポリシーを作成して定義を行う必要があります。フェールオーバー先をグルーピングしたり、処理能力の高いサーバーでフェールオーバー処理を実施する事が出来ますので、ご確認下さい。

Designing Web System Infrastructure with WAS V8.0

HAマネージャーのポリシーを設定する際の注意

- トランザクション・マネージャーに対するポリシー設定を行う場合は、注意が必要です。
  - トランザクション・マネージャーを起動できない場合、アプリケーション・サーバーの起動が出来ません。
    - トランザクション・マネージャーのポリシーで、「Nポリシー中の1つ」を設定するときには、必ず「フェールバック」を行い、トランザクション・マネージャーが自身のプロセスで、稼働するようにしてください。
    - 「オペレーション・ポリシーなし」は使用できません。

99

トランザクション・マネージャーに対するポリシー設定を行う場合は注意が必要ですので、上記をご確認下さい。

Designing Web System Infrastructure with WAS V8.0

100