

D I O S A / X T P V2.1
データ変換・通信オプション

利用の手引

輸出する際の注意事項

本製品(ソフトウェア)は、外国為替及び外国貿易法で規制される規制貨物(または役務)に該当することがあります。

その場合、日本国外へ輸出する場合には日本政府の輸出許可が必要です。

なお、輸出許可申請手続きにあたり資料等が必要な場合には、お買い上げの販売店またはお近くの当社営業拠点にご相談下さい。

はしがき

本書は、業務システムの構築を支援するD I O S A / X T P データ変換・通信オプション製品群の利用の手引です。

本書の読者としては、業務アプリケーション開発を担当し、OS、TPBASE、TAM、Oracle、その他関連 PP の使用法を一通り心得ているシステム技術者を想定しています。

2017 年 9 月 4 版

本書の関連説明書としては次のものがあります。

- DIOA/XTP 導入の手引き
- DIOA/XTP 利用の手引き
- DIOA/XTP メモリキャッシュ 利用の手引き
- DIOA/XTP データストア 利用の手引き
- DIOA/XTP データ変換・通信オプション 導入の手引き
- DIOA/XTP API リファレンス
- DIOA/XTP コマンドリファレンス
- DIOA/XTP 環境定義リファレンス
- DIOA/XTP メッセージリファレンス

備考

- (1) Microsoft、Windows は、米国あるいはその他の国における米国 Microsoft Corporation の商標または登録商標です。
- (2) UNIX は、X/Open カンパニーリミテッドが独占的にライセンスしている米国ならびに他の国における登録商標です。
- (3) HP、HP-UX は、Hewlett-Packard 社の商標または登録商標です。
- (4) Linux は、Linus Torvalds の米国およびその他の国における商標または登録商標です。
- (5) Red Hat は、米国およびその他の国における Red Hat, Inc. の商標または登録商標です。
- (6) Oracle と Java は、Oracle Corporation およびその子会社、関連会社の米国およびその他の国における登録商標です。
- (7) This product includes software developed by the Apache Group for use in the Apache HTTP server project (<http://www.apache.org/>).
- (8) その他、記載されている会社名、製品名は、各社の登録商標または商標です。

目次

- 第1章 概要..... 1
 - 1.1 目的と特徴..... 1
 - 1.1.1 目的..... 1
 - 1.1.2 特徴..... 1
 - 1.1.3 データ同期制御..... 1
 - 1.2 諸概念..... 2
 - 1.2.1 データ同期制御..... 2
 - 1.2.2 オンライン中メンテナンス..... 3
- 第2章 機能..... 5
 - 2.1 DBアクセス制御..... 5
 - 2.2 データ同期制御..... 6
 - 2.2.1 データ同期機能..... 6
 - 2.2.2 更新状況同期機能..... 6
 - 2.2.3 更新ログ出力抑止機能..... 7
 - 2.2.4 更新ログ照会機能..... 7
 - 2.3 オンライン中メンテナンス..... 8
 - 2.3.1 オンライン中DBリカバリ支援機能..... 8
 - 2.3.2 TAM再配置機能..... 16
 - 2.3.3 セーブロード機能..... 23
- 第3章 アプリケーション開発..... 26
 - 3.1 セーブロードプログラミング..... 26
 - 3.1.1 構成変更情報取得出口..... 26
 - 3.2 ユーザアプリケーションプログラム..... 29
 - 3.2.1 プログラムの構造..... 29
 - 3.3 DBアクセスプログラミング..... 31
 - 3.4 Javaアプリケーション..... 36
 - 3.4.1 センタ間データ同期制御用更新ログ出力..... 36
 - 3.5 アプリケーションの生成..... 39
 - 3.5.1 ユーザアプリケーション..... 39
 - 3.5.2 TPBASE上で動作するプログラムの生成..... 40

第1章 概要

1.1 目的と特徴

1.1.1 目的

社会インフラの重要性が増す中、大規模アプリケーションシステムにおいても、高信頼、高性能、高運用・高稼働性、そしてアプリケーション開発の高生産性・即応性への要求が益々高まっています。

本製品は、これらの要件を満足するべく、以下の機能を提供します。

- データベースサーバ障害時でもオンライン業務の継続が必要となるシステムの開発を支援する超高可用性を実現したデータベース制御機能
- フロントシステム／バックアップシステム構成で多重化しているシステムの、業務システム継続性を向上させるためのデータ同期／移行機能

1.1.2 特徴

システム構築における開発作業の効率化と維持管理の簡易化や共通化を可能とするデータベース制御機能を提供します。

1.1.3 データ同期制御

データ同期制御は、ユーザアプリケーションが更新したユーザデータと同期したデータを、同一システム上、他システム上に保持するための機能です。

データ同期制御は以下の特徴があります。

- ユーザアプリケーション処理と同一のトランザクションでは、更新内容の保存のみをおこない、それ以降のデータ同期処理は、即時実行される非同期処理でおこなうため、リアルタイムトランザクションへの影響を抑えることが可能です。
- 保存した更新内容は、トランザクションの原子性や順序性を保持して確実に処理されることが保証されるため、データ同期対象システムの運用状態などを意識する必要がありません。
- TAM 表の同期データを、別システムの TAM 表として保持するといった、単純なレプリケーションだけでなく、TAM 表と同期した Oracle 表を同一システム上、別システム上に保持するような形態も可能です。
- TAM の更新内容については、DB アクセス制御機能と連携して自動的に出力するため、ユーザアプリケーションは、データ同期のための特別な処理をおこなう必要がありません。
- 更新内容は、データ圧縮、データ分割されるため、同期処理中のリソースの消費を抑えることができます。

1.2 諸概念

1.2.1 データ同期制御

データ同期制御には、以下の共通概念があります。

(1) 更新ログ

更新ログは、フロントシステムにおける TAM/Oracle の 1 更新の更新内容を保持するデータです。

C アプリケーションが、DB アクセス制御機能を利用して TAM を更新した場合、DB アクセス制御機能によって、更新ログが自動的に生成されます。Java アプリケーションが Oracle を更新した場合、DB アクセス制御機能、データ同期制御機能を使用して更新ログを生成、登録する必要があります。

(2) ログデータ

1 トランザクションで発生した更新ログは、1 つのデータにまとめられて DIOSA/XTP データストアが管理している、TAM/Oracle 上のプールファイル(データストアの制御表)に格納されます。この、まとめられたデータのことを、ログデータと呼びます。通常 1 トランザクションで、1 ログデータが発生します。DB アクセス制御機能を利用して TAM を更新した場合、ログデータの生成、格納は自動的に実行されます。

Java アプリケーションが Oracle を更新した場合、トランザクション終了時にデータ同期制御機能のトランザクション終了処理を呼び出し、登録された更新ログを 1 つのログデータとしてプールファイルへ格納する必要があります。

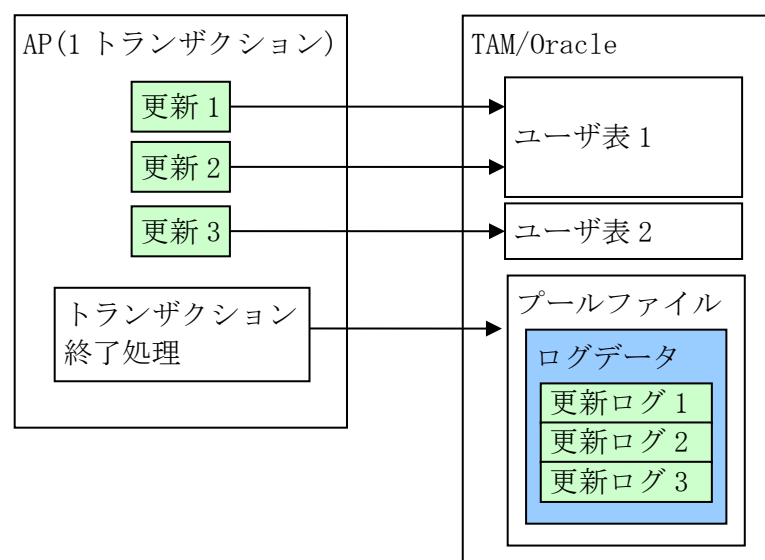


図 1.2-1 更新ログとログデータ

(3) ストリーム(スーパーストリーム)

ストリームとは、DIOSA/XTP データストアの概念で、ログデータをシーケンシャルに処理する単位です。同一ストリームに発生したログデータは、発生順に転送、処理されることが保証されます。

データストアによるログデータの送受信、更新ログの反映処理はストリーム単位に並列処理されるため、ストリーム数を増やすほど更新ログ反映処理の性能が向上します。

ただし、異なるストリームに発生した更新ログ間の処理順序は保証されないため、更新ログを複数のストリームに分散させて処理する際には注意が必要です。

なお、データストアのリファレンスなどで、スーパーストリームという記述がありますが、ストリームと同一の概念です。

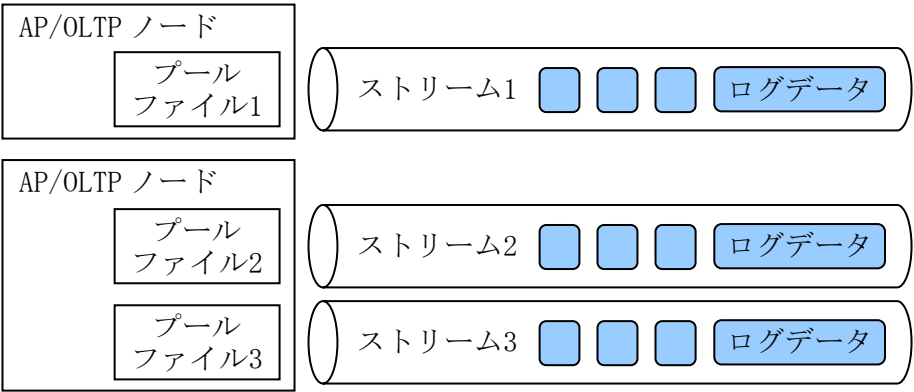


図 1.2-2 ストリーム

1.2.2 オンライン中メンテナンス

オンライン中メンテナンス機能には、以下の概念があります。

(1) **運用情報**

MAP 毎の TAM 再配置 (データベース構成変更を行うための手順) の実行状態を管理する情報を運用情報と呼びます。

運用情報には以下の 2 つの要素があります。

- ・ 運用モード
- ・ MAP 構成変更状態

運用モードは、TAM 再配置の進捗状態を把握するための状態です。以下の 5 つの状態があります。

運用モード	説明
TAM アクセスモード	通常の運用を行っている状態です。
TAM→Oracle 移行状態	TAM アクセスモードから Oracle アクセスモードに切り替える途中の状態です。
Oracle アクセスモード	TAM のメンテナンスを行う状態です。
Oracle→TAM 移行状態	Oracle アクセスモードから TAM アクセスモードに戻す途中の状態です。
未使用状態	MAP の追加直後/削除した後の状態です。

MAP 構成変更状態は MAP の構成を変更する TAM 再配置手順を実施中かを表します。

MAP の構成を変更する TAM 再配置手順を実施中は全データ削除 (TRUNCATE) を行えないため、この状態により全データ削除の実行可否を制御します。

- ・ 通常 : TAM 再配置を行っていない、または MAP の構成を変更しない TAM 再配置中の状態
- ・ 構成変更中 : MAP の構成を変更する TAM 再配置中の状態

(2) **ユーザカレントデータ群**

ユーザカレントデータ群は DIOSA/XTP が管理する論理表のうち、利用者が DB アクセス制御機能を利用してデータアクセスするものを指します。TAM 再配置機能、セーブロード機能は、ユーザカレントデータ群を

処理の対象とします。

論理表をユーザカレントデータ群とするには、以下の定義を行う必要があります。

- ・ メモリキャッシュ機能の環境定義 IMTABLECONF 節に、論理表を「表種別=1～9」（利用者が使用する論理表）として定義する。
- ・ DB アクセス制御機能の環境定義 DACENV 節に、論理表を定義する。

(3) **データ変換・通信オプション制御情報群**

データ変換・通信オプション制御情報群は DIOSA/XTP が管理する論理表のうち、DB アクセス制御機能の制御表を指します。TAM 再配置機能、セーブロード機能は、データ変換・通信オプション制御情報群を処理の対象とします。

第2章 機能

2.1 DBアクセス制御

DB アクセス制御機能では、以下の機能を提供します。

(1) DB アクセス API

DB アクセス API は、C アプリケーションが TAM のデータにアクセスするための API 群です。DB アクセス API は、DIOSA/XTP メモリキャッシュを利用して TAM にアクセスします。

また、DB アクセス API は、DB アクセス API を利用したデータ更新内容を記録した更新ログを生成し、ユーザデータと同じトランザクションでデータベースに登録します。登録された更新ログは、データ同期制御によって TAM-ORACLE データ同期やセンタ間データ同期に利用されます。これにより、TAM-ORACLE データ同期、センタ間データ同期におけるトランザクションの原子性を担保します。

(2) ユーザデータ状態管理 API

ユーザデータ状態管理 API は、TAM 上に存在すべきデータのメインキーの一覧をアプリケーションが管理するための API 群です。ユーザデータ状態管理 API で登録したメインキーの一覧は、TAM ロード機能によって Oracle から TAM にデータをロードする時に利用されます。

ユーザデータ状態管理 API の使い方によって、データの登録から削除まで TAM と Oracle のデータを同期したり、データ登録後、高速アクセスが必要な間だけ TAM にデータを置き、高速アクセスが不要になったらそのデータを TAM 上から削除して Oracle 上にだけ残したりといったデータのライフサイクルに幅を持たせることができます。

(3) Java アプリケーション向け更新ログ生成 API

Java アプリケーション向け更新ログ生成 API は、Java アプリケーションによる Oracle の更新情報から、データ同期制御によってセンタ間データ同期を行うための更新ログを生成するための API 群です。アプリケーションは、本 API で生成した更新ログをデータ同期制御機能の API を使用してデータベースに登録します。

2.2 データ同期制御

2.2.1 データ同期機能

データ同期制御では、2つのデータ同期機能を提供します。

(1) TAM-Oracle データ同期制御

TAM に対する更新内容を保存し、非同期のトランザクションで Oracle に対しても同じ更新をおこなうことで、Oracle 上のユーザデータを TAM のユーザデータと同期させます。Oracle に対する更新は環境定義で定義し、定義によって TAM 表と 1:1 に対応した Oracle 表更新だけでなく、異なる表へのアクセスなどを実行させることも可能です。

(2) センタ間データ同期制御

フロントシステムでのデータベース更新内容(TAM/Oracle)をバックアップシステムに転送/反映することで、非活性状態のバックアップシステムのデータをフロントシステムと同期させます。

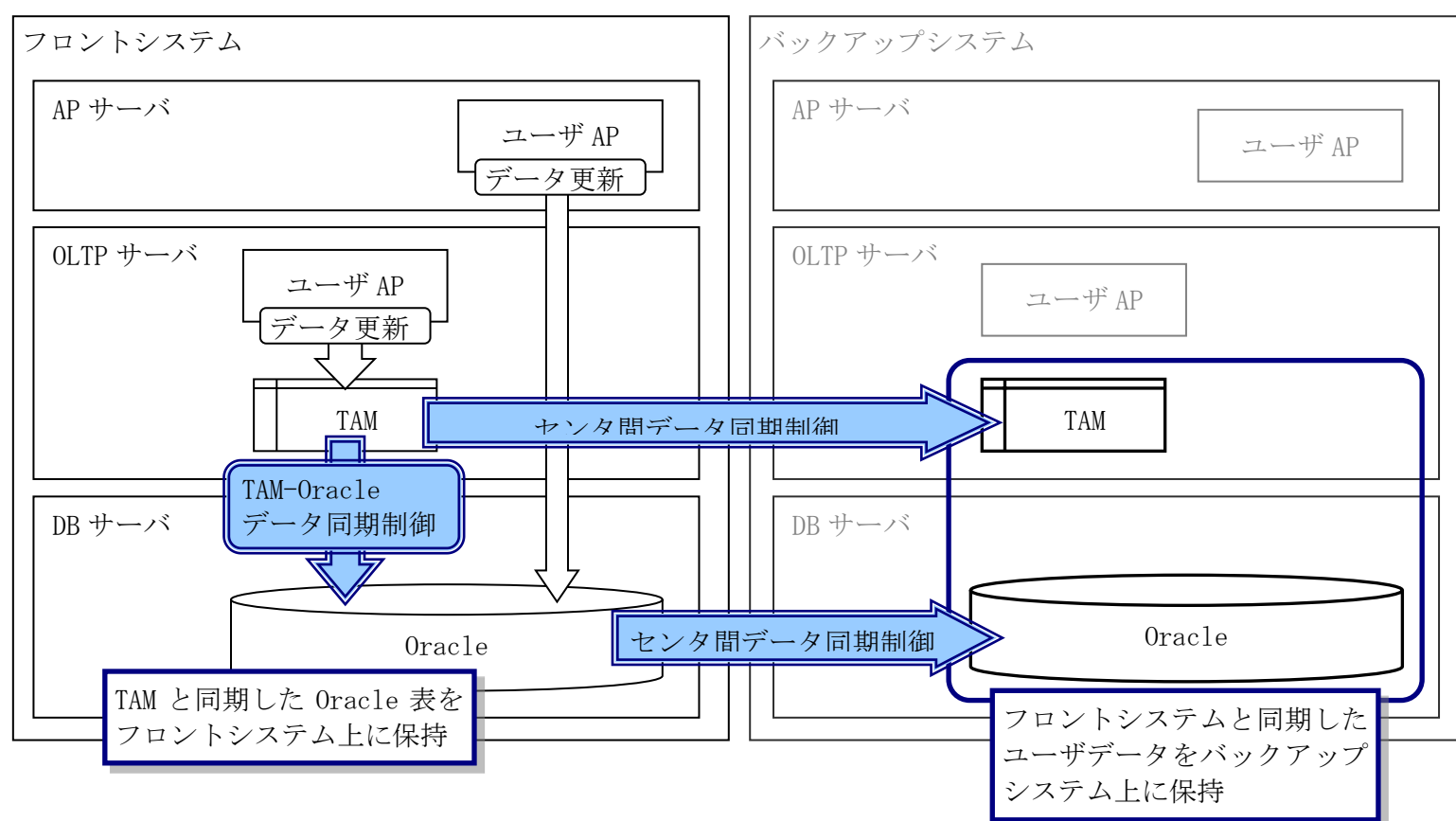


図 2.2-1 データ同期制御

2.2.2 更新状況同期機能

セーブロード機能で、TAM や Oracle のユーザデータを他システム上のデータなどから構築した場合、同期対象のシステムと DIOSA/XTP データストアの処理状態を同期させないと、データ同期制御を継続して実施することができません。

データ同期制御では、同期をおこなうためのコマンドを提供します。

2.2.3 更新ログ出力抑止機能

バックアップシステム障害などで、データ同期処理をおこなえない状態が長時間続くと、更新ログが蓄積され、格納用の領域を使い切ってしまう場合があります。

このような状況では、更新ログを新規に出力することはできないため、データ同期制御は中断しますが、オンラインアプリケーションは処理を継続することができます。

データ同期制御では、更新ログを出力できない状態になると、自動的に更新ログの出力を抑止するモードに移行します。データ同期可能な状態に復旧し、セーブロード機能で、データ同期対象のデータを復旧させたら、更新ログ出力抑止を解除するコマンドを実行することで、再び更新ログが出力される状態になります。

出力抑止については、コマンドで強制的に抑止状態にすることも可能です。ストリームを指定して出力抑止状態にすることで、該当ストリームに対してのみ、更新ログが出力されない状態となります。

2.2.4 更新ログ照会機能

更新ログのスーパーストリームや通番を指定して、更新ログの内容を表示する機能です。

データ同期でエラーが発生した場合などに、更新ログのキーや更新内容の概要を知ることができます。

2.3 オンライン中メンテナンス

オンライン中メンテナンスでは、オンライン中(※1)の障害復旧やデータベース構成変更を行うための以下の3つの機能を提供します。一部の機能は、オフライン中にも使用することができます。

- オンライン中DBリカバリ支援機能
- TAM再配置機能
- セーブロード機能

(※1) 業務運用中の状態をオンライン中と呼び、業務を停止した状態をオフライン中と呼びます。

2.3.1 オンライン中DBリカバリ支援機能

オンライン中DBリカバリ支援機能はOracle障害、及びシステム障害が発生した場合に復旧を行うためのオンライン中DBリカバリ手順を提供します。

(1) オンライン中DBリカバリ手順

オンライン中DBリカバリ手順は、Oracle障害復旧手順とプールファイル障害復旧手順、システム障害復旧手順を提供します。この手順に従って復旧することで、フロントシステム障害以外の場合は、フロントシステムでの業務を停止することなく継続運転したままの状態ですべての障害を復旧することができます。フロントシステム障害の場合は、短時間でフロントシステムでの業務を開始できます。

(a) Oracle障害復旧

OracleDB障害発生後のOracleDB上のユーザデータを復旧する手順です。OracleDB障害後、データストアログデータ実行制御(ログリーダ)による更新ログのOracle反映(非同期更新)が停止しているため、更新ログを格納するTAMのプールファイル(※1)に更新ログが蓄積し続けます。このプールファイルの状態に応じて、以下2パターンから実施する復旧手順を選択します。

復旧パターン	障害発生時の 更新ログ有無	プールファイルのオーバー フロー（※2）
X1. プールファイルの更新ログが失われていない	残っている	オーバーフローしていない
X2. プールファイルの更新ログが失われている		オーバーフローしている
	残っていない	オーバーフローしていない

(※1) プールファイルについては「データストア 利用の手引」を参照してください。

(※2) ログリーダを無効化しないまま全てのスタックファイルが満杯になるとプールファイル障害(オーバーフロー)となり、更新ログの出力が停止します。

以下に、各復旧パターンの概略を記載します。

(X1) プールファイルの更新ログが失われていない

OracleDBの復旧後、プールファイルに蓄積された更新ログをOracleDBに反映することでユーザデータを復旧します。

(X2) プールファイルの更新ログが失われている

OracleDB の復旧後、TAM のユーザデータを OracleDB にセーブすることでユーザデータを復旧します。プールファイルがオーバーフローしている場合はフロント以外の拠点のユーザデータの復旧も必要となります。

(b) プールファイル障害復旧

プールファイルがオーバーフローした際の復旧手順です。プールファイルのオーバーフローは、OracleDB 障害や通信パス障害等により更新ログが蓄積し続けることで発生します。この場合は、プールファイルの状態により、以下 4 パターンから実施する復旧手順を選択します。

復旧パターン	障害発生拠点	プールファイルの場所	転送元拠点の更新ログ有無(※1)
X3. TAM のプールファイルがオーバーフローしている場合	フロントシステム	TAM	転送元拠点は存在しない
X4. OracleDB のプールファイルがオーバーフローしている場合		OracleDB	
レプリケーションの再開	フロントシステム以外	TAM または Oracle	残っている
システム障害復旧 (H. 障害先切り離し、I. 第三拠点切り離し)			残っていない

(※1) オーバーフローしたプールファイルの最も新しい更新ログが転送元となる拠点のプールファイルに残っている場合、転送元拠点から転送を再開することでセンタ間データ同期を再開することができます。

以下に、各復旧パターンの概略を記載します。

(X3) TAM のプールファイルがオーバーフローしている場合

転送先拠点を切り離した後、オーバーフローを解消し、プールファイルに蓄積された更新ログを OracleDB に反映することでユーザデータを復旧します。

(X4) OracleDB のプールファイルがオーバーフローしている場合

転送先拠点を切り離した後、オーバーフローを解消することで復旧ができます。

レプリケーションの再開

オーバーフローの解消後、転送元拠点に更新ログが残っている場合にはレプリケーションを再開することで拠点間のデータ同期を再開します。

システム障害復旧

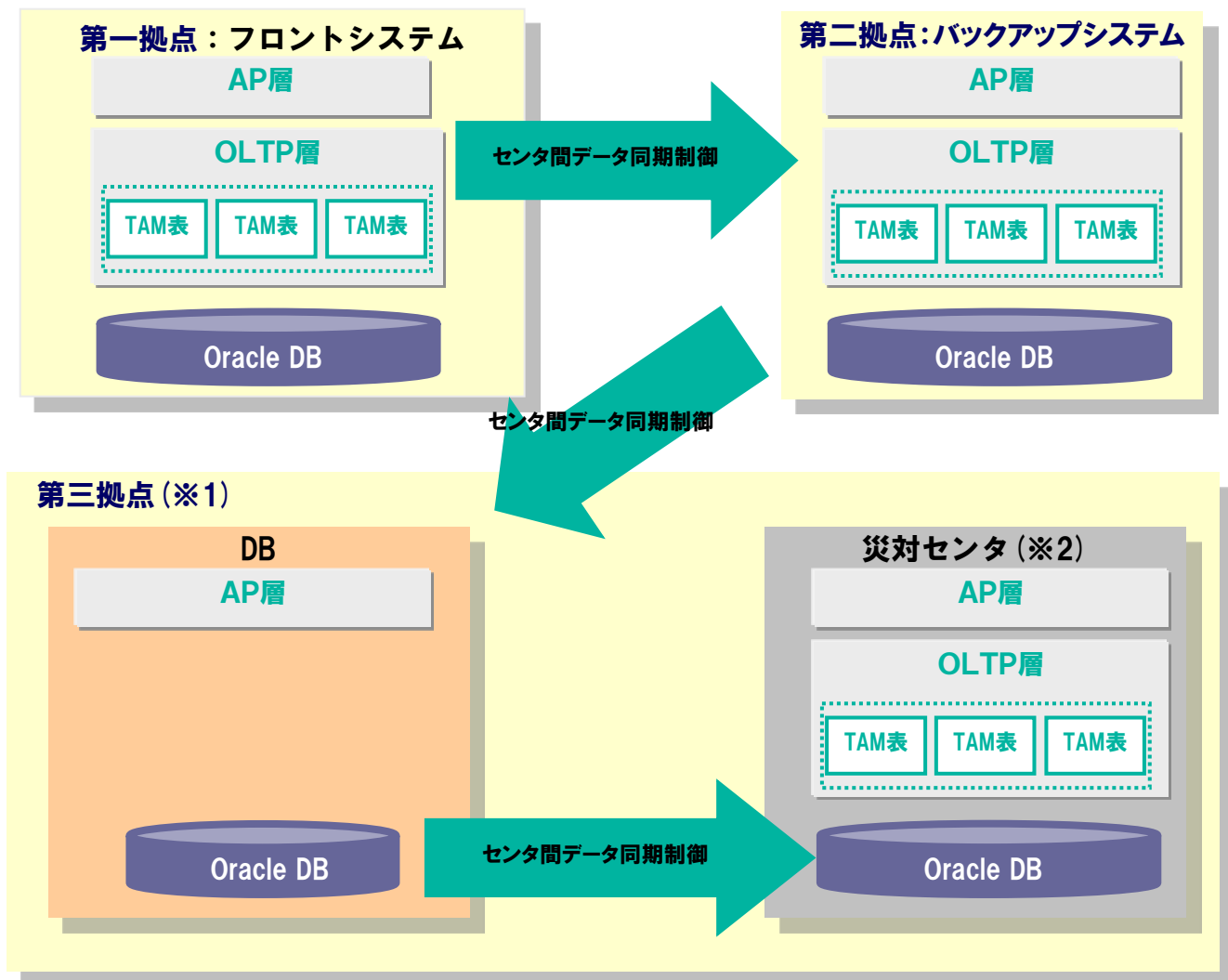
オーバーフローの解消後、転送元拠点に更新ログが残っていない場合はオーバーフローが発生した拠点をシステム障害として拠点の切り離しを行います。切り離しの手順は後述する「システム障害復旧」を参照してください。

レプリケーションの再開以外の復旧を行った場合、プールファイル障害復旧だけでなく、切り離した拠点の復旧も必要となります。この場合の復旧手順は後述する「システム障害復旧」を参照してください。

(c) システム障害復旧

システム障害復旧手順のシステム構成は、下記のシステム構成モデルをベースに手順化しています。

＜システム構成モデル＞



(※1) 第三拠点はセンタ間データ同期制御を通してデータのバックアップを行う DB と、災害時などのフロントシステム、バックアップシステムの障害に備える災対センタから構成されます。

(※2) 通常時の災対センタは非稼動状態ですが、第三拠点(DB)のログデータ実行制御機能により災対センタの OracleDB へのデータ同期が行われ、最新の状態が保たれます。

システム障害復旧手順は、以下の2段階に分けてシステムを復旧します。

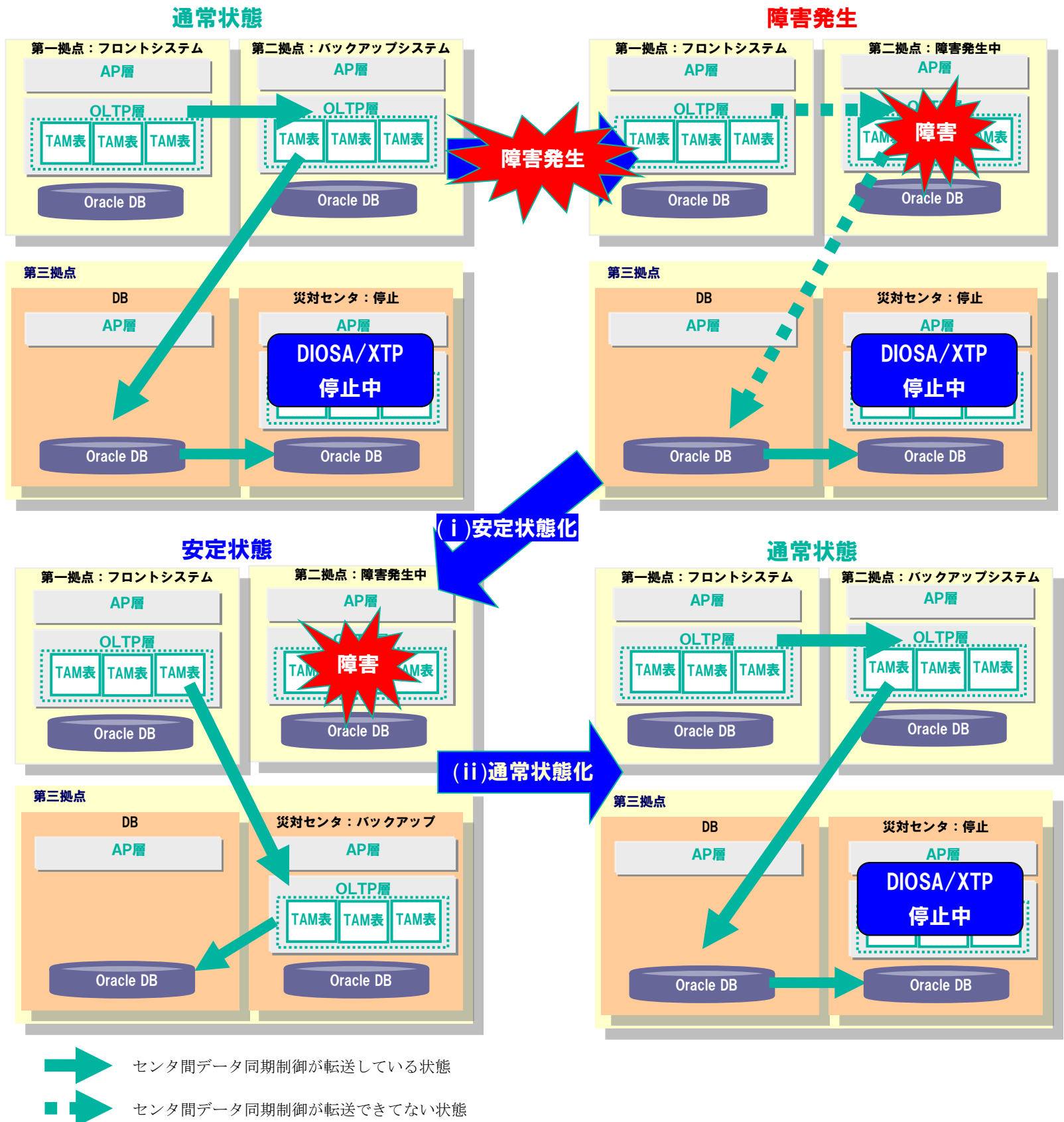
(i) 安定状態化：障害状態から安定状態への復旧

(ii) 通常状態化（完全復旧）：安定状態から通常状態への復旧

※安定状態とは、フロントシステムで業務が継続運転できる状態。

障害発生から通常状態に戻す復旧の流れ(イメージ)を以下に示します。

<復旧の流れ>



安定状態化と通常状態化の各復旧パターンの一覧を以下に示します。各復旧パターンの手順の実施内容、処理の詳細については「データ変換・通信オプション 導入の手引」を参照してください。

(i) 安定状態化復旧パターン

障害が発生しているシステムの組合せにより、以下 9 パターンから実施する復旧手順を選択します。

復旧パターン		障害発生時				安定状態			
		第一拠点	第二拠点	第三拠点		第一拠点	第二拠点	第三拠点	
				DB	災対			DB	災対
A	災対センタにバックアップ構築(DB 正常時)	フロント	×	○	停止	フロント	×	○	バック
B	災対センタにバックアップ構築(DB 障害時)	フロント	×	×	停止	フロント	×	×	バック
C	フロント-DB 間転送開始	フロント	×	○	×	フロント	×	○	×
D	バックアップ切替	×	バック	○	停止	×	フロント	○	停止
E	災対センタにフロント構築(DB 正常時)	×	×	○	停止	×	×	○	フロント
F	災対センタにフロント構築(DB 障害時)	×	×	×	停止	×	×	×	フロント
G	DB からのフロント構築	×	×	○	×	フロント	×	○	×
H	障害先切り離し	フロント	×	×	×	フロント	×	×	×
I	第三拠点切り離し	フロント	バック	×	停止	フロント	バック	×	×

<表の見方>

フロント：フロントシステムとして動作中

バック：バックアップシステムとして動作中

停止：災対センタの DIOSA/XTP を停止し、DB の OracleDB-災対センタの OracleDB 間でレプリケーションされている（DB から災対センタの OracleDB にアクセスできる）状態

○：正常動作中

×

網掛け：安定状態化により状態が変更した箇所

以下に、各復旧パターンの概略を記載します。

(A) 災対センタにバックアップ構築(DB 正常時)

バックアップシステムがシステム障害となった場合、第三拠点(災対センタ)にバックアップシステム環境を構築します。

(B) 災対センタにバックアップ構築(DB 障害時)

バックアップシステム、第三拠点(DB)がシステム障害となった場合、第三拠点(災対センタ)にバックアップシステム環境を構築します。

(C) フロント-DB 間転送開始

バックアップシステム、第三拠点(災対センタ)がシステム障害となった場合、フロントから第三拠点(DB)へ直接更新データを転送する環境を構築します。

(D) バックアップ切替

フロントシステムがシステム障害となった場合、バックアップシステムをフロントシステムに切り替えます。

(E) 災対センタにフロント構築(DB 正常時)

フロントシステム、バックアップシステムがシステム障害となった場合、第三拠点(災対センタ)にフロントシステム環境を構築します。

(F) 災対センタにフロント構築(DB 障害時)

フロントシステム、バックアップシステム、第三拠点(DB)がシステム障害となった場合、第三拠点(災対センタ)にフロントシステム環境を構築します。

(G) DB からのフロント構築

第三拠点(DB)以外の全ての論理システムがシステム障害となった場合、第三拠点(DB)で管理するデータを元にフロントシステムを構築します。

(H) 障害先切り離し

フロントシステム以外の全ての論理システムがシステム障害となった場合、障害となっている論理システムを切り離します。

(I) 第三拠点切り離し

第三拠点(DB)が障害となった場合、第三拠点(災対センタ)を切り離します。

(ii) 通常状態化（完全復旧）復旧パターン

安定状態での論理システムの状態の組合せにより以下 8 パターンから実施する復旧手順を選択します。複数の論理システムが障害となっている場合には全ての論理システムが正常動作している状態になるまで各復旧パターンの手順を実施します。

復旧パターン		安定状態				復旧後			
		第一拠点	第二拠点	第三拠点		第一拠点	第二拠点	第三拠点	
				DB	災対			DB	災対
a	災対センタにバックアップ構築 (フロント-DB 間転送時)	フロント	×	○	停止	フロント	×	○	バック
b	DB からのバックアップ構築 (災対センタ正常時)	フロント	×	○	停止	フロント	バック	○	停止
c	DB からのバックアップ構築 (災対センタ障害時)	フロント	×	○	×	フロント	バック	○	×
d	DB からのバックアップ構築 (災対がバックアップとして稼動時)	フロント	×	○	バック	フロント	バック	○	停止
e	フロントシステムからのバックアップ構築	フロント	×	×	×	フロント	バック	×	×
f	災対センタ構築	フロント	バック	○	×	フロント	バック	○	停止
g	DB 構築	フロント	バック	×	×	フロント	バック	○	×
h	計画切替	バック	フロント	○	停止	フロント	バック	○	停止

<表の見方>

フロント：フロントシステムとして動作中

バック：バックアップシステムとして動作中

停止：災対センタの DIOSA/XTP を停止し、DB の OracleDB-災対センタの OracleDB 間でレプリケーションされている（DB から災対センタの OracleDB にアクセスできる）状態

○：正常動作中

×

網掛け：安定状態化により状態が変更した箇所

以下に、各復旧パターンの概略を記載します。

(a) 災対センタにバックアップ構築(フロント-DB 間転送時)

バックアップシステムが停止していてフロントから第三拠点(DB)に更新データの転送を行っている状態の場合、第三拠点(災対センタ)にバックアップ環境を構築します。

(b) DB からのバックアップ構築(災対センタ正常時)

バックアップシステムが停止していてフロントから第三拠点(DB)に更新データの転送を行っている状態の場合、第三拠点(DB)からシステム障害拠点到データ転送してバックアップ環境を構築します。

(c) DB からのバックアップ構築(災対センタ障害時)

バックアップシステム、第三拠点(災対センタ)が停止していてフロントから第三拠点(DB)に更新データの転送を行っている状態の場合、第三拠点(DB)からシステム障害拠点到データを送送してバックアップ環境を構築します。

(d) DB からのバックアップ構築(災対センタがバックアップとして稼働時)

第三拠点(災対センタ)がバックアップシステムとして起動している場合、第三拠点(DB)からシステム障害拠点到データを送送してバックアップ環境を構築し、第三拠点(災対センタ)に代わってバックアップシステムとして使用します。

(e) フロントシステムからのバックアップ構築

フロントシステム以外の論理システムが停止している場合、フロントシステムからシステム障害拠点到データを送送してバックアップ環境を構築します。

(f) 災対センタ構築

第三拠点(災対センタ)が停止している場合、第三拠点(DB)からデータを送送して第三拠点(災対センタ)環境を構築します。

(g) DB 構築

第三拠点(DB)、第三拠点(災対センタ)が停止している場合、バックアップシステムから第三拠点(DB)にデータを送送して第三拠点(DB)環境を構築します。

(h) 計画切替

完全復旧後にフロントシステムとバックアップシステムを入れ替えたい場合、またはデータベース構成変更での計画切替を利用した TAM 再配置で、フロントシステムとバックアップシステムを入れ替える場合に実施します。

2.3.2 TAM 再配置機能

TAM 再配置機能はデータベース構成変更を行うための TAM 再配置手順と、その実現のために必要となる各機能を提供します。

(1) TAM 再配置手順

データベース構成変更を行うための TAM 再配置手順を提供します。この TAM 再配置手順に従ってデータベース構成変更をすることで、システムを停止することなく継続運転したままデータベース構成を変更することができます。また、ユーザアプリケーションは TAM 再配置実行中も、データベース構成変更を意識することなくデータアクセスを継続することができます。

(a) TAM 再配置で取り扱うデータベース構成変更

TAM 再配置で扱うデータベース構成変更は、MAP のデータ構成を変更するパターン (MAP 構成変更) と、TAM (TAM 表、TAM インスタンス) を変更するパターン (TAM メンテナンス) の大きく 2 つに分類されます。

TAM 再配置で取り扱うデータベース構成変更とその用途について以下に示します。

分類	構成変更パターン	用途
MAP 構成変更	(i) MAP の追加／削除	・データアクセスの性能問題があり、スケールアウトによる性能改善が必要となった場合 ・1つの MAP にデータ量の偏りが発生し、データ配置の変更による均一化が必要となった場合
	(ii) レプリケーショングループの追加／削除(※1)	
	(iii) OLTP ノードの追加／削除(※2) (v2.1 では未提供)	
	(iv) ハッシュ値更新	・一部のハッシュ値にデータ量の偏りが発生し、データ配置の変更による均一化が必要となった場合
	(v) ハッシュ値追加	・業務変更によりハッシュ値の算出規則が追加され、ハッシュ関数の置換が必要となった場合
TAM メンテナンス	(vi) 論理表定義変更	・業務変更により論理表のレコード項目の領域長変更やレコード項目、インデックスの追加／削除が必要になった場合
	(vii) 論理表追加／削除	・業務変更により論理表の追加／削除が必要になった場合
	(viii) 容量サイズ変更	・TAM インスタンスのメモリ不足が発生し、TAM インスタンスの容量サイズを拡張が必要となった場合

(※1) レプリケーショングループを追加すると TAM インスタンスが追加されるため、MAP を追加する場合と比べてより多くのメモリ資源が必要となります。

(※2) OLTP ノードを追加する場合はサーバの追加が必要となります。

(注意)

MAP 構成変更の TAM 再配置中に全データ削除 (TRUNCATE) を行うことはできません。

MAP 構成変更の TAM 再配置中に全データ削除を行うと、バックアップ拠点のデータストア ログデータ実行制御 (ログリーダー) が異常停止します。この場合は TAM 再配置前の環境に戻してログリーダーが TRUNCATE の更新ログを処理した後、再度 TAM 再配置を行う必要があります。

以下に TAM 再配置手順で取り扱うデータベース構成変更の概要を記載します。各データベース構成変更の処理の詳細については「データ変換・通信オプション 導入の手引」を参照してください。

(i) MAP の追加／削除

MAP を追加／削除し、各 MAP に割り当てられたハッシュ値を変更することで、データ配置が変更されます。

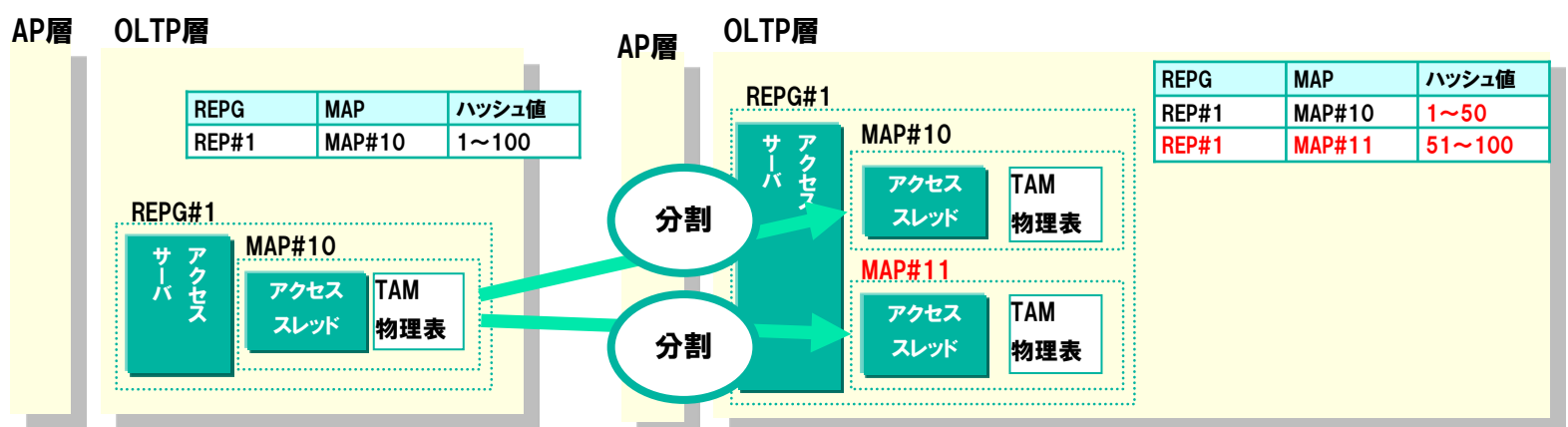
以下に MAP の追加／削除のイメージ図を記載します。

MAP追加 (分割)

追加は、MAPに割り当てているハッシュ値を2つのMAPに分割することで実現する。

追加するMAPは元のMAPと同一のレプリケーショングループ、または別のレプリケーショングループのMAPに分割する。

MAPの追加によりレプリケーショングループも追加する場合には、「レプリケーショングループ追加／削除」を実施する。

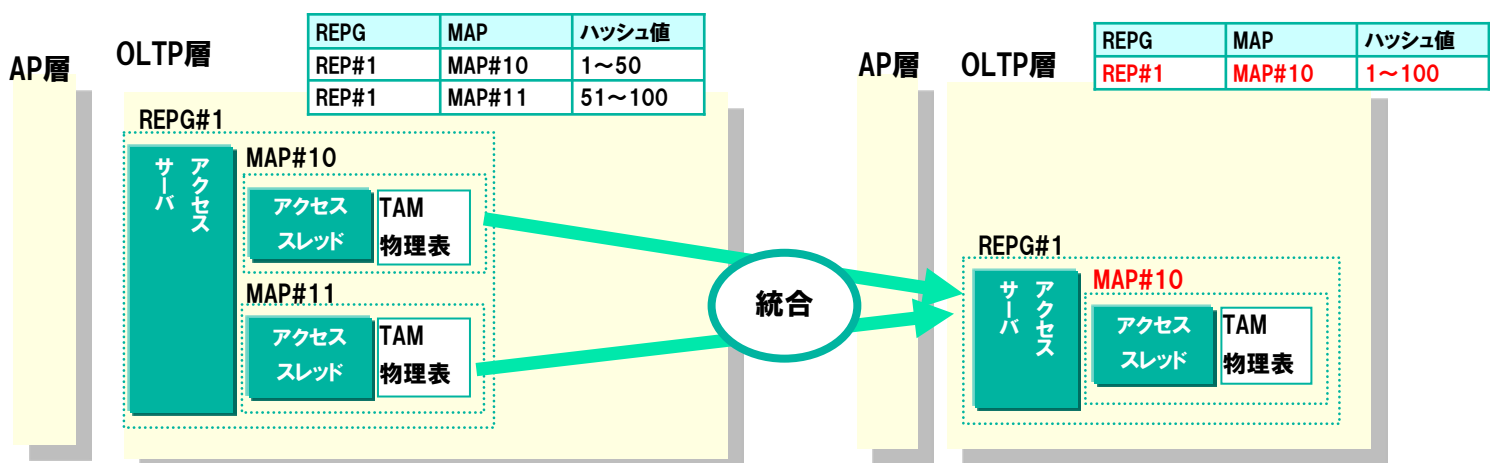


MAP削除 (統合)

削除は、複数のMAPに割り当てているハッシュ値を1つのMAPに統合することで実現する。

統合先のMAPは削除MAPと同一のレプリケーショングループ、または別のレプリケーショングループのMAPに統合する。

MAPの削除によりレプリケーショングループも削除する場合には、「レプリケーショングループ追加／削除」を実施する。



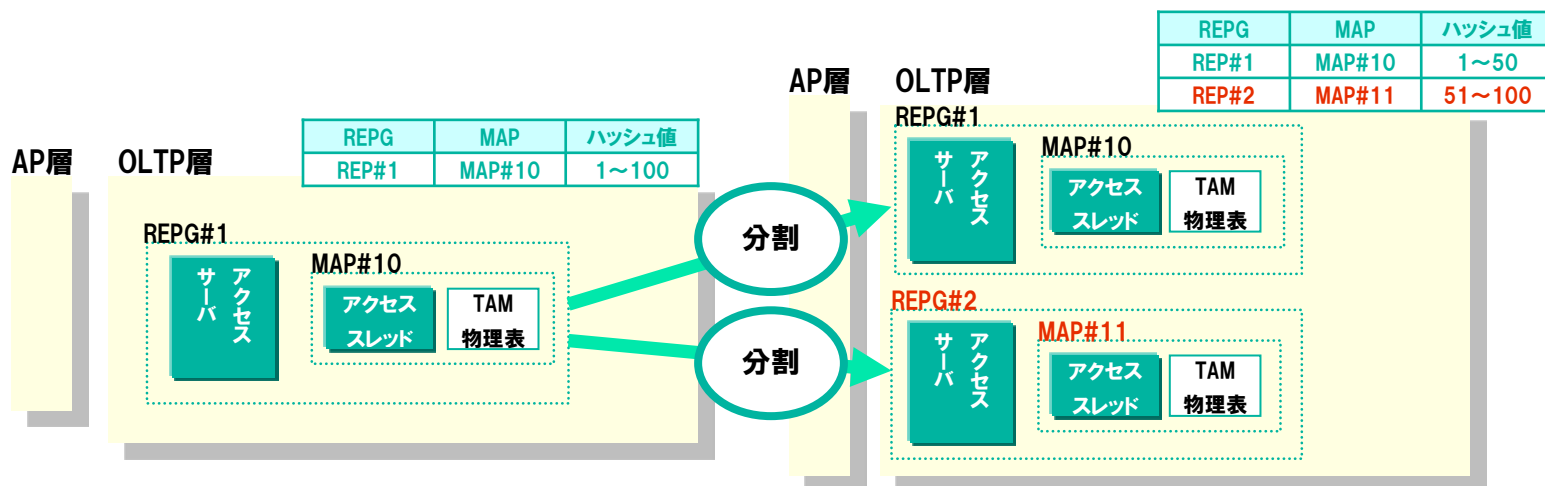
(ii) レプリケーショングループの追加／削除

レプリケーショングループを追加／削除し、各 MAP に割り当てられたハッシュ値を変更することで、データ配置が変更されます。

以下にレプリケーショングループの追加／削除のイメージ図を記載します。

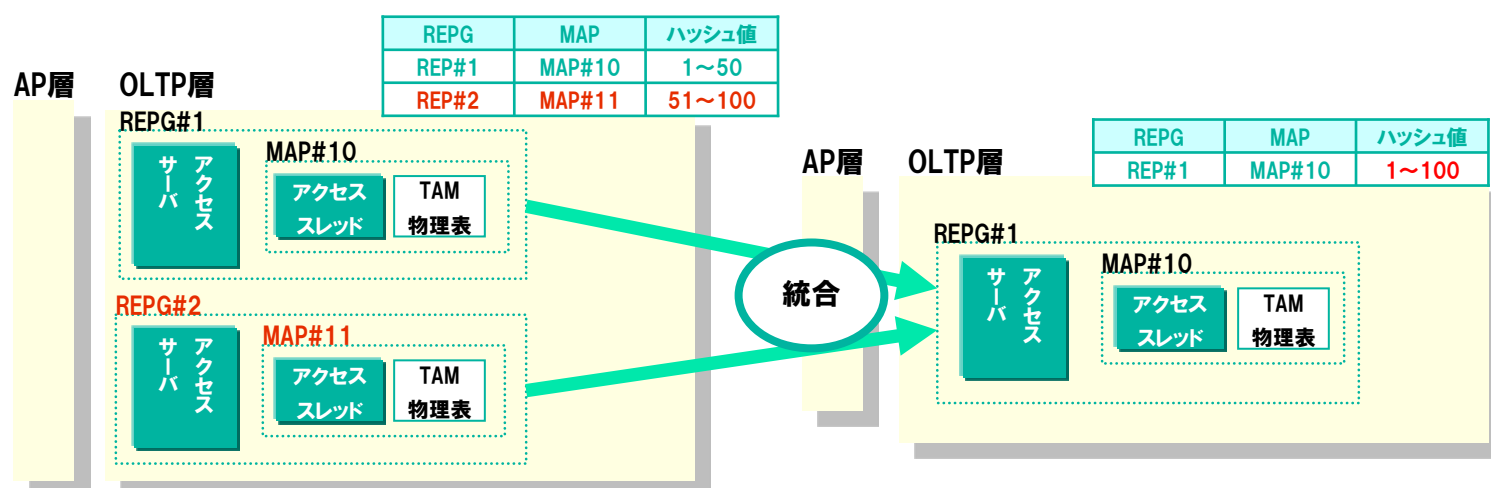
レプリケーショングループ (REPG) 追加 (分割)

追加は、MAPに割り当てているハッシュ値を、2つのMAPに分割することで実現する。



レプリケーショングループ削除 (統合)

削除は、2つのMAPに割り当てているハッシュ値を、1つのMAPに統合することで実現する。



(iii) OLTP ノードの追加／削除

本機能は V2.1 では未提供です。

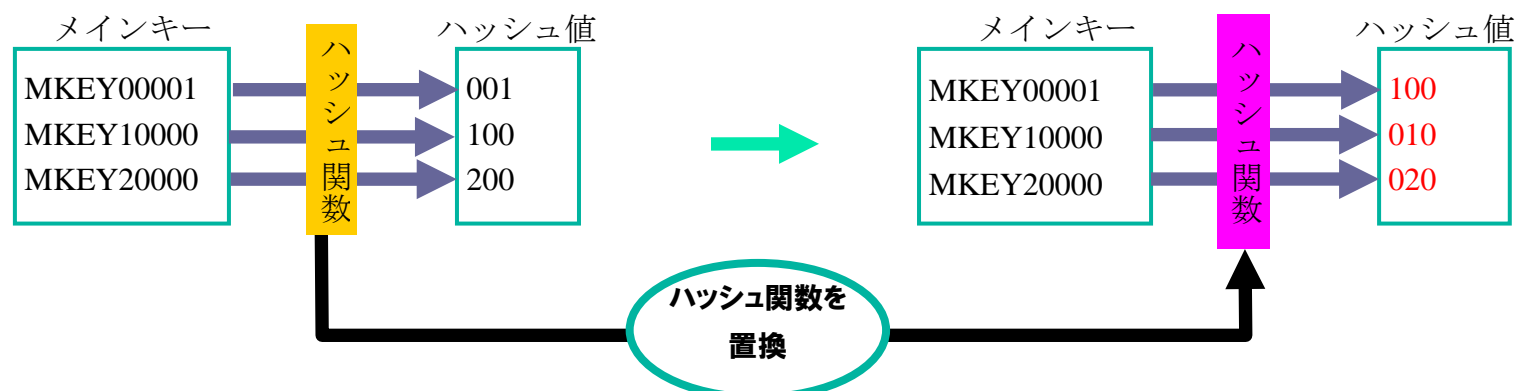
(iv) ハッシュ値更新

ハッシュ関数を含むライブラリの置換を行い、OracleDB のデータのハッシュ値を更新することで、データ配置が変更されます。

以下にハッシュ値更新のイメージ図を記載します。

ハッシュ値更新

メインキーとハッシュ値の対応を変更する場合、ハッシュ関数を置換する。(データ配置も変更される)



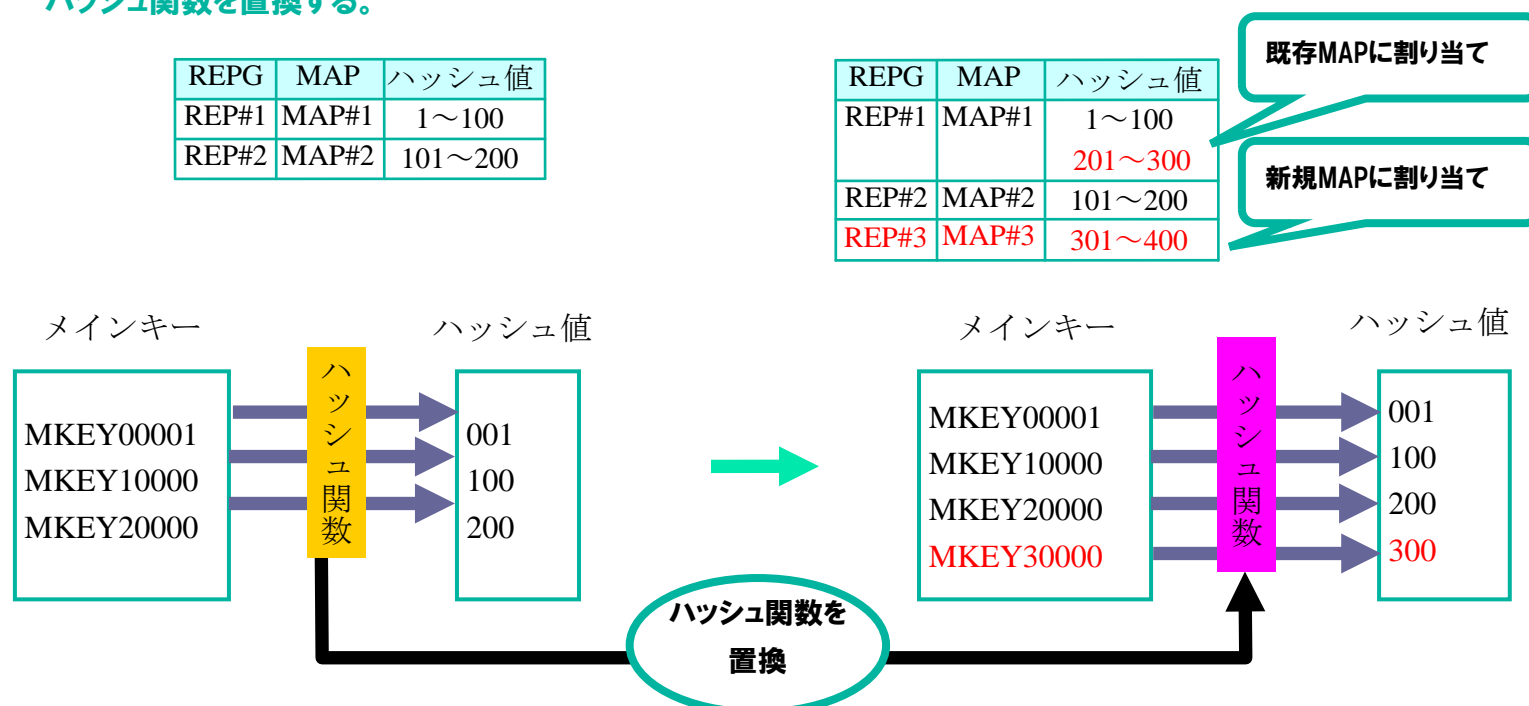
(v) ハッシュ値追加

既存のメインキーとハッシュ値の対応は変更せず、新しいメインキーとハッシュ値の対応を追加します。
追加したハッシュ値は、既存の MAP に割り当てることも、MAP やレプリケーショングループを追加し、新しい MAP に割り当てることもできます。

以下にハッシュ値追加のイメージ図を記載します。

ハッシュ値追加

既存のメインキーとハッシュ値の対応は変更せず、新しいメインキーとハッシュ値の対応を追加する場合、ハッシュ関数を置換する。



(vi) 論理表定義変更

TAM と OracleDB それぞれの定義変更をすることで、レコード項目の追加／削除やレコード項目の領域長変更、インデックス追加を行います。レコード項目の追加と削除を同時に行う場合、ユーザアプリケーションは DB アクセス制御の環境定義 DACENV 節 TABLESET 項のリビジョン番号を意識したデータアクセスを行う必要があります。

論理表定義変更は既存データへの影響の有無(※1)により、データベース構成変更の手順が異なります。

(※1) 論理表定義の変更により TAM 上の既存レコード項目のオフセット、及び TAM 上のレコード長の定義に変更が無い場合、TAM の定義変更は不要となり、既存データに影響はありません。

(vii) 論理表追加／削除

TAM と OracleDB それぞれの定義を追加／削除をすることで、論理表の追加／削除を行います。

(viii) 容量サイズ変更

TAM の定義変更をすることで、容量サイズの変更を行います。

容量サイズ変更では、TAM 物理表の容量、及び TAM インスタンスの容量を変更することができます。

(2) 運用情報照会機能

TAM 再配置手順の中で必要となる運用情報を照会することができます。

想定される使用ケースは下記の通りです。

- ・ TAM 再配置手順の中で運用情報を照会し、TAM 再配置の進捗状況や処理対象の MAPID を取得する。

(3) 運用情報更新機能

TAM 再配置手順の中で運用情報を更新します。運用情報を利用して、TAM 再配置の進捗状況、全データ削除 (TRUNCATE) の実行可否を管理します。

想定される使用ケースは下記の通りです。

- ・ TAM 再配置手順の中で運用情報を更新する。

(4) ハッシュ値更新機能

OracleDB 上のユーザカレントデータ群とデータ変換・通信オプション制御情報群の各レコードが持つハッシュ値を更新することができます。

ハッシュ値更新機能では更新対象とするレコードを選択することができます。

- ・ 全データ更新指定 (既定値)

OracleDB の論理表に存在する全てのレコードを更新対象とする。

- ・ メインキー指定

DB アクセス制御機能のユーザデータ状態管理 API を利用してメインキー登録を行ったレコードのみを
更新対象とする。

メモリキャッシュ機能が動作している環境では、メモリキャッシュ機能が管理するハッシュ関数を使用して更新を行います。

メモリキャッシュ機能が動作していない環境で使用する場合には、ハッシュ値更新コマンドがハッシュ関数を使用できるようアプリケーション動的置換機能の設定 (環境変数 DIOSA_LIBNAME または環境定義 APLIB 節) を行い、コマンドパラメータでハッシュ関数名を指定する必要があります。

コマンドパラメータで MAP を指定しなかった場合、全ての論理表を処理対象とします。

コマンドパラメータで MAP を指定した場合は、指定した MAP に属する論理表を処理対象とします。処理対象となった論理表は全レコードのハッシュ値が更新されます。これは、OracleDB 上では MAP の概念はなく、論理表内の一部のレコードのみを更新した場合に不整合が発生するためです。

(注意)

OracleDB 上の表がハッシュ値をキーとしてパーティション分割されており、かつパーティション行の移動が使用禁止 (ROW_MOVEMENT パラメータが DISABLED) されている場合、コマンド実行前に使用可能 (ENABLED) に変更し、コマンド実行後に使用禁止に戻す必要があります。

想定される使用ケースは下記の通りです。

- ・ TAM 再配置手順の中でハッシュ関数を置換した場合にメインキーとハッシュ値の対応が変わるため、OracleDB に保存されているハッシュ値を置換後のハッシュ関数で生成される値に更新する。

2.3.3 セーブロード機能

セーブロード機能は、OracleDB のデータから TAM のデータを復旧するためのロード機能を提供します。また、オンライン中に TAM のデータから OracleDB のデータを復旧するセーブ機能も提供します。

(1) TAM セーブ機能

MAP 単位に実行し、TAM のレコードを読み込み OracleDB へセーブすることができます。セーブ対象のデータは、ユーザカレントデータ群とデータ変換・通信オプション制御情報群です。DB アクセス制御機能の環境定義（DACENV 節）でレプリケーションを行わないと定義した表についても、OracleDB に表があれば処理を行います。

想定される使用ケースは下記の通りです。

- ・ オンライン中 DB リカバリ手順の中で OracleDB 障害発生時に更新ログがオーバーフローした場合、及び OracleDB のデータが復旧できなかった場合、オンライン中に TAM のデータを読み込み、OracleDB のデータを復旧する。

(注意)

TAM 上にレコードがなく、OracleDB 上のみにレコードが存在する場合でも、TAM セーブ機能が OracleDB からレコードを削除することはありません。そのため、TAM 上でのレコード削除の更新ログが失われた場合、TAM で削除されたレコードが OracleDB に残ることがあります。

(2) TAM ロード機能

MAP 単位に実行し、OracleDB から TAM へ必要なデータをロードすることができます。業務処理との排他を意識せずにデータをロードするため、オフライン中のみ実行できます。ロード対象のデータは、ユーザカレントデータ群とデータ変換・通信オプション制御情報群です。

DB アクセス制御機能の環境定義（DACENV 節）でレプリケーションを行わないと定義した表についても、OracleDB に表があれば処理を行います。

TAM ロード機能ではロード対象とするレコードを選択することができます。

- ・ 全データロード指定（既定値）

OracleDB の論理表に存在する全てのレコードをロード対象とする。

- ・ メインキー指定

DB アクセス制御機能のユーザデータ状態管理 API を利用してメインキー登録を行ったレコードのみをロード対象とする。

想定される使用ケースは下記の通りです。

- ・ DIOSA/XTP 起動時、OracleDB 上のデータから TAM へデータを反映する。
- ・ TAM 再配置実行時、OracleDB 上のデータから TAM へデータを反映する。

(3) ファイル出力 TAM セーブ機能

MAP 単位に実行し、TAM のデータを読み込み TAM 出力ファイルへセーブすることができます。

TAM 出力ファイルは、コマンドオプションで指定したディレクトリに出力されます。

出力される TAM 出力ファイル名の形式を以下に示します。

<TAM 出力ファイル名形式>

diatc_<論理システム名>_slu_<論理表名>_<MAPID>.dat
--

(説明)

- ・ 論理システム名：アプリケーション実行制御の環境定義 DIOSAMAP 節に定義した論理システム名を付加する。（最大 16 バイト）
- ・ 論理表名：メモリキャッシュ機能の環境定義 IMTABLECONF 節に定義した論理表名を付加する。（最大 255 バイト）
- ・ MAPID：メモリキャッシュ機能の環境定義 IMTABLECONF 節に定義した MAPID を 10 桁固定で付加する。（10 バイト固定）

(注意)

論理表名の最大は 256 バイトまで可能ですが、TAM 出力ファイル名がファイルシステムの最大ファイル名長を越えないように論理表名を定義する必要があります。最大ファイル名長を超えた場合、コマンドは異常終了します。

TAM セーブ機能とは以下の点で異なります。

- ・ 出力先は TAM 出力ファイルとする。
- ・ セーブ対象のデータは、ユーザカレントデータ群とし、データ変換・通信オプション制御情報群は対象外とする。

想定される使用ケースは下記の通りです。

- ・ データベース構成が TAM のみのシステムにおいての TAM 再配置実行時、構成変更前に TAM データのバックアップを TAM 出力ファイルに出力する。

(4) ファイル入力 TAM ロード機能

MAP 単位に実行し、TAM 出力ファイルから TAM へ必要なデータをロードすることができます。

TAM ロード機能とは以下の点で異なります。

- ・ 入力元は TAM 出力ファイルとする。
- ・ ロード対象のデータは、ユーザカレントデータ群とし、データ変換・通信オプション制御情報群は対象外とする。
- ・ 利用者が作成する出口関数を使用することで、データベース構成変更を可能とする。

想定される使用ケースは下記の通りです。

- ・ データベース構成が TAM のみのシステムにおいての TAM 再配置実行時、構成変更後に TAM 出力ファイルから TAM へデータを反映する。

(5) 実行状況照会機能

TAM 再配置機能、セーブロード機能の以下のコマンドの進捗状況、実行状況を照会することができます。

- ・ TAM 再配置機能 ハッシュ値更新コマンド

- ・ セーブロード機能 TAM セーブコマンド
 TAM ロードコマンド
 ファイル出力 TAM セーブコマンド
 ファイル入力 TAM ロードコマンド

想定される使用ケースは下記の通りです。

- ・ 各コマンド実行（各機能の想定される使用ケース参照）後、実行状況を確認する。

(6) **実行状況確認機能**

TAM 再配置機能、セーブロード機能のコマンドがアボートした場合に、登録された進捗状況、実行状況を削除するため、実行状況が「実行中」のプロセスが生存するか確認し、プロセスが存在しない場合は進捗状況、実行状況を削除することができます。対象となるコマンドは実行状況照会機能と同じです。

また、プロセスが存在している場合は、実行中のプロセスが存在する戻り値を返却します。

想定される使用ケースは下記の通りです。

- ・ TAM 再配置機能、セーブロード機能のコマンドがアボートした場合に使用する。

第3章 アプリケーション開発

3.1 セーブロードプログラミング

3.1.1 構成変更情報取得出口

Oracle を使用しない環境での TAM 再配置に対応するため、構成変更情報取得出口が提供されています。構成変更情報取得出口はファイル入力 TAM ロードコマンドから呼び出され、セーブしたデータに対応するハッシュ値、及びロードするデータの編集を利用者が行うための利用者出口です。

(1) 使用方法

1. 構成変更情報取得出口を含むライブラリを作成します。
2. 作成したライブラリを実行権限のあるパスに設置します。
3. AP 動的置換機能の環境変数 AP 動的置換ライブラリ定義(DIOSA_LIBNAME、DIOSA_LIBNAME_NODR)、または環境定義 APLIB 節に、設置したライブラリの定義を追加します。

(2) 呼出契機

ファイル入力 TAM ロードコマンドでコマンドパラメータに出口関数名を指定した場合、レコードの読み込みごとに呼び出されます。

(3) プログラムインタフェース

ファイル入力 TAM ロードコマンドから呼び出され、引数として読み込みを行うレコードを含む構成変更情報構造体を与えます。利用者は読み込みを行うレコードからハッシュ値の計算とレコードの編集を行い、結果を返却します。

```
int プログラム名(t_diatc_fltamdata *)

○t_diatc_fltamdata 構造体
char          *LTableName      論理表名(入力型)
char          *UserData        ユーザデータレコード(入力型)
size_t        UserDataSize     ユーザデータレコードサイズ(入出力型)
ModFlg が DIOSA_ON の場合は変更後のレコード長
unsigned int   HashValue       ユーザデータレコードのハッシュ値(出力型)
int           ModFlg           ユーザデータレコードの変更フラグ(出力型)
char          *ModBuff         変更後のユーザデータレコード格納領域(出力型)
ModFlg が DIOSA_ON の場合のみ有効
size_t        ModBuffSize      ユーザデータレコード格納領域の領域長(入出力型)
```

(4) プログラム作成方法

1. ロード先の論理表名ごとにメインキーの位置を特定し、**diosagethash()**によりハッシュ値を計算します。
2. ロード先の論理表名で指定された論理表のレコード項目に変更がある場合、ロードするユーザデータレコードを変更後のレコード定義に従って変更し、結果を変更後のユーザデータレコード格納領域に格納します。変更した場合にはユーザデータレコードの変更フラグに DIOSA_ON を設定し、ロードするユーザデータレコードサイズには変更後のユーザデータレコード長を設定します。

(5) **注意事項**

- ・ 変更後のユーザデータレコード格納領域は利用者側で用意する必要があります。使用するメモリは領域種別一括解放対象外スレッド内メモリを指定した **diosamalloc()** で確保します。
- ・ 変更後のユーザデータレコード格納領域の解放は呼び出し元で行います。
- ・ マルチスレッドのためスレッドセーフに作成する必要があります。

(6) **コーディング例**

以下に、論理表のレコード定義を変更（TABLE01 に項目の追加）する場合の構成変更情報取得出口のコーディング例を記載します。

【注意】

ユーザデータレコードの先頭にはデータ変換・通信オプション用の制御情報 40 バイトが含まれます。構成変更情報取得出口を作成する際には上記の領域を意識して作成する必要があります。

```

int USER_EXIT ( t_diatc_fltamdata *FlTamData )
{
    int          Ret = DIOSA_DONE; /* 関数戻り値 */

    if( 0 != strcmp( "TABLE01", FlTamData->LTableName ) ) {
        /* 論理表:TABLE01 以外の場合 */
        /* ハッシュ値計算 */
        Ret = diosagethash( FlTamData->UserData, &(FlTamData->HashValue) );
        if ( DIOSA_DONE != Ret ) {
            /* エラー処理 */
        }
    } else {
        /* 論理表:TABLE01 の場合 */
        /* ハッシュ値計算 */
        Ret = diosagethash( FlTamData->UserData, &(FlTamData->HashValue) );
        if ( DIOSA_DONE != Ret ) {
            /* エラー処理 */
        }
        FlTamData->ModFlg = DIOSA_ON; /* ユーザーデータレコードの変更フラグに DIOSA_ON を設定 */
        if ( NULL == FlTamData->ModBuff ) {
            /* 領域確保 */
            FlTamData->ModBuffSize = FlTamData->UserDataSize + 5;
            Ret = diosamalloc( "FLTAMLOAD", NULL, DIOSA_THRNOALL, (int)FlTamData->ModBuffSize,
                               (void*)&(FlTamData->ModBuff) );

            if ( DIOSA_DONE != Ret ) {
                /* エラー処理 */
            }
        } else if( FlTamData->ModBuffSize < FlTamData->UserDataSize + 5 ) {
            FlTamData->ModBuffSize = FlTamData->UserDataSize + 5;
            /* 領域拡張 */
            Ret = diosarealloc( "FLTAMLOAD", NULL, DIOSA_THRNOALL, (int)FlTamData->ModBuffSize,
                               (void*)&(FlTamData->ModBuff) );

            if ( DIOSA_DONE != Ret ) {
                /* エラー処理 */
            }
        }
        /* 領域初期化 */
        memset( FlTamData->ModBuff, 0, FlTamData->ModBuffSize );
        /* 変更後のユーザーデータレコードを編集 */
        memcpy( &( FlTamData->ModBuff[ 0] ), &( FlTamData->UserData[ 0] ), 40 );
        memcpy( &( FlTamData->ModBuff[40] ), "    ", 5 );
        memcpy( &( FlTamData->ModBuff[45] ), &( FlTamData->UserData[40] ),
                FlTamData->UserDataSize - 40 );
    }

    return DIOSA_DONE;
}

```

ユーザーデータレコードの先頭
32 バイトにメインキーが格納
されている

領域長には、ユーザーデータレ
コードサイズに追加項目長(例で
は5バイト)を加算する

追加項目(5 バイト)に空白を設
定する

3.2 ユーザアプリケーションプログラム

本章では、ユーザアプリケーションの開発方法について説明します。

3.2.1 プログラムの構造

ユーザアプリケーションを作るためには、必要な初期化処理や終了処理を適切な順番に従って実行する必要があります。シングルスレッド構造の場合とマルチスレッド構造の場合について、それぞれのプログラム構成を説明します。

(1) シングルスレッド構造

データ変換・通信オプションのCインターフェースを使ったユーザアプリケーションを作るためには、D I O S A / X T P とデータ変換・通信オプションの初期化処理や終了処理を必要とします。

シングルスレッド構造のプログラムでは、以下の順番で各初期化処理、終了処理を行います。

なお、例外発生等でトランザクション終了をおこなわずにプロセスを終了する場合、プロセス終了処理もおこなわないようにしてください。(利用者のプロセス終了処理は除きます)

①プロセス初期化

```
int diosaprcinit(NULL)
```

```
int diosatrnrinit(NULL)
```

```
利用者のプロセス初期化処理
```

```
int diosatrnrterm(NULL)
```

②トランザクション初期化

```
int diosatrnrinit(NULL)
```

```
int diatctrnrinit(void)
```

```
利用者のトランザクション初期化処理
```

③ユーザプログラム処理

```
データ変換・通信オプションの提供A P I を使った利用者処理 (3.5 節参照)
```

④トランザクション終了

```
利用者のトランザクション終了処理
```

```
int diatctrnrterm(void)
```

```
int diosatrnrterm(NULL)
```

⑤プロセス終了処理

```
int diosatrnrinit(NULL)
```

```
利用者のプロセス終了処理
```

```
int diosatrnrterm(NULL)
```

```
int diosaprcrterm(NULL)
```

※トランザクション初期化～トランザクション終了(上記の②～④)の一連の処理はループ可能ですが、プロセス終了後、再度プロセス初期化処理を呼び出して処理を継続することはできません。

(2) マルチスレッド構造

シングルスレッド構造のプログラムと同様に、データ変換・通信オプションのCインターフェースを使っ

たユーザアプリケーションを作るためには、D I O S A / X T P とデータ変換・通信オプションの初期化処理や終了処理を必要とします。

マルチスレッド構造のプログラムについて、メインスレッドとメインスレッド以外のスレッドのプログラム構成を説明します。

メインスレッド（プロセス初期化を行うスレッド）

シングルスレッド構造と同様です。シングルスレッド構造の説明をご参照ください。

なお、子スレッドを生成する場合は、ユーザプログラム処理（トランザクション初期化とトランザクション終了の間）のタイミングで生成してください。

メインスレッド以外のスレッド

以下の順番で各初期化処理、終了処理を行います。なお、子スレッドを生成する場合は、メインスレッドと同様に、ユーザプログラム処理のタイミングで生成してください。

①スレッド初期化

```
int diosathrinit(NULL)
int diosatrnrinit(NULL)
int diatcthrinit(int ImOpenMode )
```

利用者のスレッド初期化処理

```
int diosatrnrterm(NULL)
```

②トランザクション初期化

```
int diosatrnrinit(NULL)
int diatctrnrinit(void)
```

利用者のトランザクション初期化処理

③ユーザプログラム処理

データ変換・通信オプションの提供APIを使った利用者処理（3.5節参照）

④トランザクション終了

利用者のトランザクション終了処理

```
int diatctrnrterm(void)
int diosatrnrterm(NULL)
```

⑤スレッド終了処理

```
int diosatrnrinit(NULL)
```

利用者のスレッド終了処理

```
int diatcthrterm(void)
int diosatrnrterm(NULL)
int diosathrterm(NULL)
```

※トランザクション初期化～トランザクション終了（上記の②～④）の一連の処理はループ可能ですが、プロセス終了後、再度プロセス初期化処理を呼び出して処理を継続することはできません。

3.3 DBアクセスプログラミング

本章では、データ変換・通信オプションの API を使って DB アクセスを行うアプリケーションの開発方法について説明します。

(1) 共通事項

全てのデータアクセスの種類に共通する事項を説明します。

(a) データベースアクセス情報構造体(t_diatc_dbaccinfo)

DB アクセス API を利用したデータアクセスに必要な情報を設定するための構造体です。主にアクセス対象のメインキーまたは MAPID を指定します。詳細は API リファレンスを参照してください。なお、MAPID は DIOSA/XTP メモリキャッシュの API を利用して取得します。

(b) レコード情報構造体(t_diatc_recinfo)

DB アクセス API とアプリケーションの間でレコード情報を受け渡しするための構造体です。主にレコードの先頭ポインタとレコードのサイズを格納します。詳細は API リファレンスを参照してください。なお、レコード情報構造体は DIOSA/XTP メモリキャッシュの t_diosa_recinfo 構造体と同じ構造です。

(c) テーブル ID

DB アクセス API では、アクセス対象のテーブルをテーブル ID で指定します。diatcdbgettblid 関数でテーブル名からテーブル ID を取得します。diatcdbgettblid 関数で取得したテーブル ID は同一トランザクション内で有効なので、同一トランザクション内では DB アクセス API を呼び出す度にテーブル ID を取得しなおす必要はありません。

(d) データ変換・通信オプションの制御用領域

DB アクセス API でアクセスするテーブルは、レコードの先頭 40 バイトがデータ変換・通信オプションの制御用領域となっています。そのため、ユーザデータはレコードの先頭から 41 バイト目以降に配置しなければなりません。

また、レコード情報構造体(t_diatc_recinfo)で指定するレコードのサイズ(RecordSize)にはユーザデータのサイズに制御用領域のサイズ(40 バイト)を加えた値を指定します。なお、制御用領域のサイズは、ヘッダーdiatc.hでDIATC_RECORD_HEADER_SIZEという名前で定義されています。

(e) レコードのバイトオーダーについて

DB アクセス API を利用する場合、レコード作成、および参照はホストバイトオーダーで行ってください。TAM や DIOSA/XTP メモリキャッシュの仕様では、レコードイメージ中のキー値はネットワークバイトオーダーで作成する必要がありますが、DB アクセス API が必要に応じて変換を行います。

(f) 更新系処理について

DB アクセス API を利用して更新系処理を行う場合、事前に diosatxstart 関数を DIOSA_DB_IM 指定で呼び出す必要があります。

(2) レコード読込（1レコード）

プライマリキーの完全一致検索を行うプログラムの処理手順を説明します。

1. テーブル ID を取得する

diatcdbgettblid 関数を呼び出してテーブル名からテーブル ID を取得します。

2. 検索条件構造体を生成する

diatcdbcondsetkey 関数を呼び出して検索条件構造体を構築します。この関数のパラメータに検索キーの値と長さを指定します。

3. データベースアクセス情報構造体を設定する

t_diatc_dbaccinfo 構造体の各メンバーの値を設定します。

4. レコード読込関数を呼び出す

diatcdbread1 関数を呼び出してインメモリサーバにアクセスします。この関数のパラメータに上記のデータベースアクセス情報構造体、テーブル ID、検索条件構造体を指定します。また、レコード情報構造体、レコード格納用バッファのポインタ、バッファの大きさ、排他種類を指定します。

5. 検索結果を取得する

レコード情報構造体に格納された情報によって、検索結果のレコードにアクセスします。

注意

- 排他ありでレコードを読み込む場合、データベースアクセス情報構造体に指定するメインキーは、読込対象のレコードを追加した時に指定したメインキーと同じものを指定しなければなりません。

(3) レコード読込（複数レコード）

プライマリキーやセカンダリキーの検索を行うプログラムの処理手順を説明します。

1. テーブル ID を取得する

diatcdbgettblid 関数を呼び出してテーブル名からテーブル ID を取得します。

2. 検索条件構造体を生成する

下記のように、検索の種類に対応した検索条件設定関数を呼び出して検索条件構造体を生成します。

<完全一致の場合>

diatcdbcondsetkey 関数を呼び出します。diatcdbcondsetkey 関数のパラメータには、環境定義上のキーの長さと同じ長さの検索キーを指定します。

<前方一致の場合>

diatcdbcondsetkey 関数を呼び出します。diatcdbcondsetkey 関数のパラメータには、環境定義上のキーの長さよりも短い長さの検索キーを指定します。

<範囲指定の場合>

diatcdbcondsetrange 関数を呼び出します。

<全件検索の場合>

検索条件構造体の構築は不要です。その代わりに下記の diatcdbctxopen 関数のパラメータ IndexName と Cond に NULL を指定します。

3. データベースアクセス情報構造体を設定する

t_diatc_dbaccinfo 構造体の各メンバーの値を設定します。

4. 検索コンテキストを生成する

diatcdbctxopen 関数を呼び出して検索コンテキストを生成します。この関数のパラメータに上記のデータベースアクセス情報構造体、テーブル ID、検索条件構造体を指定します。diatcdbctxopen 関数が正常終了すると、パラメータ CtxId に検索コンテキストの識別子が設定されます。CtxId は後続の関数で指定するので検索終了まで保持してください。

5. レコード読込関数を呼び出す。

diatcdbread 関数を呼び出してインメモリサーバにアクセスします。この関数のパラメータに上記のデータベースアクセス情報構造体、検索コンテキストの識別子を指定します。また、レコード情報構造体、取得するレコード数、レコード格納用バッファのポインタ、バッファの大きさを指定します。

複数レコードを取得する時はレコード情報構造体を配列にして、配列の要素数をパラメータ RecInfoNum に指定します。

diatcdbread 関数の戻り値が DIOSA_NOENT (20) の場合は検索終了なので下記の 7 に移ります。

6. 検索結果を取得する。

diatcdbread 関数の呼び出し後、実際に取得したレコード数がパラメータ FetchedNum に格納されています。レコード情報構造体に格納された情報によって、検索結果のレコードにアクセスします。

検索が終了するまで 5 と 6 を繰り返します。

7. 検索コンテキストを破棄する

diatcdbctxclose 関数を呼び出して検索コンテキストを破棄します。この関数のパラメータに検索コンテキストの識別子を指定します。

(4) **レコード追加**

レコードを一件追加するプログラムの処理手順を説明します。

1. テーブル ID を取得する

diatcdbgettblid 関数を呼び出してテーブル名からテーブル ID を取得します。

2. データベースアクセス情報構造体を設定する

t_diatc_dbaccinfo 構造体の各メンバーの値を設定します。

3. レコード情報構造体を設定する

追加したいレコードイメージの先頭ポインタとレコードの長さを t_diatc_recinfo 構造体に設定します。

4. レコード追加関数を呼び出す

diatcdbwrite 関数を呼び出してインメモリサーバにアクセスします。この関数のパラメータに上記のデータベースアクセス情報構造体、テーブル ID、レコード情報構造体を指定します。

注意

- 一つの論理表内でプライマリキーは一意である必要があります。これを守らないと、同じプライマリキーのレコードがアクセス先の MAP とは別の MAP に存在しても TAM に追加できてしまうので diatcdbwrite 関数は正常終了します。しかし、TAM-Oracle データ同期が失敗するので注意が必要です。
- 環境定義 DACENV 節において属性が CHAR のプライマリキー項目には空文字列を格納できません。これを守らないと、diatcdbwrite 関数は正常終了して TAM にレコードが追加されますが、TAM-Oracle データ同期が失敗するので注意が必要です。

(5) レコード更新／削除

レコードを更新または削除するプログラムの処理手順を説明します。

1. 対象のレコードを排他ありで読み込む

上記(2)の手順でレコードを読み込みます。この時、排他ありで読み込むために diatcdbread1 関数のパラメータ Lock には DIOSA_LOCK_WAIT または DIOSA_LOCK_NOWAIT を指定します。

2. データベースアクセス情報構造体を設定する

t_diatc_dbaccinfo 構造体の各メンバーの値を設定します。

3. レコード更新関数またはレコード削除関数を呼び出す

diatcdbrewrite 関数または diatcdbdelete 関数を呼び出してインメモリサーバにアクセスします。この関数のパラメータに上記のデータベースアクセス情報構造体、テーブル ID、レコード情報構造体を指定します。上記1の中で取得したレコード情報構造体をそのまま利用すると DIOSA/XTP メモリキャッシュの処理性能面で有利なので強く推奨します。

注意

- データベースアクセス情報構造体に指定するメインキーは、更新または削除対象のレコードを追加した時に指定したメインキーと同じものを指定しなければなりません。

(6) レコード全件削除

テーブルのレコードを全件削除するプログラムの処理手順を説明します。

1. トランザクション開始関数を呼び出す

レコード全件削除関数は、同一トランザクションで他の更新系関数との併用ができません。

2. テーブル ID を取得する

diatcdbgettblid 関数を呼び出してテーブル名からテーブル ID を取得します。

3. データベースアクセス情報構造体を設定する

t_diatc_dbaccinfo 構造体の各メンバーの値を設定します。

4. レコード全件削除関数を呼び出す

diatcdbtruncate 関数を呼び出してインメモリサーバにアクセスします。この関数のパラメータに上記のデータベースアクセス情報構造体とアクセス対象のテーブル ID を指定します。

5. コミット関数を呼び出す

diatcdbtruncate 関数が成功したら、diosacommit 関数を呼び出してコミットします。次の項のコミット／ロールバックの説明も合わせて参照してください。

一度に複数のテーブルや複数の MAP に対してレコード全件削除を行うプログラムは、上記のレコード全件削除関数を複数回呼び出してください。その際、同じ MAP の複数のテーブルをそれぞれ全件削除したい場合は、上記 2～4 をテーブル数分繰り返し実行します。また、複数の MAP に対して全件削除を行う場合は、上記 1～5 を MAP 数分繰り返し実行します。

(7) コミット／ロールバック

diosatxstart 関数の呼び出し後におこなった上記 (4) や (5) によるレコード追加／更新／削除要求を diosacommit 関数でコミットします。また、diosarollback 関数でロールバックします。

また、上記 (6) によるレコード全件削除関数の実行後も diosacommit 関数を呼び出します。

注意

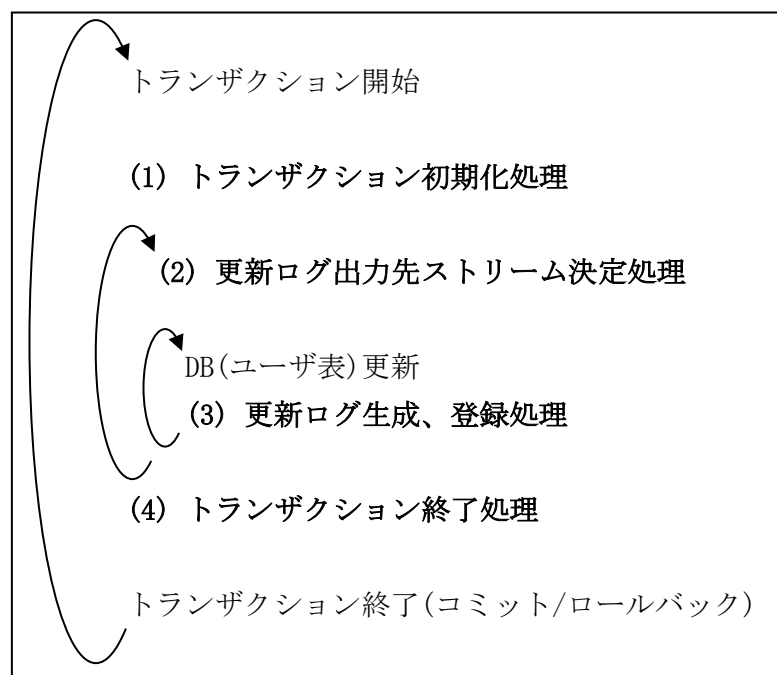
- C0 制御上で動作するアプリケーションはコミットやロールバックを行う必要はありません。これは、トランザクションのコミットやロールバックは C0 制御が行うためです。
- C0 制御上で動作するアプリケーションが実行中に明示的にコミットやロールバックを行う場合は、diosacommit 関数や diosarollback 関数を呼び出します。

3.4 Java アプリケーション

3.4.1 センタ間データ同期制御用更新ログ出力

Java アプリケーションから更新ログを生成する場合、Oracle 更新の内容を更新ログデータに変換後、更新ログ登録処理を呼び出します。

呼び出しシーケンスは以下のイメージです((1)～(4)が呼び出す必要のあるメソッド)。



(1) トランザクション初期化处理

更新ログ出力クラスのインスタンスを生成し、トランザクション初期化处理をおこないます。

[WebOTX 環境の場合]

```
DiatcDataReplicationFactory replicationFactory =  
    (DiatcDataReplicationFactory) ctx.getBean("replicationFactory");  
DiatcDataReplication rep = replicationFactory.createInstance();  
rep.tranInit();
```

※ctx は、Springfw の ApplicationContext クラスオブジェクト

[WebOTX 環境以外の場合]

```
DiatcDataReplication rep = new DiatcDataReplicationFactory().createInstance();  
rep.tranInit(con);
```

※con は、使用者側で取得した DB コネクション

(2) 更新ログ出力先ストリーム決定処理

DB 更新をおこなう前に、更新対象に対応するストリームの名前を通知する必要があります。

通知は DB 更新をおこなう場合、トランザクションごとに必ず実施する必要がありますが、トランザクション内で同一ストリームに対応する DB 更新を複数おこなう場合、最初に通知をおこなえばストリームが変

更になるか、別トランザクションになるまでは通知不要です。

ストリーム分割の設計については、導入の手引き「2.2 環境設計」を参照してください。

```
DiatcDataReplicationFactory.getInstance().setStream(ストリーム名);
```

DIOSA/XTP データストアに定義するスーパーストリーム名には、以下の命名規則で決定した文字列を指定する必要があります。

スーパーストリーム名 = ORA_{ストリーム名}_{拠点識別情報(環境変数 DIATC_CENTER_ID 設定値)}

(3) 更新ログ生成、登録処理

更新ログ生成処理を呼び出して更新ログを生成後、更新ログ登録処理を呼び出します。

更新ログ登録処理の第一パラメータには、固定で「FUNCID_DAC」を指定してください。

DB 更新と下記のメソッド呼び出しの順序は、どちらが先でも構いません。

```
更新ログ = DiatcDbUpdateLog.makeUpdateLog(SQL 文, バインド値);
```

```
DiatcDataReplicationFactory.getInstance().saveULogData(  
    DiatcDataReplicationConst.FUNCID_DAC,  
    更新ログサイズ,  
    更新ログ);
```

更新ログ生成処理（上記の makeUpdateLog メソッド）の SQL 文パラメータには、Oracle の ANSI 動的 SQL で実行可能な SQL を指定できます。DML 文だけでなく PL/SQL ブロックやストアド・プロシージャのコールも指定できます。

注意

更新ログ生成処理に指定する SQL について注意事項を説明します。

- データ同期先でも同じ SQL になるものを指定してください。
実行時に動的に値が決まるものが含まれていると、拠点間でデータの不一致が発生します。
- DDL を指定する場合は PL/SQL ブロックやストアド・プロシージャの中に定義して自律型トランザクションとして実行するようにしてください。
これは、DDL の実行前後に Oracle データベースが暗黙的な COMMIT 文を発行するためです。これを守らないと拠点間でデータの不一致が発生する可能性があります。
- オンライン中DBリカバリ支援機能を利用する場合は、「2.3.1 オンライン中DBリカバリ支援機能」の注意事項も参照してください。
- バインド変数の数は 1032 個以内にしてください。バインド変数の数が 1032 個を超える SQL を登録した場合、更新ログ反映処理でエラーが発生します。

(4) トランザクション終了処理

トランザクション終了処理をおこないます。

コミット / ロールバックに関わらず呼び出してください。

[Web0TX 環境の場合]

```
DiatcDataReplicationFactory.getInstance().tranTerm(false);  
DiatcDataReplicationFactory.removeInstance();
```

[Web0TX 環境以外の場合]

```
DiatcDataReplicationFactory.getInstance().tranTerm(con, false);  
DiatcDataReplicationFactory.removeInstance();
```

※con は、使用者側で取得した DB コネクション

(5) **拠点識別情報取得処理**

拠点識別情報が設定されているかどうかを知りたい場合に実行します。

任意のタイミングで実行可能です。

bool が true の場合拠点識別情報設定あり、false の場合設定なしです。

```
bool = DiatcDataReplication.isAvailable()
```

[注意事項]

Oracle の savepoint 機能には対応していません。

3.5 アプリケーションの生成

3.5.1 ユーザアプリケーション

(1) コンパイル

プログラムをコンパイルするために、オプションとして以下を指定します。

オプション	説明
+DD64	64bit オブジェクトを生成するように指定します。
-D_POSIX_C_SOURCE=199506L	POSIX スレッドが使用可能となるように指定します。
-I 製品インストールディレクトリ/include	本製品のインクルードファイル格納位置を指定します。(※1)
-I DIOSA/XTP インストールディレクトリ/include	DIOSA/XTP インクルードファイル格納位置を指定します。(※1)

※1 開発マシンの環境に合わせてパスを指定してください。

以下にコンパイル実行例を記載します。

```
# cc -c +DD64 -D_POSIX_C_SOURCE=199506L -I $(DIR_DIOSA)/include -I $(DIR_DIATC)/include
sample.c
```

(2) 実行ファイルの生成

実行ファイルを生成するために、オプションとして以下を指定します。

オプション	説明
-L 製品インストールディレクトリ/lib	本製品のライブラリファイル格納位置を指定します。(※1)
-L DIOSA/XTP インストールディレクトリ/lib	DIOSA/XTP ライブラリファイル格納位置を指定します。(※1)
-L Oracle インストールディレクトリ/lib	ORACLE ライブラリファイル格納位置を指定します。(※1)
-L TPBASE インストールディレクトリ/lib	TPBASE ライブラリファイル格納位置を指定します。(※1)
-L zlib インストールディレクトリ/lib	zlib ライブラリファイル格納位置を指定します。(※1)
-ldiatc	本製品の API が使用できるようにライブラリ (libdiatc.so) を指定します。
-ldxtp	DIOSA/XTP の API が使用できるようにライブラリ (libdxtp.so) を指定します。
-lpthread	POSIX スレッドのライブラリを指定します。

※1 開発マシンの環境に合わせてパスを指定してください。

以下にリンク実行例を示します。

※紙面の都合により 2 行で記載しています。

```
# cc -o sample sample.o -L $(DIR_DIATC)/lib -L $(DIR_DIOSA)/lib
-L $(DIR_ORACLE)/lib -L $(DIR_TPBASE)/lib -L $(DIR_ZLIB)/lib -ldiatc -ldxtp -lpthread
```

3.5.2 TPBASE上で動作するプログラムの生成

TPBASE 上で動作する各種ライブラリの作成方法を例示します。

(1) **コンパイル**

プログラムをコンパイルするために、オプションとして以下を指定します。

オプション	説明
+DD64	64bit オブジェクトを生成するように指定します。
-D_POSIX_C_SOURCE=199506L	POSIX スレッドが使用可能となるように指定します。
-I TPBASE インストールディレクトリ/include	TPBASE インクルードファイル格納位置を指定します。(※1)
-I 製品インストールディレクトリ/include	本製品のインクルードファイル格納位置を指定します。(※1)
-I DIOSA/XTP インストールディレクトリ/include	DIOSA/XTP インクルードファイル格納位置を指定します。(※1)

※1 開発マシンの環境に合わせてパスを指定してください。

以下にコンパイル実行例を記載します。

```
# cc -c +DD64 -D_POSIX_C_SOURCE=199506L -I$(DIR_TPBASE)/include -I $(DIR_DIOSA)/include
sample.c
```

(2) **実行ファイルの生成**

共有ライブラリを生成するために、オプションとして以下を指定します。

オプション	説明
-L 製品インストールディレクトリ/lib	本製品のライブラリファイル格納位置を指定します。(※1)
-L DIOSA/XTP インストールディレクトリ/lib	DIOSA/XTP ライブラリファイル格納位置を指定します。(※1)
-L Oracle インストールディレクトリ/lib	ORACLE ライブラリファイル格納位置を指定します。(※1)
-L TPBASE インストールディレクトリ/lib	TPBASE ライブラリファイル格納位置を指定します。(※1)
-L zlib インストールディレクトリ/lib	zlib ライブラリファイル格納位置を指定します。(※1)
-ldiatc	本製品の API が使用できるようにライブラリ (libdiatc.so) を指定します。
-ldxtp	DIOSA/XTP の API が使用できるようにライブラリ (libdxtp.so) を指定します。
-lpthread	POSIX スレッドのライブラリを指定します。

※1 開発マシンの環境に合わせてパスを指定してください。

対象プロダクトをインストールしていない場合は、指定不要です。

以下にリンク実行例を示します。

※紙面の都合により 2 行で記載しています。

```
# ld -b -o libsample.so sample.o -L $(DIR_DIOSA)/lib -L $(DIR_ORACLE)/lib  
-L $(DIR_ORACLE)/lib -L $(DIR_TPBASE)/lib -L $(DIR_ZLIB)/lib -ldiag -ldxtp -lpthread
```

D I O S A / X T P V2.1
データ変換・通信オプション
利用の手引

2017 年 9 月 4 版

日本電気株式会社
東京都港区芝五丁目 7 番 1 号
TEL (03) 3454-1111 (大代表)

©NEC Corporation 2011, 2017

日本電気株式会社の許可なく複製・改変などを行うことはできません。

本書の内容に関しては将来予告なしに変更することがあります。