Nimbusコンテキスト仕様書

2016年8月8日

目次

[1. 概要 1](#_Toc460865761)

[2. インタフェース設計 3](#_Toc460865762)

[2.1. Context 3](#_Toc460865763)

[2.2. ContextStore 4](#_Toc460865764)

[2.3. ServerInfo 5](#_Toc460865765)

[2.4. SharedContext 6](#_Toc460865766)

[2.5. SharedContextUpdateListener 9](#_Toc460865767)

[2.6. SharedContextValueDifferenceSupport 10](#_Toc460865768)

[2.7. DistributedSharedContext 12](#_Toc460865769)

[2.8. SharedContextKeyDistributor 16](#_Toc460865770)

[2.9. SharedContextTransactionManager 17](#_Toc460865771)

[3. 実装設計 19](#_Toc460865772)

[3.1. Context 19](#_Toc460865773)

[3.1.1. jp.ossc.nimbus.service.context.DefaultContextService 19](#_Toc460865774)

[3.1.2. jp.ossc.nimbus.service.context.ThreadContextService 20](#_Toc460865775)

[3.1.3. jp.ossc.nimbus.service.context.GroupContextService 21](#_Toc460865776)

[3.2. ContextStore 22](#_Toc460865777)

[3.2.1. jp.ossc.nimbus.service.context.DatabaseContextStoreService 22](#_Toc460865778)

[3.3. ServerInfo 29](#_Toc460865779)

[3.3.1. jp.ossc.nimbus.service.context.ServerInfoService 29](#_Toc460865780)

[3.4. SharedContext 31](#_Toc460865781)

[3.4.1. jp.ossc.nimbus.service.context.SharedContextService 31](#_Toc460865782)

[3.5. DistributedSharedContext 35](#_Toc460865783)

[3.5.1. jp.ossc.nimbus.service.context.DistributedSharedContextService 35](#_Toc460865784)

[3.6. SharedContextKeyDistributor 39](#_Toc460865785)

[3.6.1. jp.ossc.nimbus.service.context.MD5HashSharedContextKeyDistributorService 39](#_Toc460865786)

[3.6.2. jp.ossc.nimbus.service.context.DatabaseSharedContextKeyDistributorService 40](#_Toc460865787)

[3.7. SharedContextTransactionManager 42](#_Toc460865788)

[3.7.1. jp.ossc.nimbus.service.context.SharedContextTransactionManagerService 42](#_Toc460865789)

# 概要

コンテキストとは、特定のスコープにおいて、キーと値を紐づけて、データを保持する機能である。API的には、java.util.Mapインタフェースを継承したContextインタフェースである。

<java.util.Map>

put(key, value)

**Context**

値

キー

get(key)

key1

value1

key2

value2

スコープは、実装により異なるが、サービス単位、スレッド単位、JavaVM間共有などの実装がある。

　また、コンテキストに登録した状態を保存したり、コンテキストの状態を復元したりするコンテキストの補助機能を持つContextStoreインタフェースもある。

save()

load()

：

Context

**ContextStore**

save(context)

load(context)

：

JavaDocは、以下を参照。

<http://nimbus.osdn.jp/reports/apidocs/jp/ossc/nimbus/service/context/package-summary.html>

サンプルコードは、以下のzipファイル内のcacheを参照。

<https://osdn.jp/projects/nimbus/downloads/64236/nimbus-sample-1.2.3.zip>

# インタフェース設計

## Context

キーと値を紐づけて、データを管理する機能を提供するインタフェースである。

基本的なAPIは、データの集合管理を行うコレクションAPIであるjava.util.Mapインタフェースを継承している。異なるのは、後述するContextStoreを使った永続化機能を持つ部分である。

save()、load() etc…

put(key, value)

**Context**

ContextStore

save()、load() etc…

get(key)

以下に、ソースコードの例を示す。

import jp.ossc.nimbus.core.\*;

import jp.ossc.nimbus.service.context.\*;

Context context = (Context)ServiceManagerFactory.getServiceObject(“Context”);

context.put(“key””, “value”);

String value = (String)context.get(“key”);

以下に、Contextインタフェースの実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.context.DefaultContextService | キーと値をこのサービス単位で、メモリ中に保持する実装クラスである。 |
| 2 | jp.ossc.nimbus.service.context.ThreadContextService | キーと値をスレッド単位で、メモリ中に保持する実装クラスである。 |
| 3 | jp.ossc.nimbus.service.context.GroupContextService | Contextサービスをグルーピングして参照可能にする実装クラスである。  Contextの更新系のAPIはサポートしない。 |

## ContextStore

Contextのデータを永続化する機能を提供するインタフェースである。

Contextインタフェースの実装クラスに付与する事で、Contextの永続化API経由で呼び出され、Contextのデータを永続化する。Context全体の保存・読み込み、キーのみの読み込み、キー指定での保存・読み込み、全削除の機能を提供する。

save()

load()

loadKey()

save(key)

load(key)

save(context)

load(context)

loadKey(context)

save(context , key)

load(context , key)

Context

**ContextStore**

以下に、ContextStoreインタフェースの実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.context.DatabaseContextStoreService | Contextのデータをデータベースに永続化する実装クラスである。 |

## ServerInfo

サーバの情報（JavaVMの情報、OSの情報、サーバのアドレス等）をContextインタフェースから取得する機能を提供するインタフェースである。

Contextインタフェースを継承しているため、キーを指定して、サーバの情報を取得する事も可能であるが、各情報の静的なgetterも提供されている。また、getterが提供されている情報は、静的であるため、更新系APIでの更新はサポートしない。

save()、load() etc…

put(key, value)

**ServerInfo**

ContextStore

save()、load() etc…

get～()

以下に、ソースコードの例を示す。

import jp.ossc.nimbus.core.\*;

import jp.ossc.nimbus.service.context.\*;

ServerInfo si = (ServerInfo)ServiceManagerFactory.getServiceObject(“ServerInfo”);

String version = si.getJavaVersion();

以下に、ServerInfoインタフェースの実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.context.ServerInfoService | サーバ情報を提供する実装クラスである。 |

## SharedContext

JavaVMを跨いでデータを共有するContextインタフェースのサブインタフェースである。

Contextインタフェースの機能に加え、以下の機能を持つ。

* JavaVMを跨ぐため、データの取得や更新時に通信が発生するので、通信が必要な全ての機能において、通信タイムアウトを指定できる機能を持つ。
* 通信が発生し、更新に時間が掛かる可能性があるため、非同期に更新する機能を持つ。
* 構造的なデータを、部分的に更新する機能を持つ。
* データを同期的に更新したい場合、異なるJavaVMからも更新される可能性があり、JavaVMを跨いで同期する必要があるため、キー単位でロックを制御する機能を持つ。
* データの変更イベントの通知を受け取るSharedContextUpdateListenerを登録する機能を持つ。
* キー以外に、構造化されたデータの属性に対して、検索インデックスを設定し、データをキー以外で検索する機能を持つ。
* データが存在するノードで、データを参照処理するロジックをインタプリタ実行し、結果を取得するクエリ実行機能を持つ。

Java VM

SharedContextUpdateListener

addSharedContextUpdateListener(listener)

put(key, value, timeout)

**SharedContext**

put(key, value, timeout)

Java VM

**SharedContext**

Java VM

**SharedContext**

get(key, timeout)

いわゆるオンメモリ分散KVS（Key Value Store）で、複数のJavaVMを跨いで、データをキーに紐づけて登録し、参照する事が可能である。登録されたキーとデータのセットは、他のJavaVM上にも複製され、常に同期される。

データの取得は、KVSなので、基本はget(key)による主キーでの検索になるが、setIndex(indexName, propertyNames)で、データのプロパティに対してインデックスを設定しておき、createView()で検索ビューを作成して、インデックスを使った検索や、ANDやOR等の検索結果に対する集合論理演算が可能である。

setIndex(indexName, propNames)

put(key, value, timeout)

get(key, timeout)

**SharedContext**

createView()

SharedContextView

searchBy(data, indexName, propNames)

and()

searchBy(data, indexName, propNames)

：

getResultSet()

キー集合

SharedContextには、データを持つサーバモードと、データを持たないクライアントモードが存在する。クライアントモードでは、ローカルにデータを持たないので、データを参照する場合は、他のJavaVM上に存在するサーバモードのSharedContextと通信を行い、データを取り寄せる。

Java VM

Java VM

**SharedContext**

**（サーバモード）**

**SharedContext**

**（クライアントモード）**

get(key, timeout)

しかし、複数のデータを使って統計を行うような処理では、各データの取得で通信が発生すると、呼び出し回数に応じた通信時間が加算され、性能的な問題が生じる。その問題を解決するために、データを保持しているサーバモードのSharedContextが存在するJavaVMに、データの統計処理を依頼し結果のみを取得する事で、１度の通信で統計処理を行う機能が必要となる。また、統計処理を行うアプリケーションは、呼び出し側の要件であるため、呼び出される側にあらかじめ配置しておくのは、モジュール配置の設計上、非効率である。　これらの要件を満たす機能が、クエリ実行機能である。この機能では、SharedContext上のデータを参照して、統計処理を行うロジックを文字列で渡す事で、サーバモードのSharedContextが存在するJavaVM上で、そのロジックをインタプリタ実行して、結果のみを返すことができる。

Java VM

Java VM

**SharedContext**

**（サーバモード）**

**SharedContext**

**（クライアントモード）**

queryのインタプリタ実行

executeInterpretQuery(query, variables)

サーバモードのSharedContextを起動すると、既に起動しているサーバモードのSharedContextから、データを取得して同期を行う。その際に、サーバモードのSharedContextが複数存在すると、同じデータのはずだとは言え、どのSharedContextからデータを取得して同期するのか？という問題が起きる。そこで、サーバモードのSharedContextには、主系、副系という概念が現れる。基本的に、最初に起動したSharedContextが主系となり、他は副系となる。主系が死んだ場合は、次に起動したSharedContextが主系に成り代わる。

また、クエリ実行機能において、クエリが実行されるのも、主系にあたるJavaVMである。

以下に、SharedContextインタフェースの実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.context.SharedContextService | デフォルト実装クラスである。 |

## SharedContextUpdateListener

SharedContextへのデータの更新イベントの通知を受け取るリスナインタフェースである。

addSharedContextUpdateListener(listener)

SharedContext

イベント通知

**SharedContextUpdateListener**

　受け取る事ができる通知タイミングは、以下。

* コンテキストにデータが追加される前後。onPutBefore()、onPutAfter()
* 同期によって、コンテキストにデータが追加される前。onPutSynchronize()
* コンテキストにデータが更新される前後。onUpdateBefore()、onUpdateAfter()
* コンテキストからデータが削除される前後。onRemoveBefore()、onRemoveAfter()
* 同期によって、コンテキストがクリアされる前。onClearSynchronize()
* 主系と副系の切り替わり前。onChangeMain()、onChangeSub()

　これらのイベントの通知は、同期的に行われる。

SharedContextへのデータ更新のタイミングで、アプリケーションロジックを実行したいイベント駆動型アプリケーション開発者向けのインタフェースである。

## SharedContextValueDifferenceSupport

SharedContextへの差分更新をサポートするデータクラスが実装すべきインタフェースである。

SharedContext

**SharedContextValueDifferenceSupport**

put(key, value)

任意のデータクラス

任意のメソッドで更新を行い

更新内容を差分として取り出す

update(key, difference)

update(difference)

SharedContextValueDifference

SharedContextに登録するデータが、構造化された大きな情報量を持つクラスである場合、構造化データの一部を更新するために、毎回構造化データそのものをput(key, value)し直すのは、通信コストとしては非効率である。そこで、SharedContextは、データの部分更新をサポートする。

SharedContextValueDifferenceSupportは、SharedContextに登録するデータクラスが差分更新をサポートするために、実装すべきインタフェースである。

SharedContextValueDifferenceSupportが持つ機能は、以下である。

* 並列更新によってデータの結果不整合が起こらないようにするための更新番号を持つ。
* 更新差分情報を持つSharedContextValueDifferenceを受け取り、自分自身を更新する。
* 自分自身の複製を作る。

更新差分情報を表すSharedContextValueDifferenceインタフェースは、更新前の更新番号を持つことのみ定義されており、差分情報をどのように保持するかは、SharedContextValueDifferenceSupportを実装するクラスに、任されている。SharedContextValueDifferenceを生成する方法も、同様である。

SharedContextValueDifferenceSupportの実装クラスは、基本的には開発者が任意に実装する事を想定しているが、Nimbus内の標準実装として、提供するクラスが存在する。

以下に、SharedContextValueDifferenceSupportインタフェースの実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.context.SharedContextRecord | １次元データの構造体であるRecordクラスのサブクラスで、SharedContextValueDifferenceSupportを実装したクラスである。 |
| 2 | jp.ossc.nimbus.service.context.SharedContextRecordList | ２次元データの構造体であるRecordListクラスのサブクラスで、SharedContextValueDifferenceSupportを実装したクラスである。 |

## DistributedSharedContext

JavaVMを跨いでデータを分散共有するSharedContextインタフェースのサブインタフェースである。

SharedContextインタフェースの機能に加え、以下の機能を持つ。

* JavaVMを跨いでデータを分散するため、内部に複数のデータノードを持つ。それらの各データノードの状態を参照する機能を持つ。
* 縮退した場合に、分散したデータノードの再配置を行う機能を持つ。
* クエリ実行機能において、データが複数のJavaVM上のDistributedSharedContextに分散されているため、各JavaVM上のDistributedSharedContextで実行されたクエリの結果をマージするクエリを指定できるように、拡張されている。

DistributedSharedContextは、SharedContextの集合体として実装される。登録されるキーを内部に持つSharedContextのどこに分配するかをSharedContextKeyDistributorで決定し、データの分散を図る。この内部に配置されたSharedContextをデータノードと呼ぶ。

Java VM 1

put(key, value, timeout)

**DistributedSharedContext**

SharedContextKeyDistributor

SharedContext3

SharedContext2

SharedContext1

Java VM 2

**DistributedSharedContext**

SharedContext1

SharedContext3

SharedContext2

SharedContextKeyDistributor

get(key, timeout)

データノードであるSharedContextは、サーバモードの場合のみデータを保持する。その特性を利用して、DistributedSharedContextを持つ各JavaVMで、サーバモードのSharedContextを分散する事で、データ分散を図る。

Java VM 1

**DistributedSharedContext**

SharedContextKeyDistributor

SharedContext1

（サーバ）

SharedContext3

（クライアント）

SharedContext2

（クライアント）

Java VM 2

**DistributedSharedContext**

SharedContextKeyDistributor

SharedContext1

（クライアント）

SharedContext3

（クライアント）

SharedContext2

（サーバ）

Java VM 3

**DistributedSharedContext**

SharedContextKeyDistributor

SharedContext3

（サーバ）

SharedContext2

（クライアント）

SharedContext1

（クライアント）

上図の場合、各JavaVMは、全体の３分の１のメモリ使用量で、全てのデータを参照可能になる。

但し、この状態では、例えばJavaVM 3が突然死した場合、SharedContext3のサーバモードのノードが存在しなくなるため、そこに格納されていたデータ（全体の３分の１）が消失してしまう。これを回避するために、あらかじめ1つのSharedContextを複数のJavaVMに、サーバモードとして持つ事で、各データノードのバックアップを持つ事ができる。

Java VM 1

**DistributedSharedContext**

SharedContextKeyDistributor

SharedContext1

（サーバ）

SharedContext3

（クライアント）

SharedContext2

（サーバ）

Java VM 2

**DistributedSharedContext**

SharedContextKeyDistributor

SharedContext1

（クライアント）

SharedContext3

（サーバ）

SharedContext2

（サーバ）

Java VM 3

**DistributedSharedContext**

SharedContextKeyDistributor

SharedContext3

（サーバ）

SharedContext2

（クライアント）

SharedContext1

（サーバ）

こうすることで、１つのJavaVMが突然死した場合でも、データの消失を免れることが可能になる。但し、各JavaVMは、全体の３分の２のメモリ使用量が必要になる。

また、この構成において、JavaVM 3が死んだ場合、JavaVM全体で、SharedContext1とSharedContext3のサーバモードが１つになってしまう。この状態で、さらにJavaVM 2が死ぬと、SharedContext3のデータが消失してしまう。

そうなる事を防ぐためには、JavaVM 3が減った事によって、分散状況を再配置する必要がある。この分散の再配置をリハッシュと呼ぶ。リハッシュによって、各SharedContextのサーバモードのノード数が２つに保たれ、耐障害性が担保される。但し、この構成の場合、この時点で各JavaVMは、全てのデータをメモリ中に保持する事になる。つまり、JavaVMの消失により、全体のメモリ空間が狭くなる事には、注意が必要である。

Java VM 1

**DistributedSharedContext**

SharedContextKeyDistributor

SharedContext1

（サーバ）

SharedContext3

（サーバ）

SharedContext2

（サーバ）

Java VM 2

**DistributedSharedContext**

SharedContextKeyDistributor

SharedContext1

（サーバ）

SharedContext3

（サーバ）

SharedContext2

（サーバ）

また、DistributedSharedContext内のSharedContextにも主系、副系の概念があるが、SharedContext単体で使用する場合と異なり、起動順ではなく主系のノードが、うまく分散するように主系が決定される。

以下に、DistributedSharedContextインタフェースの実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.context.DistributedSharedContextService | デフォルト実装クラスである。 |

## SharedContextKeyDistributor

DistributedSharedContextのデータの分散を決定するインタフェースである。

DistributedSharedContext

put(key, value)

get(key)

selectDataNodeIndex (key, distributedSize)

**SharedContextKeyDistributor**

SharedContext2

SharedContext3

SharedContext1

DistributedSharedContextは、put(key, value)やget(key)などキーに紐づく処理を呼び出されると、内部に保持するデータノードであるSharedContextから、どのSharedContextに処理を依頼するかを決める必要がある。その機能の委譲先が、SharedContextKeyDistributor#selectDataNodeIndex(key, distributedSize)である。このメソッドで、指定されたキーが、どのデータノードに分散されるかを決定する。

以下に、SharedContextKeyDistributorインタフェースの実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.context.MD5HashSharedContextKeyDistributorService | デフォルト実装クラスである。  キーのMD5ハッシュ値を使って、分散するデータノードを決定する。 |
| 2 | jp.ossc.nimbus.service.context.DatabaseSharedContextKeyDistributorService | データベース上のキーからデータの分散を決定する実装クラスである。  データベースに、キーの全量が存在する前提で、キーのソート順に従って、等分になるように、分散するデータノードを決定する。 |

## SharedContextTransactionManager

SharedContextへのデータ更新に対して、トランザクション制御を付与するインタフェースである。

**SharedContextTransactionManager**

new

SharedContextTransaction

begin()

put(context, key, value)

put(key, value)

SharedContext

put(context, key, value)

put(key, value)

SharedContext

commit()

commit()

SharedContextTransactionManagerにトランザクションの開始begin()を通知すると、内部にトランザクションを管理するSharedContextTransactionが生成され、スレッドに紐づけられて管理される。

トランザクションに参加するSharedContextに、あらかじめSharedContextTransactionManagerを設定しておき、更新系のAPIを呼び出すと、SharedContextの更新は実際には行われずに、SharedContextTransactionManagerに更新内容が通知され、スレッドに紐づけられたSharedContextTransaction内に蓄積される。更新中の状態は、SharedContextTransaction内で管理され、SharedContextの参照系のAPIを呼び出すと、SharedContextTransaction内の更新中の状態が取得できる。

複数のSharedContextがトランザクションに参加可能で、最後にSharedContextTransactionManagerにトランザクションの発行commit()を通知すると、SharedContextTransaction内の更新情報が、各SharedContextに送られ実行される。複数のSharedContextがトランザクションに参加している場合で、いずれかのSharedContextへのトランザクションのコミットが失敗した場合は、全てのSharedContextに対して、トランザクション開始前の状態にロールバックされる。但し、このロールバックは、結果整合性を満たすために行われるため、コミットの時点で更新が成功したSharedContextでは、トランザクション更新後の状態になった後で、トランザクション更新前の状態に戻される。

トランザクションをコミットせずに、ロールバックした場合は、SharedContextへの更新は行われずに、SharedContextTransactionが破棄されるのみである。

DistributedSharedContextに対しても、利用可能である。

以下に、SharedContextTransactionManagerインタフェースの実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.context.SharedContextTransactionManagerService | デフォルト実装クラスである。 |

# 実装設計

インタフェース設計で示した各インタフェースのうち、機能性のあるインタフェースの実装クラスの機能を説明する。

## Context

### jp.ossc.nimbus.service.context.DefaultContextService

キーとそれに紐づくデータを、サービス内に格納する最も単純なContextインタフェースのデフォルト実装クラスである。

内部にjava.util.Mapを持ち、キーとそれに紐づくデータをMapに格納する。

save()、load() etc…

put(key, value)

**DefaultContextService**

java.util.Map

save()、load() etc…

get(key)

ContextStore

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ContextStoreServiceName | ServiceName | コンテキスト情報の永続化を行うContextStoreインタフェースを実装したサービスのサービス名を設定する。 |
| 2 | LoadOnStart | boolean | サービスの開始時に、ContextStoreサービスを使って読み込み処理を行うかどうかを設定する。  デフォルトは、false。 |
| 3 | LoadKeyOnStart | boolean | サービスの開始時に、ContextStoreサービスを使ってキーの読み込み処理を行うかどうかを設定する。  デフォルトは、false。 |
| 4 | SaveOnStop | boolean | サービスの停止時に、ContextStoreサービスを使って書き込み処理を行うかどうかを設定する。  デフォルトは、false。 |
| 5 | ClearBeforeSave | boolean | コンテキストの書き込み処理の前にストアをクリアするかどうかを設定する。  デフォルトは、trueでクリアする。 |

### jp.ossc.nimbus.service.context.ThreadContextService

キーとそれに紐づくデータを、スレッド単位に格納するContextインタフェースの実装クラスである。但し、ContextStoreを使った永続化機能はサポートしない。

内部に、java.lang.ThreadLocalを持ち、スレッド毎にjava.util.Mapを保持する事で、スレッド毎にキーとデータのマッピングを保持する。

put(key, value)

**ThreadContextService**

java.lang.ThreadLocal

java.util.Map

get(key)

サーブレットコンテナなどの、スレッドをプールして異なる処理を行うために同じスレッドを再利用する仕組みの場合、スレッドに紐づけられたMapも再利用されるため、同じスレッドが前回利用した時の状態が残る事には、注意が必要である。Nimbusでは、Nimbus AOPのThreadContextInitializeInterceptorServiceと組み合わせて使用する事で、この問題を回避する事ができる。

このサービスの属性は、存在しない。

### jp.ossc.nimbus.service.context.GroupContextService

複数のContextサービスをグルーピングして、１つのContextに見せるContextインタフェースの実装クラスである。但し、参照系の機能以外はサポートしない。

Context

get(key)

**GroupContextService**

Context

Context

同一のキーが、グルーピングされている複数のContextサービスに存在する場合は、ContextServiceNames属性に設定されている順番で探しに行くため、先に見つかったContextから取得される。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ContextServiceNames | ServiceName[] | グルーピングするContextインタフェースを実装したサービスのサービス名を配列で設定する。 |

## ContextStore

### jp.ossc.nimbus.service.context.DatabaseContextStoreService

Contextに格納されたキーと値をデータベースに永続化するContextStoreインタフェースの実装クラスである。

save()

load()

loadKey()

save(key)

load(key)

save(context)

load(context)

loadKey(context)

save(context , key)

load(context , key)

Context

**DatabaseContextStoreService**

DatabaseMapping

DatabaseSubMapping

DatabaseSubMapping

Connection

ConnectionFactory

PersistentManager

データベース

テーブル

テーブル

テーブル

ConnectionFactoryサービスを使って、JDBC Connectionを取得し、PersistentManagerサービスを経由して、データベース上のデータをContextサービスに読み込んだり、Contextサービス上のデータをデータベースに書き込んだりする。DatabaseMappingにデータベースの特定のテーブルと、Context上のキーとデータの交換方法を定義する。

データベース

ユーザID

名前

性別

ユーザテーブル

クエリ

select ユーザID->{user\_id},

名前->{name}

性別->{sex}

from ユーザテーブル

DatabaseMapping

Record

sex

name

user\_id

Context

ユーザID

キー

データ

User

・name

・sex

上図のように、データベース上のユーザテーブルから検索クエリでユーザレコードを検索し、Recordへマッピングして読み込む。そのRecordの“user\_id”値をContextのキーとし、“name”、“sex”の値をUserオブジェクトのプロパティにそれぞれ格納して、ContextにputするというようにDatabaseMappingに設定する事で、データベースからContextへの読み込みをマッピングする。同様に、Contextからデータベースへの書き込みに関しても、挿入クエリ、更新クエリ、オブジェクトのプロパティとのマッピングなどを設定し、マッピングする。

また、例えば、ユーザテーブルの1レコードに対して、電話番号テーブルのレコードが複数存在するような関係の１対多リレーションが存在する場合に、UserオブジェクトのtelephonNoプロパティに、配列としてそのユーザに紐づく複数の電話番号を格納したくなる。その場合は、DatabaseMappingに電話番号テーブルとのリレーションを表すDatabaseSubMappingを設定することで、実現できる。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ConnectionFactoryServiceName | ServiceName | JDBC接続を取得するConnectionFactoryサービスのサービス名を設定する。 |
| 2 | PersistentManagerServiceName | ServiceName | オブジェクトとSQLのマッピングを行うPersistentManagerサービスのサービス名を設定する。 |

DatabaseMappingは、addDatabaseMapping(DatabaseMapping)メソッドでインジェクションする。

以下にDatabaseMappingの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | UniqueKey | boolean | データベース上のキーが、レコードを一意に特定するユニークキーであるかどうかを設定する。言い換えると、Contextの値に格納されるデータがリスト構造の場合は、falseに設定する。  デフォルトは、true。 |
| 2 | SelectQuery | String | Contextへの読み込みに使用するクエリを設定する。 |
| 3 | KeySelectQuery | String | Contextへの読み込みで、キー単位での並列読み込みの際に使用するキーを検索するクエリを設定する。  並列読み込みを使用しない場合は、設定の必要はない。 |
| 4 | ParallelSelectQuery | String | キー単位での並列読み込みの際に使用するキーが条件句となる検索クエリを設定する。  並列読み込みを使用しない場合は、設定の必要はない。 |
| 5 | ParallelSize | int | キー単位での並列読み込みの際の並列度を設定する。  並列読み込みを使用しない場合は、設定の必要はない。 |
| 6 | SelectWhereQuery | String | キー指定での読み込みload(Context context, Object key)の際に、キーを指定して検索するクエリを設定する。 |
| 7 | InsertQuery | String | 保存の際に使用する挿入クエリを設定する。 |
| 8 | InsertKeySelectQuery | String | 保存の際に特定のキーだけを保存したい場合に、保存するキーを検索するエリを設定する。  設定しない場合は、Context上の全てのキーを保存する。 |
| 9 | UpdateQuery | String | 保存の際に使用する更新クエリを設定する。 |
| 10 | DeleteWhereQuery | String | 保存の際に使用する特定のキーに該当するレコードを削除するクエリを設定する。 |
| 11 | DeleteQuery | String | 削除の際に使用する削除クエリを設定する。 |
| 12 | KeyDatabaseRecord | Record | キーのロード時にPersistentManagerに渡すRecordオブジェクトを設定する。  Beanに直接マッピングする場合には、設定する必要はない。 |
| 13 | DatabaseRecord | Record | PersistentManagerに渡すRecordオブジェクトを設定する。  Beanに直接マッピングする場合には、設定する必要はない。 |
| 14 | DatabaseRecordList | RecordList | PersistentManagerに渡すRecordを生成するRecordListオブジェクトを設定する。UniqueKey属性がfalseの時に必要。  Beanに直接マッピングする場合には、設定する必要はない。 |
| 15 | KeyLoadPropertyName | String | 読み込んだRecordからキーとして取りだすプロパティ名を設定する。 |
| 16 | KeySavePropertyName | String | キーをRecordのプロパティへ設定するためのプロパティ名を設定する。 |
| 17 | KeyClass | Class | キーとなるBeanのクラスを設定する。  設定しない場合は、キーはRecordからKeyLoadPropertyName属性で設定されたプロパティ名で取得したオブジェクトとなる。 |
| 18 | KeyLoadPropertyMapping | Map<String, String> | 読み込んだRecordからキーのBeanへのプロパティマッピングを設定する。 |
| 19 | KeySavePropertyMapping | Map<String, String> | キーのBeanから値を取得して、Recordのプロパティへ設定するためのマッピングを設定する。 |
| 20 | ValueClass | Class | 読み込み結果となるBeanのクラスを設定する。  設定しない場合は、読み込み結果はRecordとなる。 |
| 21 | ValueRecord | Record | 読み込み結果となるRecordを設定する。 |
| 22 | ValueRecordList | RecordList | 読み込み結果となるRecordListを設定する。 |
| 23 | LoadPropertyMapping | Map<String, String> | 読み込んだRecordからBeanへのプロパティマッピングを設定する。 |
| 24 | SavePropertyMapping | Map<String, String> | Beanから値を取得して、Recordのプロパティへ設定するためのマッピングを設定する。 |
| 25 | FetchSize | int | 値を読み込む際のフェッチサイズを設定する。 |
| 26 | KeyFetchSize | int | キーを読み込む際のフェッチサイズを設定する。 |
| 27 | UseSubCursor | boolean | DatabaseSubMappingが登録されている場合に、子カーソルを使用するかどうかを設定する。  デフォルトは、true。 |
| 28 | UseSubConnection | boolean | DatabaseSubMappingが登録されている場合に、子カーソルに対して別のJDBCコネクションを使用するかどうかを設定する。  デフォルトは、false。 |
| 29 | BatchPersistCount | int | 保存する際のバッチ実行件数を設定する。 |
| 30 | BatchCommitOnPersist | boolean | バッチ実行で保存する際に、バッチ実行時にコミットするかどうかを設定する。  デフォルトは、true。 |
| 31 | ValueSort | boolean | UniqueKey属性がfalseでContextの値がBeanのリストとなる場合に、そのリストをソートするかどうかを設定する。  デフォルトは、falseでソートしない。 |
| 32 | ValueRecordListSort | Map<String[],boolean[]> | UniqueKey属性がfalseでContextの値がRecordListとなる場合に、そのリストのソート方法を設定する。  設定しない場合は、ソートしない。 |
| 33 | SynchronizedRecordList | boolean | 内部で生成するRecordListの同期化を設定する。  デフォルトは、true。 |

DatabaseSubMappingは、DatabaseMappingのaddSubMapping(DatabaseSubMapping)でインジェクションする。

以下にDatabaseSubMappingの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | SelectQuery | String | 読み込みカーソルに使用する検索クエリを設定する。 |
| 2 | SelectWhereQuery | String | 読み込みカーソルを使用しない場合に、親のキー単位で検索する条件付き検索クエリを設定する。 |
| 3 | InsertQuery | String | 保存の際に使用する挿入クエリを設定する。 |
| 4 | DeleteQuery | String | 削除の際に使用する削除クエリを設定する。 |
| 5 | DeleteWhereQuery | String | キー単位での削除の際に使用する削除クエリを設定する。 |
| 6 | DatabaseRecord | Record | PersistentManagerに渡すRecordオブジェクトを設定する。  Beanに直接マッピングする場合には、設定する必要はない。 |
| 7 | LoadPropertyMapping | Map<String,String> | 読み込んだRecordからBeanへのプロパティマッピングを設定する。 |
| 8 | SavePropertyMapping | Map<String,String> | Beanから値を取得して、Recordのプロパティへ設定するためのマッピングを設定する。 |
| 9 | KeyPropertyMapping | Map<String,String> | 読み込んだRecordのプロパティと、親キーのオブジェクトのプロパティのマッピングを設定する。  カーソルのブレイク条件となるため、カーソルを使用する場合は、設定する必要がある。 |
| 10 | IndexProperty | String | 読み込んだRecordからリストのインデックスを取得するプロパティを設定する。 |
| 11 | FetchSize | int | 読み込む際のフェッチサイズを設定する。 |
| 12 | BatchPersistCount | int | 保存する際のバッチ実行件数を設定する。 |
| 13 | BeanClass | Class | 読み込み結果となるBeanのクラスを設定する。  設定しない場合は、親のBean自体に読み込む。  但し、親BeanがRecordで、LoadBeanPropertyOfParent属性が指定してあり、そのプロパティの型がRecordListの場合は、そのRecordを生成して、読み込む。 |
| 14 | LoadBeanPropertyOfParent | String | 読み込んだ結果を、親Beanのどのプロパティにsetするかを設定する。 |
| 15 | NestedRecordList | Map<String,RecordList> | LoadBeanPropertyOfParent属性が指定してあり、そのプロパティの型がRecordListの場合に、使用するRecordListとネストレコードリスト名のマッピングを設定する。 |
| 16 | SaveBeanPropertyOfParent | String | 保存するBeanを親Beanから取得するプロパティ名を設定する。 |

## ServerInfo

### jp.ossc.nimbus.service.context.ServerInfoService

OSやJavaVMの情報をコンテキスト情報として取得可能なServerInfoインタフェースのデフォルト実装クラスである。

DefaultContextServiceのサブクラスとなっており、その機能に加え、OSやJavaVMの情報を取得するServerInfoのメソッドを実装している。ServerInfoインタフェースから取得可能な値は、Contexインタフェースのget(キー)でも取得可能だが、更新はできない。

save()、load() etc…

put(key, value)

**ServerInfoService**

ContextStore

save()、load() etc…

get～()

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ContextStoreServiceName | ServiceName | コンテキスト情報の永続化を行うContextStoreインタフェースを実装したサービスのサービス名を設定する。 |
| 2 | LoadOnStart | boolean | サービスの開始時に、ContextStoreサービスを使って読み込み処理を行うかどうかを設定する。  デフォルトは、false。 |
| 3 | LoadKeyOnStart | boolean | サービスの開始時に、ContextStoreサービスを使ってキーの読み込み処理を行うかどうかを設定する。  デフォルトは、false。 |
| 4 | SaveOnStop | boolean | サービスの停止時に、ContextStoreサービスを使って書き込み処理を行うかどうかを設定する。  デフォルトは、false。 |
| 5 | ClearBeforeSave | boolean | コンテキストの書き込み処理の前にストアをクリアするかどうかを設定する。  デフォルトは、trueでクリアする。 |

## SharedContext

### jp.ossc.nimbus.service.context.SharedContextService

キーとデータを、JavaVMを跨いで共有可能なSharedContextインタフェースのデフォルト実装クラスである。

SharedContextUpdateListener

SharedContextTransactionManager

SharedContextUpdateListener

CacheMap

**SharedContextService**

CacheMap

ContextStore

ClusterService

Interpreter

RequestConnectionFactoryService

このサービスは、様々なサービスの組み合わせで構成される。

SharedContextUpdateListenerを登録する事で、SharedContextへの更新イベントの通知を受け取ることが可能になる。

SharedContextTransactionManagerを設定する事で、SharedContextへの更新をトランザクション保証する事が可能になる。

CacheMapを設定する事で、SharedContextに登録したデータをCacheMapに登録するようになり、クライアントモードでも一時的にデータをキャッシュする事ができるようになる。また、CacheMapの機能により、一定条件に達した時に、データをあふれさせる事が可能になる。SharedContextのサーバモードと、クライアントモードであふれ制御の要件が異なる事が想定されるため、サーバモード時に使用するCacheMapと、クライアントモード時に使用するCacheMapをそれぞれ設定できる。例えば、クライアントモード時は、一定件数以上になるとキャッシュから削除するようにしておき、サーバモード時は、一定件数以上になるとファイルに退避するようにしておくなどである。モードを動的に変更しない場合は、モードにあったCacheMapのみ設定しておけば良い。

ContextStoreを設定する事で、永続化機能を使用することができる。

Interpreterを設定する事で、クエリ実行機能を使用することができる。クエリで使用する言語は、Interpreterサービスの実装に依存する。ExecuteThreadSize属性で、クエリ実行機能の処理スレッド数を設定することができる。

ClusterServiceとRequestConnectionFactoryサービスは、必須の構成となり、それぞれ、他のJavaVM上のSharedContextの存在の検知と、それらとの通信機能として使用する。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ContextStoreServiceName | ServiceName | コンテキスト情報の永続化を行うContextStoreインタフェースを実装したサービスのサービス名を設定する。 |
| 2 | LoadOnStart | boolean | サービスの開始時に、ContextStoreサービスを使って読み込み処理を行うかどうかを設定する。  デフォルトは、false。 |
| 3 | LoadKeyOnStart | boolean | サービスの開始時に、ContextStoreサービスを使ってキーの読み込み処理を行うかどうかを設定する。  デフォルトは、false。 |
| 4 | SaveOnStop | boolean | サービスの停止時に、ContextStoreサービスを使って書き込み処理を行うかどうかを設定する。  デフォルトは、false。 |
| 5 | ClearBeforeSave | boolean | コンテキストの書き込み処理の前にストアをクリアするかどうかを設定する。  デフォルトは、trueでクリアする。 |
| 6 | RequestConnectionFactoryServiceName | ServiceName | 他JavaVM上のSharedContextと通信するjp.ossc.nimbus.service.publish.RequestConnectionFactoryServiceのサービス名を設定する。 |
| 7 | ClusterServiceName | ServiceName | 他JavaVM上のSharedContext の存在を検知するjp.ossc.nimbus.service.keepalive.ClusterServiceサービスのサービス名を設定する。 |
| 8 | ClientCacheMapServiceName | ServiceName | クライアントモード時のjp.ossc.nimbus.service.cache.CacheMapサービスのサービス名を設定する。 |
| 9 | ServerCacheMapServiceName | ServiceName | サーバモード時のjp.ossc.nimbus.service.cache.CacheMapサービスのサービス |
| 10 | InterpreterServiceName | ServiceName | クエリ実行機能の、クエリのインタプリタ実行を行うjp.ossc.nimbus.service.interpreter.Interpreterサービスのサービス名を設定する。 |
| 11 | InterpretContextVariableName | String | クエリ実行機能で、クエリ中でSharedContextを参照する変数名を設定する。  デフォルトは、”context”。 |
| 12 | ExecuteThreadSize | int | クエリ実行機能を受け付けて処理するスレッドの数を設定する。  デフォルトは0で、要求受信スレッドでそのまま処理する。 |
| 13 | ExecuteQueueServiceName | ServiceName | ExecuteThreadSize属性を設定した場合の、クエリ実行受付キューとなるjp.ossc.nimbus.service.queue.Queueサービスのサービス名を設定する。  指定しない場合は、内部キューが使用される。 |
| 14 | SharedContextTransactionManagerServiceName | ServiceName | トランザクション実行のトランザクションを管理するSharedContextTransactionManagerサービスのサービス名を設定する。 |
| 15 | Subject | String | 他JavaVM上のSharedContextと通信する際の、メッセージの件名を設定する。  通常、変更する必要はないが、複数のSharedContextサービスで、通信するサービスを共用する場合は、混線しないようにそれぞれ異なるサブジェクトを設定する必要がある。 |
| 16 | Client | boolean | クライアント／サーバモードを設定する。  デフォルトは、falseでサーバモード。 |
| 17 | SynchronizeTimeout | long | 他JavaVM上のSharedContextとデータ同期を行う時のタイムアウト[ms]を設定する。  デフォルトは、5000[ms]。 |
| 18 | DefaultTimeout | long | タイムアウトを指定しないメソッドを呼び出した場合に適用されるタイムアウト[ms]を設定する。  デフォルトは、1000[ms]。 |
| 19 | SynchronizeOnStart | boolean | サービスの開始時に他JavaVM上のSharedContextとのデータ同期を行うかどうかを設定する。  デフォルトは、trueでデータ同期する。 |
| 20 | SaveOnlyMain | boolean | 主ノードのみがコンテキストの保存を行うかどうかを設定する。  デフォルトは、falseで、クライアントモード以外の全てのノードが保存する。 |
| 21 | SharedContextUpdateListenerServiceNames | ServiceName[] | SharedContextUpdateListenerサービスのサービス名配列を設定する。 |
| 22 | Index | Map<String,String[]> | インデックスを設定する。  インデックス名と、インデックスを張るデータのプロパティ名の配列を指定する。 |

## DistributedSharedContext

### jp.ossc.nimbus.service.context.DistributedSharedContextService

キーとデータを、JavaVMを跨いで分散共有可能なDistributedSharedContextインタフェースのデフォルト実装クラスである。

SharedContextUpdateListener

SharedContextTransactionManager

SharedContextUpdateListener

CacheMap

**DistributedSharedContextService**

CacheMap

ContextStore

ClusterService

Interpreter

RequestConnectionFactoryService

SharedContextKeyDistributor

このサービスは、様々なサービスの組み合わせで構成される。

内部にSharedContextServiceを持つため、構成要素のサービスは、ほとんど同じであるが、キーの分散を制御するSharedContextKeyDistributorが追加される。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | DistributedSize | int | データノードの数を設定する。  デフォルトは、2。 |
| 2 | ReplicationSize | int | データの複製数を設定する。  デフォルトは、2。 |
| 3 | SharedContextKeyDistributorServiceName | ServiceName | SharedContextKeyDistributorサービスのサービス名を設定する。  指定しない場合は、MD5HashSharedContextKeyDistributorServiceが適用される。 |
| 4 | RequestConnectionFactoryServiceName | ServiceName | 他JavaVM上のSharedContextと通信するjp.ossc.nimbus.service.publish.RequestConnectionFactoryServiceのサービス名を設定する。 |
| 5 | ClusterServiceName | ServiceName | 他JavaVM上のSharedContext の存在を検知するjp.ossc.nimbus.service.keepalive.ClusterServiceサービスのサービス名を設定する。 |
| 6 | ClientCacheMapServiceName | ServiceName | クライアントモード時のjp.ossc.nimbus.service.cache.CacheMapサービスのサービス名を設定する。 |
| 7 | ServerCacheMapServiceName | ServiceName | サーバモード時のjp.ossc.nimbus.service.cache.CacheMapサービスのサービス名を設定する。 |
| 8 | ContextStoreServiceName | ServiceName | ContextStoreサービスのサービス名を設定する。 |
| 9 | InterpreterServiceName | ServiceName | クエリ実行機能の、クエリのインタプリタ実行を行うjp.ossc.nimbus.service.interpreter.Interpreterサービスのサービス名を設定する。 |
| 10 | InterpretContextVariableName | String | クエリ実行機能で、クエリ中でSharedContextを参照する変数名を設定する。  デフォルトは、”context”。 |
| 11 | ExecuteThreadSize | int | クエリ実行機能を受け付けて処理するスレッドの数を設定する。  デフォルトは0で、要求受信スレッドでそのまま処理する。 |
| 12 | ExecuteQueueServiceName | ServiceName | ExecuteThreadSize属性を設定した場合の、クエリ実行受付キューとなるjp.ossc.nimbus.service.queue.Queueサービスのサービス名を設定する。  指定しない場合は、内部キューが使用される。 |
| 13 | ParallelRequestThreadSize | int | クエリ実行機能で、分散したノードに並列で要求を行うスレッド数を設定する。  デフォルトは0で、要求スレッドで直列に処理する。 |
| 14 | ParallelRequestQueueServiceName | ServiceName | ParallelRequestThreadSize属性を設定した場合の、クエリ実行受付キューとなるjp.ossc.nimbus.service.queue.Queueサービスのサービス名を設定する。  指定しない場合は、内部キューが使用される。 |
| 15 | SharedContextTransactionManagerServiceName | ServiceName | トランザクション実行のトランザクションを管理するSharedContextTransactionManagerサービスのサービス名を設定する。 |
| 16 | Subject | String | 他JavaVM上のSharedContextと通信する際の、メッセージの件名を設定する。  通常、変更する必要はないが、複数のSharedContextサービスで、通信するサービスを共用する場合は、混線しないようにそれぞれ異なるサブジェクトを設定する必要がある。 |
| 17 | Client | boolean | クライアント／サーバモードを設定する。  デフォルトは、falseでサーバモード。 |
| 18 | RehashEnabled | boolean | リハッシュが有効かどうかを設定する。  デフォルトは、trueで有効。 |
| 19 | SynchronizeTimeout | long | 他JavaVM上のSharedContextとデータ同期を行う時のタイムアウト[ms]を設定する。  デフォルトは、5000[ms]。 |
| 20 | RehashTimeout | long | リハッシュ処理のタイムアウト[ms]を設定する。  デフォルトは、10000[ms]。 |
| 21 | DefaultTimeout | long | タイムアウトを指定しないメソッドを呼び出した場合に適用されるタイムアウト[ms]を設定する。  デフォルトは、1000[ms]。 |
| 22 | ManagedDataNode | boolean | データノードをサービスとして登録するかどうかを設定する。  デフォルトは、falseで登録しない。 |
| 23 | LoadOnStart | boolean | サービスの開始時に、ContextStoreサービスを使って読み込み処理を行うかどうかを設定する。  デフォルトは、false。 |
| 24 | LoadKeyOnStart | boolean | サービスの開始時に、ContextStoreサービスを使ってキーの読み込み処理を行うかどうかを設定する。  デフォルトは、false。 |
| 25 | ClearBeforeSave | boolean | コンテキストの保存の前にストアをクリアするかどうかを設定する。  デフォルトは、trueでクリアする。 |
| 26 | MainDistributed | boolean | 主ノードを分散させるようにするかどうかを設定する。  デフォルトは、falseで、分散させない。  分散させない場合は、クラスタの参加順序に依存して、参加順序が先のノードで、データノードとなっているノードが主ノードになる。  分散させる場合は、クラスタの参加順序に依存して、クラスタメンバの走査開始点をノード番号でずらして、参加順序が先のノードで、データノードとなっているノードが主ノードになる。主ノードの分散の均等性を保証するものではない。 |
| 27 | SharedContextUpdateListenerServiceNames | ServiceName[] | SharedContextUpdateListenerサービスのサービス名配列を設定する。 |
| 28 | Index | Map<String,String[]> | インデックスを設定する。  インデックス名と、インデックスを張るデータのプロパティ名の配列を指定する。 |

## SharedContextKeyDistributor

### jp.ossc.nimbus.service.context.MD5HashSharedContextKeyDistributorService

キーのMD5ハッシュ値を使って分散するノードを特定するSharedContextKeyDistributorインタフェースのデフォルト実装クラスである。

DistributedSharedContext

put(key, value)

get(key)

selectDataNodeIndex(key, distributedSize)

**MD5HashSharedContextKeyDistributorService**

SharedContext2

SharedContext3

SharedContext1

ハッシュ値の分散を利用するため、厳密な分散が行われる訳ではないが、キーの数が多い場合は、それなりに分散される。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | KeyProperty | String | 分散を決定するキーのプロパティを設定する。  設定されていない場合は、キーそのもので分散を決定する。 |

### jp.ossc.nimbus.service.context.DatabaseSharedContextKeyDistributorService

キーがデータベースのテーブル上に全てあると仮定して、キーをソートしてデータノード数で等分割する事で分散するノードを特定するSharedContextKeyDistributorインタフェースの実装クラスである。

DistributedSharedContext

put(key, value)

get(key)

selectDataNodeIndex(key, distributedSize)

**DatabaseSharedContextKeyDistributorService**

SharedContext2

SharedContext3

SharedContext1

ConnectionFactory

Connection

PersistentManager

データベース

データベースに存在するキーであれば、最終的に等分散になる。但し、データベースに存在しないキーは、0番目のデータノードに振り分けられるため、偏る。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ConnectionFactoryServiceName | ServiceName | ConnectionFactoryサービスのサービス名を設定する。 |
| 2 | PersistentManagerServiceName | ServiceName | PersistentManagerサービスのサービス名を設定する。 |
| 3 | KeySelectQuery | String | キーを検索する検索クエリを設定する。 |
| 4 | DatabaseRecord | Record | データベースから検索したキーを読み込むレコードを設定する。  PersistentManagerにレコードにロードさせた後、特定のプロパティを取りだしてキーとする場合は、KeyPropertyName属性と組み合わせて設定する。 |
| 5 | KeyClass | Class | データベースから検索したキーを読み込むBeanのクラスを設定する。  PersistentManagerにこのBeanを直接ロードさせる場合は、この属性のみ設定する。  PersistentManagerにレコードにロードさせた後、特定のプロパティをレコードから取得して、このBeanに設定したい場合は、KeyPropertyMapping属性と組み合わせて設定する。 |
| 6 | KeyPropertyName | String | レコードから取得するキーのプロパティ名を設定する。 |
| 7 | KeyPropertyMapping | Map<String,String> | レコードから取得してBeanに設定するプロパティ名のマッピングを設定する。 |

## SharedContextTransactionManager

### jp.ossc.nimbus.service.context.SharedContextTransactionManagerService

SharedContextTransactionManagerインタフェースのデフォルト実装クラスである。

**SharedContextTransactionManager**

new

SharedContextTransaction

begin()

put(context, key, value)

put(key, value)

SharedContext

put(context, key, value)

put(key, value)

SharedContext

commit()

commit()

Timer

rollback()

TransactionTimeout属性が設定されている場合は、内部のTimerにタスクを登録し、指定されたタイムアウト値を過ぎても、コミットされない場合は、ロールバックを行う。

トランザクションの開始時に、指定できるロックモードを悲観ロックに指定すると、更新系のAPIを実行すると、指定されたキーのロックを自動的に取得し、他からの変更をロックする。トランザクションのコミットまたはロールバック時に、ロックは解放される。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | TransactionTimeout | long | トランザクションタイムアウト[ms]を設定する。  デフォルトは、タイムアウトしない。 |
| ２ | DefaultLockMode | int | 更新トランザクションのデフォルトのロックモードを設定する。  デフォルトは、悲観ロック。 |