目次

改訂情報

変更年月日	変更内容
2014-08-01	初版
2015-04-01	第2版 下記を追加・変更しました ■ 「*ユーザコンテキスト*」の「*プロパティ*」に、ユーザコンテキストで扱う組織はデフォルト組織セットに属するもののみである旨を追記
2015-12-01	第3版 下記を追加・変更しました ■ 「*切替*」の「*概要*」に以下の注意点を追記 ■ 依存先のアクセスコンテキストと同じ切替をサポートする必要がある ■ 切替処理はデフォルトコンテキストビルダで行う ■ 「*切替*」の「*コンテキストスイッチ用コンテキストビルダの実装*」に以下の注意点を追記 ■ Web実行環境で利用するコンテキストスイッチ処理を作成する場合は、キャッシュをサポートする必要がある ■ 「*キャッシュ*」の「*概要*」に、コンテキストスタック中のキャッシュの扱いについて追記 ■ 「*アカウントコンテキスト*」の「*プロパティ*」に、ロールID一覧はサブロールを含む旨を追記 ■ 「*付録*」の「*リソースID一覧*」に、ユーザ切替用リソースIDを追加して、一覧を最新化
2016-04-01	第4版 下記を追加・変更しました ■ 「*外部ユーザコンテキスト*」を追加しました。
2016-08-01	第5版 下記を追加・変更しました ■ 「*キャッシュ*」に TERASOLUNA Server Framework for Java (5.x) プログラミングガイドへのリンクを追記しました。
2016-12-01	第6版 下記を追加・変更しました ■ 「*アカウントコンテキスト*」の「*プロパティ*」に、数値形式のフォーマットIDを追加しました。 ■ 「*アカウントコンテキスト*」の「*ステータス遷移*」の 「図 アカウントコンテキストの各プロパティの解決順序（一般ユーザ）」に数値形式のフォーマットIDについて追記しました。
2018-08-01	第7版 下記を追加・変更しました ■ 「*アカウントコンテキスト*」の「*ステータス遷移*」の以下の説明を変更しました。 ■ 「*プロパティの解決順序*」

はじめに

本書の目的

本書ではアクセスコンテキストの仕組みについて説明します。

説明範囲は以下の通りです。

- アクセスコンテキストの用途と機能の詳細

  アクセスコンテキストの概要、および、仕様を説明し、実装を行う上で必要な情報を提供します。

対象読者

本書では次の開発者を対象としています。

- アクセスコンテキストの仕組みを理解したい。
- アクセスコンテキストを利用した実装をしたい。

次の内容を理解していることが必須となります。

- JavaEE を利用したWebアプリケーションの基礎

対象開発モデル

本書では以下の開発モデルを対象としています。

- JavaEE開発モデル
- スクリプト開発モデル

  なお、スクリプト開発モデルでは、アクセスコンテキストの実装はできません。

本書の構成

- *概要*

  アクセスコンテキストの用途、および、提供する機能の概要について説明します。

- *アクセスコンテキスト 仕様*

  アクセスコンテキストが提供する各機能の仕組みについて説明します。

- *付録*

  実装に必要な資料を提供します。

概要

この章では、アクセスコンテキストの用途、および、提供する機能の概要について説明します。

アクセスコンテキストとは

アクセスコンテキストとは、 intra-mart Accel Platform に対して現在アクセスしている処理実行者（以下、利用者）に関連する共有情報を保持するオブジェクトです。
アクセスコンテキストには、利用者が利用しているシステム情報、ログイン情報、アプリケーションなどに関する情報が含まれます。

利用者には、以下の様なアクターが含まれます。

- ログインユーザなどのブラウザからアクセスしているユーザ
- ジョブスケジューラ

アクセスコンテキストの概念

アクセスコンテキストの動作イメージは、以下の通りです。

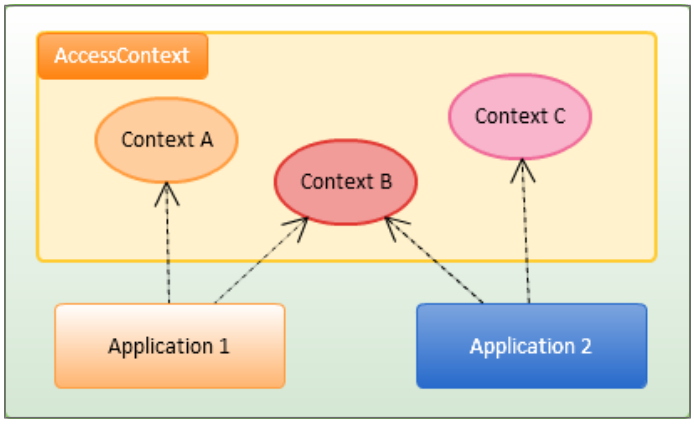

図 アクセスコンテキストの動作イメージ

intra-mart Accel Platform では、いろいろな種類のアクセスコンテキスト（図の Context A ～ Context C ）を定義することができます。
基盤機能やアプリケーション（図の Application 1 ～ Application 2 ）は、目的に応じて必要なアクセスコンテキストを利用可能です。

アクセスコンテキストの目的

一般的なWebアプリケーションでは、利用者に関連する情報の格納先として、HTTPセッションを利用します。

しかし、HTTPセッションを利用した場合、以下のような問題があります。

- HTTPセッションはWebアプリケーションに依存するため、ジョブスケジューラによる実行時は利用できません。
- HTTPセッションはWebアプリケーション内に 1 つのみであり、それらの情報はシステムやアプリケーションが自由に設定できます。
  そのため、情報を個別に管理することや整合性を保つことができません。

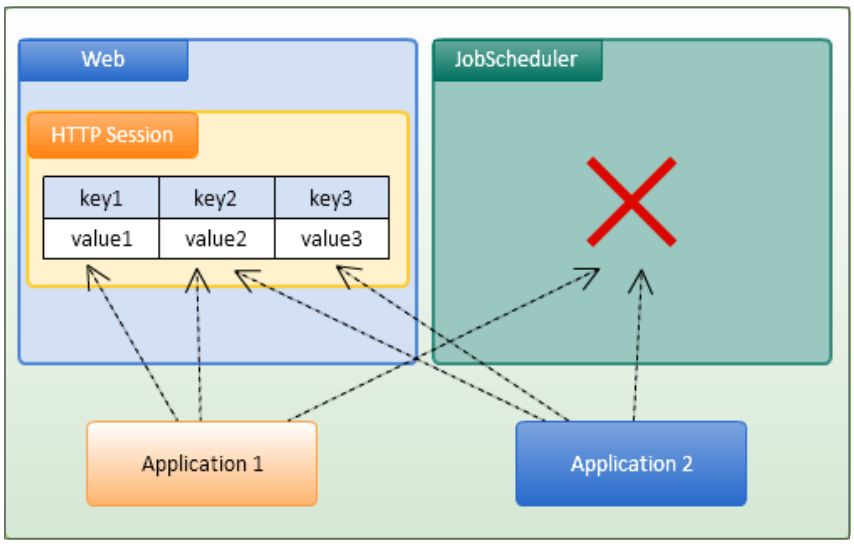

図 利用者の情報の格納先として、HTTPセッションを利用した場合

これらの問題を解決するため、利用者の情報をアクセスコンテキストとして定義します。
これにより、利用者の情報が 1 つのオブジェクトとして定義されるため、利用目的や対象とする情報が明確になります。
また、アクセスコンテキストを利用することで、Webアプリケーションを意識することなく利用者の情報を参照することができます。

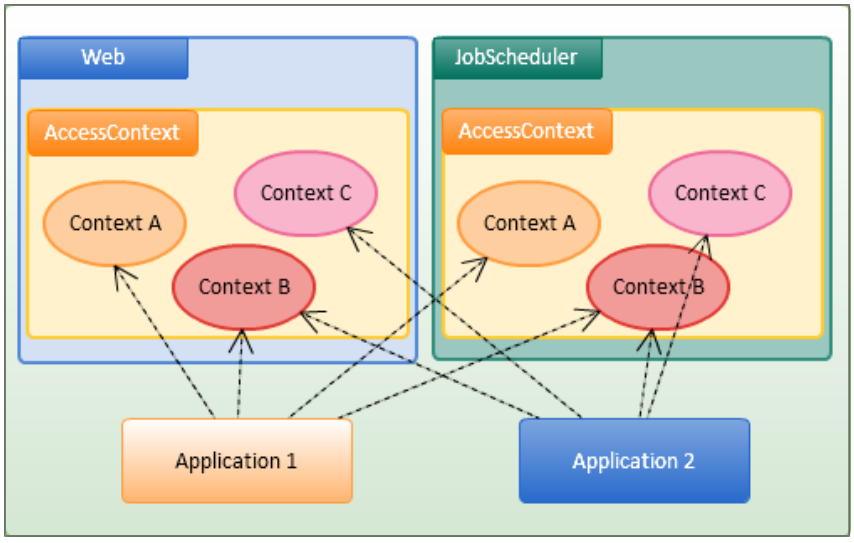

図 利用者の情報の格納先として、アクセスコンテキストを利用した場合

アクセスコンテキストの定義

アクセスコンテキストは、利用者のアクセスの開始から終了まで、利用者の情報や状態を保持します。
それらの情報は、システムおよびアプリケーションから自由に参照されます。

利用者のアクセスの開始から終了までの例は、以下のような場合です。

- ブラウザからのリクエストを受け付けてから、ブラウザにレスポンスを返すまで。
- ジョブスケジューラにより、ジョブの実行が開始してから、終了するまで。

intra-mart Accel Platform では、いろいろな種類のアクセスコンテキストを定義することができます。
これらの種類を区別する情報をコンテキスト種別と呼びます。コンテキスト種別については、「*コンテキスト種別*」を参照し

てください。

アクセスコンテキストの情報は、 intra-mart Accel Platform の基盤機能、または、アクセスコンテキストの情報を管理する機能によって設定されます。
それ以外の機能からは参照のみ可能です。

標準アクセスコンテキスト

intra-mart Accel Platform では、あらかじめいくつかのアクセスコンテキストが定義されています。
これらのアクセスコンテキストを「標準アクセスコンテキスト」と呼びます。
提供されている標準アクセスコンテキストは、以下の通りです。

表 標準アクセスコンテキスト 一覧

アクセスコンテキスト名	説明
アカウントコンテキスト	アクセスしたユーザアカウントに関連する情報を保持します。 ユーザコードやロケールなどのアカウント情報、認証状況などを取得できます。 システムで必須のアクセスコンテキストであり、 intra-mart Accel Platform 上で常に取得可能なアクセスコンテキストです。 詳細は、「 *アカウントコンテキスト* 」を参照してください。
クライアントコンテキスト	アクセスしたクライアントに関連する情報を保持します。 アクセスに利用した端末をクライアントタイプとして取得できます。 クライアントタイプには「PC」「スマートフォン」などがあります。
ユーザコンテキスト	ユーザのプロファイル情報や所属組織情報など、IM-共通マスタに紐づく情報を保持します。 詳細は、「 *ユーザコンテキスト* 」を参照してください。
認可サブジェクトコンテキスト	認可処理に関するサブジェクトの情報を保持します。 認可処理で利用され、ユーザが直接利用する必要はありません。 詳細は、「 認可仕様書 」の「 認可サブジェクトコンテキスト 」を参照してください。
ジョブスケジューラコンテキスト	ジョブやジョブスケジューラに関する情報を保持します。 ジョブやジョブネットに登録された情報や実行パラメータが取得できます。 ジョブ実行時のみ利用可能なアクセスコンテキストです。

ユーザの分類

アクセスコンテキストは利用者の状態を保持しています。
アクセスコンテキストの状態に基づいて、以下の通りにユーザが分類されます。

表 ユーザの分類一覧

未認証ユーザ	ログインしていないユーザを表します。
ログインユーザ	一般ユーザとしてログインしたユーザを表します。
システム管理者	システム管理者としてログインしたユーザを表します。
プラットフォームユーザ	基盤機能で利用する特別なユーザを表します。 ジョブの実行時は、このユーザとして実行されます。

ユーザの分類は、「 *アカウントコンテキスト* 」の情報に基づいて行われます。
アプリケーションでは、ユーザの分類の判定が必要な場合があります。
そのような場合は、「 *アカウントコンテキスト* 」-「 *ユーザの分類* 」の情報を参照して判定を行います。

アクセスコンテキストの実行環境

実行環境

実行環境とは、アクセスコンテキストを利用可能な環境を表します。
実行環境は、それぞれ利用者のアクセスの開始から終了までを表すライフサイクルを持っています。
ライフサイクルとは、アクセスコンテキストの生存期間です。

ライフサイクルについては、「 *ライフサイクル* 」を参照してください。

１つの実行環境には、あらかじめ定義された複数の種別のアクセスコンテキストが存在します。
ライフサイクル中は定義されたアクセスコンテキストを自由に利用することができます。

intra-mart Accel Platform では、通常利用される標準の実行環境が定義されています。
標準の実行環境の開始と終了処理は、基盤機能にて実行されるため、利用者が意識する必要はありません。

intra-mart Accel Platform で定義されている実行環境は、以下の通りです。

- Web実行環境

  HTTPを利用したリクエストで開始するライフサイクルを持つ実行環境です。
  HTTPセッションにアクセスコンテキストをキャッシュすることで、ユーザ情報をログインからログアウトまで保持します。

- ジョブ実行環境

  ジョブスケジューラから起動されるライフサイクルの実行環境です。

  ジョブスケジューラについては、「 ジョブスケジューラ仕様書 」を参照してください。

- 非同期実行環境

  別スレッド実行時に開始されるライフサイクルの実行環境です。
  intra-mart Accel Platform で提供する標準の非同期処理では、非同期処理実行元の実行環境がコピーされます。
  そのため、実行元と同じアクセスコンテキストを参照することができます。

  非同期処理については、「 非同期仕様書 」を参照してください。

- システム実行環境

  システム用の特別な実行環境です。
  この実行環境のライフサイクルは、Webアプリケーションと同一で、システム稼働中は常に参照することができます。
  この実行環境ではシステムデフォルトのアクセスコンテキストが管理され、その他の実行環境がライフサイクル外の場合に参照されます。

それぞれの実行環境のライフサイクルについては、「 *ライフサイクル* 」を参照してください。

上記以外の実行環境を利用するためには、独自にライフサイクルを開始する必要があります。
例えば、intra-mart Accel Platform で提供する標準の非同期処理を利用せずにスレッドを作成して処理を実行したい場合です。
その場合は、ライフサイクルを定義し、必要なアクセスコンテキストの生成処理の実装が必要となります。

標準アクセスコンテキストの実行環境

標準アクセスコンテキストが利用可能な実行環境は、以下の通りです。

表 標準アクセスコンテキストが利用可能な実行環境

アクセスコンテキスト名	Web実行環境	非同期実行環境	ジョブ実行環境	システム実行環境
アカウントコンテキスト	○	○	○	○
クライアントコンテキスト	○	○	×	○
ユーザコンテキスト	○	○	×	○

アクセスコンテキスト名	Web実行環境	非同期実行環境	ジョブ実行環境	システム実行環境
認可サブジェクトコンテキスト	○	○	×	○
ジョブスケジューラコンテキスト	×	×	○	×

切替

アクセスした状態によりアクセスコンテキストは自動で生成されますが、ユーザの操作やプログラムによってアクセスコンテキストの状態を変更したい場合があります。
アクセスコンテキストは共有情報であるため、任意のアクセスコンテキストの情報を直接変更すると、不整合が発生する可能性があります。
そのため、特別なアクセスコンテキストの変更機能を定義して呼び出すことで、整合性を保ったまま変更するようにします。
この変更機能を、アクセスコンテキストの切替機能と呼びます。

切替機能は、その目的により以下の通りに分類されます。

- コンテキストスイッチ

  以前のアクセスコンテキストは破棄され、新しいアクセスコンテキストが生成されて利用されます。
  新しいアクセスコンテキストで、継続してアクセスしたい場合に利用されます。

  例えば、複数の組織に所属しているユーザが、主所属ではない組織のユーザとしてアクセスしたい場合、コンテキストスイッチにより、所属組織を切り替えます。

  実行イメージは、以下の通りです。
  図中の「Context SWITCH」により、コンテキストスイッチが実行されます。

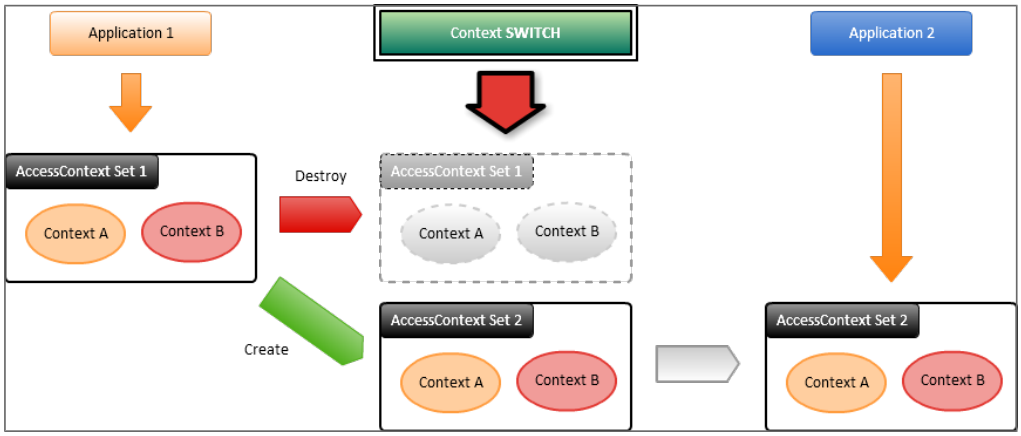

  図 コンテキストスイッチの処理イメージ

- コンテキストスタック

  以前のアクセスコンテキストは一時的に退避され、新しいアクセスコンテキストが生成されて利用されます。
  新しいアクセスコンテキストの利用が終わった場合、任意のタイミングで以前のアクセスコンテキストに戻ることができます。
  一時的に別のアクセスコンテキストでアクセスしたい場合に利用されます。

  例えば、スマートフォンでアクセス時に、特定の画面のみPC用の画面を表示したい場合は、コンテキストスタックにより、一時的にアクセスしたクライアント情報を切り替えます。
  別の画面を表示した場合は、アクセスコンテキストが以前の状態に戻っているため、スマートフォン用の画面が表示されます。

  実行イメージは、以下の通りです。
  図中の「Context STACK」により、コンテキストスタックが実行され、「Context POP」により、以前のアクセスコンテキストに戻ります。

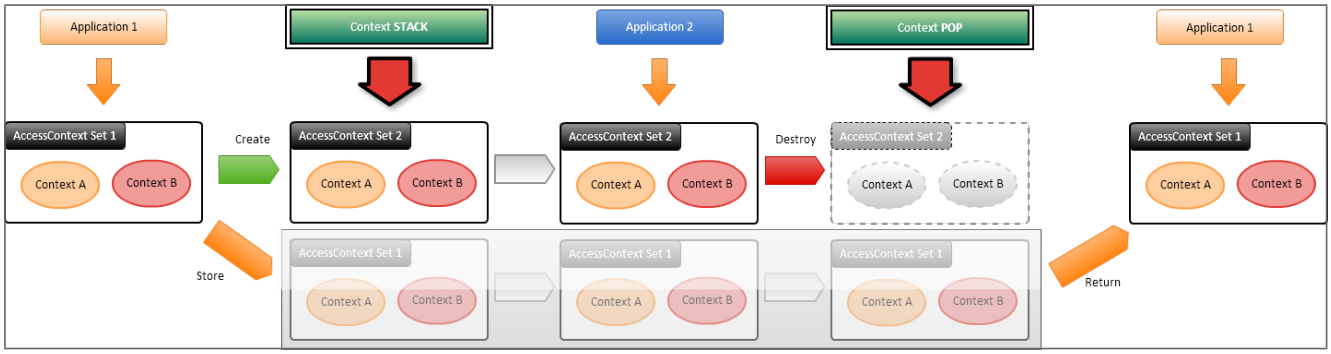

図 コンテキストスタックの処理イメージ

切替処理の詳細については、「 *切替* 」を参照してください。

アクセスコンテキスト 仕様

この章では、アクセスコンテキストの各機能の詳細を説明します。

ライフサイクル

ここでは、アクセスコンテキストのライフサイクルについて説明します。

項目

概要

アクセスコンテキストの生存期間をライフサイクルと呼びます。

アクセスコンテキストは、ライフサイクルの開始処理により、参照可能な状態となり、ライフサイクルの終了処理により、全てのアクセスコンテキストは破棄されます。

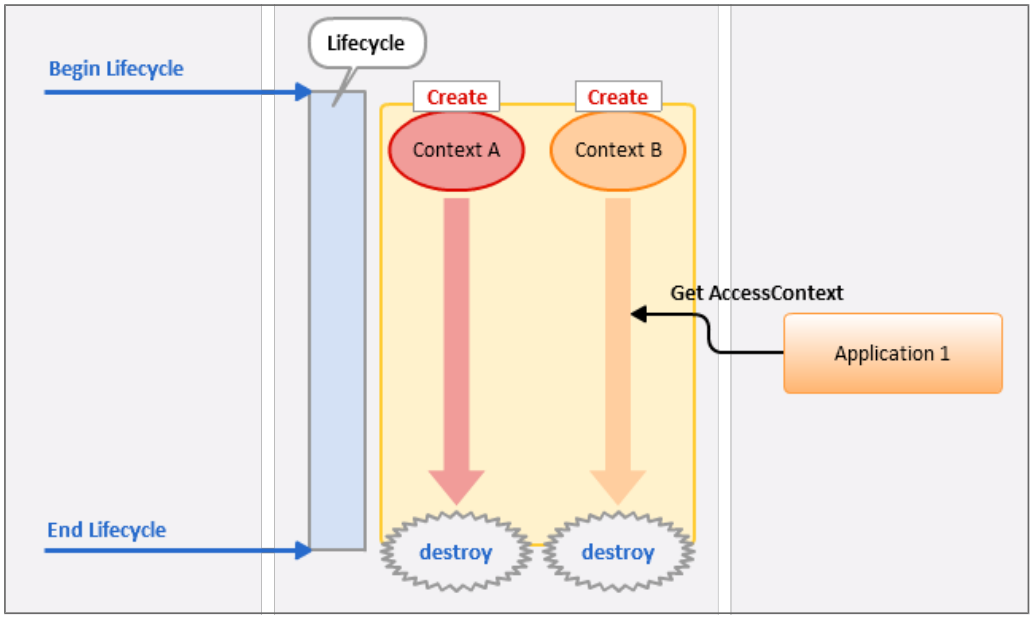

図 アクセスコンテキストのライフサイクルイメージ

ライフサイクルは、管理方式によって以下の２つに分類されます。

- システムライフサイクル

  システム実行環境のライフサイクルです。
  システム起動時に開始され、システム停止時に終了します。

- アプリケーションライフサイクル

  システム実行環境以外のライフサイクルです。
  基盤機能やアプリケーションで管理され、明示的に開始終了を行うライフサイクルです。

プログラム中からアクセスコンテキストを常に利用可能とするために、実行環境ごとにライフサイクルが定義されています。

次節より、実行環境ごとのライフサイクルを説明します。

アプリケーションライフサイクル

Web実行環境

Web実行環境では、HTTPによるリクエストごとにライフサイクルの開始と終了処理が実行されます。
この処理は `ServletFilter` で行われます。

通常、Web実行環境では連続したアクセスが行われ、HTTPセッションにより、前回のアクセス情報が継続して利用されます。
アクセスコンテキストを利用した場合でも、前回のアクセス情報を継続するため、アクセスコンテキストのキャッシュ機能を利用します。
初回アクセス時やログイン時に作成したアクセスコンテキストはHTTPセッションに格納されます。
キャッシュされた情報は、ログアウトやHTTPセッションのタイムアウトが発生するまでは、次のアクセスでも継続して利用されます。
アクセスコンテキストの内容を変更する必要がある場合は、キャッシュを破棄して再作成されます。

アクセスコンテキストのキャッシュ機能の詳細は、「 *キャッシュ* 」を参照してください。

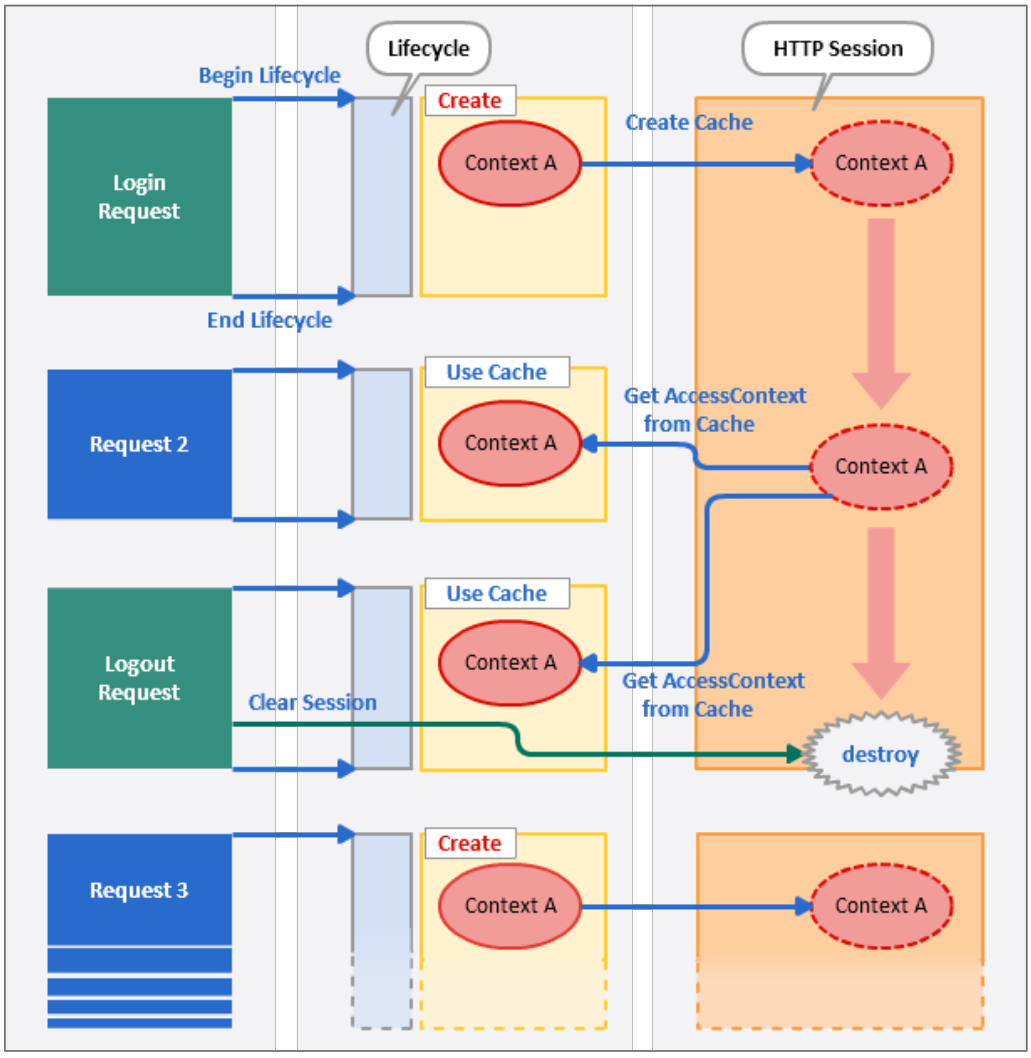

図 Web実行環境のライフサイクルイメージ

ジョブスケジューラ実行環境

ジョブ実行環境では、ジョブスケジューラによりジョブネットごとにライフサイクルの開始と終了が実行されます。
ジョブネットの開始時に、ジョブネットに登録された情報を利用してアクセスコンテキストが生成されます。

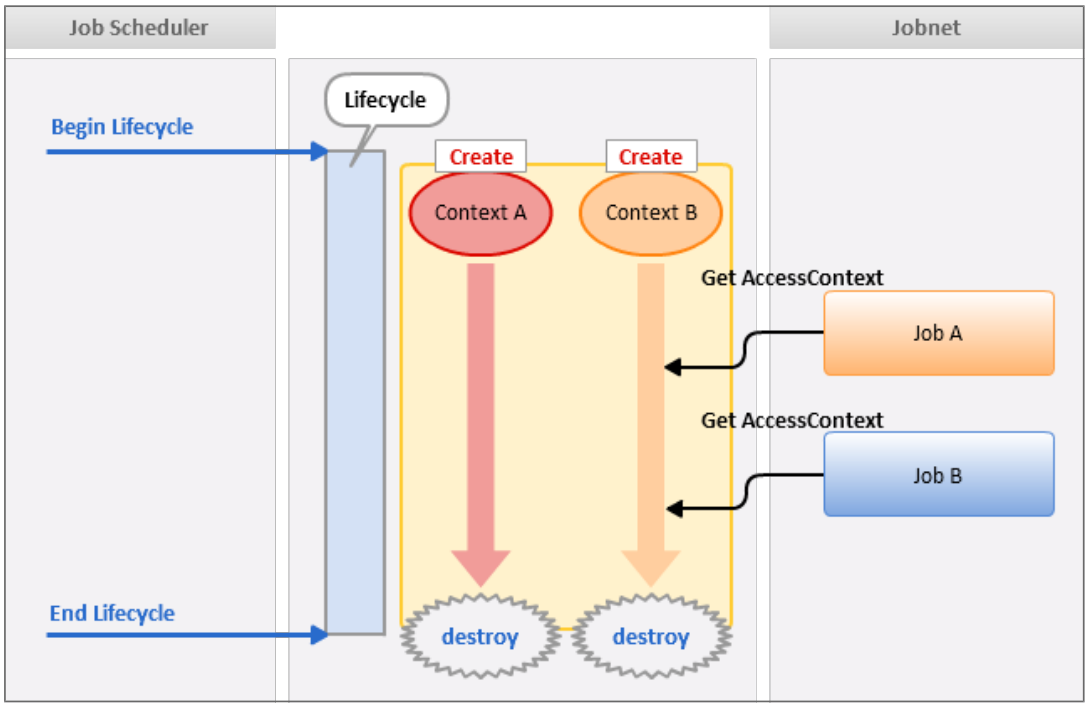

図 ジョブ実行環境のライフサイクルイメージ

ジョブスケジューラ、ジョブ、および、ジョブネットについては、「 ジョブスケジューラ仕様書 」を参照してください。

非同期実行環境

非同期実行環境では、Web実行環境のアクセスコンテキストをコピーしてライフサイクルが開始されます。
処理の流れは以下の通りです。

1. 非同期処理の要求時に、現在の Web実行環境のアクセスコンテキストを一時保存します。
2. 非同期処理の開始時に、保存されたアクセスコンテキストを復元します。
3. 非同期処理の終了時に、非同期実行環境のアクセスコンテキストを破棄します。

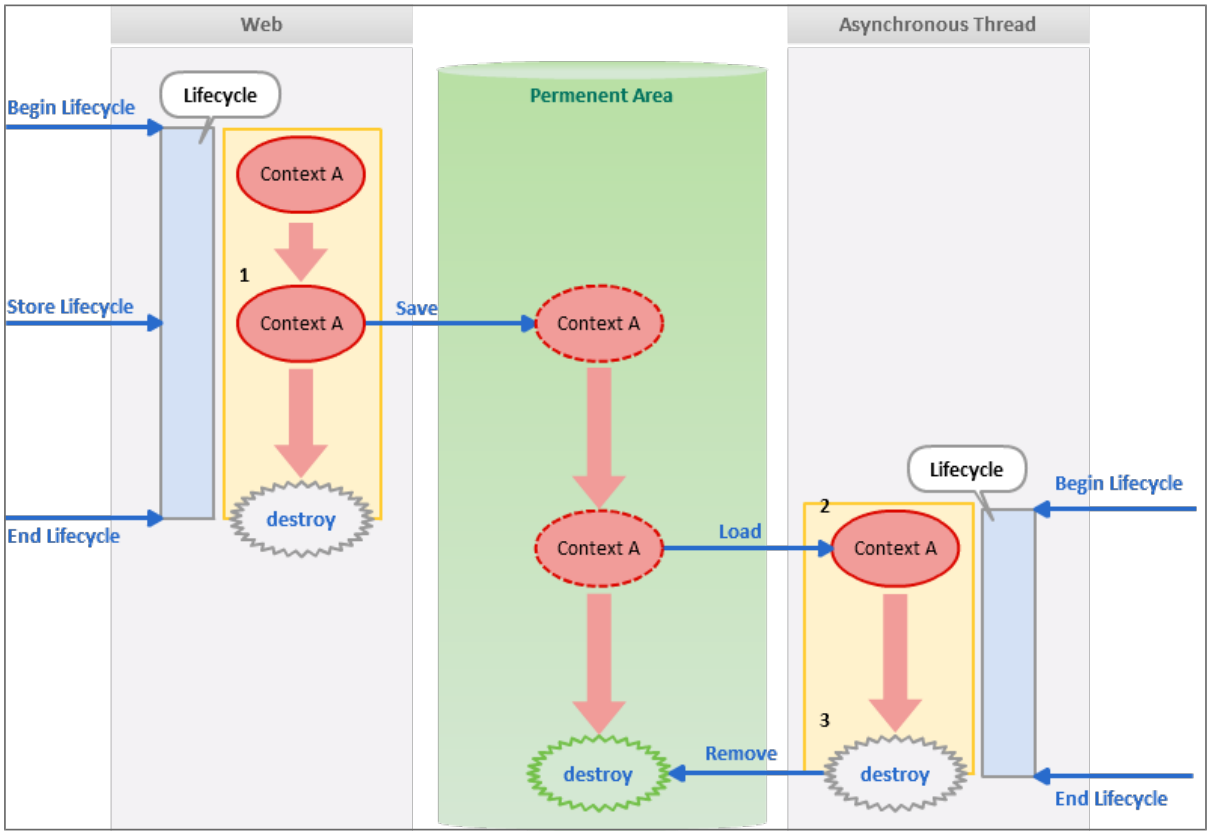

図 非同期実行環境でのライフサイクルイメージ

非同期処理については、「 非同期仕様書 」を参照してください。

システムライフサイクル

システム実行環境

システム実行環境では、システムの起動時にシステムライフサイクルが開始され、システムの終了時にシステムライフサイクルが終了します。

システムライフサイクルは、それ以外のライフサイクルとは別途管理され、シングルトンのアクセスコンテキストの生成・破棄を行います。
アクセスコンテキストの取得要求があった場合は、以下の順に取得が行われます。

1. 取得要求元の実行環境のアクセスコンテキストの取得
   取得できた場合、そのアクセスコンテキストを返却します。

2. システム実行環境のアクセスコンテキストの取得
   1. が取得できなかった場合に、システム実行環境のアクセスコンテキストを返却します。

システム実行環境のアクセスコンテキストが取得されるのは、アプリケーションライフサイクルが開始していない場合です。
具体的には、システム起動処理や、バックグラウンドで動作するプロセスなどでアクセスコンテキストを利用する場合です。

システムライフサイクルとアプリケーションライフサイクル（以下の例では、Web実行環境）の動作イメージは、以下の通りです。

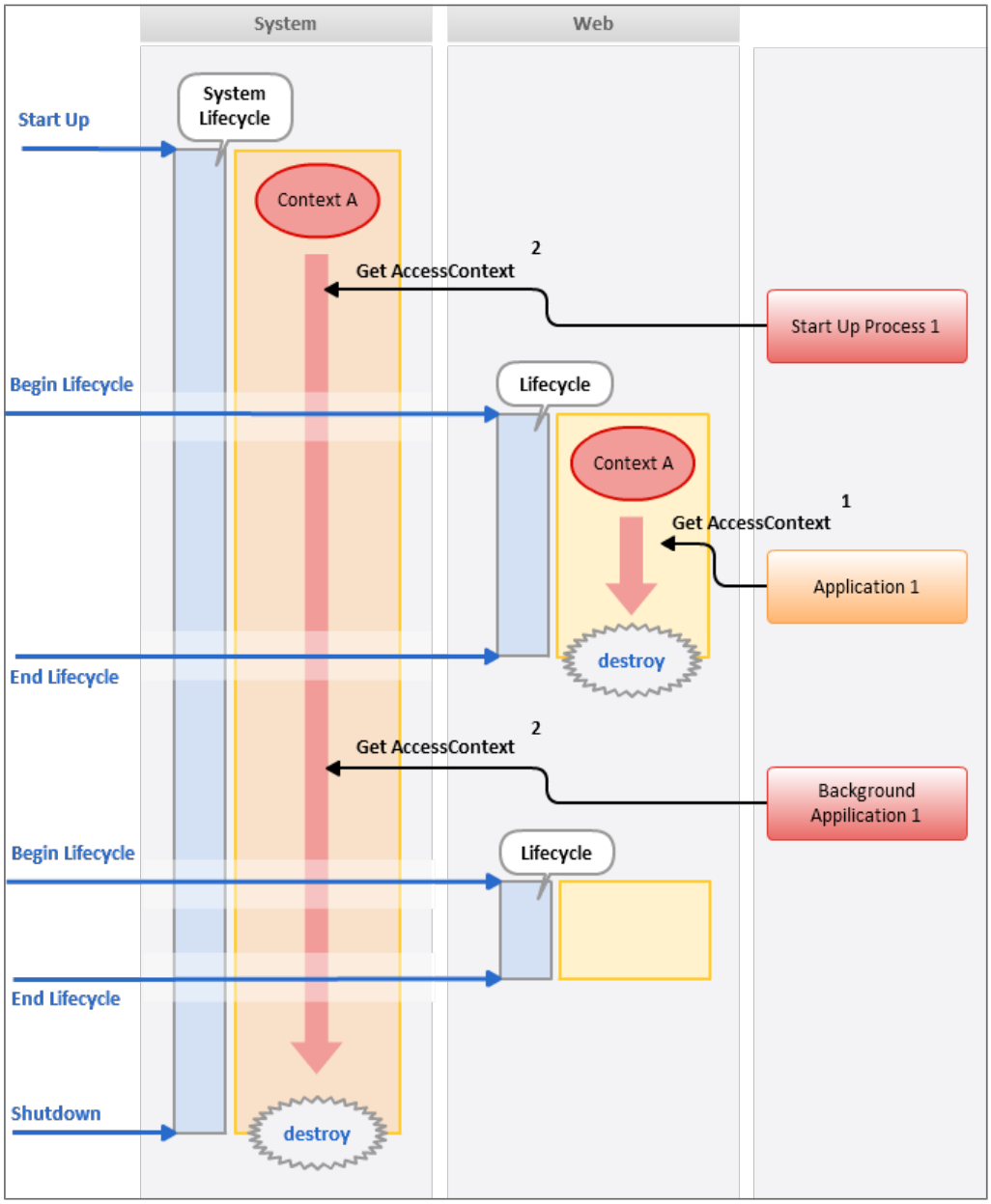

図 システム実行環境とWeb実行環境のライフサイクルイメージ

アクセスコンテキストフレームワーク

ここでは、アクセスコンテキストの機能を実現するための基本的な機構を説明します。

項目

概要

アクセスコンテキストは、ライフサイクルの操作によりアクセスコンテキストを生成することで利用可能な状態となります。
ライフサイクルの操作の種類は以下の通りです。

- ライフサイクルの開始
- ライフサイクルの切替
- ライフサイクルの終了

ここでは、アクセスコンテキストの仕組みについて、以下の順に説明します。

1. *ライフサイクルの開始フロー* について説明します。
   ライフサイクルの切替については、「 *切替* 」で説明します。
   また、ライフサイクルの終了処理は、アクセスコンテキストの破棄のみが行われるため、ここでは説明を省略します。
2. ライフサイクルの操作時に利用される、以下の要素について説明します。
   - *環境情報（Resource）*
   - *アクセスコンテキストモデル*
   - *コンテキストビルダ*
   - *コンテキストデコレータ*

ライフサイクルの開始フロー

ライフサイクルの開始により、アクセスコンテキストの生成処理が実行されます。
アクセスコンテキストの生成処理は、定義されたアクセスコンテキストごとにコンテキストビルダを呼び出して実行されます。
コンテキストビルダについては、「 *コンテキストビルダ* 」を参照してください。

アクセスコンテキストの処理は、以下の順に実行されます。

1. 開始処理の呼び出し（ `Lifecycle#begin()` ）
2. アクセスコンテキスト一覧取得
3. アクセスコンテキストごとに以下の処理を実行

1. コンテキストビルダの選出
2. コンテキストビルダの生成処理実行
3. アクセスコンテキスト保存先への格納

環境情報（Resource）

アクセスコンテキストの生成は、「環境情報（Resource）」を基に実行されます。
環境情報とは、実行環境と対象処理を表すオブジェクトで、環境情報を特定するプロパティとして、リソースIDを保持します。

環境情報を表すクラスは、以下のクラスです。

`jp.co.intra_mart.foundation.context.core.Resource`

ライフサイクルの操作ごとに環境情報を生成して利用します。
環境情報には、対象のライフサイクル操作で必要な情報を設定して、ライフサイクルの操作を実行します。

また、以降で説明する「*コンテキストビルダ*」や「*コンテキストデコレータ*」では、実行時の引数に環境情報が設定されます。
それぞれのクラスでは、ライフサイクルの操作実行時に設定された環境情報を参照できます。

リソースID

リソースIDは、任意の文字列です。
処理を分類するために、以下のように「.（ドット）」区切りで定義されます。

例） `platform.request` : Web実行環境の開始処理を表すリソースID

`platform` で始まるリソースIDは、 intra-mart Accel Platform で提供するリソースIDです。
新しいリソースIDを定義する場合は、他のリソースIDと重複しないようにアプリケーション固有の文字列を含めて定義してください。

intra-mart Accel Platform で提供するリソースIDについては、「*リソースID一覧*」を参照してください。

その他の情報

環境情報は、リソースIDの他に以下の情報を持ち、コンテキストビルダにより参照されます。

- リソース情報

  環境情報に付属する任意の情報です。
  コンテキストビルダに渡される引数として利用されます。
  `Resource#getResource()` / `Resource#setResource()` メソッドを利用して取得・設定を行います。

  ライフサイクルの操作実行者が設定する情報で、処理中は変更されません。

- リソース属性

  環境情報の追加情報です。
  キーと値で任意の情報が設定されます。

  ライフサイクルの操作の実行状態を表す情報で、ライフサイクルの操作処理プログラムの中で設定する情報です。
  例えば、アカウントコンテキスト生成中にアカウントに関する追加情報を設定し、ユーザコンテキストの生成時にその情報を参照する、といった使い方を想定しています。

  リソース属性のキーは任意の文字列です。
  ただし、キーの衝突を避けるため、利用する処理クラスの完全修飾子（FQCN）をキーとして利用してください。

  例） `jp.co.intra_mart.system.context.impl.cache.SessionContextCachePolicy.session.create`

  `Resource#getAttribute()` /`Resource#setAttribute()` メソッドを利用して取得・設定を行います。

環境情報は処理対象となる複数のアクセスコンテキストのコンテキストビルダで共有されます。
環境情報に設定する値は、複数のクラスで処理が可能な情報を設定するようにしてください。

詳しくは、「Resource クラスの APIドキュメント」を参照してください。

Web用環境情報

Web実行環境の環境情報は、以下のクラスで定義されます。

jp.co.intra_mart.foundation.context.web.HttpResource

このクラスを利用すれば、HTTPリクエスト（HttpServletRequest）およびHTTPレスポンス（HttpServletResponse）オブジェクトにアクセスが可能です。

Web実行環境用の処理クラス内で、以下のように Resource クラスをキャストして利用してください。

```
HttpResource httpResource = HttpResource.class.cast(resource);
```

アクセスコンテキストモデル

ここでは、アクセスコンテキストの実装クラスについて説明します。

アクセスコンテキストを表す実装クラス（または、インタフェース）をアクセスコンテキストモデルと呼びます。
アクセスコンテキストモデルは、以下の要素で定義されます。

- コンテキスト種別

  アクセスコンテキストを区別する情報

- プロパティ

  それぞれのアクセスコンテキストが保持する情報

また、アクセスコンテキストは便宜上、それぞれ名称（アクセスコンテキスト名）を持っています。
説明上は、アクセスコンテキスト名を利用します。

コンテキスト種別

アクセスコンテキストモデルは、コンテキスト種別を保持します。
コンテキスト種別は個々のアクセスコンテキストを区別する情報で、アクセスコンテキストのIDとして利用されます。
１つの実行環境には、あらかじめ定義されたコンテキスト種別のアクセスコンテキストが１つずつ存在します。

コンテキスト種別は、アクセスコンテキストモデルの完全修飾クラス名（FQCN）の文字列で表されます。

例）アカウントコンテキストのコンテキスト種別

jp.co.intra_mart.foundation.context.model.AccountContext

標準アクセスコンテキストのコンテキスト種別は、以下の通りです。

表 標準アクセスコンテキストのコンテキスト種別一覧

アクセスコンテキスト名	コンテキスト種別
アカウントコンテキスト	jp.co.intra_mart.foundation.context.model.AccountContext
クライアントコンテキスト	jp.co.intra_mart.foundation.context.model.ClientContext
ユーザコンテキスト	jp.co.intra_mart.foundation.user_context.model.UserContext
認可サブジェクトコンテキスト	jp.co.intra_mart.foundation.authz.context.AuthzSubjectContext

アクセスコンテキスト名	コンテキスト種別
ジョブスケジューラコンテキスト	jp.co.intra_mart.foundation.job_scheduler.JobSchedulerContext

アクセスコンテキストモデルの定義

アクセスコンテキストを定義するには、以下のインタフェースを実装します。

jp.co.intra_mart.foundation.context.model.Context

getType() メソッドを実装して、コンテキスト種別の定義を行います。

> **コラム**
>
> コンテキスト種別は、通常は自分自身の java.lang.Class オブジェクトを設定します。
> 実行環境ごとに複数の実装クラスを作成する場合は、共通のインタフェースを作成して、コンテキスト種別としてください。

アクセスコンテキストのモデルクラスは、プロパティのみを持つシンプルなオブジェクトとなるように実装します。
複雑な振る舞い（メソッド）を持つことは、共有情報の参照先としてのアクセスコンテキストの役割に反します。

アクセスコンテキストが提供する情報は、参照するための情報です。
そのため、アクセスコンテキストモデルは、 Setter のみ定義した読み取り専用のプロパティをもつインタフェースとして定義することを推奨します。
これにより、ユーザプログラムによるプロパティの変更を防ぐことができます。

アクセスコンテキストはキャッシュに格納するなどシリアライズが必要となるため、java.io.Serializable インタフェースを継承しています。
シリアライズ可能なオブジェクトとなるように実装してください。

コンテキストビルダ

コンテキストビルダとは、アクセスコンテキストを生成するための処理クラスです。
以下のライフサイクル操作時に、新しいアクセスコンテキストの生成を行います。

- ライフサイクルの開始
- ライフサイクルの切替

コンテキストビルダの定義は、設定ファイルに記述されます。
対象となるアクセスコンテキストごとに、必要なライフサイクル操作の数だけ定義されます。
コンテキストビルダを利用するための設定方法については、「 *コンテキストビルダの設定* 」を参照してください。

ライフサイクルの操作時に、リソースIDを基に実行対象のコンテキストビルダが選出され、実行されます。

コンテキストビルダの選出は以下の順に行われます。

- ライフサイクルの開始時
  1. 設定ファイルの「ターゲット」と実行時に指定された「リソースID」が一致するコンテキストビルダ
- ライフサイクルの切替時
  1. 設定ファイルの「ターゲット」と実行時に指定された「リソースID」が一致するコンテキストビルダ
  2. デフォルトコンテキストビルダ

デフォルトコンテキストビルダについては、「 *デフォルトコンテキストビルダの設定* 」を参照してください。

コンテキストビルダのインタフェース

コンテキストビルダは、以下のインタフェースを実装します。

jp.co.intra_mart.foundation.context.core.ContextBuilder

build() メソッドを実装して、アクセスコンテキストの生成を行います。
build() メソッドでは、ライフサイクル操作の実行時に渡された環境情報を参照することができます。

実装例

```
public class SampleContextBuilder implements ContextBuilder {

  @Override
  public Context build(final Resource resource) {
    // 環境情報を参照して、新しいアクセスコンテキストを生成する。
    final SampleInfo sampleInfo = SampleInfo.class.cast(resource.getResource());
    final SampleContext newContext = new SampleContext();
    newContext.setFoo(sampleInfo.getFoo());
    return newContext;
  }
}
```

build() メソッドの引数 resource には、実行時の「*環境情報（Resource）*」が設定されます。
リソース情報やリソース属性を利用して、コンテキストの生成を行います。

コンテキストビルダの抽象クラス

ContextBuilder インタフェースを実装する場合、 build() メソッド以外にも、以下のメソッドの実装が必要です。

- init() : 設定ファイルの情報を参照して初期化を行います。
- getBuilderInfo() : 設定ファイルの情報を返却します。
- getContextType() : 生成対象のコンテキスト種別を返却します。

これらのメソッドは、設定ファイルの情報を参照する機能であり、通常はコンテキストビルダごとに処理を変更する必要がありません。

そのため、新しくコンテキストビルダを作成する場合は、これらのメソッドをあらかじめ実装した、以下の基底クラスが利用可能です。

jp.co.intra_mart.foundation.context.core.ContextBuilderSupport

このクラスを継承してコンテキストビルダを作成した場合は、create() メソッドを実装してください。

実装例

```
public class SampleContextBuilder extends ContextBuilderSupport {

  @Override
  protected Context create(final Resource resource) {
    // 環境情報を参照して、新しいアクセスコンテキストを生成する。
    final SampleInfo sampleInfo = SampleInfo.class.cast(resource.getResource());
    final SampleContext newContext = new SampleContext();
    newContext.setFoo(sampleInfo.getFoo());
    return newContext;
  }
}
```

キャッシュサポート

Web実行環境では、アクセスコンテキストのキャッシュが利用されます。
Web実行環境用のコンテキストビルダを実装するためには、以下の基底クラスを継承して実装してください。
キャッシュに関する処理は基底クラスが行うため、実装が必要なメソッドはありません。

jp.co.intra_mart.system.context.core.cache.CachingContextBuilderSupport

実装例

```
import jp.co.intra_mart.foundation.context.core.Resource;
import jp.co.intra_mart.foundation.context.model.Context;
import jp.co.intra_mart.system.context.core.cache.CachingContextBuilderSupport;
import test.SampleContext;
import test.SampleInfo;

public class SampleCachingContextBuilder extends CachingContextBuilderSupport {

    @Override
    protected Context create(final Resource resource) {
        // 環境情報を参照して、新しいアクセスコンテキストを生成する。
        // キャッシュに関する処理は、CachingContextBuilderSupport が行うため、特に必要な対応はありません。
        final SampleInfo sampleInfo = SampleInfo.class.cast(resource.getResource());
        final SampleContext newContext = new SampleContext();
        newContext.setFoo(sampleInfo.getFoo());
        return newContext;
    }
}
```

キャッシュ処理の詳細については、「 *キャッシュ* 」を参照してください。

コンテキストデコレータ

コンテキストデコレータは、コンテキストビルダで作成したアクセスコンテキストをさらに拡張する場合に利用されます。
モジュールの追加により、アクセスコンテキストの生成処理をカスタマイズしたい場合などに利用されます。

コンテキストデコレータの実行は、前述した基底クラス `ContextBuilderSupport` が行います。
そのため、コンテキストデコレータを利用するためには、 `ContextBuilderSupport` を継承したコンテキストビルダを作成する必要があります。

コンテキストデコレータを利用するための設定については、「 *コンテキストデコレータの設定* 」を参照してください。

コンテキストデコレータのインタフェース

コンテキストデコレータは、以下のインタフェースを実装します。

`jp.co.intra_mart.foundation.context.core.ContextDecorator`

`decorate()` メソッドを実装して、拡張元のアクセスコンテキストを基にアクセスコンテキストの生成を行います。
`decorate()` メソッドでは、ライフサイクル操作の実行時に渡された環境情報を参照することができます。

実装例

```
public class SampleContextDecorator implements ContextDecorator {

    @Override
    public Context decorate(final Context context final Resource resource) {
        // 拡張元アクセスコンテキストを参照して、新しいアクセスコンテキストを生成する。
        final SampleContext prevContext = SampleContext.class.cast(context);
        final SampleContext newContext = new SampleContext();
        newContext.setFoo(prevContext.getFoo());
        return newContext;
    }
}
```

`decorate()` メソッドの引数 `resource` には、実行時の「 *環境情報（Resource）* 」が設定されます。
リソース情報やリソース属性を利用して、コンテキストの生成を行います。

コンテキストデコレータの抽象クラス

ContextDecorator インタフェースを実装する場合、 decorate() メソッド以外にも、以下のメソッドの実装が必要です。

- init() : 設定ファイルの情報を参照して初期化を行います。
- getDecoratorInfo() : 設定ファイルの情報を返却します。

これらのメソッドは、設定ファイルの情報を参照する機能であり、通常はコンテキストデコレータごとに処理を変更する必要がありません。
そのため、新しくコンテキストデコレータを作成する場合は、これらのメソッドをあらかじめ実装した、以下の基底クラスが利用可能です。

jp.co.intra_mart.foundation.context.core.ContextDecoratorSupport

実装例

```
public class SampleContextDecorator extends ContextDecoratorSupport {

  @Override
  public Context decorate(final Context context final Resource resource) {
    // 拡張元アクセスコンテキストを参照して、新しいアクセスコンテキストを生成する。
    final SampleContext prevContext = SampleContext.class.cast(context);
    final SampleContext newContext = new SampleContext();
    newContext.setFoo(prevContext.getFoo());
    return newContext;
  }
}
```

ライフサイクルの実行

ライフサイクルの開始・終了処理を実行するためには、 Lifecycle API を利用します。

- Lifecycle#begin(Resource) : ライフサイクルの開始
- Lifecycle#end() : ライフサイクルの終了

ただし、標準の実行環境のライフサイクルの開始・終了処理はあらかじめ実装済みのため、開発者がこれらのメソッドを利用する必要はありません。

Lifecycle API の利用方法は、「Lifecycle クラスの APIドキュメント」を参照してください。

アクセスコンテキスト設定

ここでは、アクセスコンテキストを利用するための設定方法を説明します。

項目

概要

intra-mart Accel Platform で利用するアクセスコンテキストの情報と実行環境は、アクセスコンテキスト設定ファイルで定義します。
ここでは、アクセスコンテキスト設定ファイルの詳細を説明します。

フォーマットファイル(xsd)	WEB-INF/schema/context-config.xsd
設定場所	WEB-INF/conf/context-config/{任意のファイル名}.xml

設定例

```
<?xml version="1.0"?>
<context-config
   xmlns="http://intra-mart.co.jp/foundation/context/context-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://intra-mart.co.jp/foundation/context/context-config ../../schema/context-config.xsd">

   <!-- アカウントコンテキスト -->
   <context name="jp.co.intra_mart.foundation.context.model.AccountContext"
         depends="jp.co.intra_mart.foundation.context.model.ClientContext">

      <!-- システム実行環境用コンテキストビルダ -->
      <builder target="platform">
         <builder-class>jp.co.intra_mart.system.context.impl.GenericContextBuilder</builder-class>
         <init-param>
            <param-key>context-class</param-key>
            <param-value>jp.co.intra_mart.system.admin.context.BasicAccountContext</param-value>
         </init-param>
      </builder>

   </context>

</context-config>
```

アクセスコンテキストの設定

- コンテキスト種別の設定

  アクセスコンテキストを利用するには、context タグの name 属性に、コンテキスト種別を指定します。

  ```
  <context name="sample.SampleContext"
        depends="jp.co.intra_mart.foundation.context.model.AccountContext
  jp.co.intra_mart.foundation.user_context.model.UserContext">

    :

  </context>
  ```

  context タグ中に、「*コンテキストビルダの設定*」を行うことで、実行環境が定義されます。

- 依存先アクセスコンテキストの設定

  アクセスコンテキストには、依存関係を設定することが可能です。
  依存先のアクセスコンテキストを指定することで、依存先の生成処理が完了してから、自身の生成処理を行うように順序制御することが可能です。

  依存関係を設定するためには、context タグの depends 属性に依存先のアクセスコンテキストのコンテキスト種別を指定します。
  依存先のアクセスコンテキストが複数ある場合は、「␣（スペース）」区切りで指定します。

  上記の記述例では、 SampleContext は AccountContext と ClientContext に依存するよう設定しています。

コンテキストビルダの設定

コンテキストビルダの設定には、以下の役割があります。

- コンテキストビルダが実行される実行環境を定義する。
- コンテキストビルダの実行クラスを定義する。

コンテキストビルダは「ターゲット」を持ち、設定ファイル内の target 属性に指定します。
「ターゲット」は、コンテキストビルダがどのライフサイクル操作処理で実行されるかを定義する値です。
「ターゲット」には、ライフサイクル操作処理の実行時に指定するリソースIDを設定します。

設定方法

- 基本設定

  コンテキストビルダを利用するには、context タグ内に builder タグを指定します。

```
<builder target="sample.resource.id">
  <!-- コンテキストビルダの実装クラスを指定します。 -->
  <builder-class>sample.SampleContextBuilder</builder-class>
  <!-- 初期パラメータを必要な数だけ指定します。 -->
  <init-param>
    <param-key>sample-param</param-key>
    <param-value>test</param-value>
  </init-param>
</builder>
```

  builder タグの target 属性に、ターゲットとなるリソースIDを指定します。
  リソースIDは「␣（スペース）」区切りで複数指定することが可能です。

  builder タグ内の builder-class タグには、コンテキストビルダの実装クラスを指定します。

  新しいアクセスコンテキストを作成した場合は、必要なリソースIDの数だけコンテキストビルダの設定を行います。
  必要なリソースID は、以下の通りです。

  - 対象とする実行環境の開始処理のリソースID
  - 対象とする切替処理のリソースID

  「 *リソースID一覧* 」を参照して、必要なリソースIDを確認してください。
  新しい実行環境や切替処理を定義する場合は、新しいリソースIDを定義して設定してください。

- 初期パラメータの設定

  コンテキストビルダで利用するパラメータを init-param タグで指定することが可能です。
  設定した初期パラメータは、 ContextBuilder インタフェースの init() メソッドを実装して取得します。

  実装例

```
@Override
public void init(final String contextType, final BuilderInfo builderInfo) {

  // コンテキストビルダ設定情報
  this.builderInfo = builderInfo;

  // 初期パラメータの取得
  List<InitParam> params = this.builderInfo.getInitParam();
}
```

  また、基底クラス ContextBuilderSupport を継承した場合は、継承クラス内で以下のメソッドを利用して取得することが可能です。

  getInitParameter(String key)

実行環境の定義

コンテキストビルダのターゲットに実行環境開始処理用のリソースIDを指定することで、実行環境とその実行環境で利用可能なアクセスコンテキストが有効になります。

intra-mart Accel Platform が提供する実行環境を表すリソースIDは、あらかじめ定義されています。
「 *リソースID一覧* 」を参照して、必要な実行環境のリソースIDを設定ファイルに指定してください。

デフォルトコンテキストビルダの設定

デフォルトの切替処理用コンテキストビルダの設定方法を説明します。

デフォルトコンテキストビルダとは

切替処理は、特定のアクセスコンテキストだけではなく、現在の実行環境に存在する全てのアクセスコンテキストを対象として実行されます。
そのため、切替処理を追加したい場合は、定義されている全てのアクセスコンテキストに対して切替処理を実装する必要があります。
しかし、アクセスコンテキストはモジュールとして自由に追加されるため、切替処理を定義した実装者が、別のモジュールとして追加された未知のアクセスコンテキストに対して切替処理を実装することはできません。

このような場合に対応するため、アクセスコンテキストの実装者は、あらかじめデフォルトの切替処理を定義することができます。
切替処理実行時に、設定ファイルに未定義のリソースIDが指定された場合は、デフォルトコンテキストビルダを利用して切替処理が実行されます。

例えば、 `Context A` に依存する `Context B` というアクセスコンテキストがあるとします。
`Context B` を定義するモジュールでは、 `Context A` の状態をチェックして情報を最新化するような処理を作成し、デフォルトコンテキストビルダとして定義しておきます。
特定のモジュールで `Context A` を切り替えるような新しい切替処理を定義して実行した場合、デフォルトの切替処理が実行され、 `Context B` の状態が最新化されます。

デフォルトコンテキストビルダの設定

デフォルトコンテキストビルダを定義するためには、以下の設定を行う必要があります。

1. デフォルトコンテキストビルダとして動作するコンテキストビルダを定義する。
2. ライフサイクル開始処理用のコンテキストビルダ設定に、デフォルトコンテキストビルダを利用する設定を行う。

ライフサイクルの開始処理用のコンテキストビルダは、`ContextBuilderSupport` クラスを継承している必要があります。
デフォルトコンテキストビルダには、制約はありません。
Web実行環境用の標準アクセスコンテキストは、 デフォルトコンテキストビルダをサポートしています。

以下に設定手順を説明します。
設定は、コンテキストスイッチ、コンテキストスタック、それぞれに対して行います。

1. デフォルトコンテキストビルダとして動作するコンテキストビルダを定義する。
   通常のコンテキストビルダ設定を行い、デフォルトコンテキストビルダのリソースIDを `target` 属性に指定します。
   リソースIDは任意の文字列を指定します。
   実装例

```
<!-- コンテキストスイッチ用デフォルトコンテキストビルダの定義 -->
<builder target="sample.switch.default">
    <builder-class>sample.DefaultSwitchContextBuilder</builder-class>
</builder>

<!-- コンテキストスタック用デフォルトコンテキストビルダの定義 -->
<builder target="sample.stack.default">
    <builder-class>sample.DefaultStackContextBuilder</builder-class>
</builder>
```

2. ライフサイクル開始処理用のコンテキストビルダ設定に、デフォルトコンテキストビルダを利用する設定を行う。

ライフサイクル開始処理用のコンテキストビルダ設定の初期パラメータに、デフォルトコンテキストビルダのリソースIDを指定します。

切替処理	初期パラメータのキー	設定値
コンテキストスイッチ	default-switch-resource-id	コンテキストスイッチ用のデフォルトコンテキストビルダのリソースID
コンテキストスタック	default-stack-resource-id	コンテキストスタック用のデフォルトコンテキストビルダのリソースID

実装例

```
<!-- Web実行環境開始処理用のコンテキストビルダ設定 -->
<builder target="platform.request">
   <builder-class>sample.SampleContextBuilder</builder-class>

   <!-- コンテキストスイッチ用デフォルトコンテキストビルダ設定 -->
   <init-param>
      <param-key>default-switch-resource-id</param-key>
      <param-value>sample.switch.default</param-value>
   </init-param>

   <!-- コンテキストスタック用デフォルトコンテキストビルダ設定 -->
   <init-param>
      <param-key>default-stack-resource-id</param-key>
      <param-value>sample.stack.default</param-value>
   </init-param>

</builder>
```

コンテキストデコレータの設定

- コンテキストデコレータの設定

  コンテキストデコレータを利用するには、builder タグ内に decorator タグを指定します。

```
<builder target="sample.resource.id">
   <!-- ContextBuilderSupport を継承したコンテキストビルダクラスを指定します。 -->
   <builder-class>sample.SampleContextBuilder</builder-class>
   <!-- コンテキストデコレータを必要な数だけ指定します。 -->
   <decorator>
    <!-- コンテキストデコレータの実装クラスを指定します。 -->
    <decorator-class>sample.SampleContextDecorator</decorator-class>
    <!-- 初期パラメータを必要な数だけ指定します。 -->
    <init-param>
       <param-key>decorator-param</param-key>
       <param-value>Sample Value</param-value>
    </init-param>
   </decorator>
</builder>
```

- 初期パラメータの設定

  コンテキストデコレータで利用するパラメータを init-param タグで指定することが可能です。
  設定した初期パラメータは、 ContextDecorator インタフェースの init() メソッドを実装して取得します。

  実装例

```
@Override
public void init(final String contextType, final DecoratorInfo decoratorInfo) {

   // コンテキストデコレータ設定情報
   this.decoratorInfo = decoratorInfo;

   // 初期パラメータの取得
   List<InitParam> params = this.decoratorInfo.getInitParam();
}
```

また、基底クラス ContextDecoratorSupport を継承した場合は、継承クラス内で以下のメソッドを利用して取得することが可能です。

getInitParameter(String key)

切替

ここでは、アクセスコンテキストの切替機能の詳細と実装方法について説明します。

項目

概要

切替処理は、アクセスコンテキストの情報を変更する場合に実行されます。

切替の処理イメージについては、「 *切替* 」を参照してください。

切替処理を実行すると、アクセスコンテキストの依存関係を考慮して、全てのアクセスコンテキストに対して切替処理が実行されます。
これにより、アクセスコンテキストの変更があった場合に、依存するアクセスコンテキストの情報も変更することが可能です。
切替が不要なアクセスコンテキストは、切替処理は実行されません。

切替機能は、ライフサイクル内でアクセスコンテキストの状態を変更する目的で利用されます。
そのため、ライフサイクルが開始していない状態では利用できませんので、注意してください。

切替処理の実行は、ライフサイクルの開始処理と同様に、コンテキストビルダを利用して行います。

> **注意**
>
> 依存関係があるアクセスコンテキストの場合、依存先と同じ切替（コンテキストスイッチ、コンテキストスタック）をサポートする必要があります。 サポートしていない場合、依存先のアクセスコンテキストの再生成により変更された内容を反映することができません。 この影響により、アクセスコンテキストの情報に不整合が発生する場合があります。

> **注意**
>
> 切替処理は、デフォルトコンテキストビルダで行ってください。 これは、どのようなリソースIDが拡張されても切替が行えるようにするためです。 加えて、デフォルトコンテキストビルダは、実行環境毎に定義してください。
> デフォルトコンテキストビルダについては、「 *デフォルトコンテキストビルダ* 」を参照してください。

コンテキストスイッチ

コンテキストスイッチの処理フロー

コンテキストスイッチの処理は、以下の順に実行されます。

1. 切替処理の呼び出し（ `Lifecycle#switchTo()` ）
2. アクセスコンテキスト一覧取得
3. アクセスコンテキストごとに以下の処理を実行
    1. コンテキストビルダの選出
    2. コンテキストビルダのスイッチ処理実行
    3. コンテキストストアへの格納

コンテキストスイッチ用コンテキストビルダの実装

コンテキストスイッチ用コンテキストビルダは、ライフサイクル開始時と同様に、`ContextBuilder` インタフェースを実装することで作成できます。

`jp.co.intra_mart.foundation.context.core.ContextBuilder`

ただし、そのままではアクセスコンテキストを再作成することになり、スイッチ前の情報を参照することができません。
以下のインタフェースを追加して、 `build()` メソッドの代わりに`switchFrom()` メソッドを実装することでスイッチ前の情報を参照することが可能です。

`jp.co.intra_mart.foundation.context.core.ContextSwitchSupport`

実装例

```
public class SampleContextSwitchBuilder implements ContextBuilder, ContextSwitchSupport {

  @Override
  public boolean enableSwitch(Resource resource) {
    // コンテキストスイッチを利用するため、true を返却する。
    return true;
  }

  @Override
  public Context switchFrom(Context source, Resource resource) {
    // 切り替え前のアクセスコンテキストを参照して、新しいアクセスコンテキストを生成する。
    final SampleContext prevContext = SampleContext.class.cast(source);
    final SampleContext newContext = new SampleContext();
    newContext.setFoo(prevContext.getFoo());
    return newContext;
  }

  @Override
  public Context build(Resource resource) {
    // このメソッドは利用しない。
    return null;
  }
}
```

> **注意**
>
> キャッシュをサポートするアクセスコンテキストは、Web実行環境で利用するコンテキストスイッチ処理を作成する場合、必ずキャッシュをサポートする必要があります。
>
> セッションにキャッシュを保持しつつ、コンテキストスイッチ処理でキャッシュの更新を行わなかった場合、次回アクセス時はセッションの情報を利用するため、コンテキストスイッチが行われない状態となります。
>
> また、依存するアクセスコンテキストだけがキャッシュの更新を行った場合、依存関係にあるはずのアクセスコンテキストの内容が整合性のとれないものとなります。
>
> 例えば、AccountContext のロケール切り替えのスイッチ処理をキャッシュをサポートせずに作成した場合、AccountContext は、次回アクセス時にロケールが元に戻りますが、 UserContext はロケールが変更された状態がキャッシュされるため、ユーザ名が変更されたロケールで表示されます。
>
> キャッシュをサポートするためには、「 *キャッシュ実装* 」に沿って実装してください。

キャッシュサポート

Web実行環境用の切替処理では、キャッシュの対応が必要です。
キャッシュ処理の詳細については、「 *キャッシュ* 」を参照してください。

アクセスコンテキスト設定

コンテキストスイッチを利用するためには、作成したコンテキストビルダを設定ファイルに定義する必要があります。
コンテキストビルダの設定については、「 *コンテキストビルダの設定* 」を参照してください。

コンテキストビルダのリソースIDは、以下を考慮して定義してください。

- 新しいコンテキストスイッチ処理の場合、新しいリソースIDを定義する。
- 「 *リソースID一覧* 」に定義された処理で利用するコンテキストビルダの場合、対象のリソースIDを定義する。
- デフォルトコンテキストビルダとして定義する場合、「 *リソースID一覧* 」のデフォルト用のリソースIDを定義する。

コンテキストスタック

コンテキストスタックの処理フロー

コンテキストスタックの処理は、以下の順に実行されます。

1. 切替処理の呼び出し（ `Lifecycle#stack()` ）
2. 現在のアクセスコンテキストの保存
3. アクセスコンテキスト一覧取得
4. アクセスコンテキストごとに以下の処理を実行
    1. コンテキストビルダの選出
    2. コンテキストビルダのスタック処理実行
    3. コンテキストストアへの格納
    4. （任意の処理実行）
    5. コンテキストストアの破棄とスタック前のコンテキストストアの復元（ `Lifecycle#pop()` ）

コンテキストスタック用コンテキストビルダの実装

コンテキストスタック用コンテキストビルダは、ライフサイクル開始時と同様に、`ContextBuilder` インタフェースを実装することで作成できます。

`jp.co.intra_mart.foundation.context.core.ContextBuilder`

ただし、そのままではアクセスコンテキストを再作成することになり、スタック前の情報を参照することができません。
以下のインタフェースを追加して、 `build()` メソッドの代わりに`push()` メソッドを実装することでスタック前の情報を参照することが可能です。

`jp.co.intra_mart.foundation.context.core.ContextStackSupport`

実装例

```
public class SampleContextStackBuilder implements ContextBuilder, ContextStackSupport {

    @Override
    public boolean enableStack(final Resource resource) {
        // コンテキストスタックを利用するため、true を返却する。
        return true;
    }

    @Override
    public Context push(final Context source, final Resource resource) {
        // 切り替え前のアクセスコンテキストを参照して、新しいアクセスコンテキストを生成する。
        final SampleContext prevContext = SampleContext.class.cast(source);
        final SampleContext newContext = new SampleContext();
        newContext.setFoo(prevContext.getFoo());
        return newContext;
    }

    @Override
    public Context pop(final Context source) {
        // スタックから取得したアクセスコンテキストをそのまま返却する。
        return source;
    }

    @Override
    public Context build(final Resource resource) {
        // このメソッドは利用しない。
        return null;
    }
}
```

キャッシュサポート

コンテキストスタックは、一時的な処理のためキャッシュに対する処理は行いません。
そのため、コンテキストスタック用のコンテキストビルダでは、キャッシュ対応は必要ありません。

アクセスコンテキスト設定

コンテキストスタックを利用するためには、作成したコンテキストビルダを設定ファイルに定義する必要があります。
コンテキストビルダの設定については、「 *コンテキストビルダの設定* 」を参照してください。

コンテキストビルダのリソースIDは、以下を考慮して定義してください。

- 新しいコンテキストスタック処理の場合、新しいリソースIDを定義する。
- 「 *リソースID一覧* 」に定義された処理で利用する場合、対象のリソースIDを定義する。
- デフォルトコンテキストビルダとして定義する場合、「 *リソースID一覧* 」のデフォルト用のリソースIDを定義する。

実行方法（JavaEE開発モデル）

コンテキスト切替処理を実行するためには、 `Lifecycle` API を利用します。

`Lifecycle` API の詳細は、「Lifecycle クラスの APIドキュメント 」を参照してください。

`Lifecycle` は、インタフェースです。実装クラスを取得するためには、 `LifecycleFactory` クラスを利用します。

`jp.co.intra_mart.foundation.context.core.LifecycleFactory`

実行方法

```
final Lifecycle lifecycle = LifecycleFactory.getLifecycle();
```

コンテキストスイッチ

コンテキストスイッチを実行するためには、以下のメソッドを実行します。

- `Lifecycle#switchTo(Resource)`

コンテキストスタック

コンテキストスタックを実行するためには、以下のメソッドを実行します。

- 処理の開始時： `Lifecycle#stack(Resource)`
- 処理の完了時： `Lifecycle#pop()`

実行方法

```
final Resource resource = new Resource("sample.resource.id");
final Lifecycle lifecycle = LifecycleFactory.getLifecycle();
try {

   // スタックして処理を実行する。
   lifecycle.stack(resource);

   // スタック中のアクセスコンテキストを参照。
   final SampleContext currentContext = Contexts.get(SampleContext.class);

} finally {
   // 処理の終了後、 pop() を実行すること。
   lifecycle.pop();
}
```

スタック中の処理の範囲が特定できる場合、クロージャ対応のメソッドを実行することで、 `pop()` の呼び出しを省略することが可能です。

- `Lifecycle#stack(Resource, LifecycleStackProcedure)`

  `LifecycleStackProcedure` は、スタック中の処理を定義したクロージャクラスのインタフェースです。

実行方法

```
// スタックして処理を実行する。
final Resource resource = new Resource("sample.resource.id");
LifecycleFactory.getLifecycle().stack(resource, new LifecycleStackProcedure<Boolean, Exception>() {

   @Override
   public Boolean process() throws Exception {

      // スタック中のアクセスコンテキストを参照。
      final SampleContext currentContext = Contexts.get(SampleContext.class);

      return true;
   }

});

// 処理の完了後は pop() 実行済み。
```

注意

- クロージャ対応のメソッドを利用できるのは、 2014 Spring(Granada) 以降です。
- クロージャ対応のメソッドを利用しない場合、処理の完了時には必ず `pop()` を実行する必要があります。

実行方法（スクリプト開発モデル）

コンテキスト切替処理を実行するためには、 Lifecycle API を利用します。

Lifecycle API の詳細は、「Lifecycle オブジェクトの APIドキュメント」を参照してください。

コンテキストスイッチ

コンテキストスイッチを実行するためには、以下の関数を実行します。

- Lifecycle#switchTo(String, Object)

コンテキストスタック

コンテキストスタックを実行するためには、 Lifecycle API を利用します。

コンテキストスタックを実行するためには、以下の関数を実行します。

- 処理の開始時： Lifecycle#stack(String, Object)
- 処理の完了時： Lifecycle#pop()

> **注意**
>
> 処理の完了時には必ず pop() を実行する必要があります。

キャッシュ

ここでは、Web実行環境におけるHTTPセッションを利用したアクセスコンテキストのキャッシュ機構について説明します。

項目

概要

アクセスコンテキストのライフサイクルは、 Lifecycle#begin() から Lifecycle#end() が呼び出されるまでです。
そのため、Web実行環境では 1 つのHTTPリクエストに対する処理が終了するとアクセスコンテキストは破棄されます。

一般的なWebアプリケーションにおけるユーザ情報は、ログインによりHTTPセッションに格納され、ログアウトするまで有効となります。
intra-mart Accel Platform では上記を実現するために、ユーザ情報を格納しているアクセスコンテキストをHTTPセッションにキャッシュすることで、ユーザ情報をHTTPセッションと同じスコープで保持する仕組みを提供します。

キャッシュを利用する設定は、アクセスコンテキストごとに行います。
キャッシュを利用しなかった場合、リクエストごとにアクセスコンテキストが生成されることになります。
そのため、ログイン後に生成した情報を引き継ぐことができず、毎回初期状態のアクセスコンテキストが生成されてしまいます。
また、毎回アクセスコンテキストが生成されることにより、パフォーマンスの低下を招きます。

コンテキストスイッチ実行時は、キャッシュは破棄されます。
ただし、コンテキストスタック実行中にコンテキストスイッチを実行された場合は、キャッシュの破棄、および、登録は行われ

ません。
コンテキストスイッチ前のアクセスコンテキストを引き継ぐ場合、コンテキストビルダにより適切な処理が必要です。

コラム

この機能は、アクセスコンテキストをキャッシュするための仕組みです。
頻繁にアクセスする情報をキャッシュするための機能は、 intra-mart Accel Platform の 「Cacheサービス」を利用してください。

「Cacheサービス」については、以下のドキュメントを参照してください。

- 「 SAStruts+S2JDBC プログラミングガイド 」-「 Cacheサービス 」
- 「 スクリプト開発モデル プログラミングガイド 」-「 Cacheサービス 」
- 「 TERASOLUNA Server Framework for Java (5.x) プログラミングガイド」-「 Cacheサービス 」

キャッシュの処理フロー

キャッシュ処理の流れを、以下の順に説明します。

1. ライフサイクル開始時の、キャッシュの取得・更新
2. コンテキストスイッチによるキャッシュの更新

キャッシュの取得・更新処理フロー

キャッシュ利用時は、ライフサイクル開始処理でキャッシュを利用するための処理が実行されます。
ライフサイクル開始時の、キャッシュの取得・更新処理の流れは、以下の通りです。

1. **コンテキスト生成開始**
2. **キャッシュ取得**
   キャッシュからアクセスコンテキストを取得します。
   - キャッシュからアクセスコンテキストが取得できた場合、「3. キャッシュタイムアウト判定」を行います。
   - キャッシュからアクセスコンテキストが取得できなかった場合、「5. アクセスコンテキスト生成」を行います。
3. **キャッシュタイムアウト判定**
   設定されたキャッシュタイムアウト判定方式に従って、キャッシュタイムアウトの判定を行います。
   - キャッシュタイムアウト判定によりキャッシュ有効となった場合、キャッシュを利用してライフサイクルを開始します。
   - キャッシュタイムアウト判定によりキャッシュ無効となった場合、「4. キャッシュ削除」を行います。
4. **キャッシュ削除**
   キャッシュ無効となった場合、キャッシュからアクセスコンテキストを削除します。
5. **アクセスコンテキスト生成**
   新しいアクセスコンテキストを生成します。
   この場合、キャッシュタイムアウト前のアクセスコンテキストを参照して生成することができます。
6. **キャッシュ保存**
   新しいアクセスコンテキストをキャッシュに保存します。

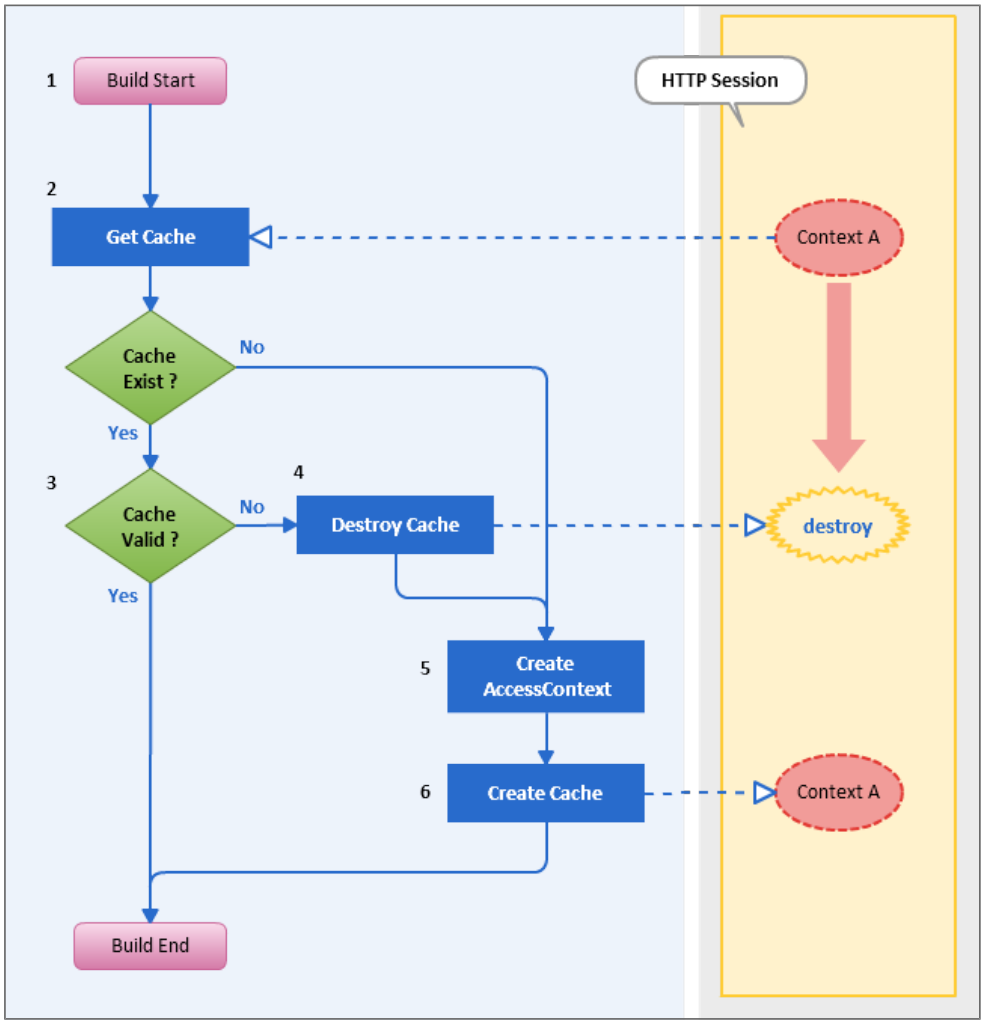

図 キャッシュの取得・更新処理フロー

コンテキストスイッチによるキャッシュの更新処理フロー

キャッシュ利用時は、コンテキストスイッチ処理でキャッシュの破棄と生成が実行されます。
コンテキストスイッチによるキャッシュの更新処理の流れは、以下の通りです。

1. **コンテキストスイッチ開始**
2. **キャッシュ削除**
   キャッシュからアクセスコンテキストを削除します。
3. **コンテキストスイッチ**
   コンテキストスイッチを実行して、新しいアクセスコンテキストを生成します。
4. **キャッシュ保存**
   新しいアクセスコンテキストをキャッシュに保存します。

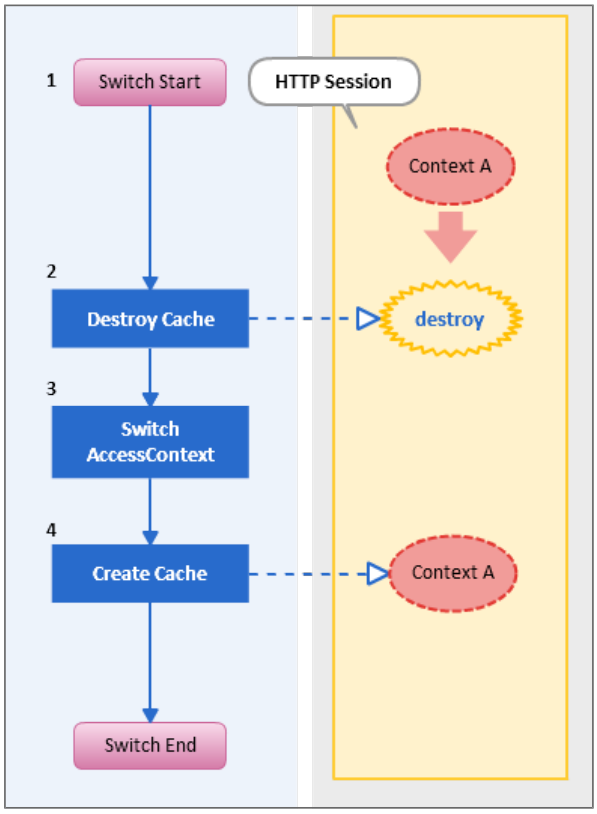

図 コンテキストスイッチによるキャッシュの更新処理フロー

キャッシュ実装

キャッシュを利用するためには、以下の対応が必要です。

- ライフサイクル開始処理では、CachingContextBuilderSupport クラスを継承したコンテキストビルダ利用する。

  Web実行環境用の標準アクセスコンテキストは、キャッシュをサポートしています。

- コンテキストスイッチ処理では、StandardSwitchableContextBuilder クラスを利用する。

  完全修飾クラス名（FQCN）

  jp.co.intra_mart.system.context.standard.StandardSwitchableContextBuilder

  StandardSwitchableContextBuilder クラスは、コンテキストスイッチでキャッシュをサポートするためのコンテキストビルダで、アクセスコンテキストの生成処理は行いません。
  新しいアクセスコンテキストを生成するためには、コンテキストデコレータを利用します。

  設定例

```
<builder target="sample.resource.id">
   <!-- キャッシュをサポートするコンテキストスイッチ用コンテキストビルダ -->
   <builder-class>jp.co.intra_mart.system.context.standard.StandardSwitchableContextBuilder</builder-class>
   <!-- コンテキストデコレータを必要な数だけ指定します。 -->
   <decorator>
    <!-- コンテキストスイッチで新しいアクセスコンテキストを生成するための、
         コンテキストデコレータの実装クラスを指定します。 -->
    <decorator-class>sample.SampleContextSwitchDecorator</decorator-class>
   </decorator>
</builder>
```

キャッシュ設定

キャッシュを利用する場合は、のコンテキストビルダは、CachingContextBuilderSupport クラスを継承している必要があり

ます。
デフォルトコンテキストビルダには、制約はありません。
Web実行環境用の標準アクセスコンテキストは、 デフォルトコンテキストビルダをサポートしています。

キャッシュを利用する場合、アクセスコンテキスト設定ファイルに設定を行います。
キャッシュの設定は実行環境開始時に決定するため、実行環境開始用のコンテキストビルダの初期パラメータに設定します。

初期パラメータのキー	cache-policy
初期パラメータの値	「 *キャッシュタイムアウト判定方式一覧* 」の設定値

Web実行環境の開始用コンテキストビルダのリソースIDは、「 platform.request 」ですので、対象のコンテキストビルダ設定に以下のように設定します。

```
<context name="sample.SampleContext">
  <builder target="platform.request">
    <builder-class>sample.SampleContextBuilder</builder-class>
    <!-- キャッシュの利用とキャッシュタイムアウト判定方式の設定 -->
    <init-param>
      <param-key>cache-policy</param-key>
      <param-value>session-user-daily</param-value>
    </init-param>
 </builder>
</context>
```

初期パラメータの値に設定する値は、「 *キャッシュタイムアウト判定* 」を指定するための設定になります。
「 *キャッシュタイムアウト判定方式一覧* 」から、適切な値を設定してください。

キャッシュタイムアウト判定

Web実行環境のアクセス時には、リクエストごとにキャッシュタイムアウトの判定が行われます。
これにより、一定期間経過後にアクセスコンテキストの情報を更新することが可能です。
ただし、コンテキストスイッチ処理では、常にキャッシュを破棄して新しいアクセスコンテキストが保存されます。

キャッシュタイムアウト判定方式一覧

以下のキャッシュタイムアウト判定方式が利用できます。

表 キャッシュタイムアウト判定方式一覧

設定値	判定方式
session-daily	1日が経過したら、キャッシュ無効とします。 1日の経過とは、システムのタイムゾーンでの時刻が 「00:00:00」 になった時点を指します。
session-user-daily	1日が経過したら、キャッシュ無効とします。 session-daily とは異なり、ユーザのタイムゾーンでの日時を利用します。 1日の経過とは、ユーザのタイムゾーンでの時刻が 「00:00:00」 になった時点を指します。
session-interval	指定時間が経過したら、キャッシュ無効とします。 指定時間は設定ファイルの初期パラメータのキー「 cache-interval 」に時間（分）を指定します。 指定時間の基準は、アクセスコンテキスト生成時（初回アクセス時、またはスイッチ時）からです。
session-infinite	常にキャッシュ有効とします。 コンテキストスイッチ実行時のみキャッシュが破棄されます。

標準アクセスコンテキストでは、以下のキャッシュタイムアウト判定方式が設定されています。

表 標準アクセスコンテキストのキャッシュタイムアウト判定方式一覧

コンテキスト種別	キャッシュタイムアウト判定方式
アカウントコンテキスト	session-user-daily
クライアントコンテキスト	session-infinite
ユーザコンテキスト	session-user-daily
認可サブジェクトコンテキスト	session-user-daily
ジョブスケジューラコンテキスト	Web実行環境で利用しないため、設定なし。

注意

依存関係があるアクセスコンテキストの場合、依存先と同じキャッシュ設定をする必要があります。
依存関係があるアクセスコンテキストのキャッシュタイムアウト判定方式が異なる場合、キャッシュタイムアウトのタイミングが異なるため、アクセスコンテキストの情報に不整合が発生する場合があります。

標準アクセスコンテキスト について

ここでは、 intra-mart Accel Platform が提供する、主な標準アクセスコンテキストの詳細について説明します。

アカウントコンテキスト

項目

概要

アカウントコンテキストとは、アカウントに関する情報を保持するアクセスコンテキストです。
ユーザコードやロケールなどのアカウント情報、認証状況などを取得できます。
intra-mart Accel Platform 上で常に利用可能です。

アカウントコンテキストを利用する場合、`AccountContext` の APIドキュメントを参照してください。

- JavaEE開発のAPIドキュメント
- スクリプト開発のAPIドキュメント

プロパティ

アカウントコンテキストのプロパティは以下の通りです。

表　アカウントコンテキストのプロパティ一覧

プロパティ名	説明

プロパティ名	説明
テナントID	現在の操作対象のテナントIDです。 アクセスしているユーザに応じたテナントIDが取得されます。 システム起動時などのユーザに依存しない処理の場合は、null が取得されます。
ユーザ種別	現在アクセスしているユーザの種別を取得します。 ユーザ種別は、以下の種類があります。 ■ 一般ユーザ ログインユーザおよび未認証ユーザです。 ■ システム管理者 ■ プラットフォームユーザ システムで利用するユーザです。 ジョブの実行ユーザは、プラットフォームユーザです。
ユーザコード	現在アクセスしているユーザのユーザコードです。 ログインしていない場合や、プラットフォームユーザの場合は、システムで定義された固定のユーザコードが使用されます。
認証状況	ログインしているかどうかを表します。 プラットフォームユーザはログインを行わないため、未認証です。
ロケール	現在アクセスしているユーザが利用するロケールです。
文字エンコーディング	現在アクセスしているユーザが利用する文字エンコーディングです。 UTF-8 のみをサポートするため、常に UTF-8 が設定されます。
タイムゾーン	現在アクセスしているユーザが利用するタイムゾーンです。
日時表示形式一覧	現在アクセスしているユーザが利用する日時表示形式情報一覧です。
週の開始曜日	現在アクセスしているユーザが利用する週の開始曜日情報です。
カレンダーID	現在アクセスしているユーザが利用するカレンダーのカレンダーIDです。
テーマID	現在アクセスしているユーザが利用するテーマのテーマIDです。
ホームURL	現在アクセスしているユーザが利用するホームURLです。
ログイン時刻	現在アクセスしているユーザがログインした時刻です。 ログインしていない場合は、null が取得されます。
ログイン署名	現在アクセスしているユーザのログイン署名情報です。 ログインしていない場合は、null が取得されます。
ロールID一覧	現在アクセスしているユーザのロールID一覧です。サブロールも含まれます。 ログインしていない場合、または、システム管理者の場合は、null が取得されます。
アプリケーションライセンス一覧	現在アクセスしているユーザが保持するアプリケーションライセンス一覧です。 ログインしていない場合、または、システム管理者の場合は、null が取得されます。
数値形式のフォーマットID	現在アクセスしているユーザが利用する数値形式のフォーマットIDです。

注意

- 「テナントID」は intra-mart Accel Platform 2014 Spring(Granada) 以降で利用可能です。
- 「数値形式のフォーマットID」は intra-mart Accel Platform 2016 Winter(Olga) 以降で利用可能です。

テナントIDについて

テナントIDは、 バーチャルテナントによる複数テナント を利用している場合の処理対象のテナントIDが設定されます。

この値は、「 テナント解決機能 」により、自動的に解決されます。
また、intra-mart Accel Platform が提供するAPIの内部で利用されるため、開発者が直接利用する必要はありません。

ステータス遷移

アカウントコンテキストはアクセスしているユーザの状態（ステータス）を保持するアクセスコンテキストで、特定の処理によってステータスが変更されます。

アカウントコンテキストのステータス変更は「 *切替機能* 」を利用して実現されます。

アカウントコンテキストのステータス変更処理は実行環境ごとに定義されますが、標準では、Web実行環境のステータス変更処理のみ定義されています。

Web実行環境のステータス

Web実行環境での初回アクセス時は、必ず未認証ユーザのアカウントコンテキストが作成されます。

標準で定義されているアカウントコンテキストのステータス変更処理は以下の通りです。

- ログイン

  未認証ユーザをログイン状態に変更します。
  ユーザの情報を読み込んで各プロパティが設定されます。
  また、ログイン時刻・ログイン署名が設定されます。

- ログアウト

  ログインユーザを未認証状態に変更します。
  テナント情報から未認証用の情報を読み込んで各プロパティが設定されます。
  また、ログイン時刻・ログイン署名がクリアされます。

- ユーザ情報更新

  ログインユーザのユーザ情報を最新化します。
  アカウントコンテキストの情報は、キャッシュ機構によりHTTPセッションにキャッシュされるため、データベースの情報を更新しただけでは、アカウントコンテキストには反映されません。
  ログイン状態を維持したまま、アカウントコンテキストの情報を最新化するためには、ユーザ情報更新処理を実行する必要があります。

  また、同時にアカウントの有効期限、ライセンス等がチェックされるため、この処理の結果、未認証状態になる場合があります。

アカウントコンテキストのステータス遷移は以下の通りです。

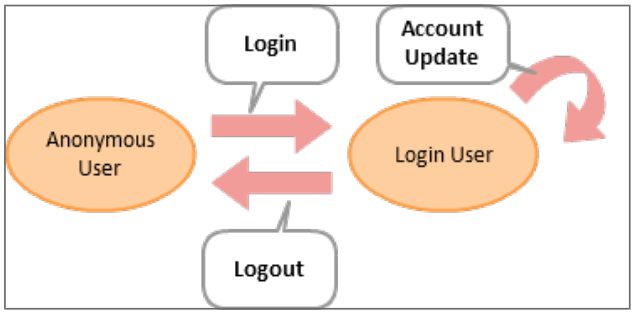

図 アカウントコンテキストのステータス遷移

プロパティの解決順序

アカウントコンテキストの情報は、ユーザの種別によらず定義された解決順序によって適切な値が取得できます。
一般ユーザの場合の、ユーザごとに設定が可能なプロパティの解決順序は、以下の通りです。

図 アカウントコンテキストの各プロパティの解決順序（一般ユーザ）

例えば、一般ユーザがロケールを取得する場合、以下の解決順序によって必ず値が取得できます。

1. アカウント情報に設定された個人設定のロケール
2. ブラウザからアクセスする際に設定されたロケール
3. テナント情報に設定されたテナントのロケール
4. システムに設定されたデフォルトのロケール

この解決順序により、ログインしていない未認証ユーザの場合でも、必ずロケールが取得できます。

> **コラム**
>
> intra-mart Accel Platform 2018 Summer(Tiffany) で、プロパティの解決順序が変更されました。
>
> - 変更前（2018 Spring(Skylark) 以前）
>   1. アカウント設定
>   2. **テナント設定**
>   3. **ブラウザ**
>   4. システム設定
>   5. 環境設定
> - 変更後（2018 Summer(Tiffany) 以降）
>   1. アカウント設定
>   2. **ブラウザ**
>   3. **テナント設定**
>   4. システム設定
>   5. 環境設定
>
> 2018 Summer(Tiffany) 以降で 2018 Spring(Skylark) 以前と同様のプロパティの解決順序を使用したい場合は、設定を変更してください。
> 設定については、「設定ファイルリファレンス」の「ブラウザロケールデコレータ設定」を参照してください。

ジョブ実行環境のステータス

ジョブ実行環境では、ジョブネット起動時にジョブネットの登録情報を利用してアカウントコンテキストが作成されます。

標準で定義されているステータス変更処理はありません。

ジョブネット起動時にアカウントコンテキストに設定される主なプロパティは、以下の通りです。

表 ジョブ実行環境のアカウントコンテキスト

テナントID	ジョブネット起動時のテナントID
ユーザ種別	プラットフォームユーザ
ユーザコード	固定値： im_job
ログイン時刻	ジョブネット起動時刻
ロール、アプリケーションライセンス	null

その他のプロパティは、テナントの情報が設定されます。

ユーザの分類

intra-mart Accel Platform では、アカウントコンテキストの「ユーザ種別」と「認証状況」に基づいてユーザの分類が行われます。

ユーザの分類の判定方法は以下の通りです。

- 未認証ユーザ
  ユーザ種別が「一般ユーザ」、かつ、認証状況が「未認証」。
- ログインユーザ
  ユーザ種別が「一般ユーザ」、かつ、認証状況が「認証済み」。
- システム管理者
  ユーザ種別が「システム管理者」。
  認証状況は、常に「認証済み」です。
- プラットフォームユーザ
  ユーザ種別が「プラットフォーム」。
  認証状況は、常に「未認証」です。

キャッシュタイムアウト

アカウントコンテキストのWeb実行環境でのキャッシュタイムアウト判定の設定は、session-user-daily です。
キャッシュタイムアウト判定については、「 *キャッシュタイムアウト判定* 」を参照してください。

ユーザのタイムゾーンを基準として日付が変わった場合、キャッシュタイムアウトとなり、アカウントコンテキストの情報を最新化します。

このため、セッションタイムアウトを設定せずに運用した場合、ログイン状態のままでも、日付が変わったタイミングでアカウントコンテキストの情報が最新の情報に変更される可能性があります。

また、同時にアカウントの有効期限、ライセンス等がチェックされるため、日付変更後のアクセスにより、未認証状態になる場合があります。

ユーザコンテキスト

概要

ユーザコンテキストとは、ユーザ名や所属組織情報など、IM-共通マスタで保有するユーザプロファイル情報を保持するアクセ

スコンテキストです。
Web実行環境で利用可能です。
現在ユーザが選択している所属組織情報が取得できます。
一時的に所属組織を切り替えた場合、切り替えた所属組織情報が取得できます。

ユーザコンテキストを利用する場合、UserContext の APIドキュメントを参照してください。

- JavaEE開発のAPIドキュメント
- スクリプト開発のAPIドキュメント

プロパティ

ユーザコンテキストが扱う情報は以下の通りです。

表　ユーザコンテキストが扱う情報一覧

プロファイル情報	ログインユーザのプロファイル情報です。 ユーザの表示名、住所やメールアドレス等が取得されます。
会社所属情報	ユーザが所属している会社情報です。 ユーザが所属している全ての会社の情報を取得して利用可能です。
組織所属情報	ユーザが所属している組織情報です。 現在処理対象とする組織（カレント組織）を取得したり、全ての所属組織を取得することが可能です。
組織所属役職情報	ユーザが所属している組織の役職情報です。 カレント組織の役職を取得したり、全ての所属組織の役職を取得することが可能です。
パブリックグループ所属情報	ユーザが所属しているパブリックグループ情報です。
パブリックグループ所属役割情報	ユーザが所属しているパブリックグループの役割情報です。
ユーザ分類情報	ユーザに紐付けられているユーザ分類情報です。

ユーザコンテキストが扱う情報のうち、多言語情報がある場合は、アカウントコンテキストのロケールに該当する情報が取得されます。
対象の情報が取得できなかった場合は、「名称未定義」が取得されます。

> **注意**
> ユーザコンテキストが扱う組織情報（「組織所属情報」、「組織所属役職情報」）は、デフォルト組織セットの情報のみです。

ステータス

ユーザコンテキストの情報は、アカウントコンテキストの情報を参照して作成されます。
そのため、ユーザコンテキストの状態も、「 *アカウントコンテキストのステータス遷移* 」に準じます。

ユーザコンテキストは、ログインユーザの情報を利用して構築されます。
そのため、ログインユーザ以外はデフォルトのユーザコンテキストが作成されます。

デフォルトのユーザコンテキストは、ほとんどの情報が未設定となっています。
また、ユーザ名に未認証ユーザを表す名称「ゲスト」が設定されます。

ユーザコンテキストを利用する場合は、アカウントコンテキストの認証状況を確認して利用してください。

カレント組織の変更

ユーザが処理対象とする所属組織をカレント組織と呼びます。
アプリケーションでユーザの所属組織情報を参照する場合は、通常カレント組織を利用します。

所属組織が複数ある場合、「 *切替機能* 」を利用してカレント組織を変更することが可能です。

- カレント組織変更

  カレント組織をユーザが所属している任意の組織に変更します。
  処理対象の所属組織を変更することで、別の組織の情報と権限を利用して以降の処理を実行可能です。

初期状態のカレント組織は、主所属の組織です。
また、主所属が設定されていない場合は、カレント組織は未設定です。

キャッシュタイムアウト

ユーザコンテキストのWeb実行環境でのキャッシュタイムアウト判定の設定は、`session-user-daily` です。
キャッシュタイムアウト判定については、「 *キャッシュタイムアウト判定* 」を参照してください。

ユーザのタイムゾーンを基準として日付が変わった場合、キャッシュタイムアウトとなり、ユーザコンテキストの情報を最新化します。

このため、セッションタイムアウトを設定せずに運用した場合、ログイン状態のままでも、日付が変わったタイミングでユーザコンテキストの情報が最新の情報に変更される可能性があります。

また、キャッシュタイムアウト時には IM-共通マスタのプロファイル情報の有効期限がチェックされます。
そのため、ログイン状態のままでも、プロファイルの有効期限が切れると、日付が変わったタイミングでプロファイルが設定されなくなります。

外部ユーザコンテキスト

項目

概要

外部ユーザコンテキストとは、外部ユーザに関する情報を保持するアクセスコンテキストです。
Web実行環境で利用可能です。
外部ユーザコンテキストは、intra-mart Accel Platform 2016 Spring(Maxima) 以降のバージョンで利用できます。

外部ユーザコンテキストを利用する場合、`ExternalUserContext` の APIドキュメントを参照してください。

- JavaEE開発のAPIドキュメント
- スクリプト開発のAPIドキュメント

プロパティ

外部ユーザコンテキストが扱う情報は以下の通りです。

表　外部ユーザコンテキストが扱う情報一覧

外部ユーザ	外部ユーザかどうかを表します。

ステータス

外部ユーザコンテキストの情報は、アカウントコンテキストの情報を参照して作成されます。
そのため、外部ユーザコンテキストの状態も、「 *アカウントコンテキストのステータス遷移* 」に準じます。

外部ユーザコンテキストは、ログインユーザの情報を利用して構築されます。
そのため、ログインユーザ以外はデフォルトの外部ユーザコンテキストが作成されます。

キャッシュタイムアウト

外部ユーザコンテキストのWeb実行環境でのキャッシュタイムアウト判定の設定は、`session-user-daily` です。
キャッシュタイムアウト判定については、「 *キャッシュタイムアウト判定* 」を参照してください。

ユーザのタイムゾーンを基準として日付が変わった場合、キャッシュタイムアウトとなり、外部ユーザコンテキストの情報を最新化します。

このため、セッションタイムアウトを設定せずに運用した場合、ログイン状態のままでも、日付が変わったタイミングで外部ユーザコンテキストの情報が最新の情報に変更される可能性があります。

付録

リソースID一覧

intra-mart Accel Platform が提供する主なリソースIDを、実行環境ごとに説明します。

新しいアクセスコンテキストを作成する場合は、この一覧から以下の確認を行ってください。

- 対象とする実行環境の開始処理のリソースIDを確認して、アクセスコンテキストの設定を行う。
- 依存先アクセスコンテキストの切替処理のタイミングを確認して、対応が必要であれば、対象処理のリソースIDを指定してアクセスコンテキストの設定を行う。
  通常はデフォルトコンテキストビルダで切替処理に対応するようにしてください。
  デフォルトコンテキストビルダについては、「*デフォルトコンテキストビルダ*」を参照してください。

> **注意**
>
> この資料は、新しいアクセスコンテキストの実装者のための資料です。
> これらのリソースIDを利用したライフサイクル操作の実行は、基盤機能やAPIが行います。
> これらのリソースIDを指定して、直接ライフサイクル操作を実行することはしないでください。

- システム実行環境

  ＜ライフサイクル開始＞

リソースID	説明
platform	システム実行環境のライフサイクル開始用。

- 共通

  ＜ライフサイクル切替＞

リソースID	対象	種別	説明
platform.switch.default	-	スイッチ	デフォルトコンテキストスイッチ用。 スイッチ前のアクセスコンテキストを元に再読込して、最新化を行います。 2015 Winter(Lydia) で `platform.request.switch.default` から変更されました。 このリソースIDは、デフォルトコンテキストスイッチ処理用のリソースIDです。 直接実行する必要はありません。 また、アクセスコンテキストごとに定義を行うため、必ずデフォルトコンテキストスイッチ用として定義されるわけではありません。

リソースID	対象	種別	説明
platform.stack.default	-	スタック	デフォルトコンテキストスタック用。 スタック前のアクセスコンテキストを元に再読込して、最新化を行います。 2015 Winter(Lydia) で platform.request.stack.default から変更されました。 このリソースIDは、デフォルトコンテキストスタック処理用のリソースIDです。 直接実行する必要はありません。 また、アクセスコンテキストごとに定義を行うため、必ずデフォルトコンテキストスタック用として定義されるわけではありません。
platform.tenant.change	アカウントコンテキスト	スイッチ	テナント切替用。 操作対象のテナントを指定されたテナントに切り替えます。 2014 Spring(Granada) で追加されました。
platform.tenant.change.stack	アカウントコンテキスト	スタック	テナント切替用。 操作対象のテナントを指定されたテナントに切り替えます。 2014 Spring(Granada) で追加されました。
platform.account.stack	アカウントコンテキスト／ユーザコンテキスト	スタック	ユーザ切替用。 操作対象のユーザを指定されたユーザに切り替えます。 2015 Winter(Lydia) で追加されました。

- Web実行環境

＜ライフサイクル開始＞

リソースID	説明
platform.request	Web実行環境のライフサイクル開始用。 未認証ユーザのアカウントコンテキストを作成します。

＜ライフサイクル切替＞

リソースID	対象	種別	説明
platform.login	アカウントコンテキスト	スイッチ	一般ユーザログイン用。 ログイン情報を基に、ログイン状態のアカウントコンテキストを作成します。
platform.logout	アカウントコンテキスト	スイッチ	一般ユーザログアウト用。 未認証ユーザのアカウントコンテキストを作成します。
platform.account.updated	アカウントコンテキスト	スイッチ	一般ユーザの個人設定再読込用。 スイッチ前のアクセスコンテキストを元に再読込して、最新化を行います。

リソースID	対象	種別	説明
platform.initial	アカウントコンテキスト	スイッチ	システム初回アクセス用。 テナント初期設定専用です。 まだシステムが未構築の場合に、テナント初期設定用のシステム管理者に変更します。
platform.admin.login	アカウントコンテキスト	スイッチ	システム管理者ログイン用。 ログイン情報を基に、システム管理者としてログイン状態のアカウントコンテキストを作成します。
platform.admin.logout	アカウントコンテキスト	スイッチ	システム管理者ログアウト用。 未認証ユーザのアカウントコンテキストを作成します。
platform.admin.updated	アカウントコンテキスト	スイッチ	システム管理者の個人設定再読込用。 スイッチ前のシステム管理者のアクセスコンテキストを元に再読込して、最新化を行います。 2014 Spring(Granada) で追加されました。
platform.device.change	クライアントコンテキスト	スイッチ	クライアントタイプ変更用。 指定されたクライアントタイプへの変更を行います。 合わせてアカウントコンテキストのテーマIDが変更されます。 利用する場合は、`ClientTypeSwitcher` API を利用します。
platform.device.change.stack	クライアントコンテキスト	スタック	クライアントタイプ変更用。 指定されたクライアントタイプへの変更を行います。 合わせてアカウントコンテキストのテーマIDが変更されます。 利用する場合は、`ClientTypeSwitcher` API を利用します。
platform.current.department.switch	ユーザコンテキスト	スイッチ	カレント組織変更用。 カレント組織を指定されたユーザの所属組織へ変更します。 2013 Spring(Climbing) で `platform.department.change` から変更されました。

- ジョブ実行環境

＜ライフサイクル開始＞

リソースID	説明
platform.job.execute	ジョブネット実行時用。 ジョブネットの実行開始時に、必要なアクセスコンテキストを作成します。