目次

改訂情報

変更年月日	変更内容
2014-08-01	初版
2015-12-01	第2版 下記を追加・変更しました ・「*アクセスコンテキストの利用*」に、「SAStruts+S2JDBC プログラミングガイド」、「TERASOLUNA Server Framework for Java (5.x) プログラミングガイド」、「スクリプト開発モデル プログラミングガイド」へのリンクを追加しました。

はじめに

本書の目的

本書ではアクセスコンテキストを利用した開発手順について説明します。

説明範囲は以下の通りです。

- 新しいアクセスコンテキストの定義方法
- 既存のアクセスコンテキストの利用方法

アクセスコンテキストの概要と仕組みについては、「 アクセスコンテキスト仕様書 」を参照してください。

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を独自に定義して提供したい。
- 提供されているアクセスコンテキストを利用したい。

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

  アクセスコンテキストを提供する、または、利用する場合。

- スクリプト開発モデル

  アクセスコンテキストを利用する場合。

サンプルコードについて

本書に掲載されているサンプルコードは可読性を重視しており、性能面や保守性といった観点において必ずしも適切な実装ではありません。
開発においてサンプルコードを参考にされる場合には、上記について十分に注意してください。

本書の構成

- *アクセスコンテキストの実装*

  アクセスコンテキストを提供する場合の実装方法を、サンプルを利用して、以下の順に説明します。

  1. 新しいアクセスコンテキストを追加する方法
  2. アクセスコンテキストにキャッシュを適用する方法
  3. アクセスコンテキストに切替機能を追加する方法

  新しいアプリケーションで利用する共有情報をアクセスコンテキストとして定義し、アプリケーション内で共通的に利用する場合に参照してください。

- *アクセスコンテキストの利用*

  提供されているアクセスコンテキストをプログラムから利用する方法を、以下の順に説明します。

  1. アクセスコンテキストへのアクセス

2. ライフサイクルの利用

intra-mart Accel Platform 、または、他のアプリケーションで提供されたアクセスコンテキストを利用して、アプリケーションを開発する場合に参照してください。

アクセスコンテキストの実装

アクセスコンテキストの実装例

この章では、アクセスコンテキスト、および、アクセスコンテキストのインスタンスを生成するためのコンテキストビルダを作成する方法を説明します。

アクセスコンテキスト、および、コンテキストビルダについての詳細は、「 アクセスコンテキスト仕様書 」-「 アクセスコンテキストフレームワーク 」を参照してください。

> **注意**
>
> この章で作成する実装例の動作確認は、intra-mart Accel Platform のサンプル機能を使用しているため、サンプルを含めて構築した環境で確認してください。

項目

機能定義と作業の流れ

この章で作成するアクセスコンテキストの要件は、以下の通りです。

- Web実行環境で利用する。
- ログインユーザの「ユーザ名」「メールアドレス」をそれぞれ文字列で保持する。
- 任意のユーザでログイン後、アクセスコンテキストの内容を更新する。

アクセスコンテキストを作成して使用可能にするための作業手順は、以下の通りです。

順序	作業内容	解説
1	アクセスコンテキストクラスを実装する。	*アクセスコンテキストの作成*
2	コンテキストビルダクラスを実装する。	*コンテキストビルダの作成*
3	アクセスコンテキスト設定を用意する。	*アクセスコンテキストの設定*
4	アクセスコンテキストの動作を確認する。	*アクセスコンテキストの動作確認*

アクセスコンテキストの作成

以下の条件を満たすアクセスコンテキストを実装します。

- ログインユーザの「ユーザ名」「メールアドレス」をそれぞれ文字列で保持する。

各情報を保持するため、アクセスコンテキストに持たせるプロパティを決定します。
今回の例では、「ユーザ名」「メールアドレス」の2つを保持するため、以下のように定義します。

プロパティ名	説明
userName	ユーザ名。IM共通マスタから取得する。 例） `青柳辰巳`
mailAddress	メールアドレス。IM共通マスタのメールアドレス 1 を使用する。 例） `aoyagi@example.com`

次に、アクセスコンテキストクラスを作成します。
クラスを作成する場合は、以下のインタフェースを実装してください。

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

Context インタフェースを実装することで、getType メソッドの実装が必須となります。

完全修飾子（FQCN）　sample.SimpleUserContext

```
package sample;

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

public class SimpleUserContext implements Context {

   /** バージョン番号（新規に採番してください）*/
   private static final long serialVersionUID = 156377866768711512L;

   /** ユーザ名 */
   private String userName;

   /** メールアドレス */
   private String mailAddress;

   public String getMailAddress() {
      return mailAddress;
   }

   /**
    * コンテキスト種別（アクセスコンテキストの種類を表すインタフェースの型）を返却します。
    * @return このクラス自身のクラスオブジェクト
    */
   @SuppressWarnings("unchecked")
   @Override
   public Class<SimpleUserContext> getType() {
      return SimpleUserContext.class;
   }

   public String getUserName() {
      return userName;
   }

   public void setMailAddress(final String mailAddress) {
      this.mailAddress = mailAddress;
   }

   public void setUserName(final String userName) {
      this.userName = userName;
   }

}
```

実装が必要なメソッドとその説明は、以下の通りです。

- getType メソッド

  自分自身のコンテキストのクラスタイプを返却してください。
  今回の例では、SimpleUserContext.class を返却します。

> **注意**
>
> Context インタフェースは Serializable インタフェースを継承しているため、serialVersionUID が必要です。
> 上記の値をそのまま使用せず、新しく採番してください。

コンテキストビルダの作成

次に、先ほど作成したアクセスコンテキストのインスタンスを生成するコンテキストビルダクラスを作成します。

クラスを作成する場合は、以下のクラスを継承してください。

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

`ContextBuilderSupport` クラスを継承することで、`create` メソッドの実装が必須となります。

完全修飾子（FQCN）　`sample.SimpleUserContextBuilder`

```
package sample;

import java.util.Date;
import java.util.Locale;

import jp.co.intra_mart.common.platform.log.Logger;
import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.core.ContextBuilderSupport;
import jp.co.intra_mart.foundation.context.core.Resource;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.context.model.Context;
import jp.co.intra_mart.foundation.exception.BizApiException;
import jp.co.intra_mart.foundation.i18n.datetime.DateTime;
import jp.co.intra_mart.foundation.i18n.timezone.SystemTimeZone;
import jp.co.intra_mart.foundation.master.user.UserManager;
import jp.co.intra_mart.foundation.master.user.model.User;
import jp.co.intra_mart.foundation.master.user.model.UserBizKey;

public class SimpleUserContextBuilder extends ContextBuilderSupport {

  private static final Logger LOGGER = Logger.getLogger(SimpleUserContextBuilder.class);

  /**
   * アクセスコンテキストのインスタンスを生成します。
   * @param resource 環境情報
   * @return アクセスコンテキストのインスタンス
   */
  @Override
  protected Context create(final Resource resource) {
    try {
      // ここに到達したことをログに出力
      LOGGER.info("SimpleUserContext created.");

      // アクセスコンテキストのインスタンスを生成して返却
      return createNewContext();
    } catch (final BizApiException e) {
      throw new RuntimeException(e);
    }
  }

  private SimpleUserContext createNewContext() throws BizApiException {
    // ログインユーザの情報をアカウントコンテキストから取得
    final AccountContext accountContext = Contexts.get(AccountContext.class);
    final String userCd = accountContext.getUserCd();
    final Locale locale = accountContext.getLocale();

    // ログインユーザの情報 (ユーザコード、ロケール) をキーとしてユーザ情報を取得
    final User user = getUser(userCd, locale);
    if (user == null) {
      // ユーザ情報が取得できないときは、空のアクセスコンテキストのインスタンスを返却
      return new SimpleUserContext();
    }

    // アクセスコンテキストのインスタンスに、ユーザ情報を設定
    final SimpleUserContext context = new SimpleUserContext();
    context.setUserName(user.getUserName());
    context.setMailAddress(user.getEmailAddress1());

    // アクセスコンテキストのインスタンスを返却
    return context;
  }
```

```
    private Date getBaseTime() {
        // ログインユーザのタイムゾーンで現在日付を取得
        final DateTime userDate = new DateTime(Contexts.get(AccountContext.class).getTimeZone());
        return new DateTime(SystemTimeZone.getDefaultTimeZone(), userDate.getYear(), userDate.getMonthOfYear(),
userDate.getDayOfMonth()).getDate();
    }

    private User getUser(final String userCd, final Locale locale) throws BizApiException {
        // IM共通マスタの検索条件を設定
        final UserManager manager = new UserManager();
        final UserBizKey bizKey = new UserBizKey();
        bizKey.setUserCd(userCd);

        // IM共通マスタからユーザ情報を検索
        final User user = manager.getUser(bizKey, getBaseTime(), locale, false);
        return user;
    }

}
```

実装が必要なメソッドとその説明は、以下の通りです。

- `create` メソッド

  自身のコンテキストビルダがサポートするアクセスコンテキストのインスタンスを返却してください。
  今回の例では、`SimpleUserContext` クラスのインスタンスを作成して、各プロパティ値を設定後、返却します。

  > **コラム**
  >
  > Web実行環境のライフサイクル開始処理では、引数 `Resource` のリソース情報プロパティは設定されていないため、ここでは使用していません。
  > 実行環境によっては、プログラムの引数としてリソース情報が設定されているため、リソース情報を参照して処理をします。

アクセスコンテキストの設定

作成したアクセスコンテキストと、コンテキストビルダを紐付ける「アクセスコンテキスト設定」を用意します。

設定ファイルに、以下のような形式で記述してください。

パス　`WEB-INF/conf/context-config/{任意のファイル名}.xml`

```
<?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">

  <context
      name="sample.SimpleUserContext"
      depends="jp.co.intra_mart.foundation.context.model.AccountContext">

    <builder target="platform.request">
      <builder-class>sample.SimpleUserContextBuilder</builder-class>
    </builder>

  </context>

</context-config>
```

> **注意**
>
> ファイル名は、他のモジュールが提供しているものと重複しないようにするために、モジュールIDを接頭子にするなどの対策を行ってください。
>
> 例） `sample-module_simple-user-context.xml`

設定が必要なタグとその説明は、以下の通りです。

- `context` タグ

  `name` 属性に、アクセスコンテキストクラスの完全修飾子（FQCN）を指定してください。
  今回の例では `sample.SimpleUserContext` を指定します。

  `depends` 属性に、依存するアクセスコンテキストクラスの完全修飾子（FQCN）を指定してください。
  今回の例では、`SimpleUserContext` 内で `AccountContext` を使用しているため、`AccountContext` が先に解決されるようにするために、依存関係を設定します。

- `builder` タグ

  `target` 属性に、呼び出すタイミングを示すリソースIDを指定してください。
  今回の例では、Web実行環境の開始処理を表すリソースID `platform.request` を指定します。

  `builder-class` タグ内に、コンテキストビルダクラスの完全修飾子（FQCN）を指定します。
  今回の例では、`sample.SimpleUserContextBuilder` を指定します。

設定ファイルの詳細は、「 アクセスコンテキスト仕様書 」-「 アクセスコンテキスト設定 」を参照してください。

アクセスコンテキストの動作確認

作成したアクセスコンテキストが動作しているか、以下の手順で確認します。

1. サーバ起動時のログを確認します。
   アクセスコンテキスト、コンテキストビルダ、および、設定ファイルが正しくアプリケーションサーバに配置されている場合、サーバ起動時に以下のログが出力されますので確認してください。

   ```
   [I.IWP.CONTEXT.MANAGER.10001] Used context. sample.SimpleUserContext
   ```

   実際の出力例

   ```
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcs9 -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.context.model.ClientContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcsa -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.context.model.AccountContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcsb -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.job_scheduler.JobSchedulerContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcsc -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.user_context.model.UserContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcsd -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.authz.context.AuthzSubjectContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcse -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. sample.SimpleUserContext
   [2014-05-01 12:00:00.000] [resin-11] INFO jp.co.intra_mart.system.service.impl.ServiceControllerImpl 5ib6n4szdi8sf
   - [I.IWP.SERVICE.00009] Initialize service "server.service.queue.management".
   [2014-05-01 12:00:00.000] [resin-11] INFO jp.co.intra_mart.system.service.impl.ServiceControllerImpl 5ib6n4szdi8sg
   - [I.IWP.SERVICE.00009] Initialize service "server.service.task.management".
   ```

2. ログイン時に、ログ、および、アクセスコンテキストの保持内容を確認します。
   任意のユーザ（例：サンプルユーザの青柳）でログイン後、`SimpleUserContextBuilder#create()` メソッドの冒頭で実装しているログが以下のように出力されますので、システムログを確認してください。

ログが出力されていれば、アクセスコンテキストの作成処理が正常に行われています。

```
[] SimpleUserContext created.
```

実際の出力例

```
[2014-05-01 12:00:00.000] [resin-port-8080-11] INFO  sample.SimpleUserContextBuilder    5ib6n4umb3mos    - [] SimpleUserContext created.
```

アクセスコンテキストの保持内容を確認するため、以下の URL にアクセスしてください。

`http://<HOST>:<PORT>/<CONTEXT_PATH>/sample/context/context_view.jsp`

「Context: sample.SimpleUserContext」カテゴリが表示され、各プロパティに想定通りの値が表示されていることを確認します。

Context: sample.SimpleUserContext

Property Name	Contexts.get()	Cached Context
Context Class	SimpleUserContext	null
mailAddress	aoyagi@example.com	null
type	class SimpleUserContext	null
userName	青柳辰巳	null

図 アクセスコンテキストの内容

アクセスコンテキストのキャッシュ機能の実装例

この章では、アクセスコンテキストをキャッシュする方法を説明します。

前章「*アクセスコンテキストの実装例*」で作成したアクセスコンテキストはキャッシュされていないため、ライフサイクルが開始されるたびに都度コンテキストビルダによってインスタンスが作成されます。
ここでは、アクセスコンテキストをキャッシュして、毎回インスタンスを作成しないようにします。

キャッシュについての詳細は、「アクセスコンテキスト仕様書」-「キャッシュ」を参照してください。

項目

機能定義と作業の流れ

この章で作成するアクセスコンテキストのキャッシュ処理の要件は、以下の通りです。

- 一度アクセスコンテキストのインスタンスを生成した場合は、キャッシュする。
- 次回アカウントコンテキストを要求されたときは、キャッシュから返却する。

アクセスコンテキストをキャッシュするための作業手順は、以下の通りです。

順序	作業内容	解説
1	キャッシュに対応するコンテキストビルダクラスを実装する。	*キャッシュに対応するコンテキストビルダの作成*
2	アクセスコンテキスト設定を追加する。	*アクセスコンテキスト設定の追加*
3	アクセスコンテキストの動作を確認する。	*アクセスコンテキストの動作確認*

キャッシュに対応するコンテキストビルダの作成

キャッシュに対応するコンテキストビルダクラスを作成します。
クラスを作成する場合は、以下のクラスを継承してください。

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

`CachingContextBuilderSupport` クラスを継承することで、`create` メソッドの実装が必須となります。
実装方法とメソッドの詳細は、前章の「*コンテキストビルダの作成*」と同様です。

前章で作成したコンテキストビルダを流用する場合は、継承しているクラスを `ContextBuilderSupport` から `CachingContextBuilderSupport` に変更してください。

完全修飾子（FQCN）　`sample.CachingSimpleUserContextBuilder`

```
package sample;

import java.util.Date;
import java.util.Locale;

import jp.co.intra_mart.common.platform.log.Logger;
import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.core.Resource;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.context.model.Context;
import jp.co.intra_mart.foundation.exception.BizApiException;
import jp.co.intra_mart.foundation.i18n.datetime.DateTime;
import jp.co.intra_mart.foundation.i18n.timezone.SystemTimeZone;
import jp.co.intra_mart.foundation.master.user.UserManager;
import jp.co.intra_mart.foundation.master.user.model.User;
import jp.co.intra_mart.foundation.master.user.model.UserBizKey;
import jp.co.intra_mart.system.context.core.cache.CachingContextBuilderSupport;

public class CachingSimpleUserContextBuilder extends CachingContextBuilderSupport {

  private static final Logger LOGGER = Logger.getLogger(CachingSimpleUserContextBuilder.class);

  /**
   * アクセスコンテキストのインスタンスを生成します。
   * @param resource 環境情報
   * @return アクセスコンテキストのインスタンス
   */
  @Override
  protected Context create(final Resource resource) {
    try {
      // ここに到達したことをログに出力
      LOGGER.info("SimpleUserContext created.");

      // アクセスコンテキストのインスタンスを生成して返却
      return createNewContext();
    } catch (final BizApiException e) {
      throw new RuntimeException(e);
    }
  }

  private SimpleUserContext createNewContext() throws BizApiException {
    // ログインユーザの情報をアカウントコンテキストから取得
    final AccountContext accountContext = Contexts.get(AccountContext.class);
    final String userCd = accountContext.getUserCd();
    final Locale locale = accountContext.getLocale();

    // ログインユーザの情報 (ユーザコード、ロケール) をキーとしてユーザ情報を取得
    final User user = getUser(userCd, locale);
    if (user == null) {
      // ユーザ情報が取得できないときは、空のアクセスコンテキストのインスタンスを返却
      return new SimpleUserContext();
    }

    // アクセスコンテキストのインスタンスに、ユーザ情報を設定
    final SimpleUserContext context = new SimpleUserContext();
    context.setUserName(user.getUserName());
    context.setMailAddress(user.getEmailAddress1());
```

```
        // アクセスコンテキストのインスタンスを返却
        return context;
    }

    private Date getBaseTime() {
        // ログインユーザのタイムゾーンで現在日付を取得
        final DateTime userDate = new DateTime(Contexts.get(AccountContext.class).getTimeZone());
        return new DateTime(SystemTimeZone.getDefaultTimeZone(), userDate.getYear(), userDate.getMonthOfYear(),
userDate.getDayOfMonth()).getDate();
    }

    private User getUser(final String userCd, final Locale locale) throws BizApiException {
        // IM共通マスタの検索条件を設定
        final UserManager manager = new UserManager();
        final UserBizKey bizKey = new UserBizKey();
        bizKey.setUserCd(userCd);

        // IM共通マスタからユーザ情報を検索
        final User user = manager.getUser(bizKey, getBaseTime(), locale, false);
        return user;
    }

}
```

アクセスコンテキスト設定の追加

アクセスコンテキストと、キャッシュに対応したコンテキストビルダを紐付ける設定を、「アクセスコンテキスト設定」に追加します。

設定ファイルに、以下のような形式で記述してください。
前章の「*アクセスコンテキストの設定*」で作成した設定ファイルに、キャッシュに関する設定を追加します。

パス　WEB-INF/conf/context-config/{任意のファイル名}.xml

```
<?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">

 <context
    name="sample.SimpleUserContext"
    depends="jp.co.intra_mart.foundation.context.model.AccountContext">

   <builder target="platform.request">
    <builder-class>sample.CachingSimpleUserContextBuilder</builder-class>
    <init-param>
     <param-key>cache-policy</param-key>
     <param-value>session-user-daily</param-value>
    </init-param>
   </builder>

 </context>

</context-config>
```

設定が必要なタグとその説明は、以下の通りです。

- init-param タグ内の cache-policy 設定

  キャッシュのポリシーを指定してください。
  今回の例では、 session-user-daily を指定します。

  キャッシュに関する設定の詳細は、「 アクセスコンテキスト仕様書 」-「 キャッシュ設定 」を参照してください。

アクセスコンテキストの動作確認

作成したアクセスコンテキストのキャッシュが動作しているか、以下の手順で確認します。

1. サーバ起動時のログを確認します。
   アクセスコンテキスト、コンテキストビルダ、および、設定ファイルが正しくアプリケーションサーバに配置されている場合、サーバ起動時に以下のシステムログが出力されます。

   ```
   [I.IWP.CONTEXT.MANAGER.10001] Used context. sample.SimpleUserContext
   ```

   実際の出力例

   ```
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcs9 -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.context.model.ClientContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcsa -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.context.model.AccountContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcsb -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.job_scheduler.JobSchedulerContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcsc -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.user_context.model.UserContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcsd -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.authz.context.AuthzSubjectContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcse -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. sample.SimpleUserContext
   [2014-05-01 12:00:00.000] [resin-11] INFO jp.co.intra_mart.system.service.impl.ServiceControllerImpl 5ib6n4szdi8sf
   - [I.IWP.SERVICE.00009] Initialize service "server.service.queue.management".
   [2014-05-01 12:00:00.000] [resin-11] INFO jp.co.intra_mart.system.service.impl.ServiceControllerImpl 5ib6n4szdi8sg
   - [I.IWP.SERVICE.00009] Initialize service "server.service.task.management".
   ```

2. ログイン時に、ログ、および、アクセスコンテキストの保持内容を確認します。
   任意のユーザ（例：サンプルユーザの青柳）でログイン後、`CachingSimpleUserContextBuilder#create()` メソッドの冒頭で実装しているログが以下のように出力されますので、システムログを確認してください。
   ログが出力されていれば、アクセスコンテキストの作成処理が正常に行われています。

   ```
   [] SimpleUserContext created.
   ```

   実際の出力例

   ```
   [2014-05-01 12:00:00.000] [resin-port-8080-11] INFO sample.CachingSimpleUserContextBuilder 5ib6n4umb3mos - []
   SimpleUserContext created.
   ```

   アクセスコンテキストの保持内容を確認するため、以下の URL にアクセスしてください。

   `http://<HOST>:<PORT>/<CONTEXT_PATH>/sample/context/context_view.jsp`

   「Context: sample.SimpleUserContext」カテゴリが表示され、各プロパティに想定通りの値が表示されていることを確認します。
   また、キャッシュが働いているため、「Cached Context」列に同じ内容が表示されます。（下図の緑枠）

   
   Context: sample.SimpleUserContext

Property Name	Contexts.get()	Cached Context
Context Class	SimpleUserContext	SimpleUserContext
mailAddress	aoyagi@example.com	aoyagi@example.com
type	class SimpleUserContext	class SimpleUserContext
userName	青柳辰巳	青柳辰巳

   図 アクセスコンテキストの内容

3. 「個人設定」を変更して、アクセスコンテキストの保持内容が**変更されていないこと**を確認します。

「個人設定」の「ロケール」を変更後にページを再表示すると、アクセスコンテキストの保持内容が変更されないことが確認できます。
これはキャッシュが正常に動作しているためです。
キャッシュが動作しているため、「個人設定」の「ロケール」を変更しても、以下のログは再度出力されません。

```
[] SimpleUserContext created.
```

「ロケール」を変更後、アクセスコンテキストの内容を変更するためには、次章で説明するアクセスコンテキストの切替処理が必要です。

アクセスコンテキストの切替機能の実装例

この章では、アクセスコンテキストを切り替える方法を説明します。

前章「*アクセスコンテキストのキャッシュ機能の実装例*」で作成したアクセスコンテキストの切替処理を行うことによって、アクセスコンテキストの内容を変更します。
ここでは、「個人設定」が変更された場合にアクセスコンテキストを切り替えます。

切替処理についての詳細は、「 アクセスコンテキスト仕様書 」-「 切替 」を参照してください。

項目

機能定義と作業の流れ

この章で作成するアクセスコンテキストの切替処理の要件は、以下の通りです。

- 「個人設定」の「ロケール」が変更されたタイミングで、コンテキストの保持内容を個人設定が反映された値に変更する。

アクセスコンテキストを切り替えるための作業手順は、以下の通りです。

順序	作業内容	解説
1	コンテキストデコレータクラスを用意する。	*コンテキストデコレータの作成*
2	アクセスコンテキスト設定を追加する。	*アクセスコンテキスト設定の追加*
3	アクセスコンテキストの動作を確認する。	*アクセスコンテキストの動作確認*

コラム

ここで説明する切替機能は、コンテキストスイッチを対象としています。
コンテキストスタックの実装の流れも、コンテキストスイッチと同じように行います。
「 アクセスコンテキスト仕様書 」-「 コンテキストスタック 」を参照して実装してください。

コンテキストデコレータの作成

コンテキストの切替に対応するコンテキストデコレータクラスを作成します。
クラスを作成する場合は、以下のクラスを継承してください。

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

`ContextDecoratorSupport` クラスを継承することで、`decorate` メソッドの実装が必須となります。

完全修飾子（FQCN）　`sample.SimpleUserContextDecorator`

```
package sample;
```

```
import java.util.Date;
import java.util.Locale;

import jp.co.intra_mart.common.platform.log.Logger;
import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.core.ContextDecoratorSupport;
import jp.co.intra_mart.foundation.context.core.Resource;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.context.model.Context;
import jp.co.intra_mart.foundation.exception.BizApiException;
import jp.co.intra_mart.foundation.i18n.datetime.DateTime;
import jp.co.intra_mart.foundation.i18n.timezone.SystemTimeZone;
import jp.co.intra_mart.foundation.master.user.UserManager;
import jp.co.intra_mart.foundation.master.user.model.User;
import jp.co.intra_mart.foundation.master.user.model.UserBizKey;

public class SimpleUserContextDecorator extends ContextDecoratorSupport {

    private static final Logger LOGGER = Logger.getLogger(SimpleUserContextDecorator.class);

    /**
     * アクセスコンテキストの内容を切り替えて、新しいアクセスコンテキストのインスタンスを返却します。
     * @param context 拡張元アクセスコンテキスト
     * @param resource 環境情報
     * @return 切替後のアクセスコンテキストのインスタンス
     */
    @Override
    public Context decorate(final Context context, final Resource resource) {
        try {
            // ここに到達したことをログに出力
            LOGGER.info("SimpleUserContext decorated.");

            // アクセスコンテキストのインスタンスを生成して返却
            return createNewContext();
        } catch (final BizApiException e) {
            throw new RuntimeException(e);
        }
    }

    private SimpleUserContext createNewContext() throws BizApiException {
        // ログインユーザの情報をアカウントコンテキストから取得
        final AccountContext accountContext = Contexts.get(AccountContext.class);
        final String userCd = accountContext.getUserCd();
        final Locale locale = accountContext.getLocale();

        // ログインユーザの情報 (ユーザコード、ロケール) をキーとしてユーザ情報を取得
        final User user = getUser(userCd, locale);
        if (user == null) {
            // ユーザ情報が取得できないときは、空のアクセスコンテキストのインスタンスを返却
            return new SimpleUserContext();
        }

        // アクセスコンテキストのインスタンスに、ユーザ情報を設定
        final SimpleUserContext context = new SimpleUserContext();
        context.setUserName(user.getUserName());
        context.setMailAddress(user.getEmailAddress1());

        // アクセスコンテキストのインスタンスを返却
        return context;
    }

    private Date getBaseTime() {
        // ログインユーザのタイムゾーンで現在日付を取得
        final DateTime userDate = new DateTime(Contexts.get(AccountContext.class).getTimeZone());
        return new DateTime(SystemTimeZone.getDefaultTimeZone(), userDate.getYear(), userDate.getMonthOfYear(),
userDate.getDayOfMonth()).getDate();
    }

    private User getUser(final String userCd, final Locale locale) throws BizApiException {
        // IM共通マスタの検索条件を設定
        final UserManager manager = new UserManager();
```

```
        final UserBizKey bizKey = new UserBizKey();
        bizKey.setUserCd(userCd);

        // IM共通マスタからユーザ情報を検索
        final User user = manager.getUser(bizKey, getBaseTime(), locale, false);
        return user;
    }

}
```

実装が必要なメソッドとその説明は、以下の通りです。

- decorate メソッド

  自身のコンテキストデコレータがサポートするアクセスコンテキストのインスタンスを返却してください。

  切替処理では通常、切替元のアクセスコンテキストを参照して、情報を引き継ぎます。
  そのような場合は、引数の「拡張元アクセスコンテキスト（context）」を参照してください。

  今回の例では、SimpleUserContext クラスのインスタンスを作成して、各プロパティ値を設定後、返却します。

  > **コラム**
  >
  > ここでは、引数 Resource のリソース情報プロパティは使用していません。
  > 切替処理によっては、プログラムの引数としてリソース情報が設定されているため、リソース情報を参照して処理をします。

アクセスコンテキスト設定の追加

アクセスコンテキストと、切替に対応したコンテキストデコレータを紐付ける設定を、「アクセスコンテキスト設定」に追加します。

設定ファイルに、以下のような形式で記述してください。
前章の「*アクセスコンテキスト設定の追加*」で作成した設定ファイルに、切替に関する設定を追加します。

パス　WEB-INF/conf/context-config/{任意のファイル名}.xml

```
<?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">

 <context
    name="sample.SimpleUserContext"
    depends="jp.co.intra_mart.foundation.context.model.AccountContext">

  <builder target="platform.request">
   <builder-class>sample.CachingSimpleUserContextBuilder</builder-class>
   <init-param>
    <param-key>cache-policy</param-key>
    <param-value>session-user-daily</param-value>
   </init-param>
   <init-param>
    <param-key>default-switch-resource-id</param-key>
    <param-value>sample.switch.default</param-value>
   </init-param>
  </builder>

  <builder target="sample.switch.default">
   <builder-class>jp.co.intra_mart.system.context.standard.StandardSwitchableContextBuilder</builder-class>
   <decorator>
    <decorator-class>sample.SimpleUserContextDecorator</decorator-class>
   </decorator>
  </builder>

 </context>

</context-config>
```

今回作成する切替処理は個人設定が変更された場合を想定していますが、特定の処理を対象とした場合、それ以外の切替処理に対応できません。
そのため、デフォルトコンテキストビルダとして定義します。
デフォルトコンテキストビルダについては、「 アクセスコンテキスト仕様書 」-「 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。

設定が必要なタグとその説明は、以下の通りです。

- builder タグ

  target 属性に、呼び出すタイミングを示すリソースIDを指定してください。
  今回の例では、デフォルト用に新しいリソースID sample.switch.default を指定します。
  デフォルトコンテキストビルダとするため、リソースID platform.request の初期パラメータに、default-switch-resource-id の設定を追加します。

  builder-class タグ内に、コンテキストビルダクラスの完全修飾子（FQCN）を指定します。
  今回の例では、jp.co.intra_mart.system.context.standard.StandardSwitchableContextBuilder を指定します。

- decorator タグ

  decorator-class タグ内に、コンテキストデコレータクラスの完全修飾子（FQCN）を指定してください。
  今回の例では sample.SimpleUserContextDecorator を指定します。

コラム

「*アクセスコンテキストの設定*」において、SimpleUserContext が AccountContext に依存する設定を行っているため、「個人設定」が変更された際に、以下の順序でコンテキストの切替が行われます。

1. AccountContext の切替。
2. SimpleUserContext の切替。

そのため、SimpleUserContext で取得する AccountContext の内容は、変更後の個人設定が反映された状態で取得できます。

アクセスコンテキストの動作確認

作成したアクセスコンテキストの切替が動作しているか、以下の手順で確認します。

1. サーバ起動時のログを確認します。
   アクセスコンテキスト、コンテキストビルダ、コンテキストデコレータ、および、設定ファイルが正しくアプリケーションサーバに配置されている場合、サーバ起動時に以下のログが出力されます。

   ```
   [I.IWP.CONTEXT.MANAGER.10001] Used context. sample.SimpleUserContext
   ```

   実際の出力例

   ```
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcs9 -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.context.model.ClientContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcsa -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.context.model.AccountContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcsb -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.job_scheduler.JobSchedulerContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcsc -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.user_context.model.UserContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcsd -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. jp.co.intra_mart.foundation.authz.context.AuthzSubjectContext
   [2014-05-01 12:00:00.000] [resin-11] INFO
   jp.co.intra_mart.system.context.manager.impl.MultipleXmlContextConfiguration 5ib6n4szdhcse -
   [I.IWP.CONTEXT.MANAGER.10001] Used context. sample.SimpleUserContext
   [2014-05-01 12:00:00.000] [resin-11] INFO jp.co.intra_mart.system.service.impl.ServiceControllerImpl 5ib6n4szdi8sf
   - [I.IWP.SERVICE.00009] Initialize service "server.service.queue.management".
   [2014-05-01 12:00:00.000] [resin-11] INFO jp.co.intra_mart.system.service.impl.ServiceControllerImpl 5ib6n4szdi8sg
   - [I.IWP.SERVICE.00009] Initialize service "server.service.task.management".
   ```

2. ログイン時に、ログ、および、アクセスコンテキストの保持内容を確認します。
   任意のユーザ（例：サンプルユーザの青柳）でログイン後、`CachingSimpleUserContextBuilder#create()` メソッドの冒頭で実装しているログが以下のように出力されますので、システムログを確認してください。
   ログが出力されていれば、アクセスコンテキストの作成処理が正常に行われています。

   ```
   [] SimpleUserContext created.
   ```

   実際の出力例

   ```
   [2014-05-01 12:00:00.000] [resin-port-8080-11] INFO sample.CachingSimpleUserContextBuilder 5ib6n4umb3mos - []
   SimpleUserContext created.
   ```

   アクセスコンテキストの保持内容を確認するため、以下の URL にアクセスしてください。

   `http://<HOST>:<PORT>/<CONTEXT_PATH>/sample/context/context_view.jsp`

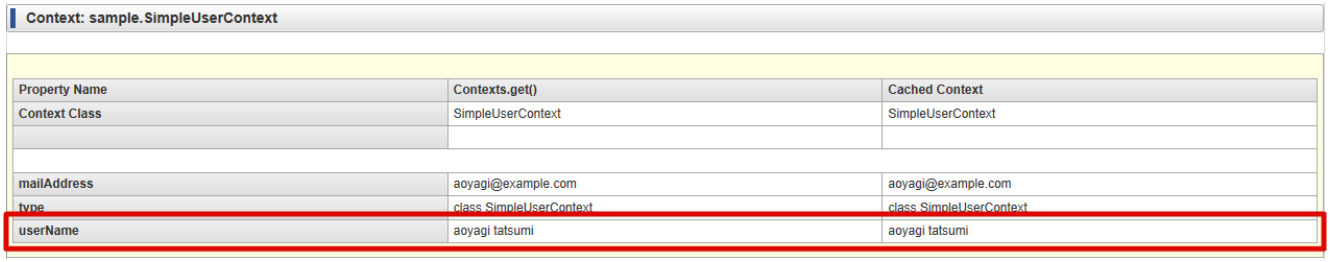
   「Context: sample.SimpleUserContext」カテゴリが表示され、各プロパティに想定通りの値が表示されていれば成功です。

   図 アクセスコンテキストの内容

3. 「個人設定」を変更して、アクセスコンテキストの保持内容が**変更されること**を確認します。

「個人設定」の「ロケール」を変更後にページを再表示すると、コンテキストの保持内容が更新されていることが確認できます。



Context: sample.SimpleUserContext

Property Name	Contexts.get()	Cached Context
Context Class	SimpleUserContext	SimpleUserContext
mailAddress	aoyagi@example.com	aoyagi@example.com
type	class SimpleUserContext	class SimpleUserContext
userName	aoyagi tatsumi	aoyagi tatsumi

図 「ロケール」変更後の内容

「個人設定」を変更したタイミングで、SimpleUserContextDecorator#decorate() メソッドの冒頭で実装しているログが以下のように出力されますので、システムログを確認してください。
ログが出力されていれば、アクセスコンテキストの切替処理が正常に行われています。

```
[] SimpleUserContext decorated.
```

実際の出力例

```
[2014-05-01 12:00:00.000] [resin-port-8080-12] INFO  sample.SimpleUserContextDecorator   default  5ib6n4umivfje 5ib6n4umisij2r2   -  [] SimpleUserContext decorated.
```

アクセスコンテキストの利用

アクセスコンテキストへのアクセス

項目

JavaEE開発モデルの場合

JavaEE開発モデルの場合、アクセスコンテキストのインスタンスを取得してコンテキストにアクセスするためには、Contexts API を利用します。

jp.co.intra_mart.foundation.context.Contexts

例えば、前章の「*アクセスコンテキストの実装*」で作成した sample.SimpleUserContext のアクセスコンテキストのインスタンスを取得する場合は、以下のように実装します。

```
final SimpleUserContext context = Contexts.get(SimpleUserContext.class);
```

また、コンテキストに保持されている値を取得するためには、コンテキストの Getter メソッドを使用します。

```
final String userName = context.getUserName();
```

intra-mart Accel Platform で提供する標準アクセスコンテキストの利用方法については、以下のドキュメントも参照してください。

- 「SAStruts+S2JDBC プログラミングガイド」-「アクセスコンテキスト - プログラミング方法」
- 「TERASOLUNA Server Framework for Java (5.x) プログラミングガイド」-「アクセスコンテキスト - プログラミング方法」

スクリプト開発モデルの場合

スクリプト開発モデルの場合、アクセスコンテキストのインスタンスを取得してコンテキストにアクセスするためには、Contexts API を利用します。

例えば、前章の「*アクセスコンテキストの実装*」で作成した sample.SimpleUserContext のアクセスコンテキストのインスタンスを取得する場合は、以下のように “get” + 「コンテキスト種別の短縮名」を使用します。

```
var context = Contexts.getSimpleUserContext();
```

また、コンテキストに保持されている値を取得するためには、コンテキストのプロパティを使用します。

```
var userName = context.userName;
```

intra-mart Accel Platform で提供する標準アクセスコンテキストの利用方法については、以下のドキュメントも参照してください。

- 「スクリプト開発モデル プログラミングガイド」-「アクセスコンテキスト - プログラミング方法」

ライフサイクルの利用

概要

アクセスコンテキストが有効な期間はライフサイクルが開始されてから終了するまでです。
ライフサイクル内でアクセスコンテキストを変更するためには、切替処理が必要になります。

切替処理についての詳細は、「 アクセスコンテキスト仕様書 」-「 切替 」を参照してください。

この章では、アクセスコンテキストの切替方法について、説明します。

> **注意**
>
> ライフサイクルの切替処理は、実行環境に依存する場合があります。
> 対象の実行環境以外で利用した場合、エラーが発生する可能性があります。

JavaEE開発モデルの場合

Lifecycle API を使用する

ライフサイクル内でアクセスコンテキストを切り替えるには、`Lifecycle` API を使用します。
アクセスコンテキストの切替方法には、「コンテキストスイッチ」と「コンテキストスタック」があります。
詳細は、「 Lifecycle クラスの APIドキュメント 」を参照してください。

また、 intra-mart Accel Platform で提供する標準アクセスコンテキストの利用方法については、以下のドキュメントも参照してください。

- 「 SAStruts+S2JDBC プログラミングガイド 」-「 アクセスコンテキスト - プログラミング方法 」
- 「 TERASOLUNA Server Framework for Java (5.x) プログラミングガイド 」-「 アクセスコンテキスト - プログラミング方法 」

コンテキストスイッチを実行する

コンテキストスイッチを実行するためには、`Lifecycle#switchTo()` メソッドを使用してください。

```
// 環境情報の作成
final String resourceId = "sample.switch.resource.id"; // あらかじめ定義されたリソースID
final Object resourceObject = new Object(); // 指定したスイッチ処理で必要なリソース情報
final Resource resource = new Resource(resourceId, resourceObject);

// コンテキストスイッチの実行
final Lifecycle lifecycle = LifecycleFactory.getLifecycle();
lifecycle.switchTo(resource);
```

引数	説明
resource	以下の情報を設定した環境情報 - あらかじめ定義されたリソースID - 指定したスイッチ処理で必要なリソース情報

コンテキストスタックを実行する

コンテキストスタックを実行するためには、`Lifecycle#stack()` メソッドを使用してください。

```
// 環境情報の作成
final String resourceId = "sample.stack.resource.id"; // あらかじめ定義されたリソースID
final Object resourceObject = new Object(); // 指定したスタック処理で必要なリソース情報
final Resource resource = new Resource(resourceId, resourceObject);

// コンテキストスタックの実行
final Lifecycle lifecycle = LifecycleFactory.getLifecycle();
lifecycle.stack(resource);
```

引数	説明
resource	以下の情報を設定した環境情報 ▪ あらかじめ定義されたリソースID ▪ 指定したスタック処理で必要なリソース情報

スタック前のアクセスコンテキストは保存されており、スタック前のアクセスコンテキストに戻すには、Lifecycle#pop() メソッドを使用してください。

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

Lifecycle API をラップして API を作成する

Lifecycle API を利用した「コンテキストスイッチ」と「コンテキストスタック」の実行時は、 環境情報として、あらかじめ定義されたリソースIDの指定と、Object 型のリソース情報の指定が必要です。
特定のアクセスコンテキストの切替を目的とする場合は、Lifecycle API をラップした別の切替用 API を作成すると便利です。

以下のコードは、クライアントタイプを変更するための、 ClientTypeSwitcher API の実装です。
ClientTypeSwitcher API では、クライアントコンテキストの切替処理をラップしています。
Lifecycle API に引き渡す引数の情報をラッパークラスが隠すことで、アクセスコンテキストの切替が簡単になります。

```
// コンテキストスイッチのラッパーメソッド
public void switchTo(final String clientTypeId) {
    if (vaildChangeClientTypeId(clientTypeId)) {
        final HTTPContext httpContext = HTTPContextManager.getInstance().getCurrentContext();
        final HttpResource resource = new HttpResource(
                MultiDeviceResourceId.CLIENT_CHANGE,
                clientTypeId,
                httpContext.getRequest(),
                httpContext.getResponse()
        );
        LifecycleFactory.getLifecycle().switchTo(resource);
    }
}

// コンテキストスタックのラッパーメソッド
public void oneTimeSwitchTo(final String clientTypeId) {
    if (vaildChangeClientTypeId(clientTypeId)) {
        final HTTPContext httpContext = HTTPContextManager.getInstance().getCurrentContext();
        final HttpResource resource = new HttpResource(
                MultiDeviceResourceId.CLIENT_CHANGE_STACK,
                clientTypeId,
                httpContext.getRequest(),
                httpContext.getResponse()
        );
        LifecycleFactory.getLifecycle().stack(resource);
    }
}
```

スクリプト開発モデルの場合

Lifecycle API を使用する

ライフサイクル内でアクセスコンテキストを切り替えるには、Lifecycle API を使用します。
アクセスコンテキストの切替方法には、「コンテキストスイッチ」と「コンテキストスタック」があります。
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。

また、 intra-mart Accel Platform で提供する標準アクセスコンテキストの利用方法については、以下のドキュメントも参照してください。

- 「 スクリプト開発モデル プログラミングガイド 」-「 アクセスコンテキスト - プログラミング方法 」

コンテキストスイッチを実行する

コンテキストスイッチを実行するためには、Lifecycle#switchTo() メソッドを使用してください。

```
// 環境情報の定義
var resourceId = "sample.switch.resource.id"; // あらかじめ定義されたリソースID
var resourceObject = new Object(); // 指定したスイッチ処理で必要なリソース情報

// コンテキストスイッチの実行
Lifecycle.switchTo(resourceId, resourceObject);
```

引数	説明
resourceId	あらかじめ定義された環境情報のリソースID
resourceObject	指定したスイッチ処理で必要な環境情報のリソース情報

コンテキストスタックを実行する

コンテキストスタックを実行するためには、Lifecycle#stack() メソッドを使用してください。

```
// 環境情報の定義
var resourceId = "sample.stack.resource.id"; // あらかじめ定義されたリソースID
var resourceObject = new Object(); // 指定したスタック処理で必要なリソース情報

// コンテキストスタックの実行
Lifecycle.stack(resourceId, resourceObject);
```

引数	説明
resourceId	あらかじめ定義された環境情報のリソースID
resourceObject	指定したスタック処理で必要な環境情報のリソース情報

スタック前のアクセスコンテキストは保存されており、スタック前のアクセスコンテキストに戻すには、Lifecycle#pop() メソッドを使用してください。

```
Lifecycle.pop();
```