EC-CUBE 3

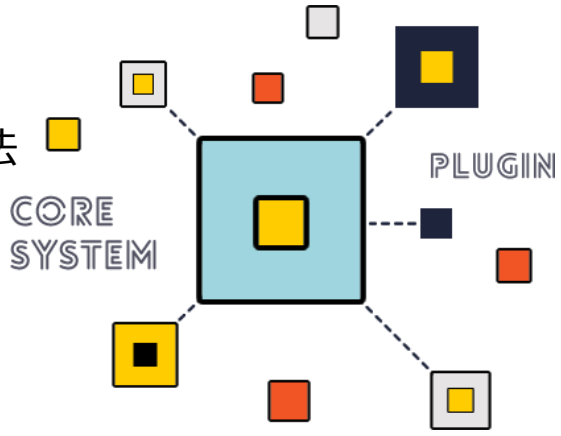
プラグイン仕様

株式会社イーシーキューブ

改定履歴

改定日	履歴	更新内容
2015/08/14	ver.1.0.0	作成
2015/08/18	ver.1.0.1	目次・見出しの調整
2015/08/19	ver.1.0.2	誤表記の修正・レイアウトの微調整
2016/2/17	ver.1.1.0	ec-cube 3.0.9のフックポイント機構について記載
2016/2/23	Ver.1.1.1	フックポイントの不具合について追記
2016/5/31	Ver.1.1.2	3.0.10で追加されたフックポイントを追記
2016/6/22	Ver.1.1.3	目次修正
2016/8/9	Ver 1.1.4	フックポイントの新旧対比の修正 マイグレーションの作成方法を追記 リダイレクト処理の実装方法・注意事項を追記
2016/8/9	Ver 1.1.5	マイグレーションの作成方法を修正
2016/8/19	Ver 1.1.6	テンプレートフックポイントの注意事項を追記
2016/8/31	Ver 1.1.7	リダイレクト処理の注記を修正

目次

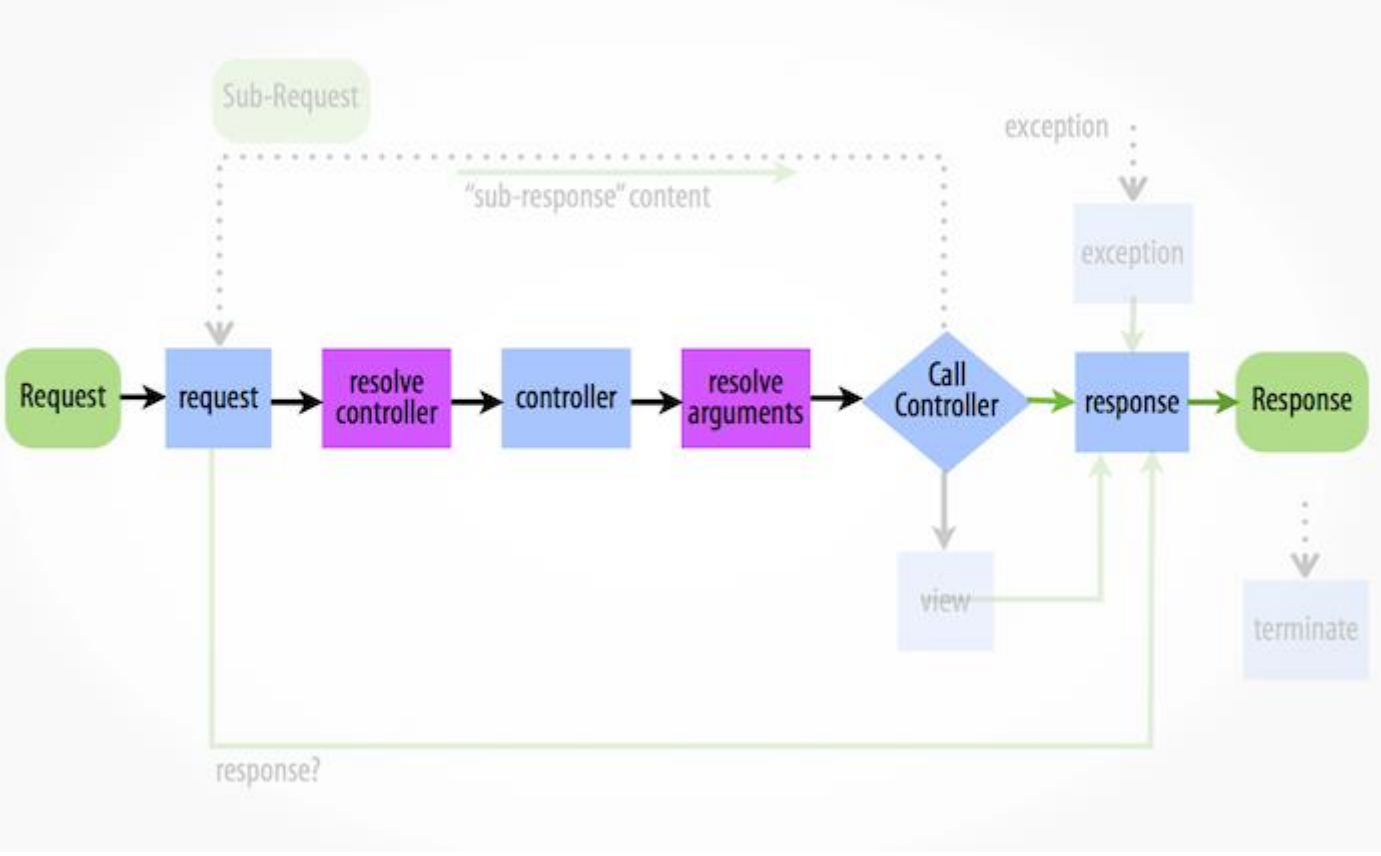


1

プラグインのアーキテクチャ

概要

EC-CUBE3は、フレームワークとしてSymfony2 ComponentsベースのSilexを利用しています。そのため、リクエストを処理しレスポンスを返すまでの内部的な動作は、Symfony2のHttp Kernelのライフサイクルにしたがって実行されます。また、Http Kernelでは、各ステップで処理を差し込めるeventという仕組みがあります。



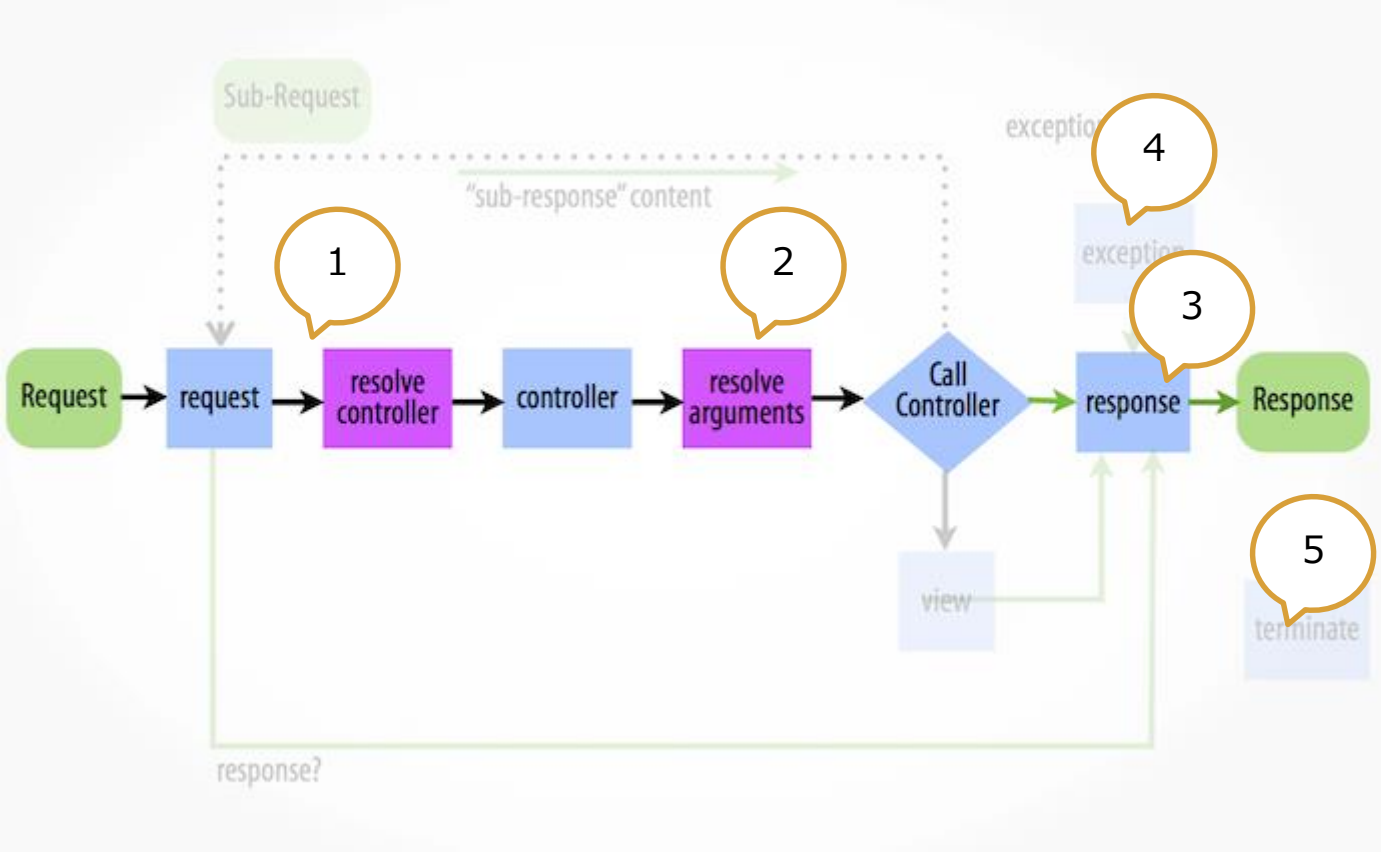
http kernelのライフサイクル：
http://symfony.com/doc/current/components/http_kernel/introduction.html

EC-CUBE3では、
- Symfony2のeventを利用したフックポイント
- コントローラ内部処理やテンプレート、メール送信など、独自に拡張したフックポイント

の2種類のフックポイントを定義しています。

共通フックポイント

共通フックポイントでは、HttpKernelのライフサイクルの各ステップで処理を差し込むことができます。



各ステップは、以下のとおりです。

1. Request
2. Controller
3. Response
4. Exception
5. Terminate

EC-CUBEではこれを拡張し、フロント画面、管理画面、ルーティング毎の4パターンで実行タイミングを定義することができます。

参考： https://github.com/EC-CUBE/ec-cube/issues/1410

共通フックポイント

注意

3.0.9では、管理画面の共通フックポイントに不具合があります。
https://github.com/EC-CUBE/ec-cube/pull/1502

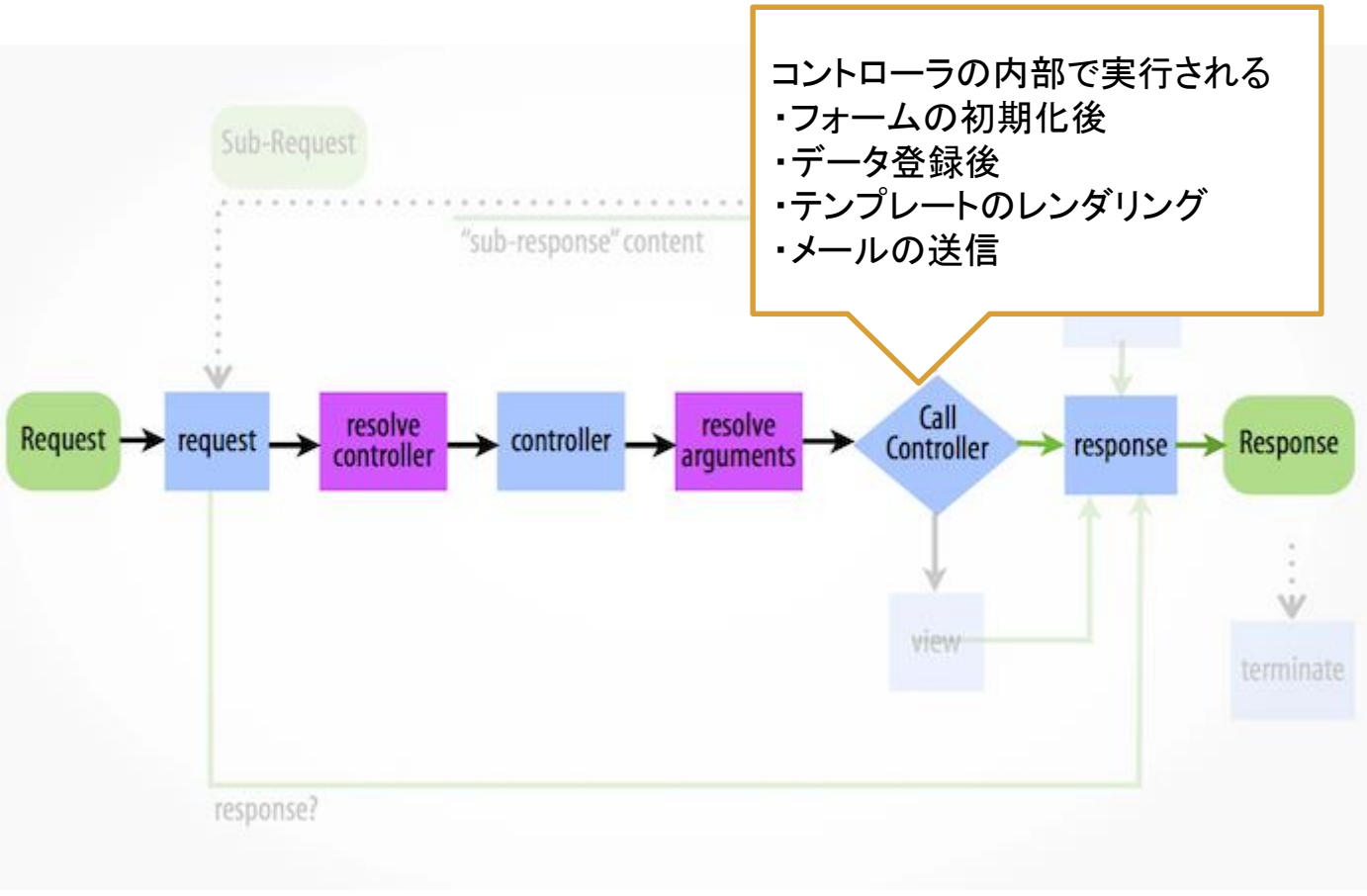
3.0.10でこの不具合は修正されています。
管理画面の共通フックポイントの動作確認を行いたい場合は、上記の修正をとりこみ確認をおこなってください。

共通フックポイント一覧

画面種別	ec-cubeフックポイント名	symfony2イベント名	実行タイミング
フロント画面共通	eccube.event.front.request	REQUEST	リクエスト生成時。v3.0.8までのBeforeフックポイントに相当。ルーティング解決後、認証処理実行前に実行される。
フロント画面共通	eccube.event.front.controller	CONTROLLER	コントローラ実行前。マイページなど認証が必要な画面では認証処理実行済の状態、ただし未ログイン時は実行されない。
フロント画面共通	eccube.event.front.response	RESPONSE	レスポンス生成時。v3.0.8までのAfter/Renderフックポイントに相当。
フロント画面共通	eccube.event.front.exception	EXCEPTION	例外発生時。
フロント画面共通	eccube.event.front.terminate	TERMINATE	アプリケーション終了時。
管理画面共通	eccube.event.admin.request	REQUEST	リクエスト生成時。v3.0.8までのBeforeフックポイントに相当。ルーティング解決後、認証処理実行前に実行される。
管理画面共通	eccube.event.admin.controller	CONTROLLER	コントローラ実行前。認証処理実行済、ただし未ログイン時は実行されない
管理画面共通	eccube.event.admin.response	RESPONSE	レスポンス生成時。v3.0.8までのAfter/Renderフックポイントに相当。
管理画面共通	eccube.event.admin.exception	EXCEPTION	例外発生時。
管理画面共通	eccube.event.admin.terminate	TERMINATE	アプリケーション終了時。
ルーティング単位	eccube.event.route.[route].request	REQUEST	リクエスト生成時。v3.0.8までのBeforeフックポイントに相当。ルーティング解決後、認証処理実行前に実行される。
ルーティング単位	eccube.event.route.[route].controller	CONTROLLER	コントローラ実行前。マイページや管理画面など認証が必要な画面では認証処理実行済の状態、ただし未ログイン時は実行されない。
ルーティング単位	eccube.event.route.[route].response	RESPONSE	レスポンス生成時。v3.0.8までのAfter/Renderフックポイントに相当。
ルーティング単位	eccube.event.route.[route].exception	EXCEPTION	例外発生時。
ルーティング単位	eccube.event.route.[route].terminate	TERMINATE	アプリケーション終了時。v3.0.8までFinishフックポイントに相当

個別フックポイント

個別フックポイントは、EC-CUBE独自に定義したフックポイントです。
コントローラ内部の処理(フォームの初期化やデータ登録処理のタイミング)やメール送信時、twigテンプレートのロード時などのタイミングでフックすることが出来ます。



これにより、既存フォームの拡張や、テンプレートの変更、メールの本文へ文言を追加したりすることができます。

参考： https://github.com/EC-CUBE/ec-cube/issues/1411

※フックポイントの一覧は以下を参照ください。
http://ec-cube.github.io/documents/hookpoints.xlsx

2

プラグインでできること

- **新規ページの作成**

  プラグインから独自のページを定義することができます。

- **既存機能の変更・拡張**

  フックポイントを利用し、フォームへの項目追加や、
  検索クエリの変更、テンプレートやビューの変更、
  メール本文の変更等が可能です。

既存機能の変更・拡張の例

- **入力フォームの拡張**
  EC-CUBE3では表示する入力フォームを構築するために、
  FormBuilderを使っています。
  - フォームを送信する前や後の任意のタイミングで、
    入力フォームを拡張することができます。
  - 定義されているすべてのフォームに対して拡張が可能です。

2

プラグインの作り方

プラグイン作成手順

1. プラグイン名、プラグインコードを決める

1. 規則・機能に則したプラグイン名・プラグインコード名の命名

2. 必要となるファイルを用意し、開発をする

1. config.yml, PluginManagerにプラグインの情報を定義
2. Migrationファイルを記載

3. 圧縮し、パッケージングする

ファイル構成

- {PluginName}
  - config.yml
  - PluginManager.php
  - ServiceProvider/*.php
  - event.yml
  - {EventName}.php
  - Resource/doctrine/*.dcm.yml
  - Resource/doctrine/migration/Version{YYYYmmddHHiiss}.php
  - Form/Type/*.php, Form/Extension/*.php

赤字は必須

- config.yml
  - プラグイン基本情報を記載
- PluginManager.php
  - プラグインのインストール時や、更新時などの処理を記載
- ServiceProvider/*.php
  - config.ymlで定義したServiceProviderで呼び出されるファイル
    ※ 詳細は後述
- event.yml
  - 利用するフックポイントを定義
- {EventName}.php
  - フックポイントに介入する処理を記載
- Resource/doctrine/migration/Version{YYYYmmddHHiiss}.php
  - テーブル作成や削除を定義
- Resource/doctrine/*.dcm.yml
  - テーブル定義を記載
- Form/Type/*.php, Form/Extension/*.php
  - 独自拡張フォームの定義を記載

注意
ディレクトリ構成は、EC-CUBE本体の構成に合わせることを推奨します。

プラグイン作成手順

config.yml

プラグイン全体の設定ファイルを記述します。

【設定項目】

- name
  インストール後に表示されるプラグイン名です。(任意の文字)

- version
  インストール後に表示されるバージョンです。(任意の文字)
  バージョンアップ管理を行う際にご活用ください。

- code
  オーナーズストアがプラグインを識別するコードです。
  (英数字/オーナーズストア内で一意)

- event
  イベントの業務処理ファイル名を記述してください。
  後述の {EventName}.php が読み込まれます。

- service
  イベント以外の任意のロジックを読み込ませるファイルを指定できます。
  (Yaml配列/複数指定可能)
  こちらに記述したファイルが `ServiceProvider` ディレクトリ以下から読み込まれます。

- orm.path
  作成するテーブル定義ファイルの配置ディレクトリを記述してください。

プラグイン作成手順

config.ymlの記述例

```
name: カテゴリコンテンツ　※必須
code: CategoryContents　※必須
version: 1.0.0　※必須
service:
    - CategoryContentsServiceProvider
orm.path
    - /Resource/doctrine
※プラグインでテーブルを追加する場合、orm.pathの記述は必須
event: CategoryContentsEvent
※eventを追加する場合、eventの記述は必須
```

プラグイン作成手順

event.yml

利用するイベントを定義するファイルです。

【設定項目】

```
{HookPointName}:
    - [{MethodName}, {Priority}]
    - [{MethodName}, {Priority}]
```

- HookPointName
  利用するフックポイント名を記述してください。
- MethodName
  Eventファイルの中の、メソッド名を記述してください。
- Priority
  FIRST / NORMAL / LASTを指定してください。

プラグイン作成手順

event.ymlの記述例

```
# front page event
Product/list.twig:
    - [onRenderProductList, NORMAL]

# admin page event
admin.product.category.index.initialize:
    - [onFormInitializeAdminProductCategory, NORMAL]
admin.product.category.index.complete:
    - [onAdminProductCategoryEditComplete, NORMAL]
```

プラグイン作成手順

Resource/doctrine/migration/Version{YYYYmmddHHiiss}.php
データ移行用ファイルです。
PluginManager.php からプラグインインストール/アンインストール時に呼ぶことができます。テーブル作成・削除、データの挿入に利用できます。

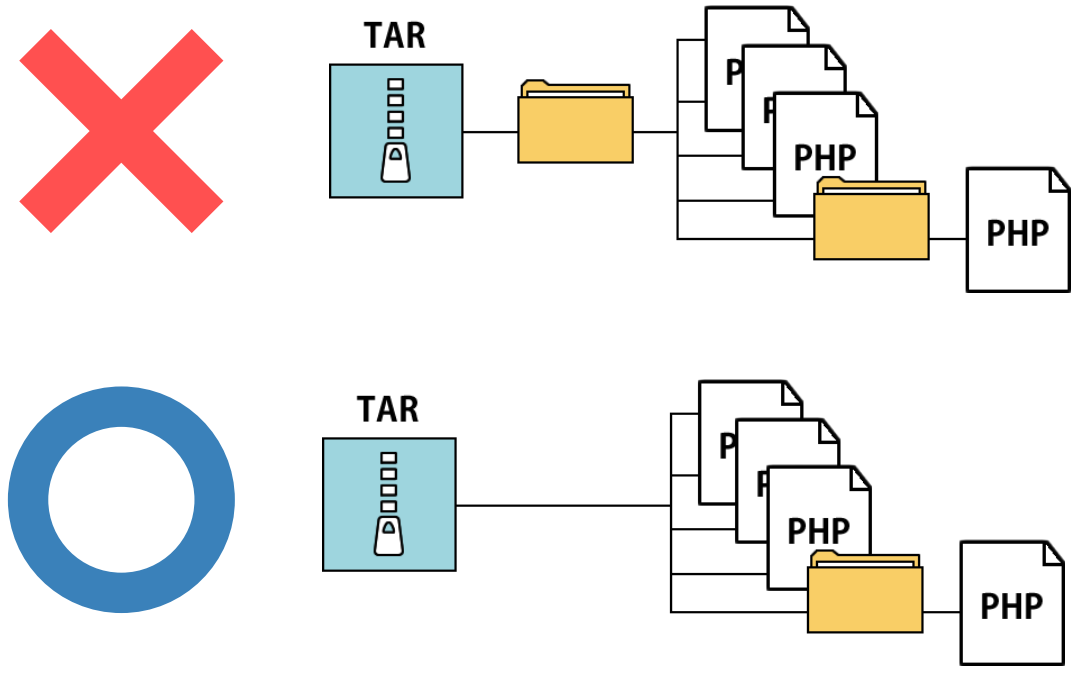
{EventName}.php
フックポイントや入力フォーム拡張時に呼ばれるファイルです。
フックポイントや入力フォームの拡張をする際の、業務処理を記述します。

PluginManager.php
EC-CUBEがプラグインを管理するファイルです。
プラグインのインストール/アンインストール/有効化/無効化時に呼ばれます。
Migration はこのファイルから呼び出します。
install(), uninstall(), enable(), disable(), update()を定義すると、それぞれインストール時、アンインストール時、有効化、無効化、更新時に実行されます。

パッケージング

プラグインファイルのアーカイブ作成方法

プラグインの圧縮方式はtar.gzをご利用ください。
圧縮する際は、**フォルダごと圧縮しない**ようにご注意ください。



命名規則

- テーブル名
  plg_を接頭辞に付与してください。

- ディレクトリ名
  config.yml内のcodeと合わせてください。
  PSR-4に準じてUpperCaseにしてください。

- その他ファイル/クラス名/構成など
  原則として**src/Eccube**と同じ構成になるようにしてください。
  EC-CUBE3の命名規則は開発ドキュメントを参照してください。

  http://ec-cube.github.io/coding_style.html

プラグインを作る

プラグイン基本設定

新規のページを追加する場合も、既存機能や画面の拡張を行う場合も、
まずはプラグイン情報を基本設定ファイルに記述する必要があります。

どちらの場合も config.yml を作成し、プラグイン情報を入力します。
name, **code**, **version** を記載してください。
必要に応じて、 ファイル構成に沿って追記してください。

```
name: カテゴリーコンテンツ
code: CategoryContent
version: 1.0.0
```

プラグインマネージャ

EC-CUBEにプラグインの管理を委譲するために、
プラグインマネージャという仕組みを導入しています。

プラグインマネージャでは、プラグインの
- インストール
- アンインストール
- 有効化
- 無効化
- 更新

のタイミングでの処理を定義することができます。

Migration はこのファイルから呼び出すことで、
テーブルの作成や削除が簡単に行うことができます。

install(), uninstall(), enable(), disable(), update()を定義すると、
それぞれインストール時、アンインストール時、有効化、無効化、更新時に
実行されます。

```
<?php

namespace Plugin¥{PluginName};

use Eccube¥Plugin¥AbstractPluginManager;

class PluginManager extends AbstractPluginManager
{
    public function install($config, $app) {}
    public function uninstall($config, $app) {}
    public function enable($config, $app) {}
    public function disable($config, $app) {}
    public function update($config, $app) {}
}
```

マイグレーション

マイグレーションは、以下のパスに配置します。

{PluginDir}/Resouce/doctrine/migration/Version[yyyymmddHHiiss].php

テーブルの作成や削除は、以下のように作成します。

```
<?php
namespace DoctrineMigrations;

use Doctrine¥DBAL¥Migrations¥AbstractMigration;
use Doctrine¥DBAL¥Schema¥Schema;
use Doctrine¥ORM¥EntityManager;
use Doctrine¥ORM¥Tools¥SchemaTool;

class Version20150706204400 extends AbstractMigration
{
    protected $entities = array(
        'Plugin¥CategoryContent¥Entity¥CategoryContent',
    );
    public function up(Schema $schema)
    {
        $app = ¥Eccube¥Application::getInstance();
        $meta = $this->getMetadata($app['orm.em']);
        // テーブル作成
        $tool = new SchemaTool($app['orm.em']);
        $tool->createSchema($meta);
    }
    public function down(Schema $schema)
    {
        $app = ¥Eccube¥Application::getInstance();
        $meta = $this->getMetadata($app['orm.em']);

        $tool = new SchemaTool($app['orm.em']);
        $schemaFromMetadata = $tool->getSchemaFromMetadata($meta);

        // テーブル削除
        foreach ($schemaFromMetadata->getTables() as $table) {
            if ($schema->hasTable($table->getName())) {
                $schema->dropTable($table->getName());
            }
        }
        // シーケンス削除
        foreach ($schemaFromMetadata->getSequences() as $sequence) {
            if ($schema->hasSequence($sequence->getName())) {
                $schema->dropSequence($sequence->getName());
            }
        }
    }
    protected function getMetadata(EntityManager $em)
    {
        $meta = array();
        foreach ($this->entities as $entity) {
            $meta[] = $em->getMetadataFactory()->getMetadataFor($entity);
        }
        return $meta;
    }
}
```

マイグレーション

PluginManagerからは、以下のように呼び出しを行って下さい。

```
namespace Plugin¥CategoryContent;

use Eccube¥Plugin¥AbstractPluginManager;

class PluginManager extends AbstractPluginManager
{
    public function install($config, $app)
    {
    }

    public function uninstall($config, $app)
    {
        $this->migrationSchema($app, __DIR__.'/Resource/doctrine/migration', $config['code'], 0);
    }

    public function enable($config, $app)
    {
        $this->migrationSchema($app, __DIR__.'/Resource/doctrine/migration', $config['code']);
    }

    public function disable($config, $app)
    {
    }

    public function update($config, $app)
    {
    }
}
```

注意
Installのタイミングでは、プラグインはまだロードされておらず、ServiceProviderで行うルーティング定義やレポジトリ定義は利用できません。そのため、enableでマイグレーションを実行しています。

マイグレーション

データ投入時の注意事項

マイグレーションでは、DBALを利用してデータの挿入や更新が可能です。
利用する場合は、executeUpdate()ではなく、insert()/update()など、組み込みのメソッドを利用することを推奨します。

推奨例

```
// INSERT
$this->connection->insert('dtb_table', array(
    'xxx' => 'aaa',
    'yyy' => 'bbb',
    'zzz' => 'ccc',
));

// UPDATE
$this->connection->update('dtb_xxx', array('xxx' => 'aaa'), array('id' => 1));
```

非推奨例

```
$this->connection->executeUpdate(
    'INSERT INTO dtb_table (xxx, yyy, zzz) VALUES(?, ?, ?)',
    array('aaa', 'bbb', 'ccc')
);

$this->connection->executeUpdate(
    'UPDATE dtb_table set xxx = ? WHERE id = ?',
    array('aaa', 1)
);
```

Topic
マイグレーション内で実行する場合、DBALのオブジェクトはconnectionプロパティに格納されています。PluginManager内で実行する場合は、$app['db']に格納されています。

新規ページの作成方法

config.ymlの作成

config.yml に **service** を記載します。

```
name: カテゴリーコンテンツ
code: CategoryContent
version: 1.0.0
service:
    - {任意のファイル名}
```

この記述によって
/{PluginName}/ServiceProvider/{任意のファイル名}.php
が呼び出されるようになります。

新規ページの作成方法

PluginServiceProvider.php

PluginServiceProvider.php

```
<?php

namespace Plugin¥{PluginName}¥ServiceProvider;

use Eccube¥Application;
use Silex¥Application as BaseApplication;
use Silex¥ServiceProviderInterface;

class PluginServiceProvider implements ServiceProviderInterface
{
    public function register(BaseApplication $app)
    {
    }

    public function boot(BaseApplication $app)
    {
    }
}
```

新規ページの作成方法

register() に以下のように記載することで、新しくルーティングが定義されます。
src/Eccube/ControllerProvider/FrontControllerProvider.php と同様に、
ルーティングを定義します。

```
Plugin/{PluginName}/ServiceProvider/{config.ymlに定義したServiceProvider}.php
public function register(BaseApplication $app)
{
    $app->match(
        '/sample', // アクセスされるURL
        ‘¥Plugin¥{PluginName}¥SampleController::sample’ // コントローラとメソッド名
    )->bind('sample'); // ルーティング名
}

Plugin/{PluginName}/SampleController.php
namespace Plugin¥{PluginName}¥Controller;

use Eccube¥Application;

class SampleController
{
    public function sample(Application $app)
    {
                return ‘Hello, Plugin World!!’;
    }
}
```

次項の「既存機能の変更・拡張」で利用している拡張方法と組み合わせて、
自由にプラグインを作成することが可能です。

既存機能の変更・拡張

フォームの拡張

既存のフォームに項目を追加するなど、フォームを拡張する場合、個別フックポイントを利用します。

フックポイントを使って処理を介入させるには、**event.yml** に以下の項目を定義します。

- 利用するフックポイント
- メソッド名
- 優先順位 (FIRST / NORMAL / LAST)

```
admin.product.category.index.initialize:
    - [onFormInitializeAdminProductCategory, NORMAL]
```

ここに記述したメソッドは、 config.yml 内の event に記載したファイルから呼ばれます。config.yml に **event** 項を追加してください。

```
name: カテゴリーコンテンツ
code: CategoryContent
version: 1.0.0
event: SampleEvent
```

このように記述することで
/Plugin/{PluginName}/SampleEvent.phpの
onFormInitializeAdminProductCategory(EventArgs $event)が呼ばれるようになります。このメソッド内で、フォームへの項目追加の定義を行います。

既存機能の変更・拡張

フォームの拡張

EventArgs $eventで利用できる引数は、呼び出し元のコントローラによって変わります。詳しくは、個別フックポイント一覧を参照してください。

```
// フォームに項目を追加する例
public function onFormInitializeAdminProductCategory(EventArgs $event)
{
    // フォームの追加
    $builder = $event->getArgument('builder');
    $builder->add('plg_category_content', 'textarea', array(
        'required' => false,
        'label' => false,
        'mapped' => false,
        'attr' => array(
            'placeholder' => 'コンテンツを入力してください(HTMLタグ使用可)',
    )));
}
```

```
# 注意
項目名には、”plg_”をプレフィクスとしてつけることで、画面に表示されます。
```

既存機能の変更・拡張

テンプレートの介入：テンプレートフックポイント

Twigのテンプレートに対し、変更を行う事ができます。
config.yml , event.yml に必要な情報を記載してください。
このフックポイントでは、 引数でTempleteEventが渡されます。取得できる値は、テンプレートファイル名、テンプレートソース、パラメータの3つです。

```
public function onRenderProductList(TemplateEvent $event)
{
    $parameters = $event->getParameters();

    $CategoryContent = $this->app['category_content.repository.category_content']
        ->find($Category->getId());

    // twigコードにカテゴリコンテンツを挿入
    $snipet = '<div class="row">{{ CategoryContent.content | raw }}</div>';
    $search = '<div id="result_info_box"';
    $replace = $snipet.$search;
    $source = str_replace($search, $replace, $event->getSource());
    $event->setSource($source);

    // twigパラメータにカテゴリコンテンツを追加
    $parameters['CategoryContent'] = $CategoryContent;
    $event->setParameters($parameters);
}
```

注意
・Responseフックポイントと異なり、twigのテンプレートソースコードが編集対象となります。Crawlerでの解析ができないため、正規表現や文字列置換で対応してください。

・twigファイルがコンパイルされる前に実行されます。パラメータの変更、追加を行う必要が有る場合に利用します。

既存機能の変更・拡張

テンプレートの介入：テンプレートフックポイント

Twigのキャッシュを効かせるため、動的な値を表示したい際は、変数化し、パラメータで渡すようにして下さい。

推奨例

```
    // twigコードにカテゴリコンテンツを挿入
    $snipet = '<div class="row">{{ CategoryContent.content | raw }}</div>';
    $search = '<div id="result_info_box"';
    $replace = $snipet.$search;
    $source = str_replace($search, $replace, $event->getSource());
    $event->setSource($source);

    // twigパラメータにカテゴリコンテンツを追加
    $parameters['CategoryContent'] = $CategoryContent;
    $event->setParameters($parameters);
}
```

非推奨例

```
    // twigコードにカテゴリコンテンツを挿入
    $content = $CategoryContent.getContent();
    $snipet = '<div class="row”’ . $content . ‘<div>';
    $search = '<div id="result_info_box"';
    $replace = $snipet.$search;
    $source = str_replace($search, $replace, $event->getSource());
    $event->setSource($source);
```

既存機能の変更・拡張

htmlコンテンツへの介入：Responseフックポイント

描画されるコンテンツや、ヘッダを書き換えることができます。
EC-CUBE内部では、フックポイントと同様の仕組みを利用しています。
config.yml , event.yml に必要な情報を記載してください。
このフックポイントでは、 引数でFilterResponseEvent が渡されます。

フックポイントの詳細は、共通フックポイント一覧を参照してください。

```
public function onRender(FilterResponseEvent $event)
{
    $request = $event->getRequest();
    $response = $event->getResponse();

    $html = $response->getContent();

    // 書き換え処理ここから
    $crawler = new Crawler($html);
    $oldElement = $crawler
        ->filter('#main');
    $oldHtml = $oldElement->html();
    $newHtml = $oldHtml . $twig;
    $html = $crawler->html();
    $html = str_replace($oldHtml, $newHtml, $html);
     // 書き換え処理ここまで

    $response->setContent($html);
    $event->setResponse($response);
}
```

注意事項
変更する要素の抽出は、idやclassでfilterして行うことを推奨します。
「 」などで抽出した場合、EC-CUBE本体の変更の影響を受けやすく、バージョンの互換性を担保しにくくなるため、避けて下さい。

既存機能の変更・拡張

管理画面メニューの追加

管理画面の左メニューを簡単に追加することができます。
EC-CUBE本体でYamlを解析しており、そこに介入が可能です。

新規ページの作成時と同じく、 ServiceProvider の register() に記述します。
受注管理の一番後ろに追加する場合は以下のように記述してください。

```
public function register(BaseApplication $app)
{
    $app['config'] = $app->share($app->extend('config', function ($config) {
        $config['nav'][1]['child'][] = array(
                'id' => 'order_sample',
                'name' => 'サンプル',
                'url' => 'sample',
        );

        return $config;
    }));
}
```

既存機能の変更・拡張

任意の入力フォームの作成：FormType

プラグイン内で利用するための独自のFormTypeを宣言できます。
FormTypeはEC-CUBE内のFormTypeと同様に作成し、
ServiceProviderにて宣言することで利用が可能になります。

```
# Form¥Type
<?php
namespace Plugin¥{PluginName}}¥Form¥Type;

use ¥Symfony¥Component¥Form¥AbstractType;
use ¥Symfony¥Component¥Form¥FormBuilderInterface;
use ¥Symfony¥Component¥OptionsResolver¥OptionsResolverInterface;

class SampleFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add('sample', 'text');
    }

    public function setDefaultOptions(OptionsResolverInterface $resolver)
    {
    }

    public function getName()
    {
        return 'sample_form';
    }
}

# ServiceProvider::register()
    $app['form.types'] = $app->share($app->extend('form.types', function ($types) use ($app) {
        $types[] = new ¥Plugin¥RelatedProduct¥Form¥Type¥Admin¥RelatedProductType();

        return $types;
    }));
```

参考：http://api.symfony.com/2.0/Symfony/Component/Form/FormBuilder.html

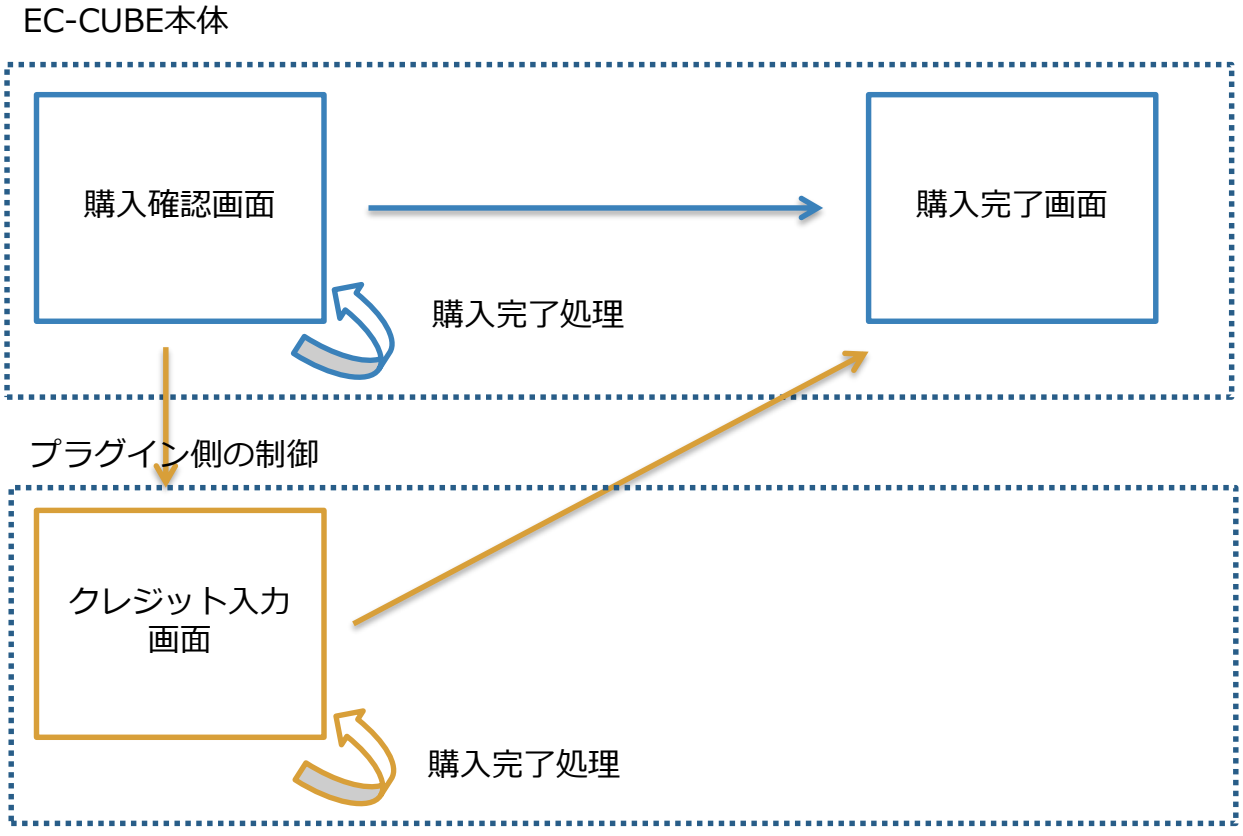
アンチパターン
$builder->remove(‘sample’)などを記載すると、他のプラグインでの拡張を妨げてしまう可能性があります。十分に注意してください。

既存機能の変更・拡張

購入完了処理を拡張する場合

決済プラグインなど、EC-CUBE本体の購入完了処理を通らず、プラグイン側で独自に購入完了を行う場合の対応を記載します。

決済プラグイン利用時の遷移フロー



決済プラグインでは独自の入力画面で、購入完了処理が行われています。
他のプラグインから決済プラグインに介入することができないため、3.0.10以降で実装された、以下のメソッドを、購入が完了するタイミングで呼び出してください。

```
# ShoppingService::notifyComplete

$app['eccube.service.shopping']->notifyComplete($Order);

# ShoppingService::sendOrderMail($Order)

$app['eccube.service.shopping']->sendOrderMail($Order);
```

既存機能の変更・拡張

購入完了処理を拡張する場合

購入完了時に処理を行うプラグインは、前項の対応を行っている決済プラグインと連動させるため、以下のフックポイントを実装してください。

event.yml

```
# 購入完了時拡張
service.shopping.notify.complete:
    - [onServiceShoppingNotifyComplete, NORMAL]
```

上記のフックポイントは、以下の箇所で実装されています。合わせて参照してください。

https://github.com/EC-CUBE/ec-cube/blob/3.0.10/src/Eccube/Service/ShoppingService.php#L1235

リダイレクト処理の実装方法

リダイレクト処理の実装方法

プラグイン内でリダイレクト処理を実装する場合は、RedirectResponseを生成することで処理できます。

```
public function onXxx($event  = null)
{
   $response = $this->app->redirect($this->app->url('xxx');
   $event->setResponse($response)
}
```

注意事項
3.0.11から、トランザクションが本体側で管理されるようになっています。
そのため、header('Location: http://xxx'); exit; のようなコードでリダイレクト処理を実装している場合、データの更新がコミットされず、正しく動作しない可能性があります。

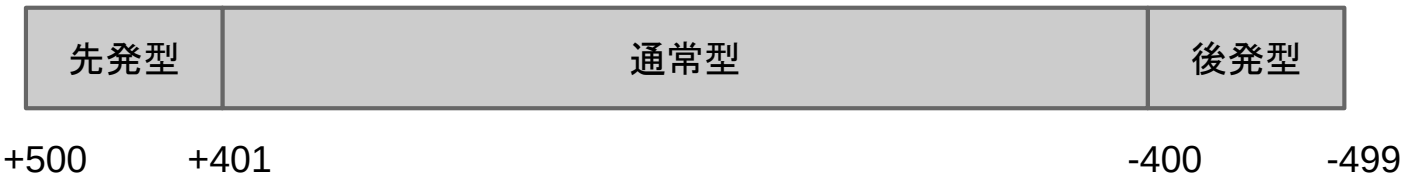
詳細は以下をご確認ください。
http://ec-cube.github.io/guideline/plugin-update-for3.0.11

4

プラグインの管理

ハンドラと優先順位

プラグインの動作の優先順位を制御するために、ハンドラと呼ばれる仕組みを使っています。ハンドラには、通常 / 先発 / 後発の３つの型があり、内部の数値で管理しています。
内部数値が0のものはハンドラを追加しない（動作しない）仕様となっています。



- 先発型：+500 ~ +401
- 通常型：+400 ~ -399
- 後発型：-400 ~ -499

プラグインインストール時の動作

各プラグイン内のevent.ymlにて定義したイベントハンドラと優先度を、
ハンドラ優先度テーブルに挿入します。
当該イベントにハンドラが1件も登録されていない場合の
デフォルト優先度は以下のようになります。

先発型(FIRST):500 / 通常型(NORMAL):400 / 後発型(LAST):-400

ハンドラが既に登録されている場合、既に存在する同型ハンドラ のイベントの優先度の最大値-1を登録します。
例：ハンドラが既に400に登録されている場合に、通常ハンドラを登録すると399に登録されます。

ハンドラ種別毎に決められた優先度枠に空きがない場合、プラグインのインストールは失敗します。

ハンドラと優先順位

Webアプリケーション実行時の動作

プラグインのイベンドハンドラをディスパッチャに登録する際、
優先度テーブルの優先度(昇順)によって各ハンドラを登録します。

イベントが実際に発生すると、登録された順にハンドラが起動します。
優先度テーブルに登録されていないイベントハンドラが定義されている場合、
優先度-500(全てのハンドラの後 / 後発型よりさらに後)として扱います。
※ 開発中などで、プラグインディレクトリを配置しただけの場合などが該当します。

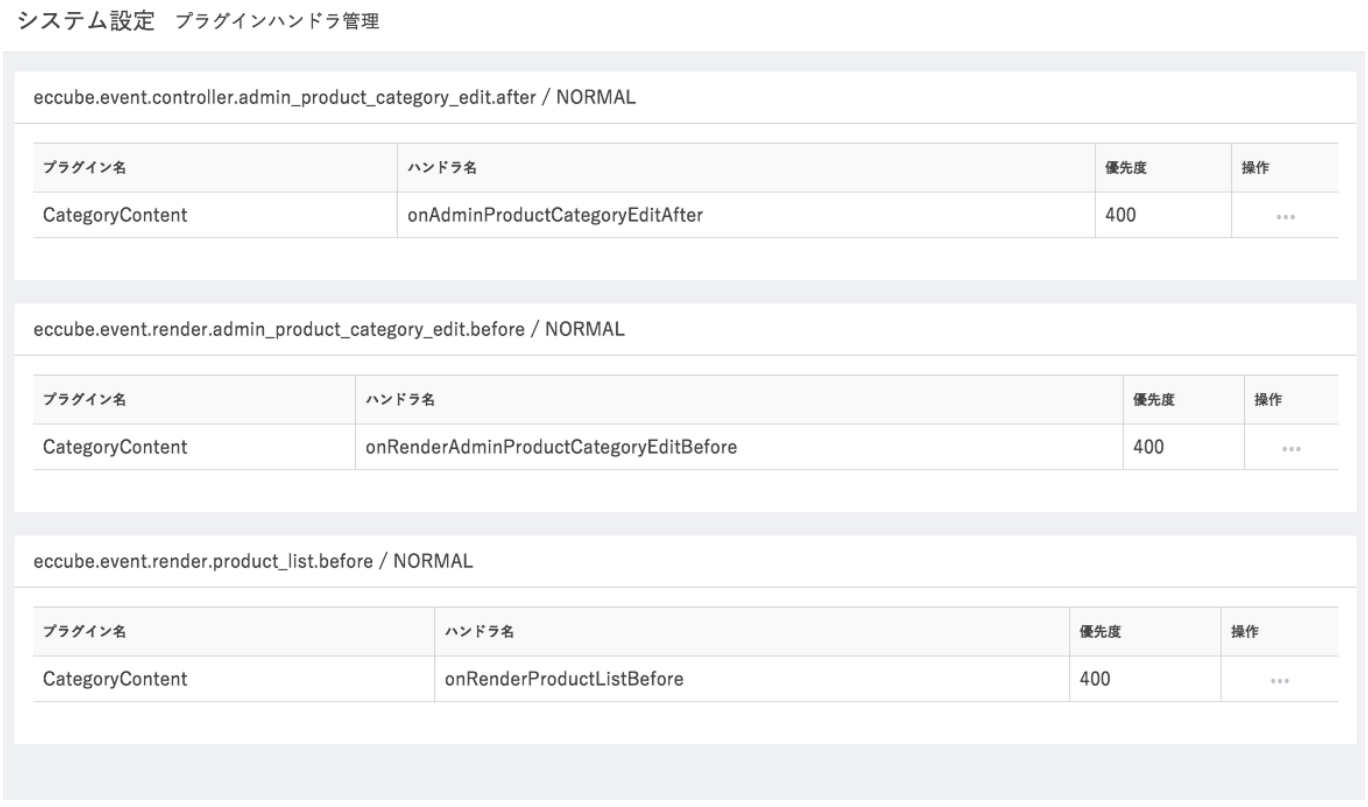
プラグイン開発者への影響

通常型プラグインとの衝突・競合を防ぐため、先発型、後発型のハンドラでは、
渡されたパラメータ(Request/Responseなど)を書き換えないように注意してください。

ハンドラと優先順位

ハンドラ優先度変更画面の動作

各ハンドラの優先度を入力に基づいてアップデートします。
ただし、優先度はハンドラ種別（先発、通常、後発）毎の範囲内に限定されます。



プラグインの設定画面の作成方法

プラグイン一覧画面からプラグイン固有の設定画面を開きたい場合、
以下の名前でルーティング名をServiceProviderに定義することで、
設定画面へのリンクボタンが表示されます。

- ルーティング
  /($app['config']['admin_route'])/plugin/{PluginCode}/config

- ルーティング名
  plugin_{PluginCode}_config

- コントローラ
  Plugin¥{PluginCode}¥Controller¥ConfigController

ここではプラグイン一覧画面に表示する情報のみを定義しているため、
ルーティングの定義や実装は、各プラグインにて行う必要があります。

```
public function register(Application $app)
{
        $app->match(
        '/{$app['config']['admin_route']/plugin/{PluginCode}/config',
        'Plugin¥{PluginCode}¥Controller¥ConfigController::index'
)
->bind('plugin_{PluginCode}_config');
}
```

ライセンス

プラグインのライセンスは基本的に**自由**です。
※プラグインの作成方法によっては、EC-CUBE本体のライセンスに抵触する場合（その場合、強制的にGPLライセンスになります）がありますので、基本ルールにのっとり作成してください。

また、EC-CUBEの商用ライセンスに矛盾するライセンス形態をとった場合、商用ライセンスご購入サイトにプラグインが導入できなくなりますので、注意が必要です。
以下の点をご参照いただき、作成者の判断にてプラグインのライセンスを選択してください。

1. **【推奨】プラグインを無料で配布する場合**（EC-CUBEオフィシャルサイトで配布する場合）
   プラグインのライセンスは商用ライセンスに矛盾しないLGPLライセンスもしくは、LGPLライセンスに矛盾しないライセンス（BSDライセンス、MITライセンス等）を推奨します。
   参照：LGPLについて
   http://ja.wikipedia.org/wiki/GNU_Lesser_General_Public_License
   ※本マニュアルでは例として、LGPLライセンスでの作成方法を記載します。

2. **プラグインを有料配布する等、再配布不可なプラグインにしたい場合**
   プラグインのライセンスは自由に決定することが可能です。
   ただし、商用ライセンスに矛盾するライセンス形態にした場合、商用ライセンスご購入サイトにて使用することができませんので、ご注意いただき、作成者の判断にて作成してください。

3. **プラグインのライセンスを指定しない場合**
   ライセンスを指定しない場合、プラグインのライセンスはEC-CUBE本体のライセンスを継承し自動的にGPLライセンスになります。
   プラグインのライセンスがGPLライセンスの場合、商用ライセンスと矛盾してしまうため、商用ライセンスご購入サイトにプラグインを導入することができなくなります。
   （EC-CUBEをGPLライセンスのまま使用される場合はプラグインがGPLライセンスでも問題なく導入できます。）

5

参考サイト

5.参考サイト

EC-CUBE3の開発に役立つ情報をWEBサイトに公開しております。

1. **EC-CUBE開発コミュニティ**
   http://xoops.ec-cube.net/
   EC-CUBEを利用しているユーザー（開発者・店舗主）の方々のための、コミュニティサイトです。EC-CUBEに関するご意見・ご要望やユーザ様の交流の為のサイトです。

2. **GitHub Wiki/Issues**
   https://github.com/EC-CUBE/ec-cube/wiki
   https://github.com/EC-CUBE/ec-cube/issues
   EC-CUBE3の開発のベースとなっているGitHubのWikiとIssueです。
   開発方針や決定事項等はWikiにて、課題やバグ報告はIssueにて管理しております。

3. **Qiita**
   https://qiita.com/tags/ec-cube3
   主に開発者やコミッターの方々が、EC-CUBEに関するノウハウを公開しているエンジニアSNSサイトです。

4. **EC-CUBE開発ドキュメント**
   http://ec-cube.github.io/
   EC-CUBEの開発ドキュメントです。コーディング規約や機能仕様、バージョンアップ手順などを記載しています。