Seasar Conference 2007
Autumn

Seasar.PHP はじめの一歩

dependency injection for PHP Developer

eXtreme Hardcore PHP Development

Agenda

- 始める前に “DI と AOP”
- Seasar.PHP の概要
- S2Container.PHP5
- S2Dao.PHP5
- Next Step.

始める前に

“DI と AOP”

What's DI ?

- Dependency Injection(依存性の注入)
- ソフトウェアパターンの１つ
- コンポーネント間の依存性を
  プログラムのソースコードから排除する

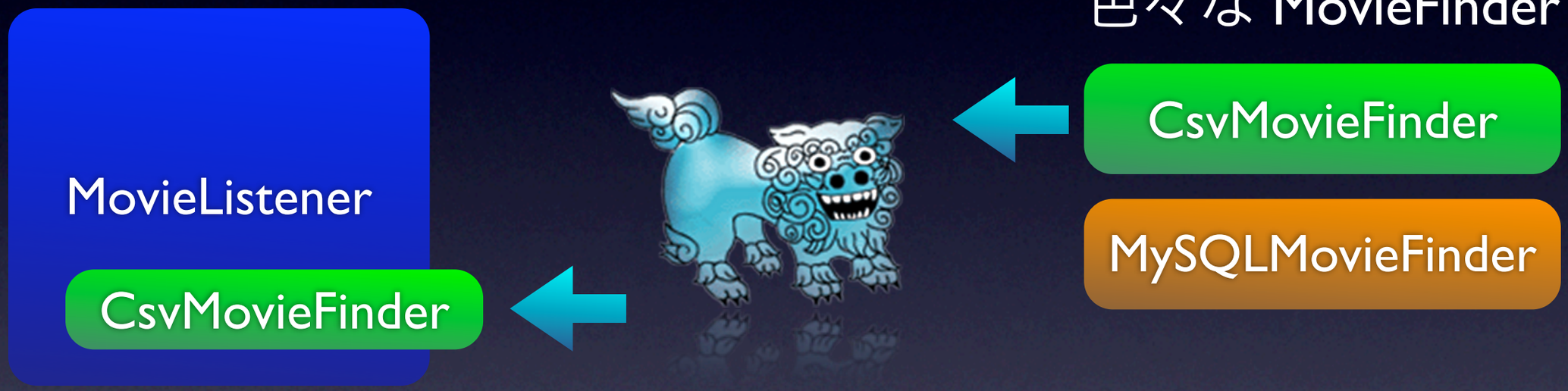
依存性とは

```
class MovieListener
{
    private $finder;
    public function __construct()
    {
        $finder = new CsvMovieFinder('movie.csv');
    }
}
```

CSV ファイルの名前が変わったら？

CSV ではなく MySQL からデータを取得したいと思ったら？

依存性の排除



オブジェクトの生成を”コンテナ”に任せる

別の finder を使いたいと思ったら設定ファイルを書き換えるだけ

What's AOP ?

- Aspect Oriented Programming
- アスペクト指向プログラミング
- 横断的関心事をプログラムのソースコードから分離する

横断的関心事とは

```
class MovieCreator
{
    public function create()
    {
        Logger::log('Call create method.');
    }
}
```

```
class MovieFinder
{
    public function find()
    {
        Logger::log('Call find method.');
    }
}
```

ログ出力のソースコードが複数のクラスに存在する

そもそもログ出力はクラスの本来の仕事とは関係ない処理

横断的関心事の分離

find がコールされたらログ出力コードとfind メソッドのコードを
合体させて実行する

Seasar.PHP の概要

eXtreme Hardcore PHP Development

Seasar.PHPとは

- Java の Seasar2 を PHP に移植
- DI & AOP の機能とそれを利用した開発フレームワーク群
- PHP5 専用

Products

S2Base.PHP5

S2Dao.PHP5

S2AnA.PHP5

S2Container.PHP5

メリット

- メンテナンス性の向上
- プログラムを拡張しやすくなる
- ユニットテストが楽

デメリット

- 開発スピードはあまり早くない
- 作成しなければいけないファイルが多い

S2Container.PHP5

eXtreme Hardcore PHP Development

S2Container.PHP5

- DIコンテナ & AOPフレームワーク
- AutoBinding で設定ファイルの量を軽減
- Class Injection
- ApplicationContext

セットアップ

- PEAR パッケージをインストール
- require_once(‘S2Container/S2Container.php’);
- require_once(‘S2ContainerSplAutoLoad.php’);

はじめての DI

```
$container = S2ContainerFactory::create('app.dicon');
$component = $container->getComponent('finder');
```

```
<components>
    <component name="finder" class="CsvMovieFinder">
        <arg>"movie.csv"</arg>
    </component>
</components>
```

はじめての AOP

```
<components>
    <component name=”finder” class=”CsvMovieFinder”>
        <aspect pointcut=”.*”>
            <component class=”S2Container_TraceInterceptor” />
        </aspect>
    </component>
</components>
```

AutoBinding

```
class MovieViewer
{
    private $message;
    public function __construct(Message $m)
    {
        $this->message = $m;
    }
}
```

```
<component name="impl" class="MessageImpl" />
<component class="MoviewViewer" autoBinding="auto">
  <!--   <arg>impl</arg>   -->
</component>
```

Class Injection

- S2Container-1.2.0 で追加された機能
- S2Container は基本的にインタフェースを実装したクラスしか Injection できない
- S2CONTAINER_PHP5_AUTO_DI_INTERFACE という定数を FALSE (デフォルト) にすると普通のクラスをインジェクションできる

ApplicationContext

- S2Container.PHP5-1.2.0 で追加された機能
- 設定ファイル０で DI & AOP が可能
- クラスにアノテーションとして設定を定義

DI (ApplicationContext 版)

```
/**
 *@S2Component('name' => 'finder')
 */
class CsvMovieFinder implements MovieFinder
{
    /**
     * @S2Binding('movie.csv')
     */
    public function __construct($csv)
    {
        .....
    }
}
```

DI (ApplicationContext 版)

```
S2ContainerApplicationContext::import('path/to/classes');
$container = S2ContainerApplicationContext::create();

if ($container->getComponentDefSize() > 0)
{
    $finder = $container->getComponent('finder');
}
else
{
    echo 'コンテナが空っぽです。';
}
```

AOP (ApplicationContext 版)

```
/**
 * @S2Aspect('interceptor' => 'traceInterceptor',
 *                 'pointcut'      => '.+Find')
 */
class CsvMovieFinder implements MovieFinder
{
    public function movieFind()
    {
        .....
    }
}
```

S2Dao.PHP5

eXtreme Hardcore PHP Development

S2Dao.PHP5

- Seasar.PHP のキラーアプリケーション
- SQL を”書ける” O/Rマッパー
- Return Type による戻り型の柔軟な変更

セットアップ

- PEAR パッケージをインストール
- require_once(‘S2Container/S2Container.php’);
- require_once(‘S2ContainerSplAutoLoad.php’);
- require_once(‘S2Dao/S2Dao.php’);
- S2ContainerClassLoader::import(S2DAO_PHP5);

はじめてのS2Dao.PHP5

1. 接続先のデータベースを設定
2. Entity クラスを定義
3. Dao インタフェースを定義
4. S2DaoInterceptor をアスペクト

接続先のデータベースを設定

```
<!-- pdo.dicon -->

<component name="dataSource" class="S2Container_PDODataSource">
  <property name="dsn">"mysql:host=localhost; dbname=s2con"</property>
  <property name="user">"root"</property>
  <property name="password">"pass"</property>
  <property name="option">
    array(PDO::ATTR_ORACLE_NULLS => PDO::NULL_EMPTY_STRING,
          PDO::ATTR_AUTOCOMMIT => false);
  </property>
</component>
```

Entity クラスを定義

```
class Employee
{
    const TABLE = ‘employee’;
    private $id;
    private $name;
    private $product;
    public function getId() {
        return $this->id;
    }
    public function setId($id) {
        $this->id = $id;
    }
    ...
}
```

Dao インタフェースを定義

```
interface EmployeeDao
{
    const BEAN = ‘Employee’;
    public function insert(Employee $employee);
    public function update(Employee $employee);
    public function delete(Employee $employee);
    public function getAllEmployeesList();
}
```

S2DaoInterceptor をアスペクト

```
<include path=’path/to/dao.dicon’>
<component name=”emp”class=”EmployeeDao”>
        <aspect>dao.interceptor</aspect>
</component>
```

コンテナから取得

```
$container = S2ContainerFactory::create();
$empDao = $container->getComponent('emp');
```

```
$employees = $empDao->getAllEmployeesList();
```

2Way SQL

- 複雑な SQL を別ファイルに記述
- 命名規則により Dao インタフェースが SQL を実行
- IF 等の簡易な制御構文も使用できる

2Way SQL

```
interface EmployeeDao
{
    const BEAN = ‘Employee’;

    public function getEmployeeByIdList($id);
}
```

EmployeeDao_getEmployeeByIdList.sql

```
SELECT * FROM employee WHERE id = /*id*/1
```

Return Type

- 命名規則によってメソッドの戻り値の型を柔軟に変更
- 本家 S2Dao よりも多彩な戻り型
  - List, Array, YAML, JSON

Return Type

```
$result = $dao->getEmployeeByIdJson(1)
echo $result;
```

```
[{"Employee":
 {"id":"1",
  "ID":"1",
  "name":"yonekawa",
  "NAME":"yonekawa",
  "product":"S2AnA.PHP5",
  "PRODUCT":"S2AnA.PHP5"}}]
```

Return Type (アノテーション)

```
/**
 * @return json
 */
public function getEmployeeById($id);
```

Next Step.

Interceptor を自作する

```
class OtherInterceptor extends S2Container_AbstractInterceptor
{
    public function invoke(S2Container_MethodInvocation $invocation)
    {
        // begin method.
        $ret = $invocation->proceed();
        // after method.
        return $ret;
    }
}
```

トランザクション制御

- pdo.dicon にトランザクション制御用の AOP アドバイスが用意されている
  - pdo.requiredTx
  - pdo.requiresNewTx
  - pdo.mandatoryTx
  - pdo.notSupportedTx
- コンポーネントにアスペクトするだけで使える

S2Pager.PHP5

- S2Dao.PHP5 の検索結果をページングするためのクラスやメソッドが用意されたユーティリティライブラリ
- JSON や YAML などの ReturnType にも対応
- S2Dao.PHP5 に同梱

S2AnA.PHP5

- 認証 & 承認用フレームワーク
- AOP の仕組みを使ってアクションを許可したり拒否したりできる
- 開発が停滞中 m(_ _)m

S2Base.PHP5

- Seasar.PHPを簡単に使える環境を用意するコマンドラインツール
- Dao や Entity など必要なファイルを自動生成し、同時にユニットテストも生成
- 既存フレームワークと Seasar.PHP の連携

Seasar.PHPは

皆様の要望・フィードバックを

お待ちしております