AbstractPage

Erklärung

Die AbstractPage Klasse ist im WSC die Mutterklasse aller darzustellenden Seiten. Egal ob es sich um eine statische oder dynamisch erstellte Seite handelt, eine Auflistung oder ein Formular, alle Seiten erben irgendwo von der AbstractPage-Klasse. Aus diesem Grund es es von wesentlicher Bedeutung die Funktionsweise dieser Klasse zu kennen und zu verstehen.

Table Of Contents

- 1 Erklärung
- 2 Die AbstractPage auf GitHub
- 3 Aufbau der Datei
  - 3.1 Reihenfolge der Funktionen in der AbstractPage
  - 3.2 readParameters()
  - 3.3 readData()
  - 3.4 assignVariables()
  - 3.5 Template
- 4 Weitere Klassenvariabeln
- 5 Komplettes Beispiel

Die AbstractPage auf GitHub

github.com/WoltLab/WCF/blob/ma…ge/AbstractPage.class.php

Aufbau der Datei

Stellen wir uns vor, wir möchten einen Shop entwickeln. Der sinnvollste Weg um ein einzelnes Produkt darzustellen, wäre die Implementierung einer Vererbung der AbstractPage, die wir in der Datei ProductPage.class.php benennen werden. Die Datei muss zwangweise über das files-PiP installiert werden, da diese sonst dem WSC nicht bekannt ist und wir diese nicht aufrufen können. Darüber hinaus muss sie sich im Ordner `Endanwendung/lib/page/` befinden. Für das grobe Grundgerüst der Datei reicht es aus, die Klasse wie folgt aufzubauen:

PHP Source Code

```
1. <?php
2. namespaceshop\page;
3. usewcf\page\AbstractPage;
4. /**
5. * Shows the example page.
6. *
7. * @author Marcel Beckers
8. * @license GNU Lesser General Public License <http://opensource.org/licenses/lgpl-license.php>
9. * @package de.yourecom.example
10. */
11. classProductPageextendsAbstractPage{
12. }
```

Display All

In Zeile 2 vergeben wir den Namespace (Namensraum) zu der Datei.
In Zeile 3 "binden" wir die Datei AbstractPage ein.
In Zeile 12 vergeben wir den Namen der Klasse, der gleichlautend mit dem Dateinamen ohne `".class.php"` sein muss und erben von der AbstractPage.
Mit dem reinem Aufruf dieser Seite führen wir nun jede Menge Funktionen innerhalb der AbstractPage aus, die wir im Anschluss nach und nach beleuchten werden.

Reihenfolge der Funktionen in der AbstractPage

Die AbstractPage-Klasse erbt von der Interface-Klasse IPage. Dort kann man schön erkennen, wann welche Funktion aufgerufen wird. Das kann wichtig sein, wenn man bestimmtem Code zu einem bestimmten Zeitpunkt ausführen lassen will.

Als erstes wird die Funktion `readParameters();` aufgerufen. Hier kann man in einem Link übergebene Variablen abfragen.

Anschließend wird die Funktion `checkModules();` aufgerufen. Hier kann man festlegen, welche Module oder Optionen

den Wert 1 bzw. TRUE haben müssen, damit die Seite aufgerufen werden kann.

Danach kommt die Funktion `checkPermissions();`. Hier können dann die benötigten Benutzergruppen-Rechte abgefragt werden.

Anschließend wird die Funktion `readData();` aufgerufen. Hier kann man zum Beispiel die Daten die schon vorhanden sind weiter bearbeiten oder auch SQL-Abfragen starten, um für die weitere Bearbeitung alle Informationen zu erhalten.

Nun nachdem wir alles zusammen haben an Informationen, die dann auf der Seite angezeigt werden sollen, kommt die Funktion `assignVariables();`. Hier kann man die einzelnen Variable auf das Template übertragen, damit wir dort die hinter den Variablen gespeicherten Informationen anzeigen können.

und zum Abschluß kommt die Funktion `show();`. Hier wird unter anderem (sofern nötig) das Menü-Icon aktiviert (mit der Funktion `UserMenu::getInstance()->setActiveMenuItem('wcf.user.menu.XXXXXXXX');`) und natürlich das Template aufgerufen durch den Durchlauf der Angabe `parent::show();`.

Somit sollte der Ablauf für die AbstractPage dargestellt sein und vor allem die Reihenfolge der einzelnen Funktionen.

readParameters()

Die Funktion readParameters ist nach der`__construct()` und `__run()` Funktion die erste Funktion, die aufgerufen wird. Diese Funktion dient beispielsweise dazu Parameter, die über die URL geliefert werden (Stichwort $_REQUEST Variable), auszulesen.

PHP Source Code

```
1. /**
2. * @see \wcf\page\IPage::readParameters()
3. */
4. public functionreadParameters() {
5. parent::readParameters();
6. // read productID parameter
7. if (isset($_REQUEST['id']))$this->productID=intval($_REQUEST['id']);
8. }
```

Wie man durch den `parent::readParameters()` Aufruf sieht, überschreibe ich an dieser Stelle nicht die Funktion der Mutterklasse, sondern erweitere diese nur. Hintergrund ist, dass in jeder Funktion der AbstractPage Klasse ein Event für EventListener abgefeuert wird, so dass man sich mit einem Drittplugin hier einklinken kann.
Da an dieser Stelle eine ID aus der URL ausgelesen wird, empfiehlt es sich, sofort die Variable mit `intval` zu bearbeiten, um so sicherzustellen, dass man definitiv mit einer Variablen mit dem Typ Integer arbeitet.

readData()

Nach der Funktion `readParameters()` wird die Funktion `show()` aufgerufen, die zur Aufgabe hat alles abzuarbeiten was zur Anzeige gehört. Die einzelnen Aufgaben sind wiederum in weiteren Funktionen aufgeteilt.
Die für uns erste relevante Funktion ist die `readData()`-Funktion, die zum Auslesen von Daten aus der Datenbank oder des Caches gedacht ist.

PHP Source Code

```
1. /**
2. * @see \wcf\page\IPage::readData()
3. */
4. public functionreadData() {
5. parent::readParameters();
6. //get product
7. $this->product= newProduct($this->productID);
8. if (!$this->product->productID) {
9. throw newIllegalLinkException();
10. }
```

```
11. }
```

Display All
In dieser Funktion holen wir nun die Produktinformationen aus unserer Datenbank durch unsere DatabaseObject Klasse für unser Produkt ab. Danach prüfen wir, ob die Klasse Product mit der angegebenen productID überhaupt ein Produkt finden konnte und werfen ggf. eine IllegalLinkException, um dem Anwender zu sagen, dass er einen ungültigen Link aufgerufen hat.

assignVariables()

Nachdem die `readData()`-Funktion erfolgreich die Daten aus der Datenbank abgefragt hat, können wir die Daten nun durch die Funktion `assignVariables()` an unser Template zuweisen, damit das Template auch unser Produkt anzeigen kann.

PHP Source Code

```
1. /**
2. * @see wcf\page\IPage::assignVariables()
3. */
4. public functionassignVariables() {
5. parent::assignVariables();
6. WCF::getTPL()->assign(array(
7. 'product'=>$this->product
8. ));
9. }
```

In der Funktion bringen wir unserem Template nun bei, dass wir im Template eine Variable namens `$product` haben möchten, die den Wert von `$this->product` erhällt. Selbstverständlich können an dieser Stelle auch mehrere Werte an das Template übergeben werden:

PHP Source Code

```
1. WCF::getTPL()->assign(array(
2. 'product'=>$this->product,
3. 'foo'=>$this->foo,
4. 'bar'=>$this->bar
5. ));
```

Tipp: In meinen Plugins haben die Namen der Variablen im Template immer denselben Namen wie in der PHP Datei. So kann man sie sich besser merken und vor allem viel leichter nachverfolgen.

Template

Nachdem wir dem Template die Werte nun zugewiesen haben, versucht die `show()`-Funktion eigenständig ein passendes Template zu finden und im Erfolgsfall einzubinden. Um diesen Automatismus zu unterstützen, muss sich das Template in derselben Endanwendung befinden, wie unsere ProductPage.class.php und gleich benannt sein, jedoch ohne "Page.class.php". Zudem muss der erste Buchstabe des Templatenamens kleingeschrieben werden. In unserem Fall hieße das Template, das eingebunden werden kann `"product.tpl"`.

Weitere Klassenvariabeln

Name	Typ	Beschreibung	Besonderheit
$activeMenuItem	string	Gibt den Namen des Menüpunktes an, der als aktiv gesetzt werden soll.	-

$action	string	Gibt den Namen für eine Aktion an (I.d.R verwendet man es zur Unterscheidung von Add oder Edit Formularen).	-
$canonicalURL	string	Gibt die kanonische URL an.	-
$forceCanonicalURL	boolean	Gibt an, ob eine kanonische URL auch beim vorhandensein von $_POST Daten erzwungen werden soll.	ab wsc 3.0
$enableTracking	boolean	Gibt an, ob der Aufenthaltsort auf dieser Seite verraten werden darf.	nur wcf 2.1
$loginRequired	boolean	Gibt an, ob der Besucher dieser Seite eingeloggt sein muss.	-
$neededModules	array	Gibt Optionen an, die aktiviert sein müssen, um diese Seite aufrufen zu können.	-
$neededPermissions	array	Gibt Benutzergruppenrechte an, die vorhanden sein müssen, um die Seite aufrufen zu können.	-
$templateName	string	Gibt den Namen des Templates zu dieser Seite an, wenn man die automatische Einbindung nicht nutzen möchte.	-
$templateNameApplication	string	Gibt die Endanwendung für $templateName an.	-
$useTemplate	boolean	Gibt an, ob überhaupt ein Template genutzt werden soll.	-

Komplettes Beispiel

PHP Source Code: ProductPage.class.php

```
1. <?php
2. namespacewcf\page;
3. usewcf\data\product\Product;
4. usewcf\page\AbstractPage;
5. usewcf\system\exception\IllegalLinkException;
6. usewcf\system\WCF;
7. /**
8. * Shows the example page.
9. *
10. * @author Marcel Beckers
11. * @license GNU Lesser General Public License <http://opensource.org/licenses/lgpl-license.php>
12. * @package de.yourecom.example
13. */
14. classProductPageextendsAbstractPage{
15. /**
16. * product ID
17. * @var integer
18. */
19. public$productID=0;
20. /**
21. * @see \wcf\page\IPage::readParameters()
22. */
23. public functionreadParameters() {
24. parent::readParameters();
25. // read productID parameter
26. if (isset($_REQUEST['id']))$this->productID=intval($_REQUEST['id']);
27. }
```

```
31.  /**
32.  * @see \wcf\page\IPage::readData()
33.  */
34.  public functionreadData() {
35.  parent::readParameters();
36.  //get product
37.  $this->product= newProduct($this->productID);
38.  if (!$this->product->productID) {
39.  throw newIllegalLinkException();
40.  }
41.  }
42.  /**
43.  * @see wcf\page\IPage::assignVariables()
44.  */
45.  public functionassignVariables() {
46.  parent::assignVariables();
47.  WCF::getTPL()->assign(array(
48.  'product'=>$this->product
49.  ));
50.  }
51.  }
```

Display All

Ihr habt Fragen oder Anregungen? Lob oder Kritik? Lasst es mich doch durch einen Kommentar von euch wissen!