Autoload

Seit PHP5.1 können Programmierer auf die __autoload Funktion in PHP zurückgreifen. Diese Funktion erleichtert dem Programmierer vorallem bei großen Projekten die Arbeit. Die __autoload Funktion wird automatisch aufgerufen, wenn im PHP Quelltext eine eigene Klasse verwendet wird. Sinn und Zweck dieser Funktion ist entsprechend, dass benötigte Klassen automatisch eingebunden werden müssen und die jeweilige Datei nicht händisch eingebunden werden muss. In diesem Artikel möchte ich euch näher erläutern mit welchen Konzepten man die __autoload Funktion in euer Projekt einbinden könnt. Der Eintrag ist für fortgeschrittene Programmierer ausgelegt.

Einfacher Aufbau (nicht empfehlenswert!)

Als erstes Konzept möchte ich euch die einfachste Umsetzungsmöglichkeit vom Autoloader zeigen. Bei diesem Konzept muss der Array mit der Information in welcher Datei welche Klasse abgelegt ist ständig händisch erweitert werden. Bei einem kleinen Projekt mag das noch machbar sein, jedoch kann man bei kleinen Projekten auch einfach ein include / require_once machen. Dieses Beispiel soll also erstmal nur den grundlegenden Aufbau von __autoload veranschaulichen und stellt keine gute Lösung dar.
Die __autoload Funktion

Quellcode

```
1.  /**
2.  * __autoload will be called automatically if requiered class isn't included yet
3.  *
4.  * @param string $class name of requiered class
5.  */
6.  function __autoload($class) {
7.  $classes = array();
8.  $classes['Database'] = "include/db/Database.class.php";
9.  $classes['Template'] = "include/db/Template.class.php";
10. if(isset($classes[$class])) {
11. require_once $classes[$class];
12. }
13. }
```

Alles anzeigen

"classes"-Ordner

Diese Version arbeitet automatisch und die Funktion muss nicht erweitert werden. Bei kleineren Projekten kann diese Version durchaus noch effektiv genutzt werden, sobald die Anzahl der eigenen Klassen jedoch recht groß wird, ist dieses Konzept unübersichtlich.
Bei dieser Umsetzung ist es wichtig, dass jede Klasse sich in einer eigenen PHP-Datei befindet und der Dateiname muss mit dem Klassennamen (vorallem Groß- und Kleinschreibung) identisch sein. Die Dateiendung ist in diesem Fall .class.php und diese Dateien müssen im Ordner "classes" abgelegt werden.

Beispiele:
Klasse: Database
Dateiname: Database.class.php
Beispielhafter Inhalt der Datei Database.class.php

Quellcode

```
1.  class Database {
2.  /**
3.  * constructor
4.  * open new database connection with given dbms
5.  *
```

```
 * @param string $dbms database management system
 * @param string $host host of dbms
 * @param string $user username for database conneciton
 * @param string $pass password for given user
 * @param string $db automatically selected database
 */
public function __construct($dbms, $host, $user, $pass, $db) { }
}
```

Alles anzeigen

Die __autoload Funtkion

Quellcode

```
function __autoload($class) {
$directory = "classes";
$filename = $class.".class.php";
if(file_exists($directory.DIRECTORY_SEPARATOR.$filename)) {
require_once $directory.DIRECTORY_SEPARATOR.$filename;
}
}
```

Namensräume und __autoload()

Auch die seit PHP 5.3 verfügbaren Namensräume können in Verbindung mit __autoload() genutzt werden, da immer der vollqualifizierte Klassenname übergeben wird. Das heißt, dass, wenn ein Objekt der Klasse \kernel\configuration\ConfigurationManager erzeugt wird, immer dieser Name übergeben wird - auch wenn die Klasse z.B. in einen Namensraum importiert wurde und über einen anderen Namen angesprochen wird.

Sofern die Namensräume der Struktur der Dateien (in der die Klassen enthalten sind) entsprechen, können die Klassen aus verschiedenen Verzeichnissen geladen werden. Für die Verständlichkeit folgt ein Beispiel.

kernel/configuration/ConfigurationManager.php

Quellcode

```
namespace kernel\configuration; // der Namensraum entspricht dem Pfad
class ConfigurationManager {
// ...
}
```

kernel/registry/Registry.php

Quellcode

```
namespace kernel\registry;
class Registry {
// ...
}
```

bootstrap.php

Quellcode

```
function __autoload($class) {
// für Linux: \ in / umwandeln und die Dateiendung anhängen
```

```
3. $class = str_replace('\\', '/', $class).'.php';
4. if(!file_exists($class)) {
5. throw new Exception('...', E_USER_ERROR);
6. }
7. include $class;
8. }
9. $configMgr = new kernel\configuration\ConfigurationManager();
10. $registry = new kernel\registry\Registry();
```

Alles anzeigen

Komplexer Lösungsansatz

Der zweite Lösungsansatz arbeitet ohne weitern Aufwand, aber wie ihr sicherlich schon festgesetllt habt, gibt es ein ziemlich großes Chaos, wenn viele Klassen hat, da diese nicht in einzelne Unterordner abgespeichert werden können. Das bedeutet, dass wir eine größere Ordnerstruktur auslesen müssen und für die Performance ist es in diesem Fall auch sehr wichtig, dass die Ordnerstruktur nicht bei jedem Aufruf von autoload durchforstet werden muss.

PHP bietet für die komplxere Verwendung auch die Möglichkeit eine Autoload-Klasse anstatt einer einfachen Funktion zu verwenden. Dazu muss jedoch mit dem Befehl spl_autoload_reigster definiert werden welche Klasse und welche darin enthaltene statische Methode die Autoload Funktion darstellt. In meinem Beispiel verwende ich dazu die Klasse Autoload und die statische Methode get.

Quellcode

```
1. spl_autoload_register(array('AutoLoad', 'get'));
```

Wie beim vorherigen Konzept muss wieder jede Klasse in einer eigenen Datei sein und der Dateiname muss entsprechend vom Klassennamen gewählt werden. Beispiel: Die Klasse Template befindet sich in der Datei Template.class.php

Die Autoload-Klasse

Quellcode

```
1. abstract class Autoload {
2. /**
3. * file paths of classes
4. *
5. * @var array $files
6. */
7. private static $files;
8. /**
9. * cache file
10. *
11. * @var string $cache
12. */
13. private static $cache = "Autoload.cache.php";
14. /**
15. * import cached file array
16. *
17. */
18. private static function init() {
19. if(file_exists(self::$ache)) {
20. self::$files = unserialize(file_get_contents(self::$cache));
21. }
22. else self::$files = array();
23. }
```

```
28.  /**
29.  * include given class
30.  *
31.  * @param string $class name of requiered class
32.  * @param bool $firsttime[optional] first call for given class
33.  */
34.  public static function get($class, $firstime=true) {
35.  // load cache into local variable
36.  if(!is_array(self::$files)) self::init();
37.  if(isset(self::$files[$class])) {
38.  require_once self::$files[$class];
39.  return;
40.  }
41.  if($firsttime) {
42.  $files = self::buildCache();
43.  file_put_contents(self::$cache, serialize($files));
44.  self::get($class, false);
45.  return;
46.  }
47.  // here you could throw an exception or output other errors if you want
48.  }
49.  /**
50.  * read filesystem recursive and build class array
51.  *
52.  * @param string $directory[optional]
53.  * @return array
54.  */
55.  private static function buildCache($directory='.') {
56.  $files = array();
57.  $dir = dir($directory);
58.  while($item = $dir->read()) {
59.  if($item == '.' || $item == "..") {
60.  continue;
61.  }
62.  if(is_directory($item)) {
63.  $found = self::buildCache($directory.DIRECTORY_SEPARATOR.$item);
64.  $files = array_merge($files, $found);
65.  }
66.  if(is_file($item) && preg_match("#(.+)\.class\.php#", $item, $match) {
67.  $class = $match[1];
68.  $files[$class] = $directory.DIRECTORY_SEPARATOR.$item;
69.  }
70.  }
71.  return $files;
72.  }
```

Alles anzeigen

Ich hoffe ich konnte euch mit diesem Eintrag behilflich sein und bei Fragen könnt ihr hier im Forum gerne ein neues Thema erstellen - es gibt genug Experten, die euch weiter helfen können 🙂