Verbindung zur Datenbank: Eine Datenbankklasse

Objektorientiertes PHP

Anders als Java erzwingt PHP nicht, dass der Anwender objekorientiert programmiert, PHP stellt dennoch alle Werkzeuge objektorientierter Programmierung zur Verfügung. Auch Dokuwiki selbst ist objektorientiert implementiert, so ist beispielsweise die `syntax.php` unseres Plugins eine von der Klasse `DokuWiki_Syntax_Plugin` abgeleitete Klasse:

```
class syntax_plugin_projekt extends DokuWiki_Syntax_Plugin
{
[...]
}
```

Datenbank-Klasse

Wir lagern nun den Zugriff auf die mysql-Datenbank in eine eigene Klasse aus.

(A1)

Erstelle eine Datei `mysqldb.php` in deinem Plugin-Verzeichnis mit folgendem Inhalt:

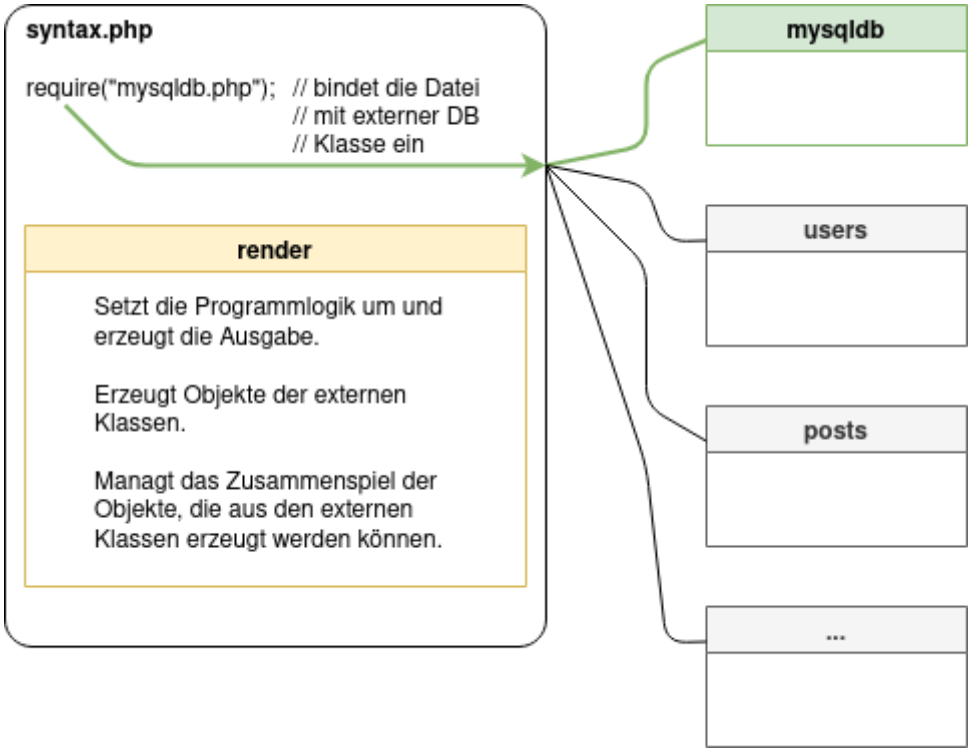
mysqldb.php

```
<?php

class mysqldb {

    /**
     * Constructor: Connect to db, return handle
     *
     * @param string        $dbusername   DB username
     * @param string        $dbpassword   DB password
     * @param string        $dbname       Database to connect to
     * @param string        $host         Database host to connect to
(optional)
     *
     * @return object       DB-Handle
```

```
     */
    function __construct($dbusername, $dbpassword, $dbname,
$host="localhost" ) {

    try {
        $pdo = new PDO("mysql:host=$host;dbname=$dbname",
"$dbusername", "$dbpassword");
        $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    } catch ( PDOException $e ) {
        echo 'Verbindung zur Datenbank fehlgeschlagen: ' .
$e->getMessage();
        return FALSE;
    }

    $this->connection=$pdo;

    }
}

?>
```

Wenn man ein neues `mysqldb` Objekt erzeugt, benötigt der Konstruktor als Argumente den Datenbankbenutzer, dessen Passwort, den Namen der Datenbank und optional einen Datenbank-Host.

Damit wir mysqldb-Objekte (und später vielleicht weitere Objekte) nutzen können müssen uns klar machen, wer in unserem Plugin die Rolle der "Steuerklasse" übernimmt. Am einfachtsen ist es, diese Funktion an die render-Methode in der Datei `syntax.php` zu delegieren. Zunächst werden dort alle Operationen ausgeführt, die nötig sind, um alle Informationen zu sammeln, dann wird – möglicherweise unter Verwendung weiterer Methoden und/oder Objekten – die HTML Ausgabe erzeugt, die dann im Wiki ausgegeben wird.



(A2)

Binde im Kopf der Datei `syntax.php` die `mysqldb.php`-Datei ein. Informiere dich, was die Einbindung mit dem Befehl `require` bewirkt. Warum sollte man hier nicht `include` verwenden?

```
[...]
// must be run within Dokuwiki
if (!defined('DOKU_INC')) {
    die();
}

// Klassendateien einbinden
require("mysqldb.php");

class syntax_plugin_projekt extends DokuWiki_Syntax_Plugin
{
[...]
```

- Ergänze innerhalb der `render`-Methode Code, der eine Datenbank Verbindung erzeugt.
- Mache dir klar, welche Funktion das dabei erzeugte Objekt `$dbhandle` im weiteren Verlauf des Programms hat.
- Teste, was passiert, wenn du eine falsches Benutzer/Passwort-Kombination, eine nicht existente Datenbank oder einen anderen Host als `localhost` angibst.

```
$mydb = new mysqldb("DBUSER", "dbuserPASS", "DBNAME");
```

Plugin-Konfiguration

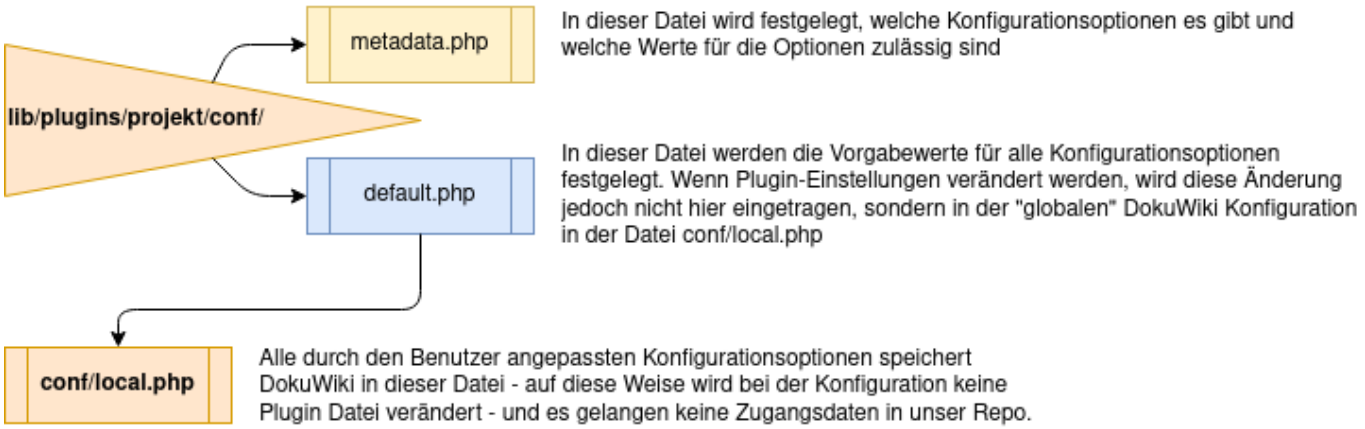
Sehr unschön ist jetzt natürlich, dass die datenbank Credentials im Quelltext des Plugins stehen - auf diese Weise kann man ein solches Plugin schlecht veröffentlichen oder weitergeben. Wesentlich sind hier 2 Implikationen:

1. Das kollidiert massiv mit der Versionsverwaltung: Man sollte unbedingt vermeiden, Benutzernamen und Passwörter in öffentlich einsehbare Git-Repos einzuchecken.
2. Wenn ein anderer Benutzer in ferner Zukunft das Plugin in seinem eigenen Wiki verwenden möchte, muss dieser, um das Plugin für sich nutzbar zu machen den Quelltext editieren um dort seine eigenen Datenbank Zugangsdaten einzutragen - das ist aus vielen Gründen Mist, einer ist z.B.: wie das Plugin jetzt auf neue Versionen aktualisiert werden soll, denn dabei würden diese Änderungen ja jedes mal wieder rückgängig gemacht.

DokuWiki bietet für dieses Problemfeld die Möglichkeit, Plugin-Einstellungen über das DokuWiki Webinterface festlegen zu können - das wollen wir im Folgenden für unser Plugin umsetzen.

Konfigurations-Konventionen

Im Plugin-Ordner gibt es ein Unterverzeichnis `conf` in diesem befinden sich zwei Dateien: `default.php` und `metadata.php`, die Funktionsweise zeigt das folgende Schaubild:



Wenn wir jetzt also unser Plugin durch DokuWiki "konfigurierbar machen wollen, müssen wir drei Fragen beantworten:

1. Welche Optionen brauchen wir und was müssen wir diesbezüglich in die Datei `metadata.php` schreiben?
2. Was sollen die Vorgabewerte für diese Optionen sein und was muss in `default.php` stehen?
3. Wie können wir im Quelltext des Plugins auf die Werte unserer Konfigurationsoptionen zugreifen?

Die **Optionen** liegen auf der Hand: DB-Username, DB-Passwort, DB-Name und DB-Host. Sinnvolle **Vorgabewerte** sind "leer", außer beim DB-Host, da dürfte `localhost` Sinn machen. Alle Optionen sind als Zeichenketten einzugeben.

In den Dateien `default.php` und `metadata.php` stellt sich das dann wie folgt dar (zunächst nur für den DB-Usernamen):

metadata.php

```php
$meta['dbusername'] = array('string');
```

default.php

```php
$conf['dbusername'] = ''; // leerer String
```

Die vollständige Entwicklerdokumentation findet sich hier, dort kann man auch die unterschiedlichen Konfigurationstypen für die `metadata.php` nachschlagen.

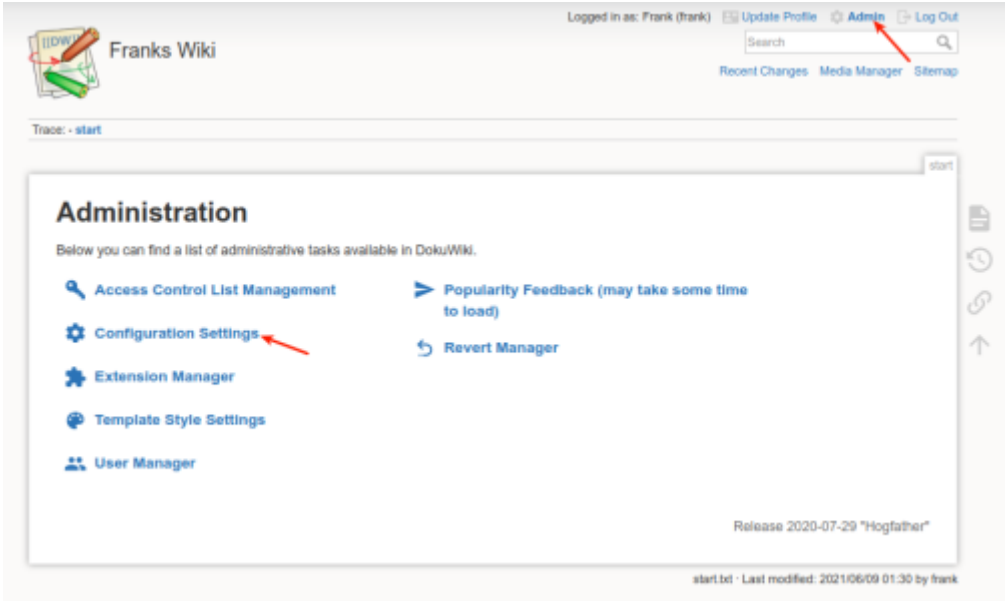
Zugriff auf die Werte der Plugin-Optionen erfolgt mit

```php
$dbuser = $this->getConf('dbusername');
```

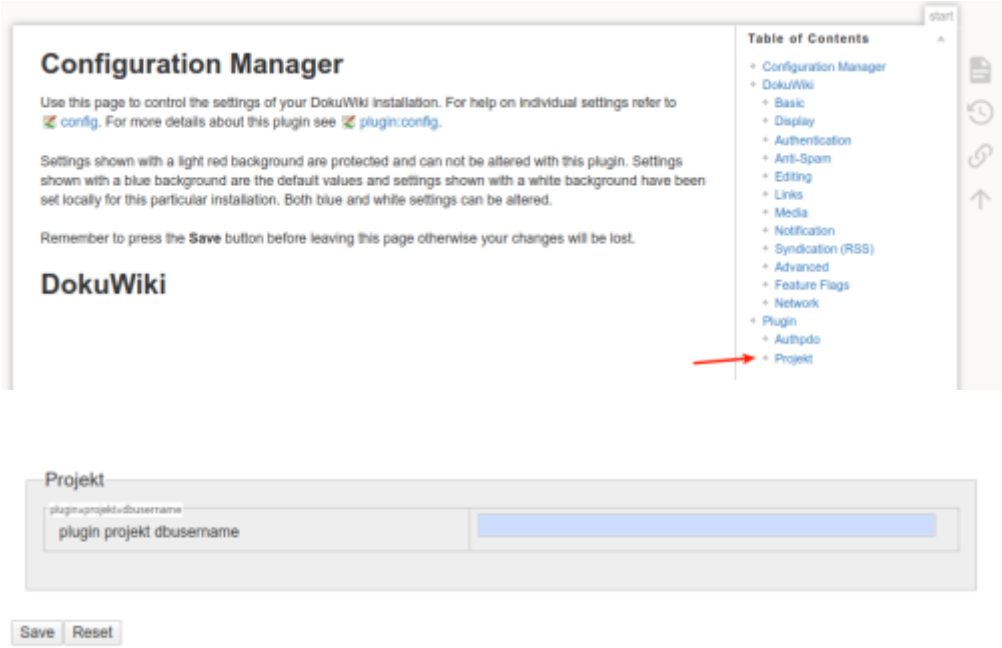
(A3)

Füge die Informationen aus dem vorigen Abschnitt deinem Plugin hinzu und mache auf diese Weise den Datenbankbenutzer konfigurierbar.

Teste die Änderungen, indem du den Konfigurationsmanager öffnest:



Dort sollte es jetzt einen Abschnitt für das Plugin `projekt` geben, in dem die Plugin-Optionen angezeigt werden.



Trage deinen DB-Benutzernamen dort ein und speichere die Einstellungen.

Modifiziere dann deine `syntax.php` so, dass der Datenbankbenutzer, der dem Konstruktor der DB-Klasse übergeben wird, aus den Einstellungen ausgelesen wird.

Teste, ob die Datenbankverbindung immer noch ordnungsgemäß zustande kommt.

Ergänze die weiteren Optionen für Passwort, Datenbankname und Datenbankhost, konfiguriere das Plugin mit den Werten, die für deinen Benutzer passen und teste die Verbindung erneut.

Hilfestellung

In der `syntax.php` kann das in etwa so aussehen - die Optionsnamen können natürlich abweichen, je nachdem wie du diese benannt hast.

```
// get settings
$dbuser = $this->getConf('dbusername');
$dbpasswd = $this->getConf('dbpasswd');
$dbname = $this->getConf('dbname');
$dbhost = $this->getConf('dbhost');
//make db connection
$dbhandle = new mysqldb($dbuser, $dbpasswd, $dbname, $dbhost);
```

Modellierung - fällt aus

Eigentlich sollte man sich an dieser Stelle überlegen, wie man seine Problemstellung (objektorientiert) modellieren möchte, das fällt uns etwas schwer, weil wir noch keine Problemstellung haben. Ein paar Überlegungen kann man an dieser Stelle dennoch anstellen.

Eine grundlegende Frage könnte z.B. sein, wie man die `mysqldb`-Klasse weiter entwickelt: Man kann weitere Methoden in der mysqldb-Klasse implementieren, die Abfragen oder Manipulationen an der

Datenbank ermöglichen. Man könnte solche Programmfunktionen jedoch auch (wie im Schema oben angedeutet) nach Objektkategorien zusammengefasst auf weitere Klassen verteilen. Das Schema demonstriert ein solches Vorgehen für ein Micro-Blog mit Benutzern, die Posts verfassen können. In diesem Setting liefert die mysqldb-Klasse lediglich das DB-Handle, das seinerseits dann an die Methoden in den user- und posts-Klassen weitergegeben wird, so dass diese die entsprechenden Funktionen - auch auf der Datenbank - erfüllen können.

Wir entwickeln zunächst weitere Methoden innerhalb unserer mysqldb-Klasse - wenn sich unsere Problemstellung konkretisisert können wir im Zuge eines Code-Refactoring weitere Aufteilungen und Modellierungsschritte vornehmen.