Simpel, alsidig, objekt-orienteret pagination (paging) i PHP

Der findes masser af bud på, hvordan man kan lave paging (pagination) i PHP. Nogle løsninger er håbløse, andre er for specifikke - og de sidste er som regel for komplekse! I det følgende præsenteres en simpel OO-tilgang, der kan håndtere de fleste behov.

Skrevet den **03. Feb 2009** af **jensgram** I kategorien **Programmering / PHP** | ★★★☆☆

[Introduktion]

I det følgende behandles emnet "paging", sideopdeling eller "pagination" (jeg benytter sidstnævnte betegnelse): Det vil sige det problem der opstår, når data fra en database skal udtrækkes i mindre portioner og serveres på individuelle sider - som eksempelvis artiklerne her på Eksperten. I og for sig er logikken ikke ret kompleks. Det store problem er derimod, at man ofte finder sig selv i færd med at kopiere store stykker kode fra pagination af gæstebogen til brugerlisten til kommentarerne til ...

Formålet med denne artikel er at gennemgå tankerne bag en simpel abstrakt PHP-klasse, der kan tage sig af alt det kedelige så du blot skal beskrive, hvordan de enkelte elementer skal renderes. (Kode følger løbende - de nysgerrige kan scrolle til bunden, hvor den samlede klasse forefindes.). Hermed rustes læseren til at tage det sidste skridt og implementere løsningen på site eget site.

Artiklen kan ved første øjekast synes lang, hvilket skyldes, at kode og forklaring følges ad. Kan du undvære forklaringen, så skimlæs og hæft dig ved den samlede kode nær artiklens bund.

(Det skal i parentes bemærkes, at jeg *er* klar over, at der på Eksperten findes mindst to andre artikler om dette emne. At dømme ud fra kritikken, lader det imidlertid ikke til at kunne skade med et nyt perspektiv.)

[De overordnede problemstillinger]

Jeg vil for det følgende angribe pagination-problematikken som to sammenhængende udfordringer:

1. Håndtering af dataudtræk fra databasen (heri benyttes MySQL; arbejder du med en anden database vil du sikkert let kunne tilpasse koden), samt opdelingen i sider

2. Problemet, der opstår ved store kollektioner, der skaber *mange* sider og derved en uoverskuelig navigation.

Sidstnævnte er i nogen grad en outsider (ihvertfald i forhold til den gængse artikel om pagination), så for at sikre fællesforståelse med læseren, tillader jeg mig at uddybe: Antag at vi har 1.800 poster i databasen og vil præsentere ti ad gangen. Den simple løsning er at liste 180 links med siderne 1-180. *Meget* uoverskueligt og derved komplet uanvendeligt.

I stedet har det været et krav til den til artiklen udviklede løsning, at sidenavigationen ville se ud à la ***1*** **[2] [3] ... [180]**, hvor asterisk indikerer den aktuelle side, mens kantede parenteser repræsenterer sidelinks. Ellipsen (...) er bogstavelig. Befinder vi os på side 65 (posterne 650-9) skal navigationen antage formen **[1] ... [63] [64] *65* [66] [67] ... [180]** etc.

[En objekt-orienteret tilgang]

Den objekt-orienterede tilgang afspejler den måde, hvorpå mennesket ser verden. Som introduktion til OO et det oplagt (og udbredt) at lave eksempler på nedarvningshierarkier, tilstand og funktionalitet på fænomener og begreber fra hverdagen, e.g. køretøjer, cykler og biler. Nogle gange er vi imidlertid tvunget til at modellere begreber, der ikke kan hentes fra omgivelserne, men som snarere er udtryk for en formaliseringsproces fra udviklerens side. I dette tilfælde eftersøger vi et begreb, der kan konkretiseres til noget, derkan håndtere pagination - resultat: Vi modellerer og udvikler en "**Paginator**":

```
<?php

abstract class Paginator {
    abstract protected function renderItem(&$row);
}

?>
```

Som omtalt ovenfor, kan stort set hele logikken bag pagination genbruges på tværs af kontekster. Faktisk er det kun et spørgsmål om at implementere en konkret renderingsmekanisme til hver af disse. Det kunne være en kommentar-rendering, gæstebogsindlæg-rendering etc. Som følge af denne betragtning har jeg valgt, at **Paginator**-klassen skal være abstrakt, idet metoden **renderItem()** skal realiseres til hver kontekst. Jeg skal senere vende tilbage til denne metode.

En **Paginator** skal i al sin enkelthed kunne håndtere følgende scenarie:

1. Find det totale antal af elementer, der skal gøres til genstand for pagination

2. Udregn det samlede antal sider = total / elementer-per-side

3. Sæt aktuel side som intern tilstand. Er ingen side angivet, antages det, at vi befinder os på første side (som internt er side nul).

4. Tilbyd rendering af den aktuelle sides elementer.

5. Tilbyd rendering af *relevante* navigationsmuligheder, jf. pkt. 2 i foregående afsnit.

Som det altid gør sig gældende, er der behov for **tilstand** og **funktionalitet**, så lad os først tilføje de nødvendige metodesignaturer:

```
<?php

abstract class Paginator {
    public function renderItems() { ... }

    public function renderNavigation() { ... }

    abstract protected function renderItem(&$row);
}

?>
```

[Den interne tilstand]

Hvert objekt af typen **Paginator** skal kende sin interne tilstand. Denne er givet gennem følgende attributter:

* **tables**: En streng med den eller de tabeller (forespørgslen, der udtrækker data fra databasen er ikke begrænset til én tabel), hvorfra data hentes. Denne værdi sættes ved instantiering.

* **fields**: En streng med de felter, der ønskes udtrukket. Denne værdi sættes ved instantiering.

* **where**: En streng med betingelsen for udtrækket (uden **LIMIT** - den del håndteres internt). Denne værdi sættes ved instantiering.

* **orderBy**: En streng med rækkernes sorteringsrækkefølge. Denne værdi sættes ved instantiering.

* **cntPerPage**: Antallet af elementer pr. side. Denne værdi sættes ved instantiering.

* **key**: GET-nøgle til at forespørge side. Som udgangspunkt benyttes "page" således at URL'erne i navigationen indeholder "?page=N". Attributten er kun medtaget for at muliggøre simpel tilpasning. Denne værdi *kan* sættes ved instantiering.

* **page**: Objektets aktuelle side. Sættes internt baseret på GET-værdien for ovennævnte **key** og vil altid være i intervallet [0; **cntPages**[.

* **cntPages**: Det samlede antal af sider. Beregnes internt som antallet elementer divideret med elementer pr. side.

Disse føjes til skelettet:

```
<?php

abstract class Paginator {
    private $tables; // (string)
    private $fields; // (string)
    private $where; // (string)
    private $orderBy; // (string)
    private $cntPerPage; // (int) Items per page
    private $key; // (string) The query string key for page navigation
    private $page; // (int) Current page
    private $cntPages; // (int) Number of pages

    public function renderItems() { ... }

    public function renderNavigation() { ... }

    abstract protected function renderItem(&$row);
}
```

```
?>
```

[Fælles funktionalitet]

Først og fremmest må vi have en konstruktør, hvormed vi kan instantiere et nyt objekt af typen **Paginator**. (Konkret kan det naturligvis først gøres når **renderItem()** er blevet realiseret, men det er en anden sag. Vi kommer dertil om et øjeblik.). Denne laver blot lidt simpel argumenthåndtering:

```
public function __construct($tables, $fields, $where, $orderBy, $cntPerPage,
$key = 'page') {
    if (intval($cntPerPage) < 1) {
        die('Illegal argument: cntPerPage must be greater than zero.');
    }

    $this->tables     = $tables;
    $this->fields     = $fields;
    $this->where      = empty($where) ? '1=1' : $where;
    $this->orderBy    = $orderBy;
    $this->cntPerPage = intval($cntPerPage);
    $this->key        = $key;

    // Perform calculations to set cntPages and page...
    $sql = sprintf('SELECT COUNT(*) FROM %s WHERE %s', $this->tables,
$this->where);
    $res = mysql_query($sql);
    $cnt = mysql_result($res, 0);

    $this->cntPages = ceil($cnt / $this->cntPerPage);
    $this->page     = (isset($_GET[$this->key]) && intval($_GET[$this->key]) >
0 ? intval($_GET[$this->key]) - 1 : 0); // ?page=1 gives $this->page = 0
    $this->page     = max(0, min($this->page, $this->cntPages - 1)); // Never
accept values greater than the number of pages
}
```

Ingen magi hér! Konstruktøren sørger for, at den initielle tilstand sættes, hvorefter det udregnes, hvor mange sider der skal benyttes, samt hvilken side der forespørges (i.e. der tildeles værdier til **cntPages** og **page**). **Bemærk**, at side 1 repræsenteres som **$this->page = 0**.

renderItem() er erklæret abstrakt, så vi kan være sikre på, at metoden er realiseret (implementeret) inden et konkret **Paginator**-objekt kan oprettes. Idéen hermed er, at **renderItems()** (bemærk "s" - der tales hér om den offentlige metoder, der sørger for at renderer den aktuelle sides elementer) blot kan kalde **renderItem()** for hver række fra databasen, der skal være en del af den aktuelle side:

```
public function renderItems() {
    $out = '';
```

```
    $sql = sprintf('SELECT %s FROM %s WHERE %s ORDER BY %s LIMIT %d, %d',
$this->fields, $this->tables, $this->where, $this->orderBy, $this->page *
$this->cntPerPage, $this->cntPerPage);
    $res = mysql_query($sql);
    while ($row = mysql_fetch_assoc($res)) {
        $out .= $this->renderItem($row); // Calling the concrete
implementation for each row
    }

    return $out;
}
```

[Navigationen]

Ovenstående er nu et fungerende skelet, der i sig selv kan håndtere pagination (når man har realiseret **renderItem()** - eksempel følger i næste afsnit). Det er imidlertid ikke meget ved det, førend man også har mulighed for at navigere mellem siderne. Som allerede omtalt, ønsker jeg at lave et system, der kan håndtere endda meget store kollektioner, i.e. mange sider. Desuden ønsker jeg at kunne styre udseendet meget præcist, så hvert element skal forsynes med relevante CSS-klasser:

* **seqFirst**: Første side
* **seqLast**: Sidste side
* **seqCurrent**: Den aktuelle side - side 1 som udgangspunkt
* **seqBefore**: Sider før den aktuelle
* **seqAfter**: Sider efter den aktuelle
* **seqGap**: Udeladte sider (ellipse)

Eksemplet fra tidligere - **[1] ... [63] [64] *65* [66] [67] ... [180]** - ønskes renderet som følger:

```
<ul class="PaginatorNavigation">
    <li class="seqFirst seqBefore"><a href="">1</a></li>
    <li class="seqBefore seqGap">...</li>
    <li class="seqBefore"><a href="?page=63">63</a></li>
    <li class="seqBefore"><a href="?page=64">64</a></li>
    <li class="seqCurrent">65</li>
    <li class="seqAfter"><a href="?page=66">66</a></li>
    <li class="seqAfter"><a href="?page=67">67</a></li>
    <li class="seqAfter seqGap">...</li>
    <li class="seqLast seqAfter"><a href="?page=180">180</a></li>
</ul>
```

Og så et lille intermezzo: Da ingen sideangivelse svarer til at få serveret side 1, ønsker jeg ikke, at der

bliver genereret et link til "?page=1". Desuden skal **Paginator** være let at bruge, så jeg må på en måde gøre det muligt for folk at kunne håndtere selve genereringen af sidelinkenes URL'er. Det kan eksempelvis være, at man i ét specifikt tilfælde vil angive sidetallet på det, jeg kalder "mod_rewrite"-vis: http://.../gaestebog/65 i stedet for som GET-parameter. Man kan ikke tage hensyn til alle typer scenarier, så i stedet overlader jeg muligheden til brugeren ved at lave en metode - **generateLinkUrl()** - der *kan* overskrives, hvis man ønsker mere fleksibilitet end blot at kunne omdøbe GET-parametren (jf. attributten **key**):

```
protected function generateLinkUrl($linkPrefix, $key, $page) {
    $var = (strpos($linkPrefix, '?') === false ? '?' : '&') . $key . '=' .
$page;

    return $linkPrefix . ($page > 1 ? $var : ''); // Don't add page data if
link goes to first page
}
```

Argumenterne turde være selvforklarende. De to sidste kunne have været undværet, da implementøren blot kan tilgå **$this->key** hhv. **$this->page**. Personligt foretrækker jeg dog denne afkobling.

Jeg vil ikke gå i dybden med hver linie i **renderNavigation()**, da jeg i nogen grad finder koden selvforklarende. I store træk går det ud på at iterere over alle siderne (1 - **cntPages**) og påklistre de korrekte klasser, jf. ovenstående liste. Der er imidlertid ingen grund til at gennemgå *alle* sider, så jeg springer over hvor jeg kan (i.e. hvor sider er udeladt).

Metoden er forsynet med to parametre: **renderNavigation($linkPrefix = '', $cntAround = 2)**. Det første videresendes til **generateLinkUrl()** og kan benyttes til at angive sidelinkenes udgangspunkt. Andet argument er antallet af sidelinks direkte omkring den aktuelle side. Sættes den til 1 bliver det løbende eksempel til **[1] ... [64] *65* [66] ... [180]**, sættes den til 0 fås **[1] ... *65* ... [180]**. Negative værdier tolkes som et fravalg af udeladte sider, hvilket betyder, at der genereres sidelinks til alle sider. (Det var netop det vi ville undgå, men det kan jo være, at du er af en anden opfattelse, så nu har du muligheden. Jeg gengiver forresten *ikke* eksemplet hér!)

Da denne metode er den sidste, ser jeg ingen grund til at vise den for sig. I stedet præsenterer jeg - ta-daaaah...

[Den samlede klasse]

Vejen hertil var lang, men her er det du kom for:

```
<?php

abstract class Paginator {
    private $tables; // (string)
    private $fields; // (string)
    private $where; // (string)
    private $orderBy; // (string)
    private $cntPerPage; // (int) Items per page
    private $key; // (string) The query string key for page navigation
```

```
    private $page; // (int) Current page
    private $cntPages; // (int) Number of pages

    public function __construct($tables, $fields, $where, $orderBy,
$cntPerPage, $key = 'page') {
        if (intval($cntPerPage) < 1) {
            die('Illegal argument: cntPerPage must be greater than zero.');
        }

        $this->tables     = $tables;
        $this->fields     = $fields;
        $this->where      = empty($where) ? '1=1' : $where;
        $this->orderBy    = $orderBy;
        $this->cntPerPage = intval($cntPerPage);
        $this->key        = $key;

        // Perform calculations to set cntPages and page...
        $sql = sprintf('SELECT COUNT(*) FROM %s WHERE %s', $this->tables,
$this->where);
        $res = mysql_query($sql);
        $cnt = mysql_result($res, 0);

        $this->cntPages = ceil($cnt / $this->cntPerPage);
        $this->page     = (isset($_GET[$this->key]) &&
intval($_GET[$this->key]) > 0 ? intval($_GET[$this->key]) - 1 : 0); // ?page=1
gives $this->page = 0
        $this->page     = max(0, min($this->page, $this->cntPages - 1)); //
Never accept values greater than the number of pages
    }

    public function renderItems() {
        $out = '';
        $sql = sprintf('SELECT %s FROM %s WHERE %s ORDER BY %s LIMIT %d, %d',
$this->fields, $this->tables, $this->where, $this->orderBy, $this->page *
$this->cntPerPage, $this->cntPerPage);
        $res = mysql_query($sql);
        while ($row = mysql_fetch_assoc($res)) {
            $out .= $this->renderItem($row); // Calling the concrete
implementation for each row
        }

        return $out;
    }

    public function renderNavigation($linkPrefix = '', $cntAround = 1) {
        $out     = '';
        $isGap   = false;
        $classes = ' ';

        for ($i = 0; $i < $this->cntPages; $i++) {
            $isGap    = false;
            $classes  = '';
            $classes .= ($i == 0 ? 'seqFirst ' : '');
            $classes .= ($i == $this->cntPages - 1 ? 'seqLast ' : '');
            $classes .= ($i == $this->page ? 'seqCurrent ' : '');
```

```
            $classes .= ($i < $this->page ? 'seqBefore ' : '');
            $classes .= ($i > $this->page ? 'seqAfter ' : '');

            // Are we at a gap?
            if ($cntAround >= 0 && $i > 0 && $i < $this->cntPages - 1 &&
abs($i - $this->page) > $cntAround) {
                $isGap    = true;
                $classes .= 'seqGap ';

                // Skip to next linked item (or last if we've already run past
the current page)
                $i = ($i < $this->page ? $this->page - $cntAround :
$this->cntPages - 1) - 1;
            }

            $lnk = ($isGap ? '...' : ($i + 1));
            if ($i != $this->page && !$isGap) {
                $lnk = '<a href="' . $this->generateLinkUrl($linkPrefix,
$this->key, $i + 1) . '">' . $lnk . '</a>';
            }
            $out .= "\t<li class=\"" . trim($classes) . '">' . $lnk .
"</li>\n";
        }

        return "<ul class=\"PaginatorNavigation\">\n" . $out . '</ul>';
    }

    protected function generateLinkUrl($linkPrefix, $key, $page) {
        $var = (strpos($linkPrefix, '?') === false ? '?' : '&') . $key . '=' .
$page;

        return $linkPrefix . ($page > 1 ? $var : ''); // Don't add page data
if link goes to first page
    }

    abstract protected function renderItem(&$row);
}

?>
```

[Brugseksempel]

Du er nu veludrustet (!) til at realisere din egen **Paginator**. Jeg har gjort således:

```
class PaginatedGuestbook extends Paginator {
    protected function renderItem(&$row) {
        $o = "\nIndlæg fra " . $row['navn'] . "<br />\n"
           . "Overskrift: <em>" . $row['overskrift'] . "</em><br />\n"
```

```
            . "Indhold: " . $row['indhold'] . "<br />\n";
        // Her kunne man udskrive flere data fra rækken ($row)

        return $o;
    }
}

$gb = new PaginatedGuestbook('gaestebog', 'tidspunkt, navn, overskrift,
indhold', 'hidden=0 AND deleted=0', 'tidspunkt DESC', 5);
print $gb->renderItems();
print "<br />\n\n";
print $gb->renderNavigation('minside.php');
```

Husk, at der næsten ikke er noget overarbejde. Du skulle alligevel have skrevet renderingskode. Eneste forskel er blot, at du nu skal pakke det ind i en på forhånd navngivet metode. **Resten får du forærende!** Simpelt? Anvendeligt? Jæs! Det er *nu* du begynder at vinde pengene tilbage!

[Afslutning]

Jeg håber, at du som læser har fået noget ud af artiklen. Den bliver noget længere end planlagt, men jeg håber så, at der måske har sneget sig et par fornuftige ord om objekt-orientering ind. Kopiér koden og se, om ikke den kan bruges i *dit* unikke tilfælde.

Kunne du godt tænke dig at se, hvordan pagination-klassen opfører sig, så kig forbi http://www.jensgram.dk/sjov/gaestebog/ (og skriv gerne en hilsen). Den benyttede CSS finder du på http://www.jensgram.dk/styles/_core.css (der skal ledes lidt).

Eventuelle fjel og wrangler bekagles.

Med venlig hilsen
- Jens Gram, http://www.jensgram.dk/

PS: På http://www.jensgram.dk/web/objects/ kan du finde klassen **JgPaginator**, der er lige til at gå til uden copy-paste.

PPS: Formateringsmulighederne på Eksperten lader stadig noget tilbage at ønske. På http://www.jensgram.dk/web/e-artikler/1265 fremstår artiklen som jeg havde tiltænkt.

Change log:

```
2009/01/13: Første version publiceret
2009/01/14: Beskrivelse af negativ cntAround-argument til renderNavigation
2009/03/07: Reformateret til E5 (ingen rettelser)
```

Kommentar af superb d. 30. Jan 2009 | 1

nu jeg ret ny i OOP, hvorfor er det du siger $this->?

hvad gør "function __construct"
og hvad betyder det at noget er "erklæret abstrakt"?

Kommentar af zurekk d. 23. Jan 2009 | 2