Börja programmera i solidba.se

Jonatan Olofsson

28 oktober 2009

Innehåll

1 Inledning

För att underlätta för nya användare att använda sig av ramverket i solidba.se har jag samlat ihop grunderna till en snabb förståelse i detta dokument.

Solidba.se grundar sig starkt i den objektorienterade sidan av PHP, och i alla fall en grundläggande förståelse för hur klasser ser ut och fungerar i PHP är nödvändigt.

solidba.se är i stort ett ramverk för att smidigt och snabbt skapa välstrukturerade, enhetliga och genomtänkta funktioner för webben. Tanken är att det ska sköta så mycket som möjligt själv och underlätta så mycket som möjligt för programmeraren, och även i slutändan användaren. Därför tillhandahåller solidba.se enkla funktioner för att, till exempel, på ett snyggt och abstraherat sätt använda databasen, göra formulär eller fixa säkerheten och inloggning och dylikt.

2 PHP

Det förmodligen viktigaste att förstå innan man börjar kolla på källkoden till solidba.se är klass-strukturen i PHP och förekomsten av *magiska funktioner*. De viktigaste av dessa är __get() , __set() och i viss mån __call().

Dessa tre funktioner är PHP's fallback när språket inte vet vad den ska göra med en rad kod. Till exempel, säg att objektegenskapen $object->property efterfrågas, men inte är definierad någonstans i koden. PHP kommer då att skicka vidare frågan till funktionen __get() (i det här fallet) som förhoppningsvis vet vad den ska göra med det. Motsvarande gäller för __set() vid tilldelning av okända egenskaper ($object->property = 3) eller för __call() vid anrop av okända metoder.

Nästa sak som är viktig att förstå är PHP's struktur för arv av egenskaper. I PHP kan man utöka klasser med en överbyggnad; ex: class Page extends MenuItem. Den nya klassen (Page i exemplet) får alla egenskaper och metoder som MenuItem, och dessutom de nya som definieras i Page-klassen. Anrop bubblar uppåt, så om en eftersökt metod eller egenskap inte finns i klassen som objektet i fråga är av, frågar den vidare om klassen den utökar vet vad som efterfrågas.

Detta gör att metoder och egenskaper överskuggas av metoder/egenskaper av samma namn "högre upp"i hierarkin.

3 solidba.se

3.1 Uppstart

För att börja introducera idéerna med solidba.se ska jag börja med att förklara vad som händer vid en sidladdning. Kom ihåg att när allt kommer omkring är det inget annat än en vanlig index.php-fil som körs, precis som vilken annan php-fil som helst.

Det första som händer är att några objekt som är viktiga senare startas upp. Dessa är till exempel databas-objektet, men allt det här är egentligen irrelevant nu, med undantaget att vi vet att de redan finns tillgängliga när vi börjar göra saker. Vi blir redan här identifierade med den användare vi är inloggade som.

Efter den första uppstartsprocessen så tar vi reda på vilken sida som eftersöks. Detta görs helt enkelt genom att kolla på $_REQUEST['id'] och se vad som efterfrågas där. Det viktiga är dock att en tabell i databasen här håller reda på vilken klass objektet i fråga är för något. Om $id = 8$ motsvarar framsidan på hemsidan finns en rad i databasen som associerar 8 => FrontPage, och man skapar alltså ett objekt av klassen FrontPage.

Notera att om användaren inte får se sidan av rättighetsskäl så pekas den redan här om till en annan sida. Detta betyder dock inte att du inte behöver pröva användaren för rättigheter! Det är möjligt, och ofta troligt, att kollen som görs här är mindre strikt än den du egentligen vill applicera på dina specifika sidor och funktioner.

Objektet som skapas här lagras i globalen $CURRENT, och det enda man gör med objektet i det här läget är att anropa funktionen $CURRENT->RUN.

3.2 Funktionen RUN

Varje typ av objekt som skall kunna göra något på hemsidan har funktionen RUN(). Denna anropas när objektet är det centrala intresset på sidan som visas för användaren, det vill säga när ett objekt av just den klassen efterfrågas. Objektet har redan när det skapades matats med det ID som associeras med objektet, så den har alla möjligheter att ta reda på mer om sig själv.

Funktionen RUN() innehåller den PHP-kod som på något sätt skapar innehåll till sidan. Detta sker i princip på helt traditionellt sätt som i vilket PHP-projekt som helst.

När man skapat innehållet, som är typiskt för klassen, körs normalt koden $Templates->render();, varpå mn överlåter åt template-klassen att rita ut den för sidan valda HTML-mallen. Sedan returnerar RUN-funktionen och anropet avslutas

3.3 REQUEST, GET och POST

Genom att använda något som kallas SPL i PHP kan man bland annat på ett intressant sätt göra objekt som fungerar med vektor-syntax; För att exemplifiera

```
class A extends ArrayObject {
...
}
$a = new A();
$a['index'] = 2;
```

Något man kan använda detta till är för att skriva över den superglobala variablerna $_REQUEST, $_GET och $_POST och få superglobala objekt som innehåller samma sak som deras tidigare namne, med undantaget att man på ett snyggt, effektivt och globalt sätt kan sätta upp regler för vad dessa variabler får innehålla. För att sätta upp dessa regler används följande syntax

```
$_REQUEST->setType('id', 'numeric');
$_REQUEST->setType('title', 'string');
$_REQUEST->setType('checkbox', 'any');
$_REQUEST->setType('userdefined', '#<regexp>#');
$_REQUEST->setType('arrayofstrings', 'string', true);
```

```
$_REQUEST->addType('id', 'string');
```

'string' i sammanhanget tar bort HTML-formatering och ser till att strängen är snäll, och 'numeric' ser till så att variabeln är numerisk. Skulle en inkommen variabel inte uppfylla kraven kommer $_REQUEST att uppföra sig som om det aldrig kommit någon data. Detta betyder också att man måste vara noggrann med att definiera datatyper innan man använder variablerna, annars kan svårfunna fel uppstå.

3.4 Användbara funktioner och objekt

3.4.1 Controller

I nätet av objekt i solidba.se finns också en spindel. Denna kallas Controller och är den som tar hand om alla objekt i minnet. I och med att varje objekt tilldelas ett globalt ID kan man centralt ta hand om så att varje objekt bara behöver laddas en gång. När man behöver komma åt ett objekt som man ska göra någonting med - kom ihåg att allt är objekt av de klasser som finns definierade; Du arbetar följaktligen ganska ofta med objekt i solidba.se - när du vill komma åt dessa, då ber du Controllern ladda det åt dig. Chansen är ofta stor att någon annan, tidigare i koden, har laddat samma objekt och då är det ju onödigt att göra det igen. Controllern behåller pekare till alla objekt den skapar, vilket gör att du snabbt och enkelt kan komma åt alla objekt du behöver var du än är. Du kan också välja att hämta flera objekt samtidigt, vilket gör att kontrollern kan rationalisera databasförfrågningarna och ladda flera objekt parallellt, istället för ett åt gången.

```
$p = $Controller->pageEditor;
$p = $Controller->alias('pageEditor');
$p = $Controller->$id; //Numeriskt ID
$p = $Controller->get($id, EDIT);
$p = $Controller->get(array(1,2,3,4));
$Controller->$id(EDIT, 'User');
$Controller->get($id, OVERRIDE);
```

Som synes finns ett flertal sätt att smidigt få tag på de objekt man önskar. Frågar man efter flera objekt returneras de i samma ordning som de frågas efter. OVERRIDE och EDIT svarar ovan mot vilken rättighetskontroll som skall tillämpas vid laddning av objektet. Det näst sista exemplet är lite extra

speciellt, och för närmare förståelse hänvisas till PHP-dokumentationen av __call()

3.4.2 Databasabstraktion

Databas-objektet i solidba.se är tänkt att vara väldigt lättanvänt och logiskt, och automatisera generering och kontroll av SQL-kod, som annars lätt tar upp stor del av koden och ser fult ut. Även om det går att skriva ren SQL också, så är det sällan det behövs.

För att exemplifiera

```
$class = $DB->spine->getCell(array('id' => $id), 'class');
$aliases = $DB->aliases->asList(array('id' => $id), 'alias');
```

I det första exemplet görs associationen mellan id och klass som nämndes ovan. Den andra raden hämtar alla alias som är associerade med objektet med id $id. asList som används i exemplet kommer att returnera motsvarande

```
array('alias1', 'alias2', 'annatAlias')
```

Det finns fler parametrar och flera andra metoder i databas-klassen, och de finns alla dokumenterade mer i huvuddokumentationen. Där hitta du också med information om andra parametrar till de exemplifierade funktionerna.

Något väl värt att notera är nyckel => värde-strukturen som finns i den första parametern, som motsvarar WHERE-stycket i en SQL-fråga. Detta möjliggör nämligen för databasklassen att automatiskt kontrollera så att ingen skadlig kod tar sig in i den bakomliggande SQL-frågan.

Nycklarna i vektorn som är den första parametern kan även ha några magiska tecken sist i sig för att modifiera betydelsen. Exempel på sådana är t.ex >,>=,! och , som motsvarar LIKE i SQL-syntax.

```
$class = $DB->content->asArray(array(
  'published>=' => time()),
  'id,section,content');
```

Ovanstående exempel hämtar allt innehåll (som finns i tabellen 'content', som är publicerat efter nuvarande tid. De kolumner från databasen vi vill ha är id, section och content. Vi vill dessutom ha resultatet i form av en array,

vilket innebär att varje rad i den returnerade vektorn, i sin tur är en vektor innehållande den rad i databasen som hämtats.

3.4.3 Formulär och flikar

För att enkelt kunna generera formulär för administration finns färdiga klasser som enklast visas med ett exempel

```
$Form->Collection(
  new Tabber('uniqueName',
    new Tab(__('Page settings'),
      new Input(__('Title'), 'title', @$page->Name, 'required'),
      new Input(__('Alias'), 'alias', @$page->alias),
      new TextArea(__('Description'), 'desc', @$page->description)
    ),
    new eTab(__('Content'),
      '<h3>'.__('Modify section content').'</h3>',
      new Tabber($unique_tabber_id,
        $contentArray
      )
    )
  )
);
```

Koden ovan ger lite blandade exempel på formulär, och hur de kan placeras i flikar.

Tabber är en klass som i sin konstruktor tar emot först ett unikt namn, sedan en godtycklig mängd objekt av olika Tab-klasser. Skillnaden mellan Tab och eTab är marginell, men kort sagt så är Tab något mer specialiserat på att visa formulärfält rätt i, medan eTab ("emptytab, dvs inget extra) hellre fylls med mera välformaterat innehåll, som i det här fallet ännu en flikrad. Input och textarea är klasser för att enkelt skriva ut sina HTML-motsvarigheter.

Komplett dokumentation om alla formulärobjekt finns i den stora dokumentationen, men generellt gäller parameterföljden <Titel, unikt html-namn, initialt värde>.

3.4.4 Metadata

Ett objekt kopplas normalt till data av olika slag. Framför allt kan det röra sig om lite längre innehåll, eller om korta attribut, såsom titel, associerad bild etc. För dessa små attribut finns en klass kallad metadata. Klassen används främst på två olika sätt, som båda visas i exemplet nedan.

```
private $_Cal = false;
private $_Image = false;
private $_mine;

function __set($property, $value) {
  if(in_array($property, array('Cal', 'Image')) {
        if($this->{'_'.$property} !== false) {
      Metadata::set($property, $value);
    }
    $this->{'_'.$property} = $value;
  }
}

function __construct($id, $language=false) {
  parent::__construct($id, $language);
  $this->getMetadata('', false, array('Cal', 'Image'));
}
```

I konstruktorn kallas bas-klassens funktion getMetadata(), där den tredje parametern visar att vi minst vill ha värden till egenskaperna 'Cal' och 'Image'. Funktionen kör sedan motsvarande

```
$this->Cal = $old_value_Cal;
$this->Image = $old_value_Image;
```

Där $old_value_... satts av en tidigare användning av Metadata::set(*property*,value) och nu finns i databasen. Om inget gammalt värde finns ansätts tomma strängen, vilket gör att nästa gång __set() fångar upp att egenskapen tilldelas ett värde kommer detta att skrivas till databasen och lagras.

Tanken med metadata-klassen är dock att man ska slippa skapa tabeller och en hel del extrakod bara för att spara småsaker, helt enkelt för att slippa bry sig om att krångla med databasen direkt.

3.5 Att skriva en ny modul

Varje modul i solidba.se representeras som en klass i PHP. Ofta vill du göra nya typer av sidor som visar någon typ av innehåll, och vad man vill göra då, är att utöka page

```
class MinKlass extends Page {
...
}
```

Klassen måste egentligen bara innehålla funktionen RUN(), men som vanligt med PHP-klasser kan man ha en konstruktor, som i så fall skall inledas på följande sätt

```
class MinKlass extends Page {
  function __construct($id, $language=false)
  {
        parent::__construct($id, $language);
        ...
  }

  function RUN()
  {
    global $Templates;
    ...

    $this->setContent('main', $content);
    $Templates->render();
  }
  ...
}
```

I exemplet ovan är också inkluderat hur man visar innehåll på sidan. Den första parametern i setContent() är vilken sektion på sidan innehållet skall placeras i. Normalt är 'main', men även 'header' eller 'footer' kan tänkas. Vilka sektioner som finns definieras i stilmallen, men 'main' skall alltid finnas.

Resten av klassen, och funktionen RUN, kan nu bestå av den funktionalitet du önskar för din modul. Genom att kolla igenom klasserna som du bygger ut kan du se vilka funktioner som du har tillgängliga ifrån tidigare nivåer.

För att testa din modul lägger du den i en lämplig mapp i solidba.se's mappstruktur och går in på administrationsområdet, under "Installer", hittar din nya modul och väljer att lägga till den i menyn.

3.6 Mer avancerade klasser

3.6.1 Installer etc

För att deklarera att en klass går att installera, uppdatera respektive ta bort läggs följande kod till i klassen

```
static public function INSTALLABLE() {return __CLASS__;}
static public function UPGRADABLE() {return __CLASS__;}
static public function UNINSTALLABLE() {return __CLASS__;}
```

Till varje av dessa funktioner måste en motsvarande funktion som faktiskt gör något, finnas function Install() function Upgrade() function Uninstall() En lämplig funktion för Install-metoden vore till exempel att sätta upp databasen för klassen. Install-metoden bör också returnera ett värde som på något sätt identifierar versionen av klassen. Detta returvärde används när funktionen Upgrade anropas, dit den skickas som enda argument.