EVENING

splitedragon@hotmail.com

2020.11.30.

Table of Contents

1. 설치 및 실행하기 5
2. 라우팅 6
   - 라우팅 디버깅 7
   - Controller 와 Method 7
   - URL 에 Argument 함께 전송하기 8
   - URL 에 GET 데이터 전송하기 8
   - POST 데이터 전송하기 9
   - 기정경로 9
   - 모듈화된 URL 10
   - 특정라우팅설정 11
   - .htaccess 의 Rewrite 기능 13
3. 파일구조 15
4. MVC 구조화 18

- Model 18
- View 18
- Controller 18

5. Controller 19
   1) 기본규정 19
   2) 메소드정의 19
   3) 전처리메소드와 후처리메소드 20
   4) 전역 전처리메소드와 후처리 메소드 21
   5) View 지정 22
   6) Model 지정 22
6. View 23
   1) 기본규정 23
   2) view 파일 로드 _setView() 23
   3) view 파일 출력 23

- 기본출력 output() 23
- 부분출력 partial() 24
- 캐쉬출력 cacheOutput() 24

4) PHP 변수를 페이지에 전송하기 24

5) 고급기능 25
- setHttpStatusHeader() 25

7. Model 26

1) 기본규정 26

2) Model 작성의 기본 26
- 모델클래스명정의 26
- 모델속성정의 26
- 모델속성값 설정메소드 정의 27

3) SQL 조작 28
- DB 접속 설정 28
- SQL 문 설정 28
- SQL 문 값대입 29
- 다중조회 30
- 단일조회 30
- 추가 30
- 삭제 31
- 변경 31
- SQL 문 직접 실행 31
- Transaction 기능 32
- SQL 구문 미리보기 33

8. Controller 와 Model 연결 35

1) 모델로드 37

2) 모델조작 38

9. 다중언어지원기능 40

1) 어종파일작성 40
2) 어종파일적재 40
3) 어종콘텐츠사용 40
10. Middleware 기능 43
11. 기본설정기능 해석 44
1) URL, Path 작성 44
2) 클래스맵설정 44
3) gURL 의 유용성 45
4) DB 관련설정 46
5) Namespace 맵설정 47
6) Debug 방식설정/해제 49
12. 유용한 기능들 50
1) Filter 50
2) Session 50
3) FileUtils 50
4) Benchmark 51
종장 52

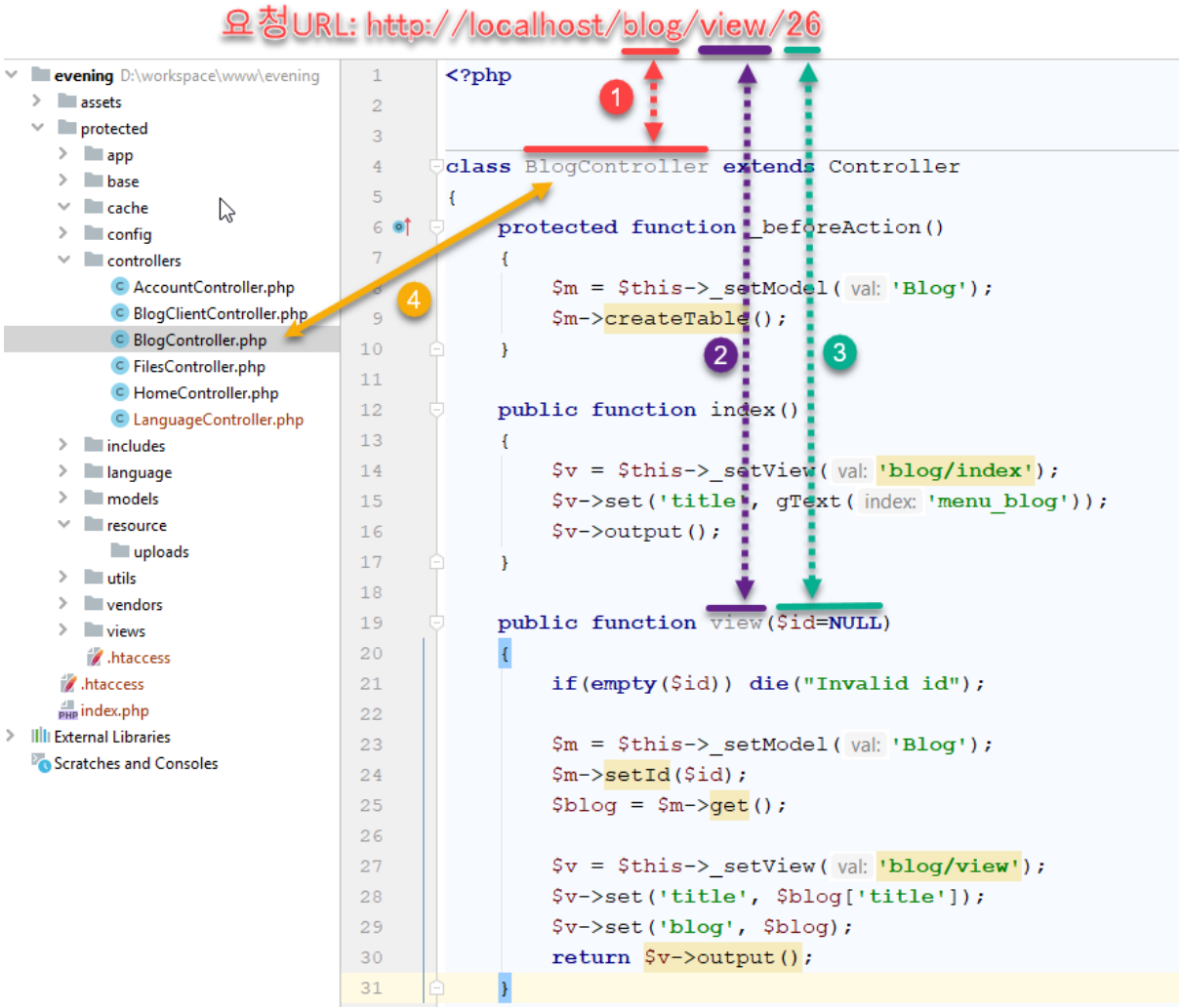
1. 설치 및 실행하기

- evening.zip 압축파일을 푼다.
- /evening 폴더내용을 웹서비스 루트경로에 복사해놓는다.
  웹서비스 루트경로라고 할 때에는 Apache httpd.conf 설정파일에서 DocumentRoot 로 지정된 경로를 의미한다.
- 이해가 편리하도록 MySQL 엔진을 사용하며 evening 이라는 DB 를 먼저 창조한다.
  DB 접속정보들은 /config/db.config.php 에서 정의하였다.
- 웹브라우저에서 http://localhost/를 호출한다.
  만일 DocumentRoot 안에 evening 이라는 폴더를 그대로 넣었다면 http://localhost/evening 으로 호출하여야 할것이다.

2. 라우팅

모든 웹개발에서 말단에서 어떤 URL 로 요청을 보냈을때 어느 로직이 맡아서 수행하겠는가 하는 라우팅이 해당 프로젝트의 완전성, 무결성, 특징을 결정하게 된다.

이브닝의 라우팅방식은 아래의 실례와 같다.




```
1- Controller
   Controller_Name: blog
   Controller Class Name: BlogController
2- Method
   Method_Name: view
3- Argument
   Argument1: 26 -> view($id=NULL)
4- File Path
   /protected/controllers/BlogController.php
```

이브닝은 요청을 Controller 가 받아서 처리하도록 구성한다.

http://localhost/{Controller_Name}/{Method_Name}/{Argument1}/{Argument2}...

이 요청을 받아 처리하는 파일은 Controller_Name+'Controller.php' 이다.
가령 Controller_Name 이 home 이라면 파일명은 HomeController.php 로 된다.
이 파일의 위치는 /controllers/안에 있어야 한다.
파일위치: /controllers/{Controller_Name}Controller.php
Controller 클래스는 다음과 같이 작성한다.

```
class {Controller_Name}Controller extends Controller
{
    public function {Method_Name}($arg1, $arg2,…)
    {
    }
}
```

- 라우팅 디버깅

  구체적으로 해당 요청시 매 Controller 와 Method, Argument 등이 어떻게 결정되는지를 확인하고 싶을때 Middleware::run() 메소드에서 Router::getUrl() 결과를 뿌려보면 알수 있다.

  /base/Middleware.php

  ```
  public static function run()
  {
      var_dump(Router::getUrl());
  }
  ```

  편리상 이 라우팅을 다 파악할때까지 라우팅 디버깅을 사용하는것이 좋을것이다.

  여러가지 실례를 통하여 URL 에 해당한 코드들의 관계를 설명한다.

- Controller 와 Method

  URL: http://localhost/home/first

  파일: /controllers/HomeController.php

  Controller: HomeController

  Method: HomeController 클래스의 first()

```
class  HomeController extends Controller
{
    public  function first()
    {
        echo 'Hello';
    }
}
```

- URL 에 Argument 함께 전송하기

  URL: http://localhost/home/first/abc/123

  파일: /controllers/HomeController.php

  Controller: HomeController

  Method: HomeController 클래스의 first()

  파라미터 1: abc

  파라미터 2: 123

```
class  HomeController extends Controller
{
    public  function first($param1, $param2)
    {
        echo 'Hello!';
        echo $param1; // abc
        echo $param2; // 123
    }
}
```

[참고] 만일 파라미터를 전송하지 않을수도 있는데 이경우 위와 같이 메소드를 정의하면 오류가 발생한다. 때문에 파라미터가 없는 경우를 고려하여 매 파라미터의 기정값을 지정해주는것이 코드품질을 높여주는 방법이다.

```
    public  function first($param1='', $param2=NULL)
```

- URL 에 GET 데이터 전송하기

  URL: http://localhost/home/first/abc/123?userid=jina

  파일: /controllers/HomeController.php

  Controller: HomeController

Method: HomeController 클래스의 first()

파라미터 1: abc

파라미터 2: 123

GET: $_GET['userid']

```
class HomeController extends Controller
{
    public function first($param1, $param2)
    {
        echo 'Hello!';
        echo $param1; // abc
        echo $param2; // 123
        echo $_GET['userid']; // jina
    }
}
```

- POST 데이터 전송하기

URL: http://localhost/home/first

POST 방식으로 userid=jina 를 전송했다고 가정하면

파일: /controllers/HomeController.php

Controller: HomeController

Method: HomeController 클래스의 first()

```
class HomeController extends Controller
{
    public function first()
    {
        echo 'Hello!';
        echo $_POST['userid']; // jina
    }
}
```

- 기정경로

만일 요청 URL 에 controller_name 이나 method_name 이 없는 경우 기정으로 실행되어야 할 controller 나 method 를 지정해줄수 있다.

/config/route.config.php 에서

```
function gDefaultRoute()
{
    return array(
        'controller'=>"home",
        'action'=>"index",
        'argument'=>NULL
    );
}
```

위와 같이 정의하면 controller_name 이나 method_name 이 없는 경우 자동적으로 HomeController, index() 메소드가 실행되게 된다.

http://localhost/

HomeController, index() 실행

http://localhost/users

UsersController, index() 실행

- 모듈화된 URL

프로젝트의 특성이나 규모에 따라 요청 URL 을 부분적으로 세분화해야 할 경우가 제기된다.

실례로 일반 사용자용 UsersController 와 관리자용 UsersController 를 따로 작성해야 하지만 파일명이 똑같아서 /controllers/ 바로 안에 함께 만들수 없다고 하면, 관리자용으로는 별도로 admin 이라는 폴더안에 모든 controller 를 모아두고 처리하도록 하고 싶을때 모듈화기능을 사용한다.

http://localhost/admin/home/first 를 호출했을때 /controllers/admin/HomeController.php 파일이 처리하도록 하려면 /config/route.config.php 에서

```
function gModules()
{
    return array(
        'admin', // 동일함: admin/, /admin, /admin/
    );
}
```

라고 정의만 하면 된다.

만일 이렇게 반영하지 않으면

/controllers/AdminController.php 에서 home()이라는 메소드가 first 라는 한개의 파라미터값을 받아 처리하게 된다.
따라서
http://localhost/{Module_Path}/{Controller_Name}/{Method_Name}
이 요청을
/controllers/{Module_Path}/{Controller_Name}Controller.php 가 처리하도록 하려면
/config/route.config.php 에서 다음과 같이 그 **Module_Path** 를 반영하면 된다.

```
function gModules()
{
    return array(
        '{Module_Path}',
    );
}
```

우선순위(적용순위)는 먼저 설정한것이 후에 설정한것보다 더 높다.

- 특정라우팅설정

기본원리

gRoutes()

/config/route.config.php

어떤 경우에는 요청 URL 에 대하여 앞에서처럼 명시적인 규칙에 따르지 않고 그 해당 처리부를 완전히 다르게 대응시켜야 할 필요가 있을수 있다.

즉 어떤 요청에 대하여 특정하게 어떤 Controller, Method 가 실행되라고 설정해줄수 있다. NodeJS 의 Express 의 라우팅설정과 비슷한 개념이다.

가령 http://localhost/contact/send 라는 요청이 왔을때 기본라우팅규칙에 따르면 ContactController, send() 가 처리하지만 이를 CommunicationController, client() 가 처리하게 하려면

Route.config.php 파일의 gRoutes() 함수에서 다음과 같이 정의해준다.

```
function gRoutes()
{
    return array(
        // Request method is GET
```

```
        'GET' => array(
            'contact/send'=> array('communication',
'client'),
        ),
        // Request method is POST
        'POST' => array(

        ),
        // Request method is PUT
        'PUT' => array(

        ),
        // Request method is DELETE
        'DELETE' => array(

        ),
    );
}
```

HTTP 요청형식별로 정의하였다. 다음과 같은 규칙에 따라 정의하면 된다.

`'요청 URL'=> array('controller 명', 'method 명', 'controller 파일경로')`

파일경로를 지정하지 않으면 /controllers/에서 해당 파일을 찾게 된다.

적용순서

매 설정들은 우선순위가 먼저 설정된것이 후에 설정된것보다 더 높다.

```
'GET' => array(
    '/a'=> array('aa', 'bb'),
    'a/b'=> array('aaa', 'bbb'),
    'a/b/c'=> array('aaaa', 'bbbb'),
),
```

이제 http://localhost/a/b/c 를 호출했다면 aaaaController, bbbb() 가 호출되지 않고 우선순위가 높은 **'/a'=> array('aa', 'bb')** 에 매칭되어 aaController, bb()가 실행된다. 그리고 나머지 b/c 는 URL argument b,c 로 취급된다.

[참고] **'/'=> array('main', 'index')** 이것은 오직 http://localhost/, http://localhost/index.php 에 대응한다.

Controller 만 지정하기

가령

/blog/client/view 는 BlogClient->view()

/blog/client/ 는 BlogClient->index()

가 대응되게 작성하였다고 하자.

이때, 그 작성순서에 따라 바라던대로 라우팅이 안될수 있다. 즉 /blog/client 규칙을 먼저 정의했다면, 그 다음의 /blog/client/view 는 우선권이 낮아 무시되며 결국 마지막/view 는 URL Argument 로 취급되고 만다.

바로 이 경우를 극복하기 위해 controller 만 지정해줄수 있다.

```
'GET' => array(
    'blog/client'=> array('BlogClient'),
),
```

이렇게 정의하면 /blog/client/* 형식의 모든 요청들은 BlogClient 가 담당하게 되며 /blog/client/ 다음에 있는 URL 부분들은 method name 추출부터 시작해서 자동적으로 method, argument 를 결정하여 BlogClient 에 넘어가게 된다.

즉 /blog/client/view/123 은 BlogClient->view(123) 으로 호출된다.

'blog/client'=> array('BlogClient') 과

'blog/client'=> array('BlogClient', '') 은 동일한 의미를 가진다.

모든요청조종설정

가령 홈페이지를 정비할 필요가 있다고 하자. 이때 모든 요청에 대하여 정비중이라는 페이지를 응답하게 하여야 한다면 바로 이 설정을 사용하면 유용할것이다.

```
'GET' => array(
    '*'=> array('wait', 'index')
)
```

모든 GET 요청에 대하여 WaitController, index() 가 실행된다.

이 규칙은 그것이 정의된 순서와 무관계하게 다른 설정들보다 적용순서가 제일 낮다.

[참고] * 설정을 활성화하면, 라우팅의 기본규칙은 무시된다. 즉 http://localhost/users/login 에 대응한 UsersController, login() 이 있다고 해도 그것은 무시되고 *설정에 있는 controller 와 method 가 실행된다.

- .htaccess 의 Rewrite 기능

프레임워크의 루트에는 URL 을 재쓰기 하는 .htaccess 파일이 있다.

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1 [L]
```

이렇게 하면 모든 요청 URL 앞에 index.php/를 자동적으로 첨부해준다.

즉

http://localhost/users/login

은 사실상 내부적으로

http://localhost/index.php/users/login

으로 재구성되어 PHP 에 넘겨진다.

이 Rewrite 기능을 사용하자면 Apache httpd.conf 파일에서

LoadModule rewrite_module modules/mod_rewrite.so

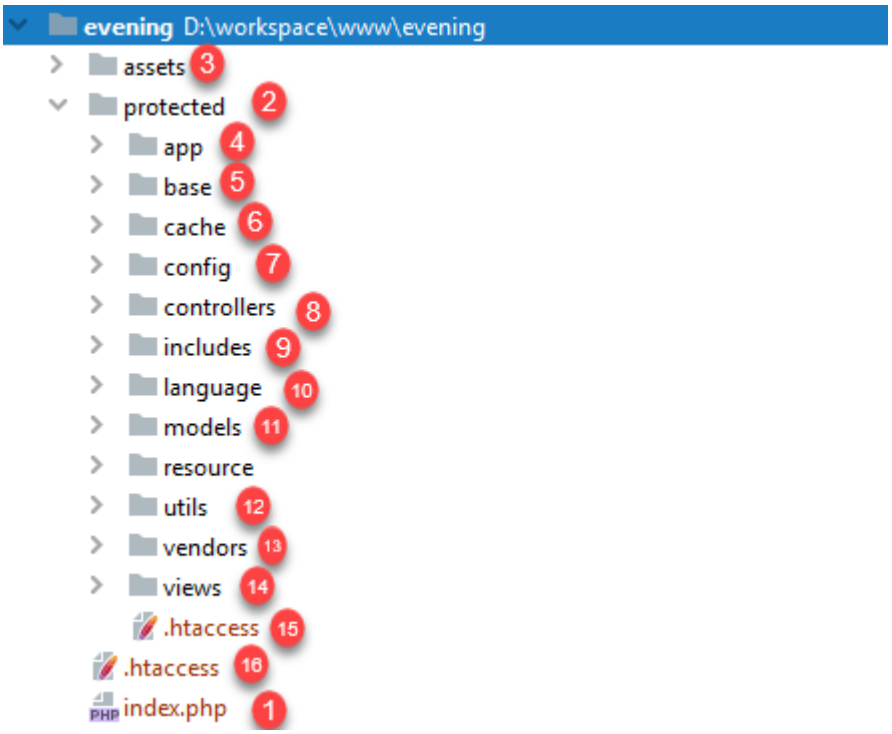
와 같이 mod_rewrite.so 모듈을 탑재해야 한다. 대체로 모든 Apache 에서 기정으로 이 기능은 지원한다.

만일 이기능을 사용할수 없는 경우에는 요청 URL 을 작성할때 인위적으로 index.php 를 붙여주어야 한다.

http://localhost/index.php/home/index

3. 파일구조

모든 폴더 및 파일명은 대소문자를 구분한다. (Windows 에서는 대소문자구분이 없지만 Linux 같은데서는 대소문자구분이 있기때문이다.)



1 – index.php

이브닝의 부트파일이다.

모든 로직은 이 파일에서부터 시작된다.

16 - .htaccess

말단 요청 URL 앞에 모두 index.php/를 붙여주는 역할을 한다.

만일 이 기능이 작동하지 않는다면 모든 URL 에 인위적으로 http://localhost/index.php/users/login 형식으로 붙여주어야 한다.

2 – protected

말단에서 직접 호출이 불가능한 코드들의 루트이다.

즉 웹브라우저에서 http://localhost/proctected/ 로 접근 할수 없다.

15 - .htaccess

/proctected 안의 모든 파일을 말단에서 직접 접근하지 못하게 하는 기능을 수행한다.

이 파일이 없으면 마음대로 접근 가능해진다.

3 – assets

웹브라우저에서 적재되는 css, js, image, video, audio, 등 파일들이 위치한다.

즉 웹브라우저에서 직접 호출가능하다.

http://localhost/assets/css/style.css

4 – app

해당 프로젝트의 특성에 맞게 별도로 작동시킬 코드들이 위치한다.

실례로 해당 유저의 세션을 관리하는 코드를 별도로 클래스화하여 여기에 위치해놓을수 있다.

대체로 프로젝트와 직접 련관된 로직이지만, 공통적으로 사용해야 하며 Controller 나 Models 에 위치하는것이 적합치 않는 경우 여기에 위치한다.

5 – base

프레임워크의 기초클래스들이 존재한다.

이 부분의 코드들을 변경하여 새로운 프레임워크로 확장할수 있다.

수정하면 코드 전반에 영향을 주기때문에 심중하여야 한다.

6 - cache

말단에 현시할 페이지를 캐쉬파일로 여기에 저장한다.

물론 코드에서 캐쉬기능을 사용할때에만 여기에 캐쉬파일이 창조된다.

7 - config

Database, Router, URL, Language, Namespace 설정파일들이 있다.

8 - controllers

controller 파일들이 위치한다.

파일명은 클래스이름과 일치하여야 하며 ...Controller.php 로 끝나야 한다.

첫 글자는 언제나 대문자이여야 한다.

UsersController.php

9 – includes

페이지현시부분에서 공동으로 사용해야 할 코드들을 놓는다.

Header, Footer 부분은 공동으로 사용하므로 이 부분 코드들은 여기에 별도로 내오고

```
include_once(gURL('includes').DS.'menu.inc.php');
```

와 같이 사용할수 있다.

10 – language

어종별로 변경되어야 할 내용들을 적은 어종파일들이 여기에 존재한다.

en.php, ru.php ...

11 – models

Database 조작관련 클래스들이 존재한다.

파일명은 클래스이름과 일치하여야 하며 ...Model.php 로 끝나야 한다.

첫 글자는 언제나 대문자이여야 한다.

UsersModel.php

12 – utils

파일조작, Validator 등 각종 PHP 유용기능들을 모아둔다.

파일명과 클래스명이 일치하여야 한다.

13 – vendors

PHPMail, ImageMaster 와 같은 외부 라이브러리들을 여기에 놓는다.

14 – views

말단에 현시할 페이지 파일들이 위치한다.

4. MVC 구조화

이브닝은 MVC 구조화를 위해 다음과 같은 관례를 규정하고 있다.

- Model

  Database 관련 조작을 담당한 파일들이 위치한다.

  폴더: /models/

  모든 파일이름은 첫글자가 대문자이여야 하며 항상 Model.php 로 끝나야 한다.

  UsersModel.php

  클래스 이름은 파일명과 같아야 하며 Model 클래스를 계승하여야 한다.

```
public class UsersModel extends Model
{
}
```

  /models/ 안에서 폴더구조를 더 세분화할수 있다.

- View

  말단에 현시해주는 페이지를 담당한 파일들이 위치한다.

  폴더: /views/

  /views/ 안에서 폴더구조를 더 세분화할수 있다.

- Controller

  모든 요청을 받아 처리하는 파일들이 위치한다.

  폴더: /controllers/

  모든 파일이름은 첫글자가 대문자이여야 하며 항상 Controller.php 로 끝나야 한다.

  UsersController.php

  클래스 이름은 파일명과 같아야 하며 Controller 클래스를 계승하여야 한다.

```
class UsersController extends Controller
{
}
```

  /controllers/ 안에서 폴더구조를 더 세분화할수 있다. [2. 라우팅/모듈화된 URL]

5. Controller

Controller 는 매 URL 요청을 직접 맡아 처리하는 프레임워크의 중심로직을 담당한다.

요청 URL 로부터 Controller 와 Method 를 규정하는 규칙은 [2. 라우팅] 을 참고한다.

1) 기본규정

- URL 의 controller_name 과 파일명, controller class 명은 일치하여야 하며 첫 글자는 항상 대문자이여야 한다.

  URL 의 controller_name 의 첫 글자를 소문자로 주어 URL 요청을 해도 프레임워크는 자동적으로 대문자로 변경된 파일과 클래스를 찾아 로드하게 된다.

- 파일명과 클래스명의 마지막은 항상 Controller.php, Controller 로 끝나야 한다.

  http://localhost/users/login

  /controllers/UsersController.php 파일을 로드한다. 소문자로 시작된 usersController.php 를 로드하지 않는다.

  ```
  class UsersController extends Controller
  ```

- 모든 Controller 는 클래스화되어야 하며 기초클래스인 Controller 를 계승하여야 한다.
- 파일의 경로는 /controllers/안에 위치하며 route.config.php 파일에서 gModules() 에 정의함에 따라 /controllers/안에서 보다 세분화할수 있다. [2.라우팅/모듈화된 URL]

2) 메소드정의

매 요청에 대하여 실지 실행되어야 할 로직을 구현한 메소드들을 정의한다.

- 요청 URL 의 method_name 과 해당 controller 클래스에 정의할 메소드 이름은 동일하여야 한다.
  http://localhost/home/index
  에서 controller 는 HomeController.php, method 는 index()로 규정된다. 따라서

  ```
  class HomeController extends Controller
  {
        public function index()
        {
        }
  }
  ```

- 메소드의 permission 은 public 속성으로 지정해주어야 한다.
- 요청 URL 에 파라미터들이 첨부된다면 메소드의 내부파라미터변수로 그것을 받아 처리할수 있다.

http://localhost/Users/login/jina

```
class UsersController extends Controller
{
    /* 만일 파라미터를 붙이지 않고 URL 을 요청하면 오류가 발생하기때문에 파라미터의 기정값을 정의해주어야 한다.  메소드의 내부파라미터 자체를 선언안하면 URL 에서 넘어오는 파라미터를 무시하는것으로 된다.*/
    public function login($param1='')
    {
        echo $param1; // jina 출력
   }
}
```

3) 전처리메소드와 후처리메소드

매 요청에 해당한 기본 메소드가 실행되기전과 실행된 후에 항상 실행되어야 할 로직이 있는 경우 전처리메소드, 후처리메소드를 정의할수 있다.

전처리, 후처리 메소드들은 정의되면 자동적으로 실행된다. 전처리, 후처리메소드들은 해당 기본 메소드가 내부 파라미터를 가진다면 꼭 같이 가질수 있다.

자동 전처리, 후처리기능은 실례로 어떤 요청을 수행하기전에 사용자가 로그인한 상태일때만 실행되도록 하기 위해서 전처리메소드에 로그인상태여부를 따지는 로직을 구현하는 등에 유용하다. 결국은 매 기본메소드의 전처리, 후처리기능이다.

- 전처리 메소드

  기본메소드의 이름앞에 _before 를 붙여주어 선언한다.

  http://localhost/home/index

  _beforeIndex()를 선언하면 기본메소드 index()가 실행되기전에 자동적으로 _beforeIndex()가 먼저 실행된다.

```
protected function _beforeIndex()
{
    echo 'index()메소드를 실행하기전에 항상 먼저 실행!';
}
```

꼭 permission 을 **protected** 로 선언한다.

그것은 이렇게 하지 않으면 요청 URL 에서 메소드명으로 _beforeIndex 를 붙여 직접 호출할수 있기때문이다. 즉

http://localhost/home/_beforeIndex 라고 호출하면

_beforeIndex() 메소드가 직접 말단요청에 의해 실행되므로 이를 방지하기 위해 permission 을 **protected** 로 선언해주어야 하는것이다.

- 후처리 메소드

  기본메소드의 이름앞에 _after 를 붙여주어 선언한다.

  http://localhost/home/index

  _afterIndex()를 선언하면 기본메소드 index()가 실행된후 자동적으로 _afterIndex()가 실행된다.

```
protected function _afterIndex()
{
    echo 'index()메소드를 실행한 다음 항상 실행!';
}
```

꼭 permission 을 **protected** 로 선언한다.

그것은 이렇게 하지 않으면 요청 URL 에서 메소드명으로 _afterIndex 를 붙여 직접 호출할수 있기때문이다. 즉

http://localhost/home/_afterIndex 라고 호출하면

_afterIndex() 메소드가 직접 말단요청에 의해 실행되므로 이를 방지하기 위해 permission 을 **protected** 로 선언해주어야 하는것이다.

4) 전역 전처리메소드와 후처리 메소드

해당 controller 에서 요청 URL 에 대응하여 정의된 모든 기본메소드들에 대하여 전역적으로 전처리, 후처리 메소드를 정의할수 있다.

실례로 index(), second() 등의 기본메소드들이 있다고 할때 매 메소드가 실행될때 마다 그전과 그후에 자동적으로 실행되는 전역 전처리메소드와 후처리 메소드를 정의할수 있다.

Contoller 기초클래스에는 _beforeAction(), _afterAction() 추상메소드가 있다.

이 메소드를 재정의하면 된다. 역시 permission 을 **protected 로** 선언하여야 한다.

```
class UsersController extends Controller
{
    protected function _beforeAction()
    {
        /* login(), logout() 실행되기전에 자동적으로 실행! */
    }

    public function login($param1='')
```

```
    {
    }

    public function logout($param1='')
    {
    }

    protected function _afterAction()
    {
        /* login(), logout() 실행후 자동적으로 실행! */
    }

}
```

전역 전처리, 후처리 기능은 실례로 모든 요청에 대한 기본메소드가 로그인한 말단에 대해서만 실행가능하여야 하는 전제조건이 있는 경우, 매 메소드별로 전처리메소드를 정의하지 말고 전역적으로 전처리메소드를 정의하여 로그인여부에 따라 기본메소드에 넘어가도록 하는데 유용하다. 결국은 해당 Controller 의 전처리, 후처리라고 볼수 있다.

[참고] PHP 의 실행자체를 중지하는 die, exit 등 구문이 사용되는 경우 자동적으로 실행하게 되어 있는 모든 전처리, 후처리 메소드 또는 기본 메소드 자체가 바라는대로 실행되지 않을수 있다.

5) View 지정

controller 에는 내부메소드로 _setView() 메소드가 있다.

```
$v = $this->_setView('home/index');
$v->output();
```

이렇게 하면 /views/home/index.php 를 말단에 현시해주게 된다.

상세내용은 [6. View] 를 참고한다.

6) Model 지정

Controller 에는 내부메소드로 _setModel() 메소드가 있다.

```
$m = $this->_setModel('admin/Users');
```

이렇게 하면 /models/admin/UsersModel.php 에 있는 UsersModel 클래스가 로드된다.

상세내용은 [7. Model] 을 참고한다.

6. View

View 는 말단에 현시할 부분이다.

1) 기본규정

- view 파일들은 모두 /views/안에 위치한다.
- 파일명은 대소문자를 구분한다.
- 폴더구조를 보다 세분화할수 있다.

```
class HomeController extends Controller
{
    public function index()
    {
        $v = $this->_setView('home/index');
        $v->set('title', '첫 페이지');
        $v->output();
    }
}
```

2) view 파일 로드 _setView()

뷰파일을 로드하는 방법에는 크게 두가지가 있다.

한가지는 Controller 클래스의 내부메소드 _setView()를 사용하는것과 다른 한가지는 View 클래스의 실체를 선언하는것이다.

- Controller 클래스에서 _setView()메소드를 호출하여 해당 view 를 로드할수 있다.

  $v = $this->_setView('home/index'); // .php 를 붙이지 않는다.

  /views/home/index.php 파일이 로드된다.
- View 기초클래스로부터 뷰파일을 로드할수 있다. 이 경우 뷰파일의 전체경로를 주어야 한다.

```
$v = new View('D:/www/evening/protected/views/home/index.php');
```

이 기능은 가령 전혀 다른곳에 있는 view 파일을 지정해야 할 경우 유용하다.

3) view 파일 출력

- 기본출력 output()

  말단에 실지 출력해주기 위해서는 View 클래스의 output()메소드를 호출한다.

  $v->output();

- 부분출력 partial()

  View 클래스의 partial() 메소드는 output() 와 달리 직접 말단에 출력해주지 않고 출력해야 할 내용을 리턴해준다.

```
$v1 = $this->_setView('home/header');
$header = $v1->partial();
$v2 = $this->_setView('home/index');
$body = $v2->partial();
$v3 = $this->_setView('home/footer');
$footer = $v3->partial();
echo $header.$body.$footer;
```

- 캐쉬출력 cacheOutput()

  응답속도를 높이기 위해 뷰파일을 다시 출력하지 않고 이미 출력했던 뷰파일의 캐쉬파일을 출력해줄수 있다.

  View 클래스의 cacheOutput()메소드를 사용한다.

  이 메소드는 캐쉬파일이 존재하지 않으면 새로 창조한다. 그렇지 않으면 해당 뷰파일의 절대경로를 md5 으로 변환한 파일명을 가진 캐쉬파일을 출력한다.

```
$v = $this->_setView('home/index');
$v->cacheOutput();
```

`cachePartial()은 직접 출력하지 않고 캐쉬내용을 리턴해준다.`

4) PHP 변수를 페이지에 전송하기

페이지가 출력될때 PHP 변수를 함께 전송하여 그 변수값에 따라 현시부분을 조종해야 할 필요가 있을수 있다.

실례로 해당 페이지의 <title></title>내용을 PHP 변수로 동적으로 반영하려면

```
$v = $this->_setView('home/index');
$v->set('title', '첫 페이지');
$v->output();
```

이렇게 하면, /views/home/index.php 파일에서는

```
<title><?php echo $title; ?></title>
```

로 호출할수 있다.

만일 배열을 전송했다면 역시 배열그대로 사용가능하다. 즉 전송할때의 PHP 변수형은 그대로 유지된다.

5) 고급기능

- setHttpStatusHeader()

  HTTP 응답 상태 코드를 지정하여 응답 Header 부에 반영할수 있다.

```
$v->setHttpStatusHeader(500); // status code 를 500 즉 Internal Server Error 상태
```

7. Model

Database 조작을 담당한다.

1) 기본규정

- 모델은 Database 조작이 필요하거나, 또는 Database 와 비슷한 부분을 구성하여야 할 필요가 있을때 정의하고 사용한다.
- 파일명은 class 명과 일치하여야 하며 첫 글자는 항상 대문자이여야 한다.
- 파일명은 Model.php 로 끝나야 한다. 클래스명도 Model 로 끝나야 한다.
  /models/UsersModel.php

```
class UsersModel extends Model
```

- Model 기초클래스를 계승하는것이 기본이며 경우에 따라 자체로 정의할수 있지만 꼭 클래스로 정의하여야 한다.
  Model 기초클래스는 Database 조작에 편리한 메소드들을 제공하고 있다.
- 파일위치는 /models/에 있어야 하며 필요에 따라 그 안에서 세분화될수 있다.

2) Model 작성의 기본

- 모델클래스명정의

모델은 Model 기초클래스를 계승하는것을 기본으로 한다.
특성상 이 요구를 지키지 않아도 된다.(database 조작은 없이 모델로직으로서의 역할을 수행하게 하는것 등)

```
class UsersModel extends Model
```

모델클래스명은 해당 모델클래스가 대상하는 테이블의 이름을 반영하는것이 좋다. 즉 물리적 Database 에 대응한 론리적모델처럼 구성하는것을 의미한다.

- 모델속성정의

일반적으로 Model 클래스의 속성들은 대상하는 물리적 DB 테이블의 column 마당과 대응되게 정하는것이 좋다.
가령 users 라는 테이블에 id, name, email, password 라는 마당이 있다고 할때 이 테이블을 조작하는 모델인 UsersModel 은 아래의 코드처럼 $_id, $_name, $_email, $_password 을 속성을 가지도록 정의한다.
중요한것은 이 속성의 permission 을 **private** 로 설정해주어 외부에서 직접 이 속성값들을 조작하지 못하게 하는것이다.

```
private $_id = NULL; // users 테이블의 id 마당
private $_name = NULL; // users 테이블의 name 마당
```

```
private $_email = NULL; // users 테이블의 email 마당
private $_password = NULL; // users 테이블의 password 마당
```

- 모델속성값 설정메소드 정의

private 로 선언된 모델속성들에 값을 설정한다는것은 물리적 DB Table 의 해당 마당에 값을 대입하는 행위와 같은 흐름으로 보아야 한다.
때문에 **속성값들에 대한 설정**은 SQL 주입공격과 같은 보안성측면을 고려하며 개발자가 Database 의 data type 에 맞게 값대입이 되도록 일정한 로직을 통과시키는 내부메소들을 통하여 진행하도록 하여야 한다.
가령 $_id 값은 정수값이여야 하므로 setId()라는 메소드를 정의하고 여기서 입력되는 값을 정수형으로 무조건 변환시키도록 Filter::toInt($val) 을 적용할수 있다.
또한 $_password 값은 전문암호해쉬값이여야 하므로 setPassword()라는 메소드를 정의하고 입력값을 Filter::hashPassword() 거쳐 값을 받아들이도록 할수 있다.

```
class UsersModel extends Model
{
    private $_id = NULL;
    private $_name = NULL;
    private $_email = NULL;
    private $_password = NULL;

    public function setId($val)
    {
        $this->_id = Filter::toInt($val);
    }
    public function setName($val)
    {
        $this->_name = Filter::safeString($val);
    }
    public function setEmail($val)
    {
        $this->_email = Filter::safeString($val);
    }
    public function setPassword($val)
    {
        $this->_password = Filter::hashPassword($val);
    }
}
```

3) SQL 조작

Model 기초클래스는 PHP 가 제공하는 PDO 엔진을 사용한다. 때문에 다양한 DB 엔진에 대한 조작이 가능하다. 이에 대해서는 /base/DB.php 에 상세히 반영되어 있다.

기정으로 MySQL 엔진을 지정하였다.

/config/db.config.php 에서

```
define('DB_TYPE', 'mysql');
```

이 부분이다.

이브닝은 기초적인 DB 조작관련 메소들을 제공하고 있다.

즉 다중조회, 단일조회, 추가, 삭제, 변경 등 메소드를 지원한다.

이러한 메소드들의 이름은 개발자들이 정의하는 메소드명과 중복되지 않도록 하기 위해 public 속성이지만 _기호로 시작하도록 하였다.

- DB 접속 설정

  모델클래스에서 직접 설정하지 않는다. DB 설정에 따라 모델클래스실체가 창조될때 자동적으로 접속된다.

  /config/db.config.php 에서 설정한다.

  ```
  define ('DB_HOST', 'localhost'); // host 명
  define ('DB_USER', 'root'); // user 명
  define ('DB_PASS', ''); // db 암호
  define ('DB_NAME', 'evening'); // db 명
  ```

  이렇게만 설정하면 해당 모델클래스의 실체가 구성될때 DB 에 자동적으로 접속하게 된다.

  Model 기초클래스의 구축자메소드를 보면

  ```
  $this->_db = DB::init();
  ```

  바로 이부분에서 DB 접속이 자동적으로 진행된다.

- SQL 문 설정

  ```
  _set($sql, $data)
  ```

SQL 문은 매우 단순한것으로부터 복잡한것까지 그 형태는 셀수 없이 많다. 때문에 이브닝프레임워크에서는 다른 프레임워크에서 지원하는 QueryBuilder 같은 것을 특별히 제안하지 않고 매 개발자들이 고유한 SQL 문을 작성하여 쓰도록 한다. 이것은 자기가 작성한 SQL 문을 시각적으로 분석하기에도 편리할것이다.

실례로, users 테이블의 모든 레코드를 얻는 SQL 문을 설정하려면.

```
public function getUserAll()
{
    $sql = "SELECT * FROM users";
    $result = $this->_set($sql)->_getAll();
    if(empty($result)) return NULL;
    return $result;
}
```

와 같이 작성한다.

_set($sql, $data=NULL) 메소드는 두개의 파라미터를 받는데 첫번째 파라미터는 SQL 문이고 두번째 파라미터는 이 SQL 문에 대입할 값들이다.

이 메소드는 모델클래스실체 자체를 돌려준다.

- SQL 문 값대입

```
_set($sql, $data=NULL)
```

$data 는 array 형으로 SQL 문에 대입할 값들의 모임이다.

PDO 의 문맥에 따라 값을 대입하면 그 안전성도 비교적 높다.

아래에 값대입을 위한 세가지 방식에 대한 실례를 보여준다.

명시적인 값대입

```
$sql = "SELECT * FROM users WHERE name=:name OR
email=:email";
$data = array(':name'=>$this->_name, ':email'=>$this-
>_email);
$result = $this->_set($sql, $data)->_getRow();
```

익명의 값대입

```
$sql = "INSERT INTO users (name, email, password) VALUES
(?,?,?)";
$data = array($this->_name, $this->_email, $this-
>_password);
$result = $this->_set($sql, $data)->_insert();
```

직접적인 값대입

필요에 따라 직접적인 값대입을 하는 SQL 구문을 작성할수도 있다. 이 경우 보안성문제에 관심을 돌려야 한다.

```
$sql = "SELECT * FROM users WHERE name='".$this->_name."'";
$result = $this->_set($sql)->_getRow();
```

- 다중조회

`_getAll()`

조회결과값이 여러개인 경우 _getAll()메소드를 사용한다.

```
public function getUserAll()
{
    $sql = "SELECT * FROM users";
    $result = $this->_set($sql)->_getAll();
    if(empty($result)) return NULL;
    return $result;
}
```

- 단일조회

`_getRow()`

조회결과값이 한개일때 _getRow()메소드를 사용한다.

```
public function getUserById()
{
    $sql = "SELECT * FROM users WHERE id=?";
    $data = array($this->_id);
    $result = $this->_set($sql, $data)->_getRow();
    if(empty($result)) return NULL;
    return $result;
}
```

- 추가

`_insert()`

```
public function add()
{
    $sql = "INSERT INTO users (name, email, password) VALUES
(?,?,?)";
    $data = array($this->_name, $this->_email, $this-
>_password);
    $result = $this->_set($sql, $data)->_insert();
    if(empty($result)) {
        return false;
    }
    $id = $this->_lastInserted();
```

```
    return $id;
}
```

- 삭제

`_remove()`

```
public function delete()
{
    $sql = "DELETE FROM users WHERE id=?";
    $data = array($this->_id);
    $result = $this->_set($sql, $data)->_remove();
    return $result;
}
```

- 변경

`_update()`

```
public function updateEmail()
{
    $sql = "UPDATE users SET email=:email WHERE
name=:name";
    $data = array(':email'=>$this->_email,
':name'=>$this->_name);
    $result = $this->_set($sql, $data)->_update();
    return $result;
}
```

- SQL 문 직접 실행

`_run()`

CRUD(Creating, Reading, Updating, Deleting)기능말고도 테이블구조변경, 속성변경 등 여러가지 SQL 구문들이 많은데 이런 SQL 구문들은 _run()메소드로 실행할수 있다.

```
public function createTable()
{

    $sql = "CREATE TABLE IF NOT EXISTS `users` (
      `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
      `name` varchar(30) CHARACTER SET utf8 COLLATE
utf8_general_ci NOT NULL,
      `email` varchar(255) CHARACTER SET utf8 COLLATE
utf8_general_ci NOT NULL,
      `password` varchar(255) CHARACTER SET utf8 COLLATE
utf8_general_ci NOT NULL,
      `reg_date` timestamp(0) NOT NULL DEFAULT
CURRENT_TIMESTAMP(0),
      PRIMARY KEY (`id`) USING BTREE,
      UNIQUE INDEX `name`(`name`) USING BTREE,
```

```
      UNIQUE INDEX `email`(`email`) USING BTREE
    ) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET = utf8
COLLATE = utf8_general_ci ROW_FORMAT = Compact";

    $result = $this->_set($sql)->_run();

    return $result;
}
```

- Transaction 기능

DB 조작을 보다 안전하게 할 필요가 있다면 transaction 기능을 사용해야 한다.

이브닝은 transaction 기능을 위해 두가지 방법을 제공한다.

_transBegin(), _transCommit(), _transBack()

```
public function add()
{
    $this->_transBegin();
    $sql = "INSERT INTO users (name, email, password)
VALUES (?,?,?)";
    $data = array($this->_name, $this->_email, $this-
>_password);
    $result = $this->_set($sql, $data)->_insert();
    if(empty($result)) {
        $this->_transBack();
        return false;
    }
    $id = $this->_lastInserted();

    // Call nested transaction
    $this->_transBegin();
    $sql = "UPDATE users SET email=:email WHERE
name=:name";
    $data = array(':email'=>$this->_email,
':name'=>$this->_name);
    $result = $this->_set($sql, $data)->_update();
    if(empty($result)) {
        $this->_transBack();
    }
    $this->_transCommit();

    $this->_transCommit();
    return $id;
}
```

보는것처럼 transaction 을 여러개 창조해도 그의 순서를 유지한다. 즉 Nested 구조이다.

_transSwitch(true|false)

Transaction 기능을 열기|끄기 하는 기능이다. 이렇게만 설정하면 내부적으로 SQL 구문 실행이 성공하면 transCommit()이 실행되고 실패하면 transBack()이 실행된다.

Nested Transaction 기능을 지원하지 못한다.

```
public function createTable()
{
    $this->_transSwitch(true);

    $sql = "CREATE TABLE IF NOT EXISTS `users` (
      `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
      `name` varchar(30) CHARACTER SET utf8 COLLATE
utf8_general_ci NOT NULL,
      `email` varchar(255) CHARACTER SET utf8 COLLATE
utf8_general_ci NOT NULL,
      `password` varchar(255) CHARACTER SET utf8 COLLATE
utf8_general_ci NOT NULL,
      `reg_date` timestamp(0) NOT NULL DEFAULT
CURRENT_TIMESTAMP(0),
      PRIMARY KEY (`id`) USING BTREE,
      UNIQUE INDEX `name`(`name`) USING BTREE,
      UNIQUE INDEX `email`(`email`) USING BTREE
    ) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET =
utf8 COLLATE = utf8_general_ci ROW_FORMAT = Compact";

    $result = $this->_set($sql)->_run();

    $this->_transSwitch(false);

    return $result;
}
```

- SQL 구문 미리보기

`_dumpQuery()`

명시적 혹은 익명으로 값대입을 하는 SQL 구문을 분석하기 위해서 값대입후 SQL 구문을 볼 필요가 있을때 사용한다.

```
public function getUserById()
{
    $sql = "SELECT * FROM users WHERE id=?";
    $data = array($this->_id);
    $target = $this->_set($sql, $data);
    var_dump($this->_dumpQuery()); // 값대입후 SQL 문출력
    $target->_getRow();
    if(empty($result)) return NULL;
```

```
    return $result;
}
```

만일 $this->_id 값이 12 이라고 하면 값대입후 SQL 구문은

SELECT * FROM users WHERE id=12 로 출력된다.

반드시, _set()를 실행한 다음에 _dumpQuery()를 호출해야 해당 SQL 문을 고찰할수 있다.

8. Controller 와 Model 연결

먼저 /models/UsersModel.php 를 다음과 같이 구성하였다고 본다.

```
class UsersModel extends Model
{
    private $_id = NULL;
    private $_name = NULL;
    private $_email = NULL;
    private $_password = NULL;

    public function setId($val)
    {
        $this->_id = Filter::toInt($val);
    }
    public function setName($val)
    {
        $this->_name = Filter::safeString($val);
    }
    public function setEmail($val)
    {
        $this->_email = Filter::safeString($val);
    }
    public function setPassword($val)
    {
        $this->_password = Filter::hashPassword($val);
    }

    public function getUserAll()
    {
        $sql = "SELECT * FROM users";
        $result = $this->_set($sql)->_getAll();
        if(empty($result)) return NULL;
        return $result;
    }

    public function getUserById()
    {
        $sql = "SELECT * FROM users WHERE id=?";
        $data = array($this->_id);
        $target = $this->_set($sql, $data);
        $result = $target->_getRow();
        if(empty($result)) return NULL;
        return $result;
    }

    public function getUserByNameOREmail()
    {
        $sql = "SELECT * FROM users WHERE name=:name OR
email=:email";
        $data = array(':name'=>$this->_name, ':email'=>$this-
>_email);
        $result = $this->_set($sql, $data)->_getRow();
        if(empty($result)) return NULL;
        return $result;
```

```
    }

    public function updateEmail()
    {
        $sql = "UPDATE users SET email=:email WHERE name=:name";
        $data = array(':email'=>$this->_email, ':name'=>$this-
>_name);
        $result = $this->_set($sql, $data)->_update();
        return $result;
    }

    public function add()
    {
        $this->_transBegin();
        $sql = "INSERT INTO users (name, email, password) VALUES
(?,?,?)";
        $data = array($this->_name, $this->_email, $this-
>_password);
        $result = $this->_set($sql, $data)->_insert();
        if(empty($result)) {
            $this->_transBack();
            return false;
        }
        $id = $this->_lastInserted();

        // Call nested transaction
        $this->_transBegin();
        $sql = "UPDATE users SET email=:email WHERE name=:name";
        $data = array(':email'=>$this->_email, ':name'=>$this-
>_name);
        $result = $this->_set($sql, $data)->_update();
        if(empty($result)) {
            $this->_transBack();
        }
        $this->_transCommit();

        $this->_transCommit();
        return $id;
    }

    public function delete()
    {
        $sql = "DELETE FROM users WHERE id=?";
        $data = array($this->_id);
        $result = $this->_set($sql, $data)->_remove();
        return $result;
    }

    public function createTable()
    {
        $this->_transSwitch(true);

        $sql = "CREATE TABLE IF NOT EXISTS `users` (
          `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
          `name` varchar(30) CHARACTER SET utf8 COLLATE
utf8_general_ci NOT NULL,
```

```
            `email` varchar(255) CHARACTER SET utf8 COLLATE
utf8_general_ci NOT NULL,
            `password` varchar(255) CHARACTER SET utf8 COLLATE
utf8_general_ci NOT NULL,
            `reg_date` timestamp(0) NOT NULL DEFAULT
CURRENT_TIMESTAMP(0),
            PRIMARY KEY (`id`) USING BTREE,
            UNIQUE INDEX `name`(`name`) USING BTREE,
            UNIQUE INDEX `email`(`email`) USING BTREE
          ) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET = utf8
COLLATE = utf8_general_ci ROW_FORMAT = Compact";

          $result = $this->_set($sql)->_run();

          $this->_transSwitch(false);

          return $result;
      }
}
```

1) 모델로드

두가지 방식이 있다.

- Controller 의 _setModel() 메소드로 로드한다.
  아래의 실례에서는 /models/UsersModel.php 를 로드하여 UsersModel 클래스실체를 불러온다. 그리고 UsersModel 클래스에서 정의한 getUserAll()메소드를 호출하여 users 라는 테이블에 있는 모든 레코드를 조회하여 출력한다.

```
class UsersController extends Controller
{
    public function index()
    {
        $m = $this->_setModel('Users'); // Model 을 덧붙이지
않음
        $results = $m->getUserAll();
        var_dump($results);
    }
}
```

이 메소드로는 모델파일의 위치가 세분화되어도 그 상대경로를 지정해주어 로드할수 있는 우점이 있다.

실례로, /models/admin/UsersModel.php 파일을 로드하려면

_setModel('admin/Users') 로 작성하면 된다.

- 직접 모델클래스실체를 창조하여 로드할수 있다.

```
$m = new UsersModel(); // /models/UsersModel.php 파일 로드
```

이 경우에는 오직 /models/ 바로 안에 있는 모델파일들에서만 찾아 로드하므로 파일위치가 세분화된 모델은 로드할수 없는 결함이 있다.

2) 모델조작

Controller 에서 모델클래스의 속성값을 설정하고 DB 조회결과값을 얻는 조작에 대한 실례는 다음과 같다.

```
class UsersController extends Controller
{
    public function index()
    {
        /* /models/UsersModel.php 로드 */
        // $m = new UsersModel();
        $m = $this->_setModel('Users');

        /* users 테이블 창조 */
        $result = $m->createTable();
        echo 'users 테이블 창조: '.(empty($result)?'성공':'실패');

        /* users 테이블의 모든 레코드 가져오기 */
        $result = $m->getUserAll();
        echo '등록된 유저수: '.count($result);

        /* users 테이블에 새 레코드 추가 */
        $m->setName('test');
        $m->setEmail('test@test.com');
        $m->setPassword('123');
        $addedId = $m->add();
        echo '등록된 id: '.$addedId;

        /* 새 레코드 추가에 해당한 SQL 문 출력 */
        echo '실행된 SQL 구문: '.$m->_dumpQuery();

        /* 앞에서 새로 등록한 유저정보 가져오기 */
        $m->setId($addedId);
        $result3 = $m->getUserById();
        echo $result3['id'].':'.$result3['name'];

        /* test 유저의 이메일 변경 */
        $m->setName('test');
        $m->setEmail('evening@evening.com');
```

```
        $m->updateEmail();
        echo '실행된 SQL 구문: '.$m->_dumpQuery();
    }
}
```

9. 다중언어지원기능

여러가지 어종별로 출력할 콘텐츠들을 정의하고 필요에 따라 해당 어종콘텐츠를 호출할수 있는 기능이다.

1) 어종파일작성

- 모든 어종파일들은 /language/ 에 위치하여야 한다.
- 파일명은 어종을 구분하기 쉽게 정한다.

  English: en.php

  Japanese: jp.php

  Russian: ru.php
- 매 어종파일들은 $GLOBALS['gLang'] 라는 전역배열을 정의한다.

  en.php

```
$GLOBALS['gLang'] = array(
    'menu_home' => 'Home',
    'menu_blog' => 'Blog',
);
```

  ko.php

```
$GLOBALS['gLang'] = array(
    'menu_home' => '홈',
    'menu_blog' => '게시판',
);
```

- 어종콘텐츠는 key=>value 쌍으로 정의하여주며 모든 어종파일들에 똑같은 key 가 존재하여야 한다.

2) 어종파일적재

gImportLanguage($lang='en')

전역함수 gImportLanguage()를 사용하여 해당 어종파일을 로드할수 있다.

이 경우 파라미터를 주지 않으면 기정으로 en.php 를 로드한다.

파일확장자를 없애고 파라미터로 어종파일명을 입력해주면, /language/에서 해당 파일을 찾아 로드하게 된다.

3) 어종콘텐츠사용

어떻게 어종파일에 정의한 콘텐츠를 사용하는가를 실례를 들어본다.

- 전역함수 gText()

  HomeController index()에서 /views/home/index.php 페이지를 출력해줄때 페이지의 title 을 어종별로 변환해서 출력해주는 실례는 다음과 같다.

  HomeController.php

```
class HomeController extends Controller
{
    public function index()
    {
        gImportLanguage('ko');

        $title = gText('menu_home'); // '홈'

        $v = $this->_setView('home/index');
        $v->set('title', $title);
        $v->output();
    }
}
```

  /views/home/index.php

```
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <title><?php echo $title; ?></title><!-- '홈' 출력 -->

</head>

<body>
</body>

</html>
```

- 전역함수 gTextE()

  gText() 는 해당 어종콘텐츠를 리턴해주지만 gTextE()는 저래 출력해준다.

  즉 echo gText()나 같은 기능이다.

  <title><?php gTextE('menu_home'); ?></title>

- 동적작성

가령 “jina 가입회수 40 회” 를 출력하는데 jina, 40 이라는 글자는 동적으로 변경해야 한다면?

/language/ko.php

```
$GLOBALS['gLang'] = array(
    'menu_home' => '##id## 가입회수##count##회',
);
```

Controller 나 View 에서는

```
gText('menu_home', array('id'=>$id, 'count'=>$count))
```

즉 동적변경값을 ##key##로 정의하고 gText, gTextE 에서는 두번째 파라미터로 array(key=>value) 을 보내준다.

10. Middleware 기능

모든 요청처리에 앞서 먼저 실행하여야 할 중간로직이 있다면 middleware 기능을 사용할수 있다.

Middleware 클래스의 run()메소드에서 필요한 로직을 호출하면 된다.

```
class Middleware
{
    /**
     * Runs before any request from client.
     * A developer can customize the method to be suited an
application.
     */
   public static function run()
    {
        gImportLanguage('en');

        var_dump(Router::getUrl());
    }
}
```

위의 코드는 모든 요청에 대하여 먼저 어종파일 en.php 를 로드하고, 요청 URL 에 해당한 라우팅규칙이 어떻게 적용되었는지 상세정보를 출력시킨다.

11. 기본설정기능 해석

프레임워크의 구조화를 위한 기본설정들에 대하여 고찰한다.

1) URL, Path 작성

전역함수 gURL()

/config/base.config.php

```
function gURL($index)
{
    $g_config_url = array(
        'http_root'=>    MOD_REWRITE? BASE_URL : BASE_URL.'/index.php',
        'http_assets'=> BASE_URL.'/assets',

        // for file system
        'doc_root'=>     DOC_ROOT,
        'protected'=>    DOC_ROOT.DS."protected",
        'app'=>          DOC_ROOT.DS."protected".DS."app",
        'base'=>         DOC_ROOT.DS."protected".DS."base",
        'controllers'=> DOC_ROOT.DS."protected".DS."controllers",
        'models'=>       DOC_ROOT.DS."protected".DS."models",
        'views'=>        DOC_ROOT.DS."protected".DS."views",
        'utils'=>        DOC_ROOT.DS."protected".DS."utils",
        'includes'=>     DOC_ROOT.DS."protected".DS."includes",
        'config'=>       DOC_ROOT.DS."protected".DS."config",
        'resource'=>     DOC_ROOT.DS."protected".DS."resource",
        'vendors'=>      DOC_ROOT.DS."protected".DS."vendors",
        'cache'=>        DOC_ROOT.DS."protected".DS."cache",
        'language'=>     DOC_ROOT.DS."protected".DS."language",
        'assets'=>       DOC_ROOT.DS."assets",

        // Add more custom url
        'upload'=>       DOC_ROOT.DS."protected".DS."resource".DS."uploads",
    );

    return isset($g_config_url[$index])? $g_config_url[$index] : '';
}
```

파일구조에 따라 해당 경로에 접근하려면 gURL('upload')와 같이 호출하는것이 좋다.

모든 URL, Path 를 gURL()로 작성하고 호출하면 폴더구조가 바뀌거나 폴더명이 달라졌다고 해도 base.config.php 에서 설정부분만 바꾸어 주면 전반적인 코드에 영향을 주지 않고 깨끗이 반영된다.

2) 클래스맵설정

전역함수 gClassMap()

/config/base.config.php

클래스실체를 창조할때 어느 폴더에서 그 클래스파일을 찾겠는가를 정의하는 클래스맵을 설정할수 있다.

가령 /app/Security.php 파일을 위치하고 new Security() 했을때 이 파일을 로드하여 클래스실체를 창조하고 싶다면,

먼저 gURL()에 app 폴더의 절대경로를

```
'app'=> DOC_ROOT.DS."protected".DS."app",
```

와 같이 정의해주고 /base.config.php 의 gClassMap()에 'app'인덱스를 추가해주면 된다.

```
function gClassMap()
{
    return array(
        gURL('base'),
        gURL('controllers'),
        gURL('models'),
        gURL('app'),
        gURL('utils'),
    );
}
```

위의 코드는 어떤 클래스를 호출할때에 /base, /controllers/, /models, /app, /utils 순서대로 매 폴더에서 해당 클래스파일을 찾아 로드하게 된다.

3) gURL 의 유용성

gURL()

/config/base.config.php

앞에서 잠간 언급한것처럼 gURL()기능을 사용하면 폴더명이 바뀌거나 그 구조가 변경되었을때 세부적인 코드를 따라가며 변경하지 않고 gURL()에서만 변경해주면 쉽게 호환성을 맞출수 있다.

- http URL 작성에서 gURL 의 사용

/assets/ bootstrap-3.3.2/css/bootstrap.min.css 파일을 로드한다고 하면

```
<link rel="stylesheet" type="text/css" href="<?php echo gURL('http_assets');?>/bootstrap-3.3.2/css/bootstrap.min.css" media="all">
```

이렇게 하면 가령 루트폴더명인 assets 를 public 으로 바꾸었다고 해도 gURL 에서

'http_assets'=> *BASE_URL*.'/assets', 을 'http_assets'=> *BASE_URL*.'/public' 으로 변경만 하면 모든것이 무탈해진다.

이렇게 하지 않고 그냥 명시적인 상대경로를 사용한다면 즉

```
<link rel="stylesheet" type="text/css" href="assets/bootstrap-3.3.2/css/bootstrap.min.css" media="all">
```

어차피 구체적인 이 코드를 따라가면서 assets/ 를 public/으로 수정해주어야 할것이며 이런 호출부분이 만일 10 개이라면 10 개소를 따라 가며 다 수정해주어야 할것이다.

gURLE()가 있는데 이것은 echo gURL()과 같은 기능을 수행한다.

- File path 작성에서 gURL 의 사용

물리적 파일조작에서도 경로지정을 gURL()을 사용하는것이 좋다.

```
<?php include_once(gURL('includes').DS.'menu.inc.php');?>
```

즉 URL, Path 작성에 필요한 루트폴더나 중요 폴더에 대한 참조값을 gURL 에 정의해주고 gURL 을 통하여 URL, Path 를 작성하도록 하는것이 좋은 선택이다.

4) DB 관련설정

/config/db.config

- DB 접속설정

```
define ('DB_HOST', 'localhost');
define ('DB_USER', 'root');
define ('DB_PASS', '');
define ('DB_NAME', 'evening');
```

어떤 경우에는 특정 port 를 지정할 필요도 있을것이다. 이경우

define ('DB_PORT', 1024); 와 같이 정의하고 사용하면 될것이다.

- DB 종류추가

gDbLink()

현재 이브닝은 MySQL, PGSQL 만을 지원한다.

만일 MSSQL 과 같은 새로운 DB 형을 추가하려고 한다면 gDbLink()에서 그 링크정보를 작성해주면 된다.

```
function gDbLink($index)
{
    $g_db_link = array(

'mysql'=>'mysql:host='.DB_HOST.';dbname='.DB_NAME.';charset=utf8',
'pgsql'=>'pgsql:dbname='.DB_NAME.';host='.DB_HOST,
'mssql'=>'sqlsrv:Server='.DB_HOST.';Database='.DB_NAME
,
    );

    return isset($g_db_link[$index])? $g_db_link[$index] : null;
}
```

링크정보는 PHP PDO 의 DB 별 DSN 규칙을 그대로 사용하면 된다.

- 사용할 DB 종류 지정

```
define('DB_TYPE', 'mysql');
```

여기서 gDbLink()에 정의한 DB 형중 어느 하나를 지정해주면 된다.

5) Namespace 맵설정

gNamespaceMap()

/config/namespace.config.php

Namespace 가 붙은 클래스나 객체를 다루려면 해당 파일의 물리적위치를 정확히 지정해주어야 한다.

유명한 PHPMailer 라이브러리를 사용해야 한다고 하자.

PHPMailer 의 모든 클래스들은

```
namespace PHPMailer\PHPMailer;
class PHPMailer
{}
```

와 같이 `PHPMailer\PHPMailer` 라는 `namespace` 가 붙어있다.

이제 `Mailer.php` 에서 `PHPMailer` 라이브러리를 사용한다고 할때

```
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\SMTP;
use PHPMailer\PHPMailer\Exception;
```

```
class Mailer
{
    $mail   = new PHPMailer();
}
```

와 같이 사용하게 된다.
사실상 PHPMailer 클래스실체를 찾으려면 gClassMap()에 정의된 맵에서 찾게 되지만 namespace 가 붙은 대상은 gNamespaceMap()에 정의된 맵에서 찾도록 구성하였다.

```
use PHPMailer\PHPMailer\PHPMailer
$mail   = new PHPMailer();
```

이것은 원래
```
$mail   = new PHPMailer\PHPMailer\PHPMailer();
```
를 의미한다.

그런데 사실은 PHPMailer.php 파일이
/PHPMailer/src/PHPMailer.php 에 존재한다.
결국 PHPMailer\PHPMailer\PHPMailer() 은

require "PHPMailer/src/PHPMailer.php";

new PHPMailer();

와 같은 로직을 요구하게 된다.

이와 같은 물리적경로찾기를 자동적으로 쉽게 하도록 하기 위해 gNamespaceMap()에 namespace 별 절대경로를 설정해준다.

```
function gNamespaceMap()
{
    return array(
        'PHPMailer\\PHPMailer' =>
NAMESPACE_BASEPATH.DS.'PHPMailer'.DS.'src',
}
```
이렇게 설정하면 require "PHPMailer/src/PHPMailer.php"와 같은 코드작성이 필요없다.

/vendors/ 에는 외부라이브러리들을 위치한다.

한가지 실례를 더 보자.

Image 조작용 라이브러리인 ImageMaster3 을 사용하려고 할때

```
namespace Gumlet;
class ImageResize
{}
```

이고 ImageResize.php 파일이 /vendors/ImageMaster3/lib/ImageResize.php 에 있다고 할때

이 라이브러리를 사용하는 코드에서

```
use \Gumlet\ImageResize;
new ImageResize()
```

로 했다면

gNamespaceMap()에서 다음과 같이 설정해주면 된다.

```
function gNamespaceMap()
{
        return array(
        'PHPMailer\\PHPMailer' =>
NAMESPACE_BASEPATH.DS.'PHPMailer'.DS.'src',
        'Gumlet' =>
NAMESPACE_BASEPATH.DS.'ImageMaster3'.DS.'lib',
    );
}
```

6) Debug 방식설정/해제

일단 운영을 시작하면 경고와 같은 일체 비정상적인 내용들이 출력되지 말아야 한다.

그러나 개발과정에는 이러한 내용들을 보아야 문제원인을 쉽게 해명하고 대책할수 있다.

이를 위해 프레임워크에서는 Debug 방식을 설정하거나 해제하는 설정기능을 지원한다.

index.php 파일에서

```
    define ('DEBUG', 1); // 0: ignore debug, 1: allow debug.
```

바로 이부분이다.

1 로 설정하면 Debug 방식이며 0 이면 해제한다.

즉 프로젝트를 배포할때에는 0 으로 설정해주어야 할것이다.

Evning.php, Loader.php, Model.php 에서 DEBUG 상수값을 사용한 부분을 참고한다.

12. 유용한 기능들

/utils/에는 파일조작, 벤치마크, 유효성검사 등 유용한 기능들을 지원하는 클래스파일들이 존재한다. 매 프로젝트의 내부로직에 종속되지 않고 PHP 전반적인 기능으로써 사용할수 있는 라이브러리형식의 기능들은 다 여기에 작성해준다. 외부라이브러리들은 /vendors/에 위치해준다.

1) Filter

- Filter::safeString($input);

  입력값에서 ‘, ”, <, >, &, \0, \xla, \x8 등 보안위협을 내포하는 특수기호들을 무의미성고유기호로 변환한다.

- Filter::unsafeString($input)
  safeString() 에 의하여 변환된 특수기호들을 원래대로 복원한다.
- Filter::toStr()
  입력값을 문자열형으로 변환한다. 문자열형으로 변환할수 없는 입력값은 null 로 돌려준다.
- Filter:toInt()
  입력값을 정수형으로 변환한한다. 실패하면 null 로 돌려준다.
- Filter::hashPassword()
  입력값을 암호해쉬화한다. 암호해쉬는 PHP 의 **password_hash()** 함수를 사용하므로 PHP5.5 부터 정상작동한다.
- Filter::checkPassword($val, $password)
  원시값 $val 을 암호해쉬한것과 $password 와 같은가를 비교판단하여 결과를 돌려준다.

2) Session

- Session::start()
  안전하게 session 을 시작한다.
- Session::clear()
  모든 session 을 삭제한다.
- Session::setSession($name, $val)
  세션명이 $name 인 세션에 $val 값을 할당한다.
- Session:getSession($name)
  세션명이 $name 인 세션값을 얻는다.

3) FileUtils

- FileUtils::output($path, $outname)

$path 에 있는 파일을 $outname 명으로, 해당 MIME 헤더로 말단에 파일내용을 돌려준다.

- FileUtils::getExtension1($path)
  $path 에 지정된 파일의 확장자를 안전하게 추출한다.
  PHP 의 pathinfo() 는 일부 정상값을 출력하지 못한다.
- FileUtils::save($path, $data)
  $path 에 해당 $data 를 써넣는다.
- File::replaceContents($file, $search, $replace, $to=null)
  $file 의 내용을 읽어 $search 부분을 $replace 로 전부 교체하고 $to 로 새로 파일을 보관하거나 $to 가 없으면 $file 을 덧쓰기한다.
- FileUtils::getFileSize($file)
  $file 의 크기를 정확히 얻는다. PHP 의 filesize()는 32 비트 체계에서 2GB 이상의 파일크기를 얻지 못하는 결함이 있다.
- FileUtils::findAllFiles($dir)
  $dir 와 하위폴더를 포함하여 존재하는 모든 파일들을 조회하여 리스트를 돌려준다.

4) Benchmark

해당 로직의 실행시간을 미리초단위로 계수할수 있다.

```php
$b = new Benchmark();
$b->mark('start');
/* 어떤 로직 */
$b->mark('end');
echo $b->eplased_time('start', 'end'); // 로직실행시간
```

이외에도 GUID, Mime, Captcha, FileDownloader 등이 더 있다.

공통적으로 사용하기에 좋다고 생각되는 메소드들은 이렇게 /utils/에 따로 정의해주면 그것이 그대로 프레임워크의 유용한 기능으로 고착되는것이다.

종장

이브닝은 프레임워크를 어떻게 만드는가 하는것을 보여주는 샘플이다.

/base/Router.php 클래스의 라우팅로직을 자기식대로 교체하면 전혀 새로운 프레임워크를 만들수도 있다.

요청 URL 에 대응한 라우팅방식을 자기식대로 규제하고 그 구현을 Router::prepare() 에서 구체적으로 반영하면 될것이다.

DB 관련부분이 미약하다고 생각되면 현존 구조에서 그 기능을 더 확장해주면 된다.

자기만의 프레임워크를 만들수 있는 감각을 주자는것이 바로 이브닝의 목표이며 완벽한 새로운 프레임워크를 누군가 또 만들 자신심을 가지게 하는것이 이브닝의 목적이다.