

<ASP.NET MVC3 강좌 리스트>

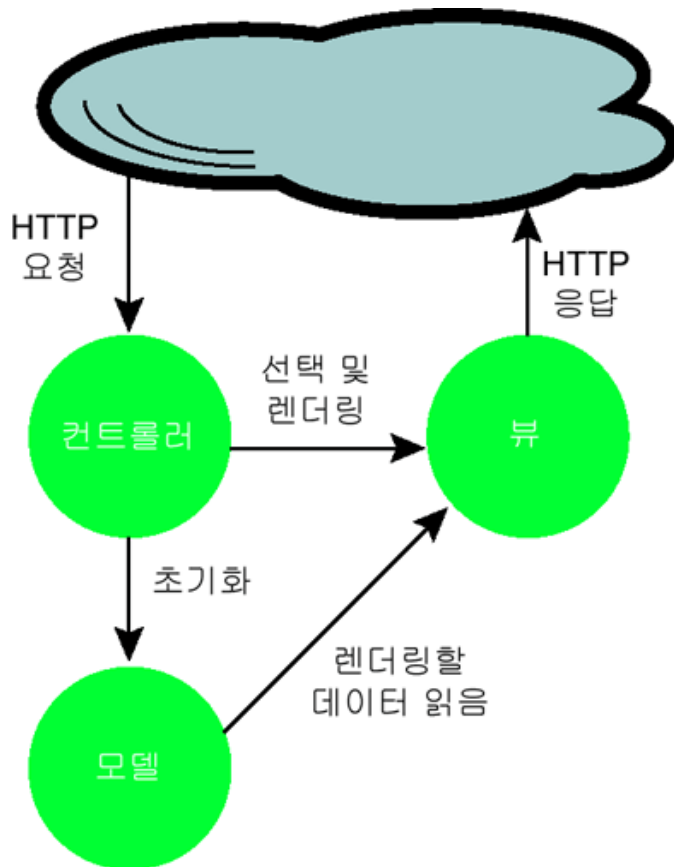
목차

[ASP.NET MVC3 강좌] 1. MVC(Model- View – Controller)	2
[ASP.NET MVC3 강좌] 2. MVC 환경 세팅.....	5
[ASP.NET MVC3 강좌] 3. HelloWorld MVC	9
[ASP.NET MVC3 강좌] 4. Layout , Partial Page	15
[ASP.NET MVC3 강좌] 5. Model 을 View 에서 표현하기	19
[ASP.NET MVC3 강좌] 6. MVC 의 매력적인 기능 스캐폴딩!(With EF).....	25
[ASP.NET MVC3 강좌] 7. 유효성 검사	30
[ASP.NET MVC3 강좌] 8. ModelBinder	34
[ASP.NET MVC3 강좌] 9. MVC ActionResult 종류 살펴보기	37
[ASP.NET MVC3 강좌] 10. MVC 처리 프로세스 & ActionFilter	41
[ASP.NET MVC3 강좌] 11. Global.asax.....	44
[ASP.NET MVC3 강좌] 12. (번외편) Repository Pattern	47
[ASP.NET MVC3 강좌] 13. MEF(Managed Extensibility Framework) in MVC	50
[ASP.NET MVC3 강좌] 14. Javascript in MVC	54
[ASP.NET MVC3 강좌] 15. Javascript Intellicense in Visual Studio.....	57
[ASP.NET MVC3 강좌] 16. JSON in MVC - 1.....	59
[ASP.NET MVC3 강좌] 17. JSON in MVC - 2.....	62
[ASP.NET MVC3 강좌] 18. HTML5 in MVC	65
[ASP.NET MVC3 강좌] 19. MVC Tips 1 - HandleUnknownAction	69
[ASP.NET MVC3 강좌] 20. MVC Tips 2 - Cache ActionFilter	71

[ASP.NET MVC3 강좌] 1. MVC(Model- View – Controller)

MVC 는 제록스 펠러앨토 연구소에서 스몰토크 관련 일을 하던 Trygve Reenskaug 에 의해 1979 년에 처음으로 고안되었다. 상당히 고전적인 패턴중의 하나로 MVC 라는 개념은 수 많은 형태의 패턴을 재창조해 낼 만큼 뛰어난 패턴으로 평가 받고 있다. 이 패턴에 영향 받은 대표적인 패턴은 Microsoft 가 1990 년에 사용한 MVP 패턴이 있으며 이것은 ASP.NET 에도 사용되어서 많은 사용자에게 인기를 얻었다. 최근에 WPF 에서 많이 사용되고 있는 MVVM 패턴도 MVC 에서 파생된 패턴 중에 하나라고 할 수 있다.

MVC 는 사용자 인터페이스로부터 비즈니스 로직을 분리하여 서로에게 영향을 주지 않고 수정할 수 있게 하는 것을 목표로 하고 있으며 각각 분리된 모듈로써 서로가 해야 할 역할이 확실하게 정의되어 있다.



[그림 : MVC 의 도식도 From MSDN]

- Model : MVC의 모델은 실질적인 데이터 풀링 소스의 데이터 타입에 근거한다. 종종 이것들은 Database Table 의 row 로써 많이 사용되게 된다. 최대한 원형을 보존한 상태지만 실제 어플리케이션에서 사용할 수 있는 형태의 최소한의 가공이 이루어지게 되는데 예를 들면 바이너리 객체를 ImageType으로 만들어 주거나, xml stream을 XDocument 객체로 만들어주는 것 등이 있다. DAL(Data-Access-Layer) , BLL(Business Logic Layer) 에서DAL에 해당하는 부분이다.
- View : MVC에서의 View는 사용자에게 렌더링 되어서 보여질 UI단의 작업을 담당하고 있다. 이곳에서 데이터의 재 가공이 이루어지면, 이미 MVC로써의 분리에는 실패한 것 이라고 볼 수 있다. (물론 DateTime 을 년/월/일 식으로 표현하거나, DateTimePicker 등에 호환시키기 위한 형 변환 등은 가능하다.) 간단하게 View는 데이터를 보여주는 것에만 집중 하는 것이라고 볼 수 있다.
- Controller : MVC에서의Controller 은 BLL(Business Logic Layer) 에 가장 가까운 것으로써, 실제 View 에서 데이터를 보여주거나, Model 을 검사하고, 그에 따른 렌더링 방식을 결정하는 것을 목표로 한다. 즉, 실질적인 로직은 여기에 대부분 포함된다.

MVC 가 처음 세상에 나왔을 때, 사용자들은 기존 ASP.NET 과 ASP.NET MVC 중 어떤 것을 선택해야 하는지 많이 혼동 하였다. 하지만 오래지나지 않아 MVC 프레임워크가 사용하기에 따라서 ASP.NET 보다 더 많은 생산성을 창출해 낼 수 있다는 사실이 알려지게 되었다. 물론 ASP.NET Classic 을 전부 커버할 수는 없지만, MVC 가 기존 ASP.NET 을 앞지를 수 있는 몇 가지 근거가 있다.

1. Url Routing 이 자동적으로 지원되어 사용자 친화 URL 을 만들기 용이
2. JQuery 프레임워크와 그 개발 시기가 일맥상통하여, 상대적으로 컨트롤을 만들기가 용이
3. 10 년 이상된 기존 ASP.NET 보다 최신 기술을 삽입하기가 용이
4. ViewState 가 없기 때문에 웹페이지가 가볍다.
5. 페이지 별로 Controller 가 존재하던 기존 ASP.NET 과 달리 모든 코드가 Method 형태로 되어 있어 단위테스트에 강하다.
6. 카테고리 별로 Controller 을 묶을 수 있다.

그렇지만 ASP.NET Classic 이 MVC 보다 강한 점도 있다.

1. 컨트롤의 삽입이 빠르다.
2. 이벤트 중심이어서 해당 이벤트를 작성하는 것만으로도 빠르게 사용자 요청을 처리할 수 있다.
3. 한 페이지에 많은 기능을 빠른 시간 내에 삽입할 수 있다.

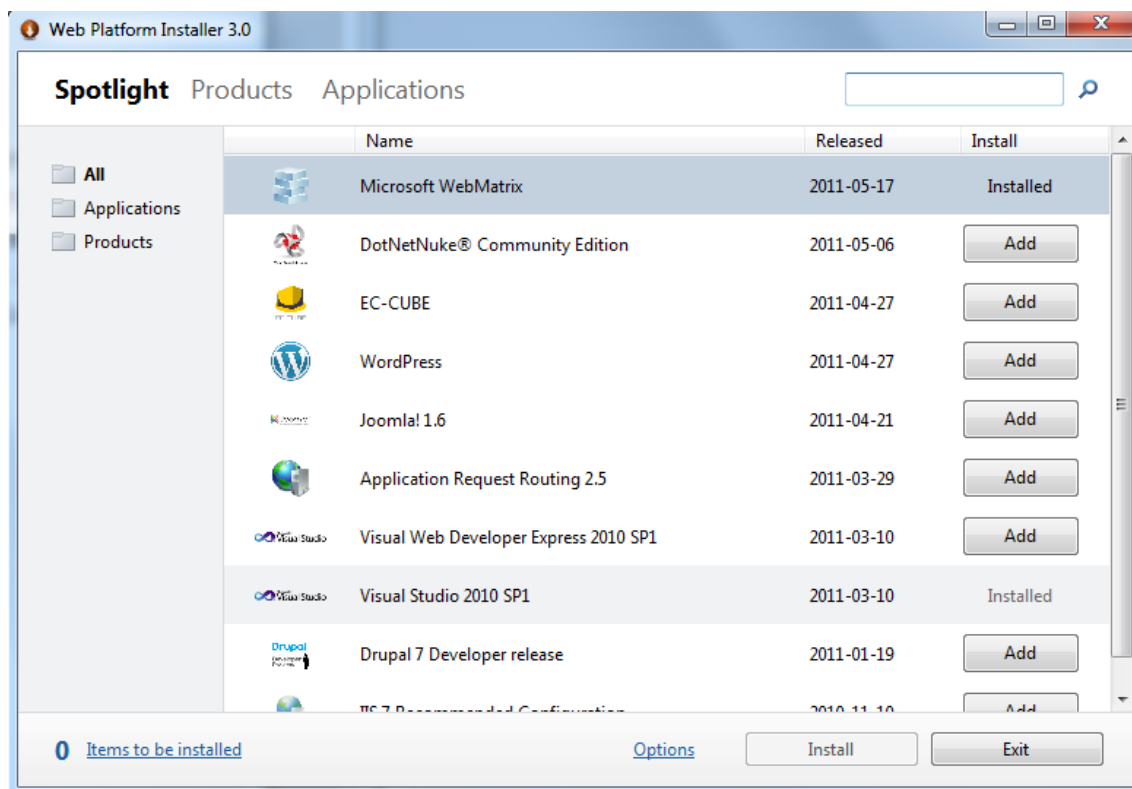
Summary

웹 어플리케이션의 성격에 따라서 MVC 로 개발해야 할 웹 어플리케이션 과 ASP.NET Classic 으로 개발해야 하는 웹 어플리케이션이 구별이 가능하다. 분명 한 페이지에 많은 기능이 삽입되면서, 결과를 빠르게 도출해 내야 하는 페이지에서는 ASP.NET Classic 을 사용하는 것이 유리하며 많은 사용자가 몰리고, 접근성이 중요하며, 사용자의 사용 경험이 중요한 사이트 (특히 요즘의 SNS 가 그렇죠)에서는 MVC 가 강점을 가진다고 볼 수 있다.

[ASP.NET MVC3 강좌] 2. MVC 환경 세팅

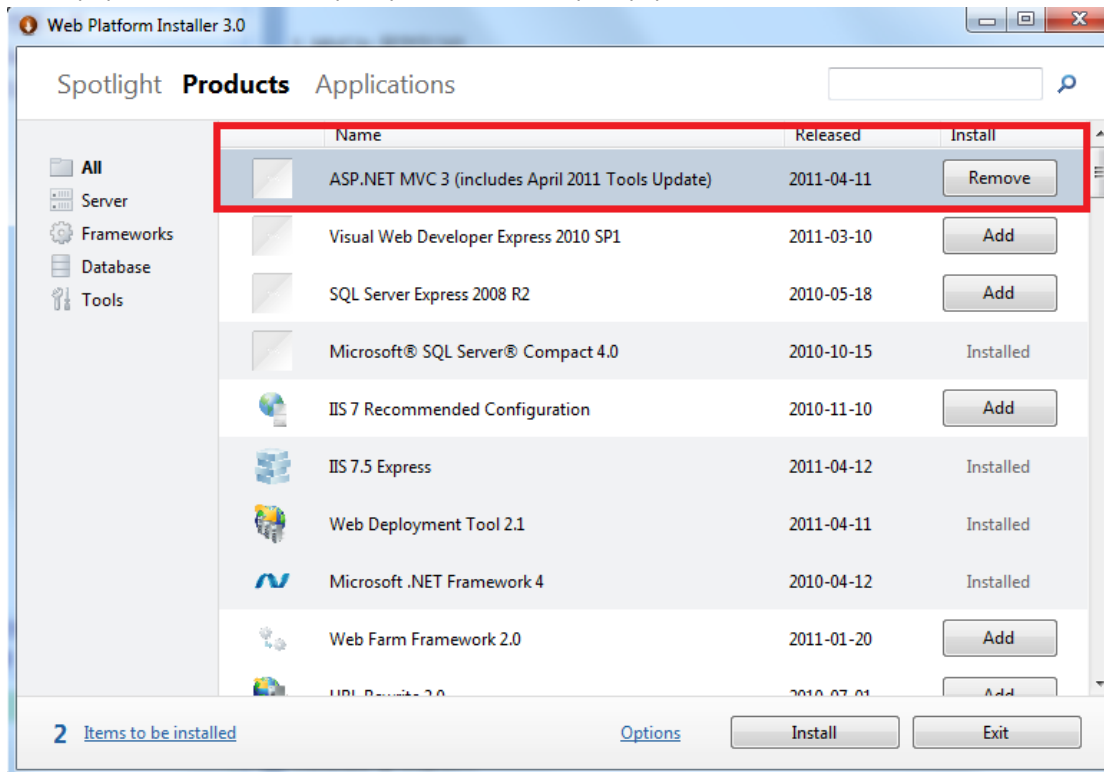
MVC 를 사용하기 위해서는 기본적으로 닷넷 3.5 이상 Visual Studio 2008 이상의 사양이 필요 하다. MS 에서는 WPI(Web Platform Installer)라는 것을 선보이게 된다. WPI 는 웹 개발에 필요한 모든 요소를 설치하고 심지어 고급 규모의 웹 어플리케이션까지 설치할 수 있는 툴로서 개발 초반에 개발 환경을 세팅할 때 많이 사용하는 툴이다.

1. 먼저 Visual Studio 2010 을 설치 한다. 다운로드 는 이곳(<http://www.microsoft.com/visualstudio/ko-kr/download>) 에서 가능 하다. 평가판 외에도 MS 는 여러 곳에서 공식적으로 툴을 무료로 사용하는 방법(DreamSpark 등) 을 제공하니 그런 기회를 살펴보는 것도 좋을 것 같다.
2. 웹메트릭스로 잘 알려진 Web Platform Installer 를 설치한다. 관련 정보는 이곳(<http://www.microsoft.com/web/downloads/platform.aspx>)에서 찾아볼 수 있다.
3. 이제 설치를 진행 한다.

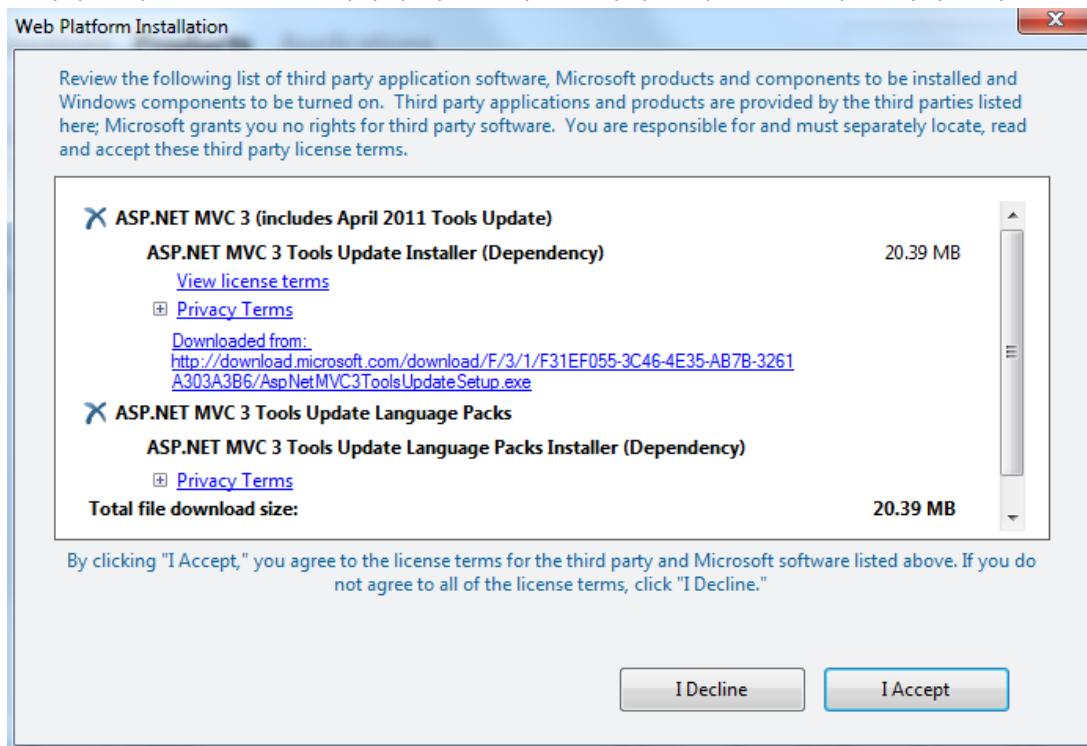


[그림 : Web Platform Installer]

간단하게 Product 탭을 선택한 후 MVC3 을 선택합니다.

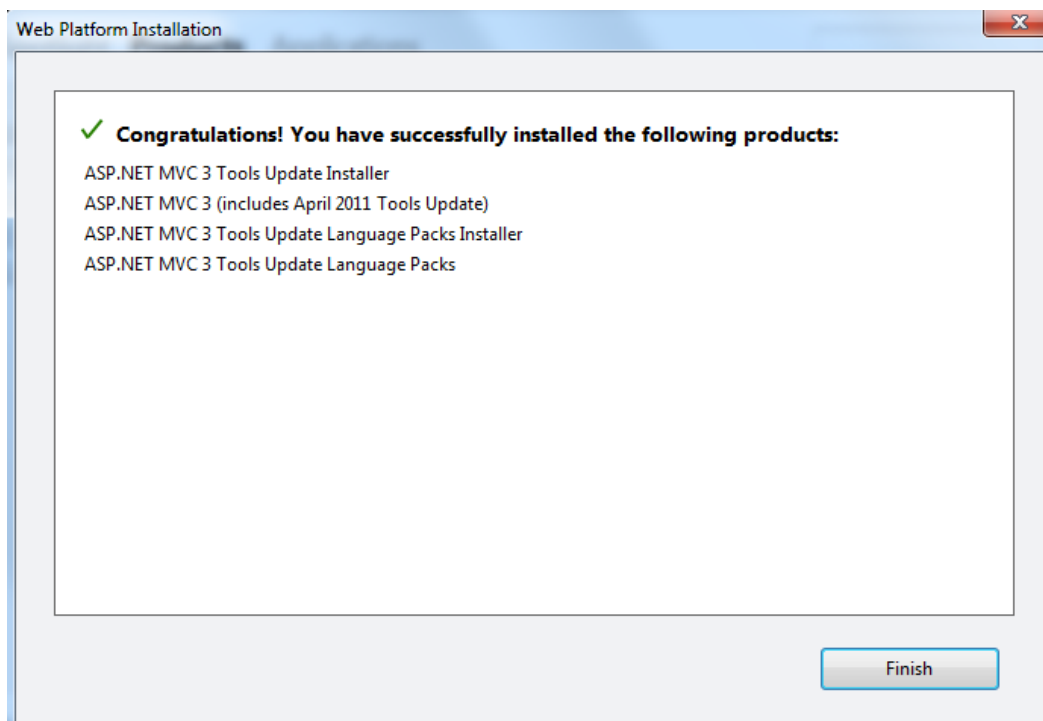
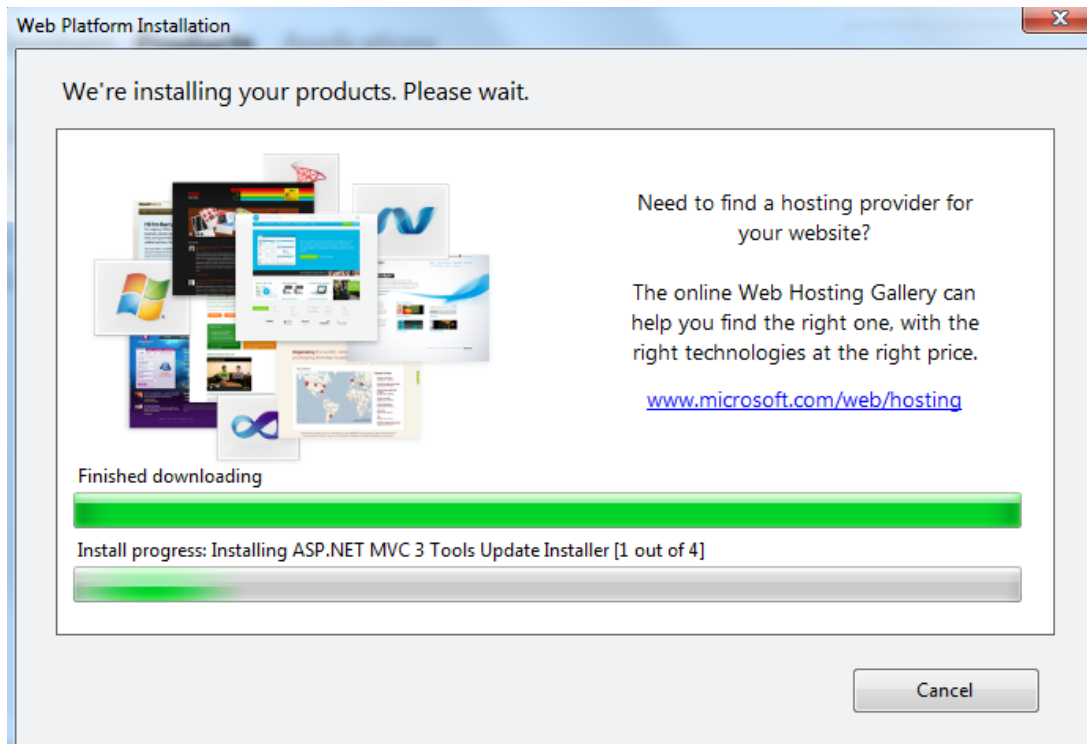


선택하고 나면 , MVC 를 설치하기 위한 종속성을 가지는 제품들을 전부 설치해 준다.

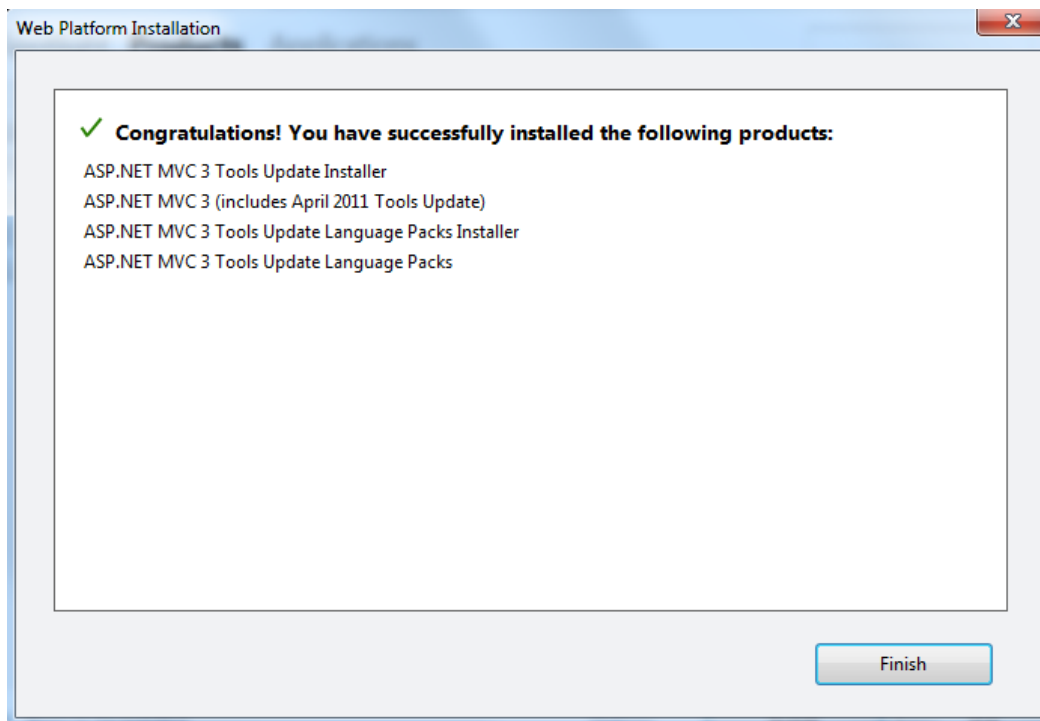


[그림 : MVC3 에 종속성을 가지는 모든 컴포넌트 설치]

간단하게 클릭 하나만으로 설치가 완료 된다.



[그림 MVC3 관련 제품 설치중]



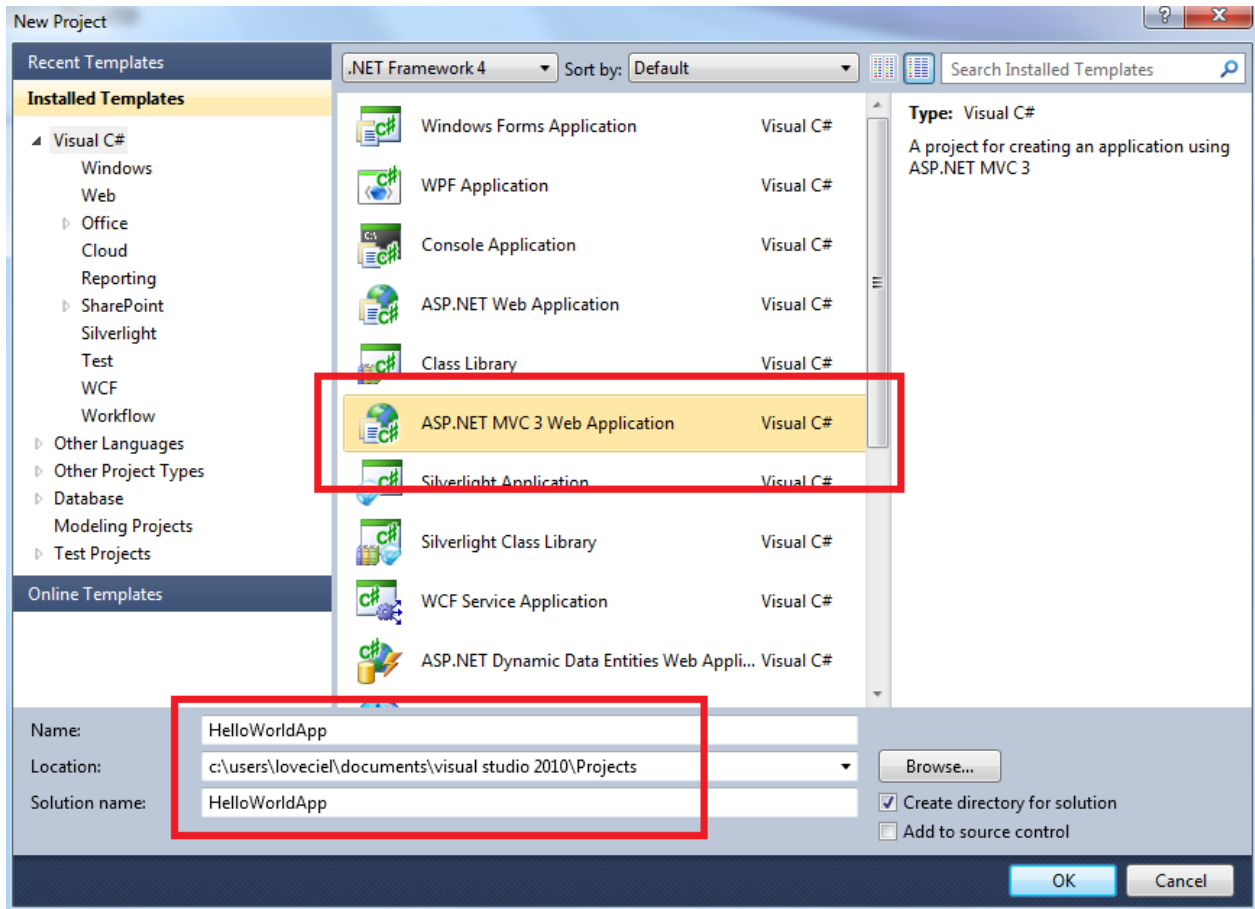
[그림 : MVC3 설치 완료]

이로써 MVC 를 위한 개발이 준비가 되었다.

Summary

기존에는 하나의 플랫폼을 설정하기 위해 이곳 저곳에서 필요한 컴포넌트를 다운받았었는데, 웹 플랫폼 인스톨러의 출현 이후에는 그럴 필요가 거의 없어졌다.

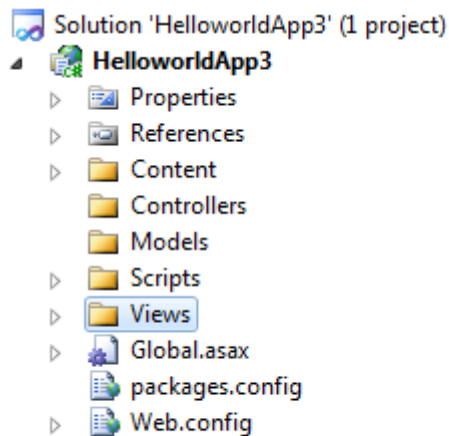
[ASP.NET MVC3 강좌] 3. HelloWorld MVC



WPI 를 이용해서 MVC 설치를 정상적으로 진행하셨다면 위와 같이 MVC3 을 개발할 수 있는 환경이 준비 되었다. 여기서는 기본적인 어플리케이션을 통해 MVC 의 대략적인 구조를 실습하려고 한다.

먼저 Project > WebApplication > MVC3 을 설치하시고 , Empty Project 를 선택한다.

MVC 프로젝트는 데이터를 가져와서 재가공하는 Model 과 그것을 핸들링하는 Controller, 그리고 핸들링된 결과값을 보여주는 View 로 이루어져 있다. 그러나 기본적인 MVC 어플리케이션 템플릿에는 Model 이 따로 존재하지 않는다.



1. Routing

MVC 는 기초적으로 SEO 친화적인 URL 인 Routing 을 기본골격으로 하고 있다. 따라서 Routing 에 대한 이해는 중요하다.

아무튼 이 Routing 은 Global.asax 에 정의가 되어 있다.

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

routes.MapRoute(
    "Default", // Route name
    "{controller}/{action}/{id}", // URL with parameters
    new { controller = "Home", action = "Index", id = UriParameter.Optional }
);
```

Routing 은 위와 같은 형식으로 이루어져 있으며, 이곳에서 실제 이동해야 할 페이지를 정의 한다. 자신이 원하는 Routing 형식이 있을 때는 routes.MapRoute() 메서드를 이용해서 자신이 원하는 라우팅 방식을 삽입하면 된다.

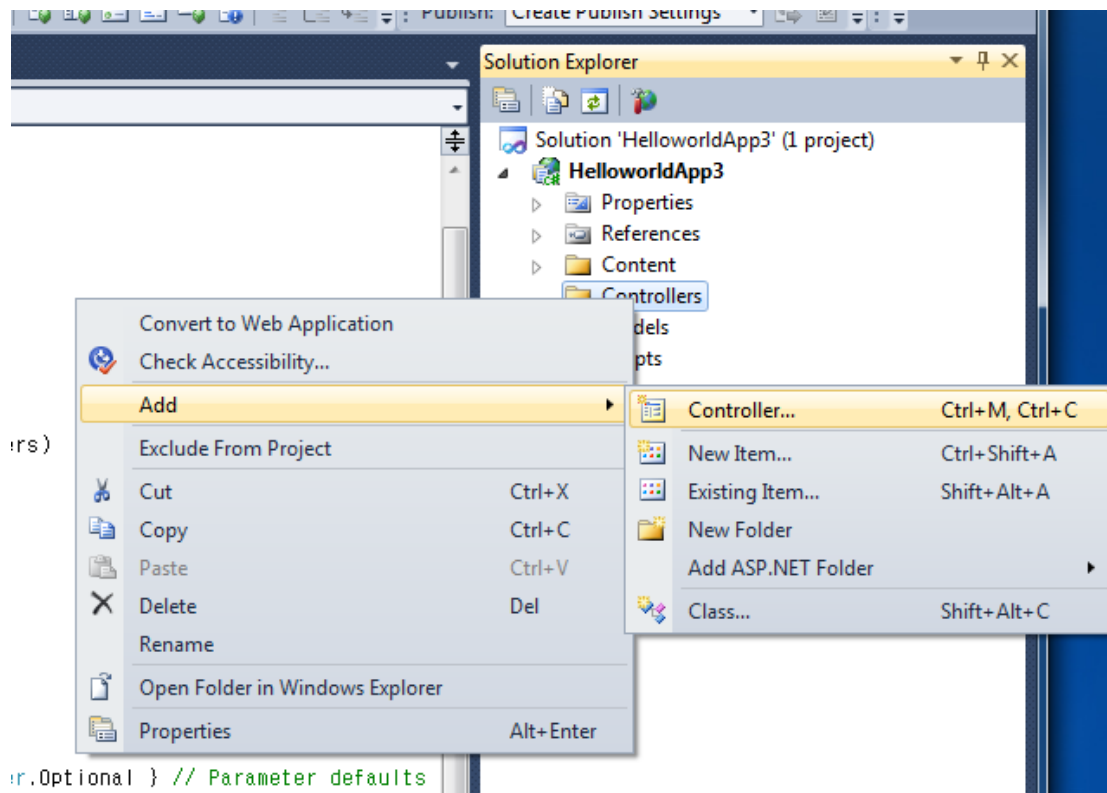
routes.MapRoute("라우팅 이름", "라우팅 규칙", "기본값")

현재 Routing 에서 정의된 것은 웹 페이지명을 controller 명/action 명 /변수 id 로 정의한 라우팅이라고 풀이할 수 있다. 그 뒤에 기본값이 정의됨으로써, 사용자가 정의한 라우팅이 아닌 경우 기본값에 따른 페이지를 읽어올 수 있다. 지금 정의된 라우팅에서는 controller 값이 home 으로 action 값이 index 로 정의 됨으로써, <http://기본호스트/> 를 입력하였을 때 <http://기본호스트/home/index> 로 이동한다는 것을 보여준다. 이 라우팅은 절차적인 방식으로 정의 되며, 위에서 정의된 라우팅과 밑에서 재정의된 라우팅이 겹칠 경우 위에서 정의된 라우팅으로 동작하니 주의해야 한다.

그 외에 global.asax 에서는 글로벌 필터 같은 MVC3 에서 추가된 기능이 사용있다.

2. Controller

Routing 에 따르면 현재 <http://localhost> 를 호출하게 되면 Home 컨트롤러에 index Action 을 호출하도록 설정되어 있는 것을 알 수 있다. 그렇지만 현재 MVC 프로젝트에 있는 컨트롤러에는 아무것도 정의되어 있지 않음을 알 수 있다. 먼저 올바르게 페이지를 표시하기 위해 HomeController 를 생성해 볼 수 있다. 컨트롤러폴더에서 오른쪽 클릭해서 add 를 선택하면 기존 asp.net 과 달리 controller 를 선택하는 메뉴가 새롭게 있는 것을 알 수 있는데 이것은 asp.net MVC 에서 선보이는 스캐폴딩이라고 하는 기능으로써 , 원하는 코드를 자동으로 생성해주는 개념이다. 이 Controller 를 선택하고 HomeController 를 생성하도록 한다.



[그림 : 컨트롤러 생성]

```

namespace HelloWorldApp3.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/

        public ActionResult Index( )
        {
            return View();
        }
    }
}

```

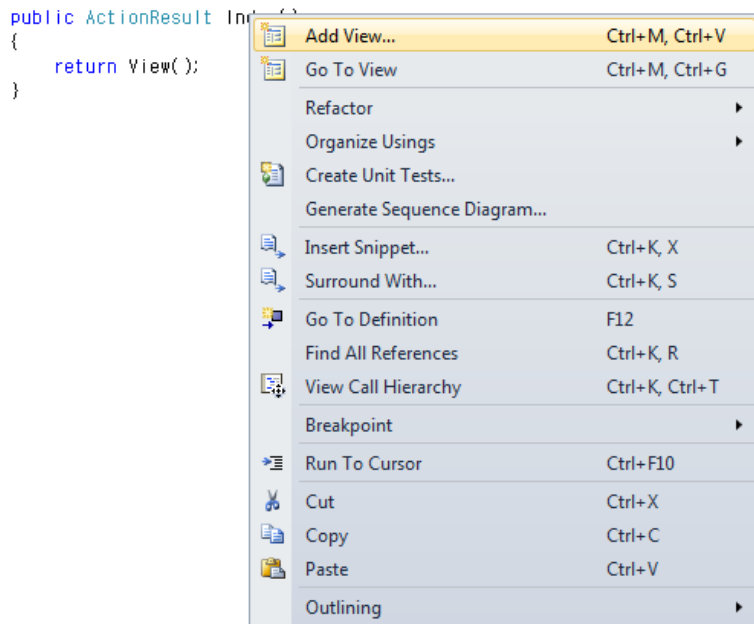
[코드 : 생성된 컨트롤러와 액션]

위와 같이 생성된 컨트롤러에 자동으로 Index Action 이 추가됨을 알 수 있다.

이로써 사용자는 Index 에 사용자가 원하는 행동을 삽입해 View 에 보여질 Model 을 가공해서 보여줄 수 있게 된다.

3. View

Controller 는 생성되었지만, 여전히 이 Controller 의 결과값을 보여줄 View 는 생성되지 않았다. 이 View 또한 간단하게 생성이 가능하다.



위와 같이 Action 에서 오른쪽 클릭을 하게 되면 AddView 라는 컨텍스트 메뉴가 보이는 것을 알 수 있다. 이를 이용해서 개발자는 간단하게 View 까지도 생성할 수 있다. 이제 localhost 를 실행하면 정상적으로 화면이 출력됨을 알 수 있다.

4. HelloWorld

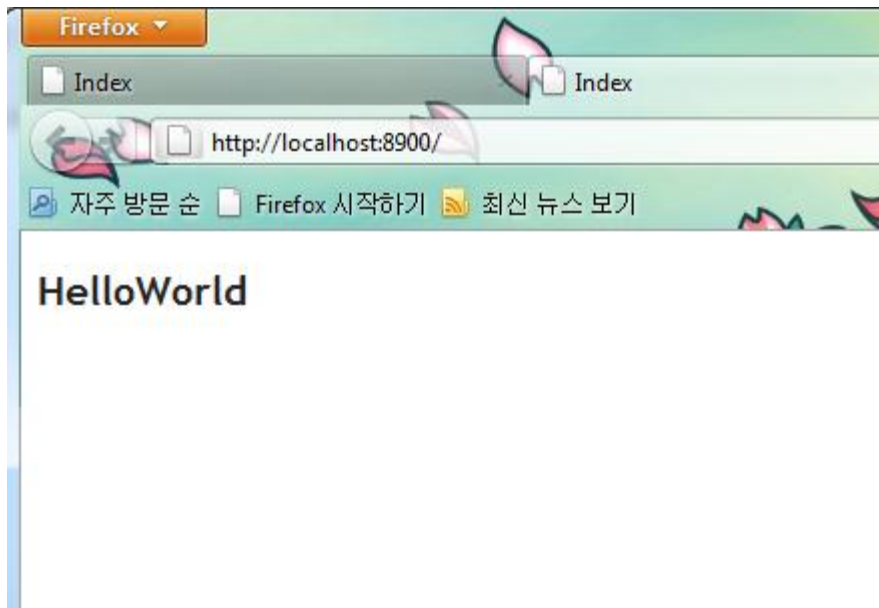
이제 Controller 에서 View 를 조작하는 법을 설명할 차례이다. 먼저 Index Action 에 다음과 같은 코드를 삽입합니다.

```
public ActionResult Index()
{
    ViewBag.HelloWorld = "HelloWorld";
    return View();
}
```

이후에 Home/Index.cshtml 에 Index 라는 글자 대신 다음의 코드를 삽입한다.

```
<h2>@ViewBag.HelloWorld</h2>
```

이제 결과값을 보면 Index Action 에서 정의한 ViewBag.HelloWorld 의 값이 정상적으로 반영되어 있는 모습을 확인할 수 있다.



Summary

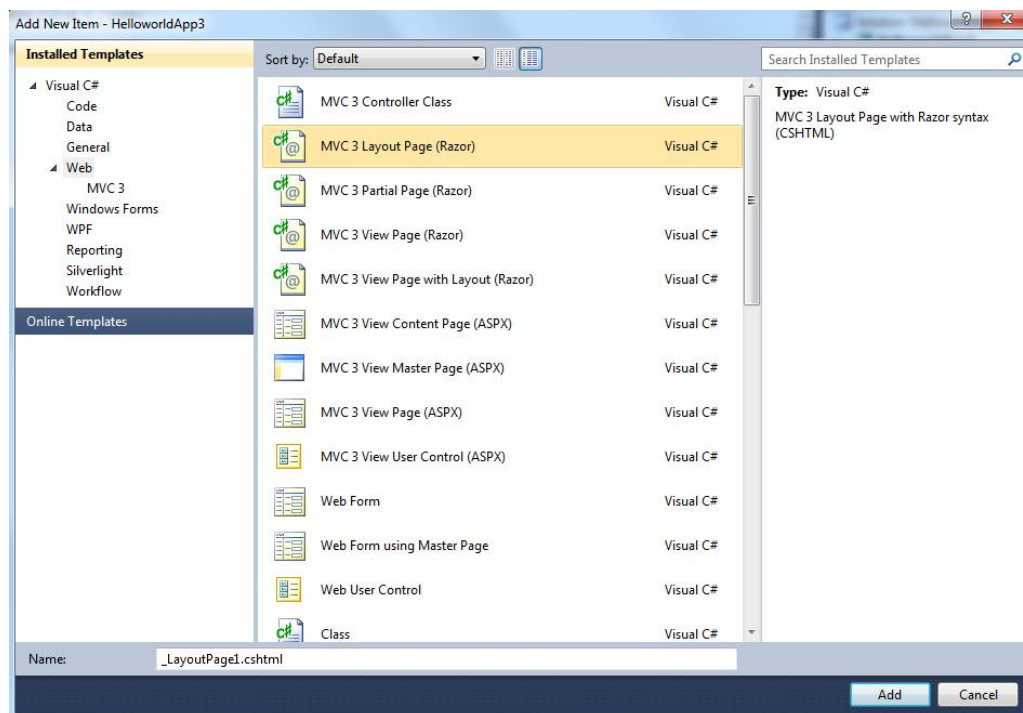
이번 장에서는 MVC 프레임워크의 기본적인 구조를 살펴보았다. 아직 실제 실무에서 사용하기에는 많은 내용이 소개가 되지 않았지만, 어떻게 페이지가 구성되는지에 설명은 되었다.

[ASP.NET MVC3 강좌] 4. Layout , Partial Page

여기에서는 실무에 적용하기 위한 필수적인 몇 가지 부가기능에 대해서 알아보도록 하겠다.

1. Layout(MasterPage)

이전 ASP.NET 2.0 에 Master Page 라는 기능이 추가가 되었다. 이 기능은 중복되는 html 페이지의 헤더와 푸터와 같이 광범위하게 사용되는 부분을 공통적인 master Page 라는 하나의 aspx 페이지로 분리한 후, 각각의 콘텐츠 페이지가 그 마스터페이지의 골격을 공유해서 사용하는 기능으로 이 기능이 탑재된 후 ASP.NET 에서는 많은 코드량을 줄일 수 있었으며, 곧 ASP.NET 에서 효율적으로 개발을 하기 위한 하나의 공통적인 개발 방법론이 되었다. 이러한 편리한 기능인 마스터 페이지를 MVC 에서 사용할 수 없다는 건 납득하기 어려울 수도 있을 것 같다. 그래서 MVC 는 이전 ASP.NET 보다도 더 쉬운 방법으로 마스터 페이지를 지원합니다. MVC 에서는 또한 ViewBag 을 마스터페이지와 공유하는 것이 가능함으로써, 개발에 있어서 더 큰 유연성을 제공한다.



[그림 : 마스터페이지 레이아웃 페이지]

1) 보통 마스터페이지는 View 폴더 아래의 Shared 폴더에 추가하게 된다. 이곳에 마스터페이지를 추가한다.

2) 이 마스터페이지를 다음과 같이 수정한다.

```
<!DOCTYPE html>
<html>
<head>
  <title>@ViewBag.Title</title>
</head>
<body>
  <div>Header – 마스터페이지 테스트</div>
  <div>
    @RenderBody()
  </div>
  <div>Header – 마스터페이지 테스트 </div>
</body>
</html>
```

이곳에서 @RenderBody() 부분이 실제로 콘텐츠로 치환될 영역이다. 뷰 페이지를 생성할 때 마스터 페이지를 생성하게 되면 저 부분에 실제 콘텐츠가 채워지게 된다.

3) 3 번째 글에서 작성했던 Home/Index.cshtml 페이지를 다음과 같이 수정한다.

```
@{
  Layout = "~/Views/Shared/_LayoutMaster.cshtml";
  ViewBag.Title = "Index";
}
<h2>@ViewBag.HelloWorld</h2>
```


4) 이제 실행결과를 확인 한다.



붉은색으로 표시된 부분은 다른 페이지에서도 공유해서 쓸 수 있는 마스터페이지 콘텐츠 내용이다.

2. Partial

파셜 컨트롤은 이전 클래식 ASP.NET 에서 UserControl 과 같은 기능을 한다. 이것은 View 페이지와 Master 페이지에 같이 사용될 수 있다. ViewBag 을 공유하지만 이쉽게도 Model 은 공유하지 못한다. 하지만 각각의 파셜뷰는 각각의 독립적인 Model 을 소유할수 있으며, 이는 View 페이지에서 파셜 컨트롤을 렌더링할 때 넘겨줄 수 있다.

1) Shared 폴더에 Add->NewItem 으로 파셜 페이지를 두 개 생성 한다. 각각의 이름은 Header 과 Footer 로 정의한다

2) 각각의 코드에 다음과 같은 코드를 삽입한다.

```
<div>Header - 파셜페이지 테스트</div>  
[Header.cshtml]
```

```
<div>Footer - 파셜페이지테스트</div>  
[Footer.cshtml]
```

3) LayoutMaster.cshtml 의 코드를 다음과 같이 수정한다.

```
<!DOCTYPE html>

<html>
<head>
  <title>@ViewBag.Title</title>
</head>
  <body>
    @Html.Partial("Header")
    <div>
      @RenderBody()
    </div>
    @Html.Partial("Footer")
  </body>
</html>
```

이제 실행시켜서 결과를 확인 한다.



결과값이 아까와 동일 하다.

Summary

파살 컨트롤과 마스터페이지는 약간 복잡한 성격의 페이지에서 프로그램의 복잡도를 떨어뜨릴 수 있는 아주 강력한 기본기능 중에 하나이며 이는 MVC 뿐 아닌 ASP.NET 프로그래밍의 기본이 되는 요소로써, 필수적으로 습득해야 할 기본지식이다.

[ASP.NET MVC3 강좌] 5. Model 을 View 에서 표현하기

이제 실제 Model 을 View 로 표현해보도록 하겠다. Model 은 Controller 를 통해서 가공되고 최종적으로 형성된 Model 이 View 를 통해서 보여지게 되는데, 여기서 Controller 가 Model 을 View 로 보내는 방법은 2 가지가 존재한다. 두 가지 방법은 다음과 같다.

- ViewBag 객체를 이용
- StrongView 를 이용.

개발자는 두 가지 방법을 전부 이용하여 데이터를 보내는 방법을 익혀야 한다. 각각의 방법은 서로간의 장 단점을 가지고 있으며, 이를 적절히 이용해야 더 유연하게 MVC 를 이용할 수 있을 것이다.

두 가지 방법의 장 단점은 다음과 같다.

종류	기능설명
ViewBag(ViewData)	데이터에 대한 추가적인 생성없이 Model 을 전송할수 있다. ViewBag 이전버전인 ViewData 의 경우 무조건 object형태로 전송되었기 때문에 변환이 필요하였으나 , 이번 4.0 에 추가된 Dynamic Type인 ViewBag 은 그런 변환이 필요없이 사용이 가능하다. 그러나 Dynamic Type은 Runtime 시에만 형값의 추론이 가능하므로 실제 개발시에 올바른 코드작성을 도와주는 인텔리센스가 동작하지 않는다.
StrongView	Controller 에서 Model 을 전송하는 보편적인 방법이다 , Model 객체에 이 값이 담기게 되며 , 모든 값을 추론가능하기 때문에 인텔리센스에서도 사용이 가능하다. 다양한 Model 을 포함하지 못하는 단점이 존재한다.

그러면 모델을 View 로 보내는 두 가지 방법을 전부 살펴보도록 하겠다. 일단 먼저 전송할 Model 을 작성한다.

- 1) Models 폴더에 ClassLibrary 파일을 하나 만들고 이름을 HelloModel 로 작성한다
- 2) 아래와 같은 프로퍼티를 생성한다

```
namespace HelloworldApp3.Models
{
    public class HelloModel
    {
        public String TodayDate { get; set; }
        public String HelloMsg { get; set; }
    }
}
```

이제 해당 모델을 ViewBag 와 StrongView 를 이용해서 연결해 보도록 하자.

1. ViewBag

ViewBag 의 가장 큰 특징은 간편하게 데이터를 넘겨줄 수 있다는 것이다. 또한 ViewBag 은 MasterPage 와 PartialPage 에서 공유할 수 있다는 장점도 있다. 그런 반면 스캐폴딩 같은 코드 자동완성을 사용하지 못하는 것은 큰 취약점으로 나타낼 수 있다.

그러면 이제 ViewBag 을 이용해서 Model 을 View 로 연결해 보도록 하겠다.

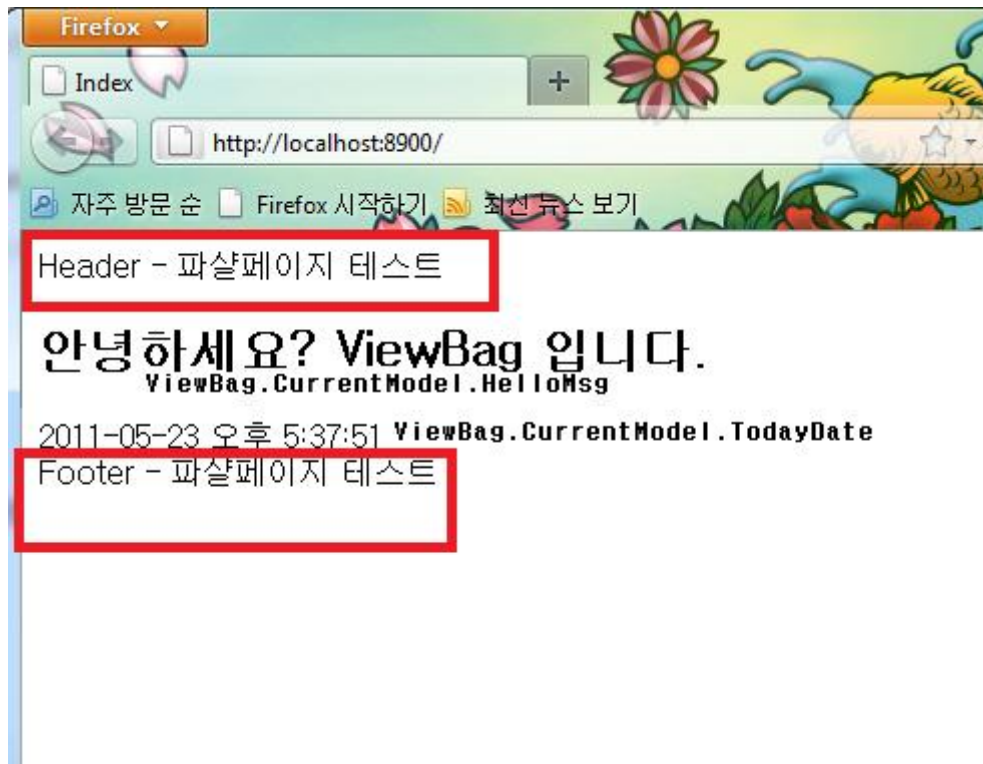
- 1) 먼저 HomeController 의 Index Action 을 다음과 같이 수정한다.

```
public ActionResult Index()  
{  
    ViewBag.CurrentModel = new HelloWorldApp3.Models.HelloModel {  
        TodayDate = DateTime.Now.ToString(),  
        HelloMsg = "안녕하세요? ViewBag 을 통한 Model 전송입니다."  
    };  
    return View();  
}
```

- 2) Index.cshtml 은 다음과 같이 수정된다. 인텔리센스가 지원되지 않으므로, 철자에 유의 하도록 하자. 해당 코드는 ViewBag.CurrentModel 모델이 있다고 가정한 후, 그곳에 해당 Model 을 정의해서 작성한 것이다. 이는 컴파일 타임에 오류를 검출하는 대신 런타임 시에 타입을 검사해서 유연하게 컴파일링이 되도록 한다 (이는 반대로 런타임 시까지 오류를 검출하기 힘들다는 문제점을 낳기도 한다.)

```
@{  
    Layout = "~/Views/Shared/_LayoutMaster.cshtml";  
    ViewBag.Title = "Index";  
}  
<h2>@ViewBag.CurrentModel.HelloMsg</h2>  
@ViewBag.CurrentModel.TodayDate
```

3) 실행결과를 확인한다.



[그림 : 실행화면]

위와 같이 적용이 된 것을 볼 수 있을 것이다. 위의 페이지 결과와 같이 ViewBag.CurrentModel 은 View 페이지에 정상적으로 데이터를 넘겼으며, 그 결과가 잘 출력되고 있다. 이제 다음으로 StrongView 에 대해 살펴해보도록 하자.

2. StrongView

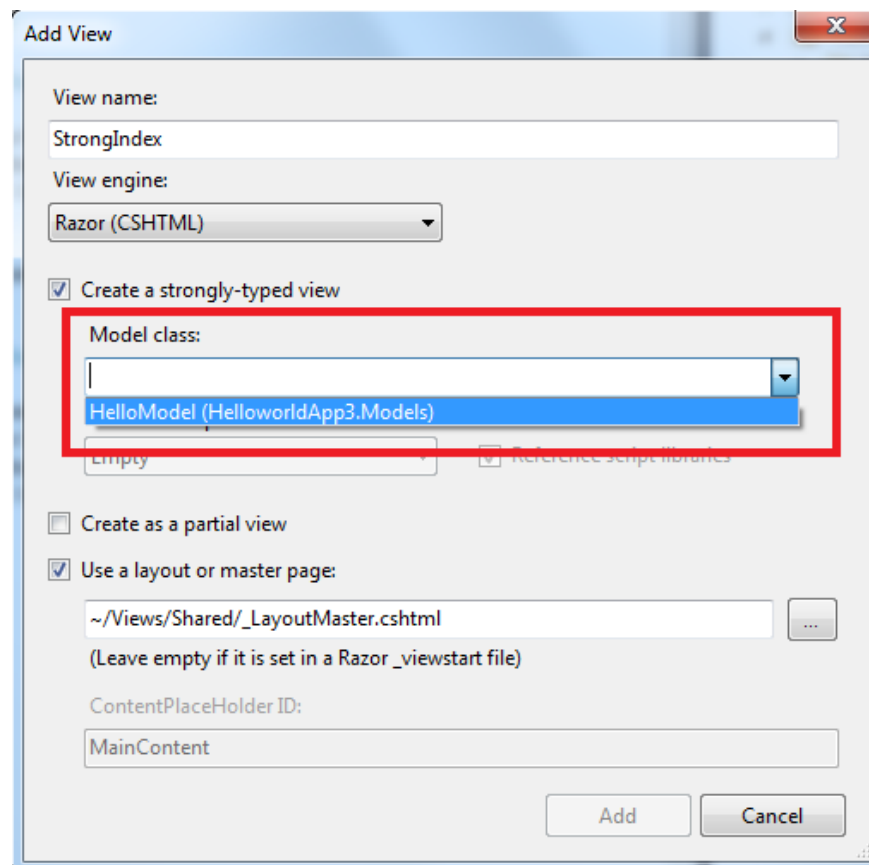
강력한 뷰는 가공된 Model 을 뷰에 바로 투영시키기 위해 고안되었다. 단순히 View 를 이용해서 Model 에 바인딩하는 것은 ViewBag 과 큰 차이가 없어 보인다. StrongView 의 가장 큰 장점은 컴파일시에 타입을 체크할 수 있다는 것과, 온전한 범위의 스캐폴딩을 지원한다는 것이다. 또한 구현방식 또한 실제 View 에 Data 를 주입시킨다는 느낌으로 구현함으로써 ASP.NET 의 가장 큰 장점이었던 웹페이지를 클라이언트처럼 개발한다는 철학에 의거한다고 할 수 있다.

- 1) 먼저 StrongView 를 구현하기 위해 StrongIndex 라는 Action 을 만든다. 그 후 아래와 같이 구현한다.

```
public ActionResult StrongIndex()  
{  
    return View(  
        new HelloworldApp3.Models.HelloModel {  
            TodayDate = DateTime.Now.ToString(),  
            HelloMsg = "안녕하세요? StrongView 를 통한 Model 전송입니다."  
        });  
}
```

해당 코드는 뷰에 완성된 Model 을 넘겨주는 것으로 코드를 마무리 하고 있다. 정말 이 작업이 끝이다.

- 2) 이제 3 장에서 했던 것처럼 AddView 를 이용해서 새로운 뷰를 추가한다. 이번에는 StrongView 를 작성할 것이기 때문에, 아래의 그림과 같이 View 에서 사용할 모델을 지정한다.



[그림 : Model 연결]

- 3) 이제 StrongIndex 뷰를 작성해보자. View() 에 들어간 값은 Model 객체에 있으며, 이를 이용해서 전송된 Model 에 편리하게 접근할 수 있다.

```
@model HelloworldApp3.Models.HelloModel
```

```
@{  
    ViewBag.Title = "StrongIndex";  
    Layout = "~/Views/Shared/_LayoutMaster.cshtml";  
}  
<h2>@Model.HelloMsg</h2>  
@Model.TodayDate
```

- 4) 이제 실행 결과를 확인해보자.

결과값은 단순하지만, 훨씬 간단하게 코딩할 수 있다.

3. Bonus! StrongPartial

클래식 ASP.NET 을 어느 정도 다룰 수 있는 개발자의 경우는 WebUserControl 을 핸들링하기 위해 다양한 변수를 삽입하거나, WebUserControl 의 behind 코드에 데이터 연결을 위한 코드를 작성하곤 했다. MVC 에서는 이것이 불가능하지만, 그대신 View 가 받은 데이터를 Partial 로 보내는 것은 가능하다. 이제 Strong Partial Page 를 작성해보도록 하겠다.

- 1) 먼저 Footer 페이지를 아래와 같이 수정한다.

```
@model HelloworldApp3.Models.HelloModel  
<div>Footer - 파살페이지 테스트</div>  
@Model.TodayDate
```

해당 코드에서는 첫 줄에 사용할 model 을 정의하였다. 한번에 여러 개의 model 을 정의할 수 없는 부분이 약간 아쉬운 부분이다. 세 번째 줄에서는 첫 페이지에서 정의한 모델을 가져와서 사용하는 모습을 보여준다. 이는 View 페이지에서 사용했던 코딩과 완전히 동일한 방식이다.

- 2) StrongIndex 의 Partial 을 호출하는 코드를 삽입하고 두 번째 인자로 Model 을 넘겨준다.

전체 코드는 다음과 같다

```

@model HelloWorldApp3.Models.HelloModel

@{
    ViewBag.Title = "StrongIndex";
    Layout = "~/Views/Shared/_LayoutMaster.cshtml";
}
<h2>@Model.HelloMsg</h2>
@Model.TodayDate
@Html.Partial("Footer",Model)

```

@Html.Partial()은 두 가지 인자를 받도록 오버로딩 되어 있는데 첫 번째 인자는 사용할 partialpage 의 페이지 이름, 두 번째 인자는 넘겨 받을 Model 이다.

이러한 방식으로 PartialPage 에 모델을 전송할 수 있다.



[그림 : 파샬 페이지에 데이터를 정상적으로 전달한 모습]

Summary

ViewBag 을 이용한 방법과 StrongView 모두 MVC 에서 데이터를 View 로 전송하는 가장 간단하면서도 중요한 방법이다. 다음 장에서는 데이터베이스 연결과 함께 스캐폴딩에 대해서 알아보도록 하겠다.

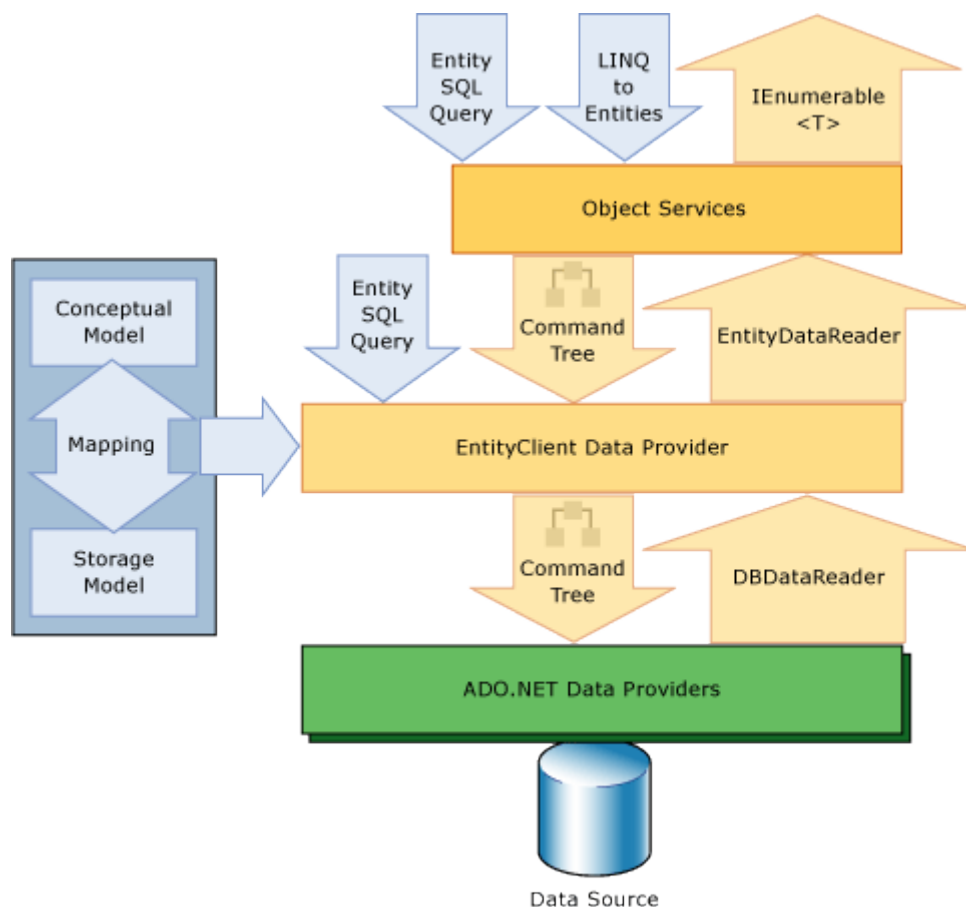
[ASP.NET MVC3 강좌] 6. MVC의 매력적인 기능 스캐폴딩!(With EF)

우리는 지금까지 기본적인 MVC의 구조에 대해 살펴보았다. 이번 시간에는 실제 데이터베이스와의 연동을 살펴보도록 하겠다. 이로써 MVC로 좀더 실무에 근접한 어플리케이션을 만들 수 있을 것이다.

Entity Framework

엔티티 프레임워크는 Microsoft에서 개발한 ORM 모델중의 하나로써, 기존의 ADO.NET보다 좀더 구조적이고 객체적으로 접근하기 위해서 고안되었다. 이 엔티티 프레임워크는 또한 MS에서 개발된 기술이기 때문에, MVC와도 좋은 호환성을 보여준다.

이번 장에서는 EF와 MVC의 연동만을 다루려고 한다.



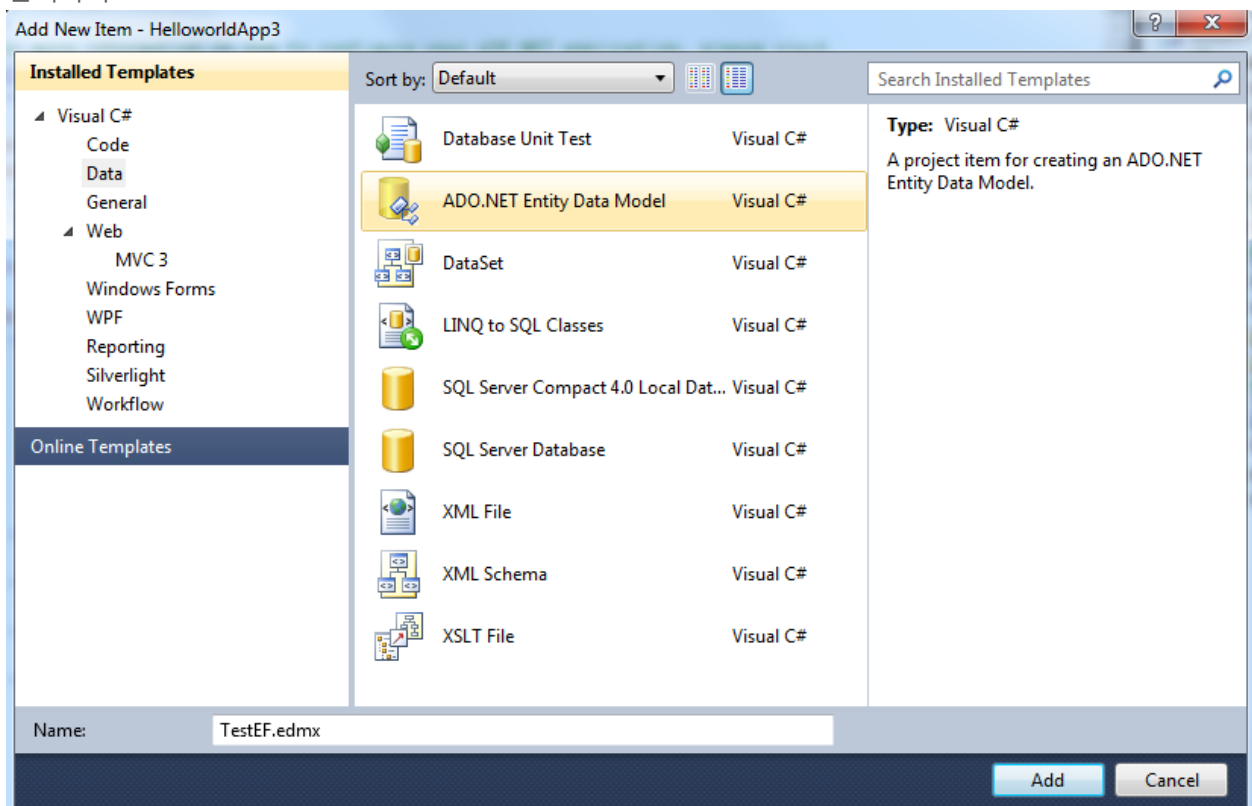
[그림 : Entity Framework의 기본적인 구조]

<MVC 스캐폴딩>

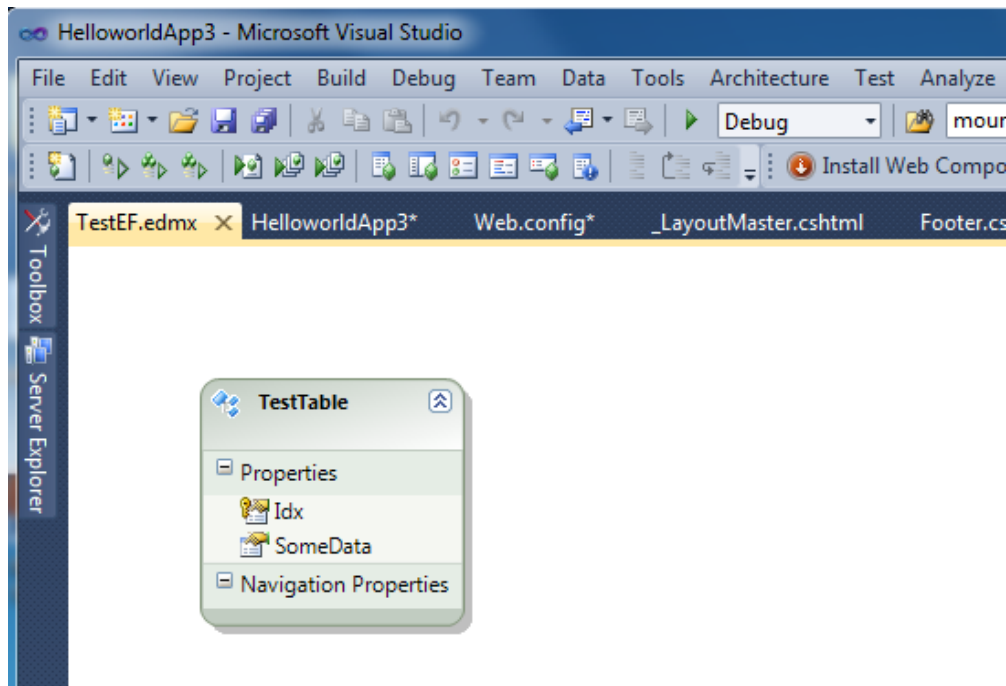
본디 스캐폴딩이란 기술은 Ruby on Rails 에서 처음 선보였던 기술이다. 이는 Model 을 투영해서 CRUD(Create Read Update Delete)를 한번에 코드로 뽑아내주는 것으로써 Ruby on Rails 가 가장 인기를 얻게 된 원동력이기도 하다. MVC 도 물론 이런 스캐폴딩을 ASP.NET MVC 전반에 걸쳐 폭 넓게 지원하고 있다. 특히 EF 와의 연동으로 인해 이는 더 큰 효과를 보여주고 있다.

<MVC 와 EF 를 연동한 스캐폴딩 구현>

1. 먼저 기존프로젝트의 Model 에서 Data 탭을 선택하고 템플릿 중에 ADO.NET Entity Data Model 을 선택하자.

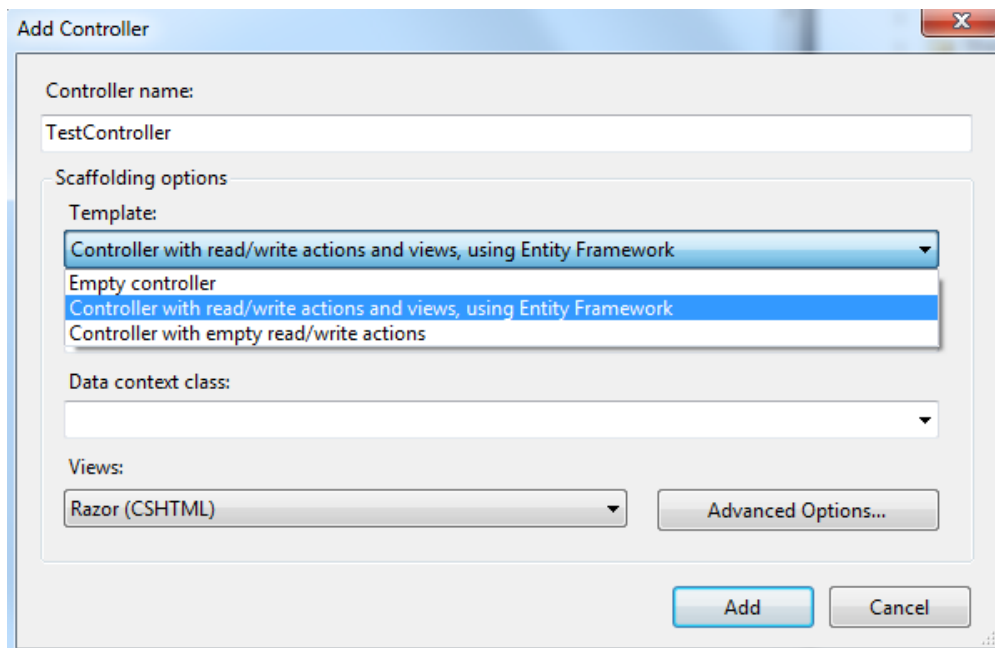


2. 엔티티 프레임워크를 설치하게 되면 처음에 자신이 참조해야 할 데이터베이스를 찾게 된다. 이때 미리 설정해놓은 테이블을 체크하고 생성하게 되면 정상적으로 EF 가 생성되게 된다.

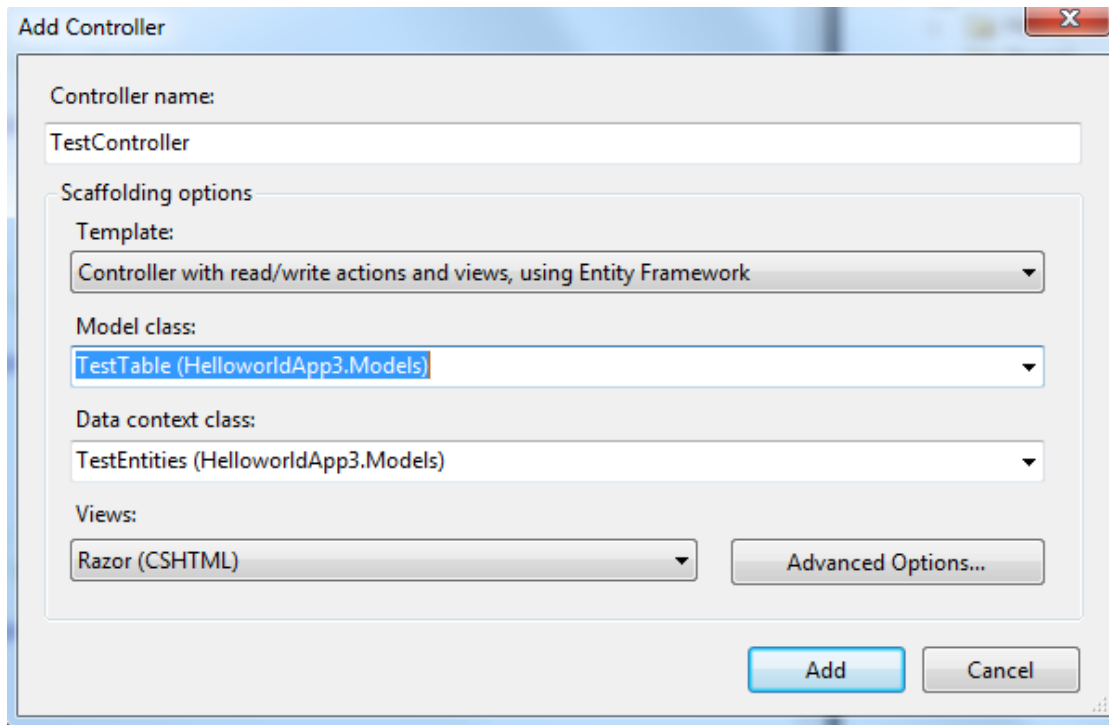


[그림 : 생성된 Entity Framework]

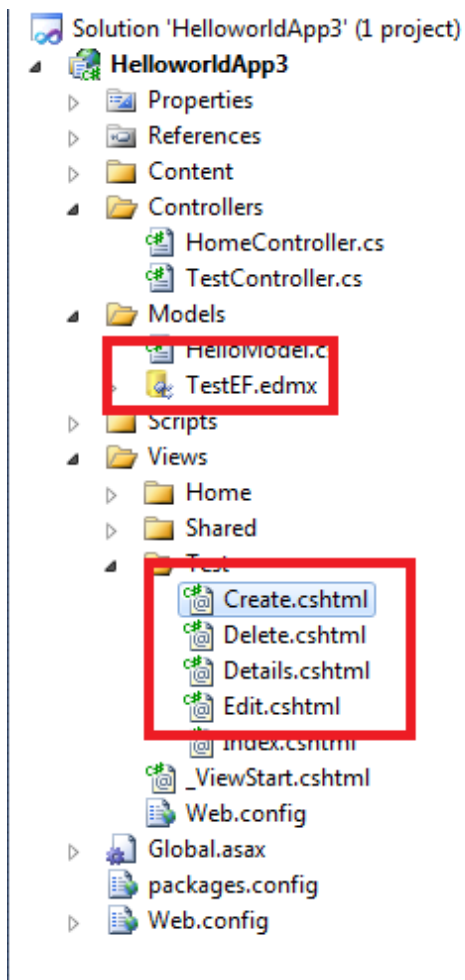
3. 이제 컨트롤러를 만들 시간이다. 기존에 만들었던 컨트롤러와 달리 EntityFramework 을 추가하고 나면 다음과 같이 컨트롤러를 만드는 창이 바뀐다.



4. 여기서 아래그림과 같이 세팅한다.



Template 는 어떤 스캐폴딩을 사용할 것인지를 정하는 건데 필자가 선택한 건 모든 상황에 대한(CRUD) 뷰를 만드는 것이다. ModelClass 는 실제 모델링에 사용되었던 테이블명을 말한다. 마지막으로 DataContext 는 Entity Framework 의 이름을 지칭한다.



CRUD 모든 로직이 완성된 코드 전체가 자동생성 된 것을 확인할 수 있다.

Summary

Entity Framework 의 스캐폴딩은 데이터베이스에 의거한 당신의 로직을 가장 쉬운 방법으로 적용하는 방법을 보여준다. 이것은 매우 쉬울 뿐 아니라 다른 어떠한 추가적인 클래스도 필요 없다. MVC 는 이 개발 방법론을 가장 효율적으로 적용하기 좋은 대표적인 프로젝트 템플릿이 될 것임엔 두말할 여지가 없다. 다음 장에서는 데이터를 연동할 때 가장 필수적인 유효성 검사에 대해 알아보도록 하겠습니다.

[ASP.NET MVC3 강좌] 7. 유효성 검사

유효성검사는 데이터베이스에 올바른 데이터를 적제하기 위한 필수적인 기능 중에 하나이다. 요즈음에는 이런 유효성검사를 여러 가지 방법으로 구현하고 있는데, 기본적으로 MVC 에도 이러한 유효성 검사 기능이 폭넓게 구현되어 있다. 이러한 유효성 검사는 그 형태에 따라 몇 가지 형태로 구분 되어진다.

1. 입력값에 들어간 값에 대한 검증.

2. Model 에 대한 유효성 검증.

입력값에 들어간 값에 대한 검증은, 해당 컨트롤에 대해서 수동으로 검사하고 에러메시지를 삽입하는 것 뜻한다. 그러나 이러한 수동적인 작업 없이, Model 자체에 대해 유효성 검사를 실시할 수도 있다. 입력값에 들어간 값에 대한 검증을 하고, 에러메시지를 표시하는 기능 자체는 개발자의 기호와 디자인에 따라서 사용할 수 없을 수도 있으나, MVC 자체의 모델을 검증할 수 있기 때문에 실무에서도 유용하게 쓰이는 편이다. 이번 장에서는 두 가지 유효성 검사 방법을 살펴보도록 하겠다.

Note : 해당 기능을 정상적으로 활용하기 위해서는 layout 페이지에 jquery 를 추가해줘야 한다.

```
<!DOCTYPE html>

<html>
<head>
    <title>@ViewBag.Title</title>
    <script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")" type="text/javascript"></script>
</head>
<body>
    @Html.Partial("Header")
    <div>
        @RenderBody()
    </div>

</body>
</html>
```

[코드 : layoutmaster.cshtml]

이전에 작업했던 소스에서 Create.cshtml 소스를 살펴보도록 하겠다

```
@model HelloWorldApp3.Models.TestTable

@{
    ViewBag.Title = "Create";
    Layout = "~/Views/Shared/_LayoutMaster.cshtml";
}

<h2>Create</h2>

<script src="@Url.Content("~/Scripts/jquery.validate.min.js")" type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")" type="text/javascript"></script>

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>TestTable</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.SomeData)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.SomeData)
            @Html.ValidationMessageFor(model => model.SomeData)
        </div>

        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```

이곳에 이미 기본적인 유효성검사 로직이 삽입되어 있는 것을 알 수 있다. 이를 먼저 살펴보자.

@Html.ValidationSummary(true)

에러메시지를 종합적으로 보여주는 메서드이다.

@Html.ValidationMessageFor(model => model.SomeData)

해당 모델에 해당되는 메시지이다.

이로써 기본적인 validation 체크가 가능해진다. DateTime 같은 경우나, 해당 필드에 빈 값이 들어갈 수 없는 경우 MVC의 Validation은 자동으로 이를 감지해준다

Create

TestTable

SomeData

RegDate

The RegDate field is required.

Create

[Back to List](#)

[그림 : 등록일자에 빈 값이 들어가는 경우, 빈 값을 넣을 수 없기 때문에 validation 메시지가 나타난다.]

Create

TestTable

SomeData

RegDate

The value '2011-02-30' is not valid for RegDate.

Create

[Back to List](#)

[그림 : 위와 같이 날짜 포맷이 정상적이지 않을 때도(2 월에 30 일은 없음) 정상적으로 오류를 감지해준다]

이 외에도 자신이 설정한 메시지를 validation 메시지에 내보내고 싶은 분도 있을 것이다.

이런 경우 간단한 코딩으로 이 기능을 구현할 수 있다.

Create 페이지를 다음과 같이 수정한다.

```
[HttpPost]
public ActionResult Create(TestTable testtable)
{
    if (testtable.SomeData == "LoveCiel")
    {
        ModelState.AddModelError("SomeData", "LoveCiel 테스트용 에러메시지");
    }
    if (ModelState.IsValid)
    {
        db.TestTable.AddObject(testtable);
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(testtable);
}
```

ModelState 는 현재 View 에서 전송 받은 Model 의 상태를 점검하는 객체이다. 이곳에 ModelError 객체가 포함되어 있으면 ModelState.IsValid 가 False 를 반환하게 되고, 이를 에러로 감지해 낼 수 있게 된다.

Create



TestTable

SomeData
LoveCiel LoveCiel 테스트용 에러메시지

RegDate
2010/01/01

Create

[그림 3 : 사용자 정의 ErrorMessage]

Summary

다음 장에서는 실질적인 Validation 을 사용하기 위해 ModelBinder 에 대해 알아보도록 하겠다. 이 두 장을 습득함으로써 MVC 의 유효성 검사에 관한 기능은 전부 익힐 수 있을 것이다.

[ASP.NET MVC3 강좌] 8. ModelBinder

모델바인더는 MVC가 내세우는 강력한 기능 중에 하나로써, 객체의 유효성을 직접 그 객체에 정의하는 방법이다. 이것을 이용하여 사용자는 유효성검사를 좀더 쉽고 직관적이며, 적은 코딩으로 이루어낼 수 있다.

먼저 ModelBinder를 사용하기 위해 기존의 코드를 수정하자

```
public class HelloModel
{
    public String TodayDate { get; set; }
    [Required(ErrorMessage = "안녕이라고 말해주세요.")]
    [RegularExpression("^안녕")]
    public String HelloMsg { get; set; }
}
```

[코드 : Models/HelloModel]

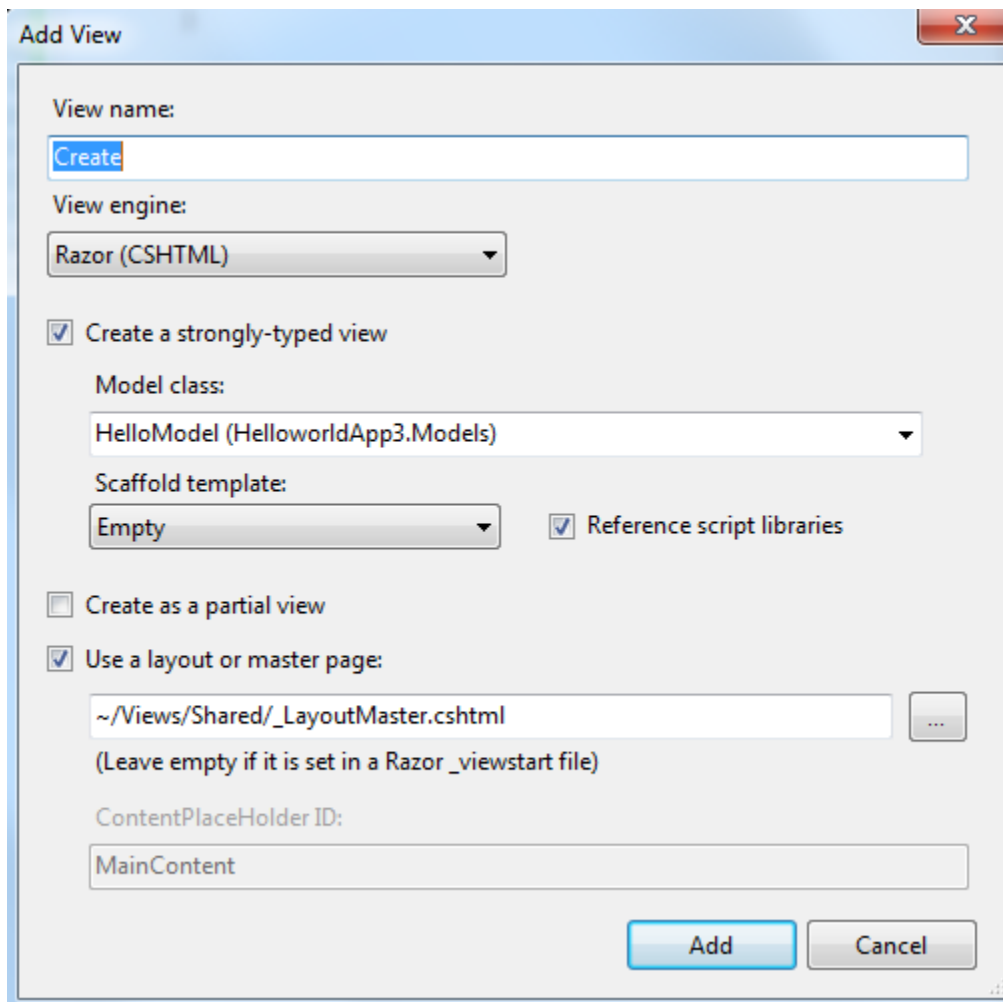
위의 코드에서 우리는 단순한 인자 값에 Required와 RegularExpression을 포함하였다. 이는 해당 모델에 항상 값이 있어야 하며, 정규식을 통과해야 함을 의미한다.

이제 해당 모델을 구현하는 뷰를 만들어보자. 먼저 HomeController를 생성하고 Create 액션을 코딩한다.

```
public ActionResult Create()
{
    return View();
}
```

[코드 : Create Action. 기본코딩으로 아무것도 손을 대지 않았다.]

해당 코드에서 우클릭을 하면 새로 뷰를 추가할 수 있다. AddView 를 클릭한 후 다음그림과 같이 입력한다.



View name:
Create

View engine:
Razor (CSHTML)

☒ Create a strongly-typed view

Model class:
HelloModel (HelloworldApp3.Models)

Scaffold template:
Empty

☒ Reference script libraries

☐ Create as a partial view

☒ Use a layout or master page:
~/Views/Shared/_LayoutMaster.cshtml
(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID:
MainContent

Add Cancel

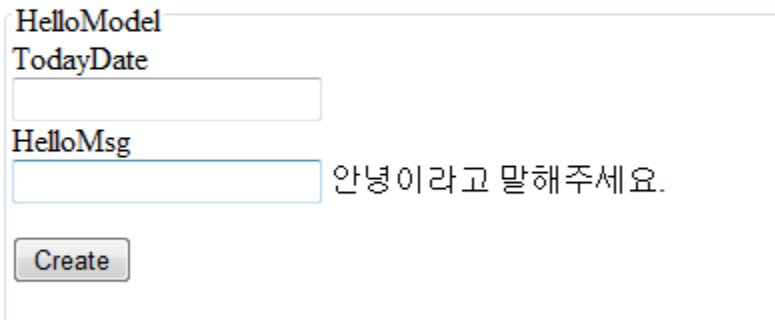
[그림 : 설정]

해당 화면에서 우리는 Create 페이지를 생성한다는 것, 그리고 해당 Model 에 바인딩되는 StrongView 를 생성한다는 것, 그리고 그 모델로써 HelloModel(ModelBinder 가 설정된)을 사용한다는 것을 알 수 있다. 이로써 우리는 ModelBinder 를 사용한 예제를 완성하였다. 너무 간단하지 않은가? 그렇지만 기능은 전부 구현되었다!

<http://호스팅된페이지/Hello/Create> 페이지로 이동해서 결과를 확인해보도록 하자.

Header - 파살페이지 테스트

Create



[Back to List](#)

[그림 : HelloMsg 가 미 입력된 화면 , 해당 메시지는 항상 입력되어야 한다]

이제 HelloMsg 에 입력값을 넣어보면 다음과 같이 변하는 것을 알 수 있다.

Header - 파살페이지 테스트

Create



[Back to List](#)

[그림 : 안녕이라는 메시지가 포함되지 않은 경우 포함되는 에러메시지]

Summary

이로써 MVC 의 유효성 검증에 대한 모든 것을 살펴보았다. 허무하리만큼 구현하긴 간단하지만 잘만 사용하면, 강력한 기능을 구현하는 것이 이 ModelBinder 라고 할 수 있다. 다음 아티클에서는 실제 MVC 에서 사용하는 반환값들의 모든 종류에 대해서 알아보도록 하겠다.

[ASP.NET MVC3 강좌] 9. MVC ActionResult 종류 살펴보기

MVC 의 ActionResult 는 MVC 에서 보내질 View 의 형태를 결정하는 것으로써 이것을 정복하는 것 만으로도 MVC 를 어느 정도 능숙하게 다룰 수 있을 것이라고 기대할 수 있다. 이번 장에서는 MVC 에서 주요하게 사용되는 ActionResult 를 상속받는 구현 객체들에 대한 이야기를 풀어보도록 하겠다.

1. ActionResult 의 원형

```
public abstract class ActionResult {  
    public abstract void ExecuteResult(ControllerContext context);  
}
```

[코드 : ActionResult 의 원형]

위에서 살펴볼 수 있듯이 이것은 단지 ExecuteResult 를 호출하는 아주 간단한 추상 클래스이다. 이 메서드는 해당 컨트롤러 컨텍스트를 전달받는데, 이는 사 모든 http 서버 측 객체에 접근할 수 있다는 것을 의미하며, 또한 해당 컨트롤러를 참조할 수 있다는 것을 의미한다.

다음과 같은 객체들이 ActionResult 를 상속받아 구현된다.

System.Web.Mvc.ContentResult

ASP.NET 의 Response.Write 와 같은 기능을 한다. Content(); 에 전달되는 인자를 그대로 출력한다.

System.Web.Mvc.EmptyResult

메서드명과 같이 아무것도 반환하지 않는다.

System.Web.Mvc.FileResult

Stream 형태로 해당 뷰를 반환한다.

System.Web.Mvc.HttpUnauthorizedResult

권한이 없는 http 오류코드(401)를 리턴한다

System.Web.Mvc.HttpStatusCodeResult

각종 http 상태코드를 반환한다

System.Web.Mvc.JavaScriptResult

자바스크립트 형태의 Stream 을 리턴한다.

System.Web.Mvc.JsonResult

Json 형태의 Stream 을 리턴한다. ContextType 은 (text/json)이다.

System.Web.Mvc.RedirectResult

해당 페이지로 리다이렉트 한다 ASP.NET Classic 의 Response.Redirect 와 동일하다.

System.Web.Mvc.RedirectToRouteResult

해당 MVC 안의 라우팅 되는 페이지로 리다이렉트 한다.

System.Web.Mvc.ViewResultBase

해당 뷰를 렌더링한다. 때에 따라 View 로 Model 을 보내거나, View 로 사용할 페이지를 지정할 수 있다.

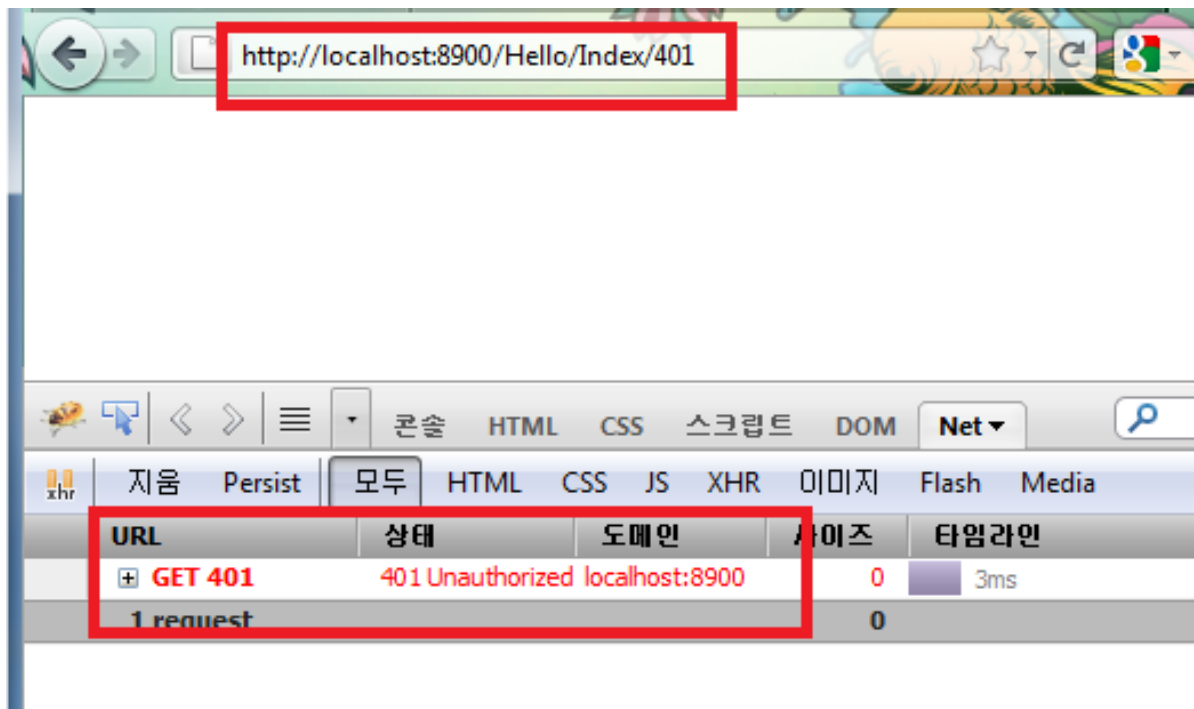
이렇게 여러 가지 ActionResult 가 있지만 실질적으로 고유한 Result 는 ContentResult 와 ViewResult, RedirectResult, JsonResult, HttpStatusCodeResult, 그리고 FileResult 정도이다. 다른 리턴값은 단지 아래의 클래스를 래핑해서 사용자가 좀더 이용하기 쉽게 만들었을 뿐이다.

예를 들어 HttpStatusCodeResult 와 HttpUnauthorizedResult 를 살펴보도록 하자.

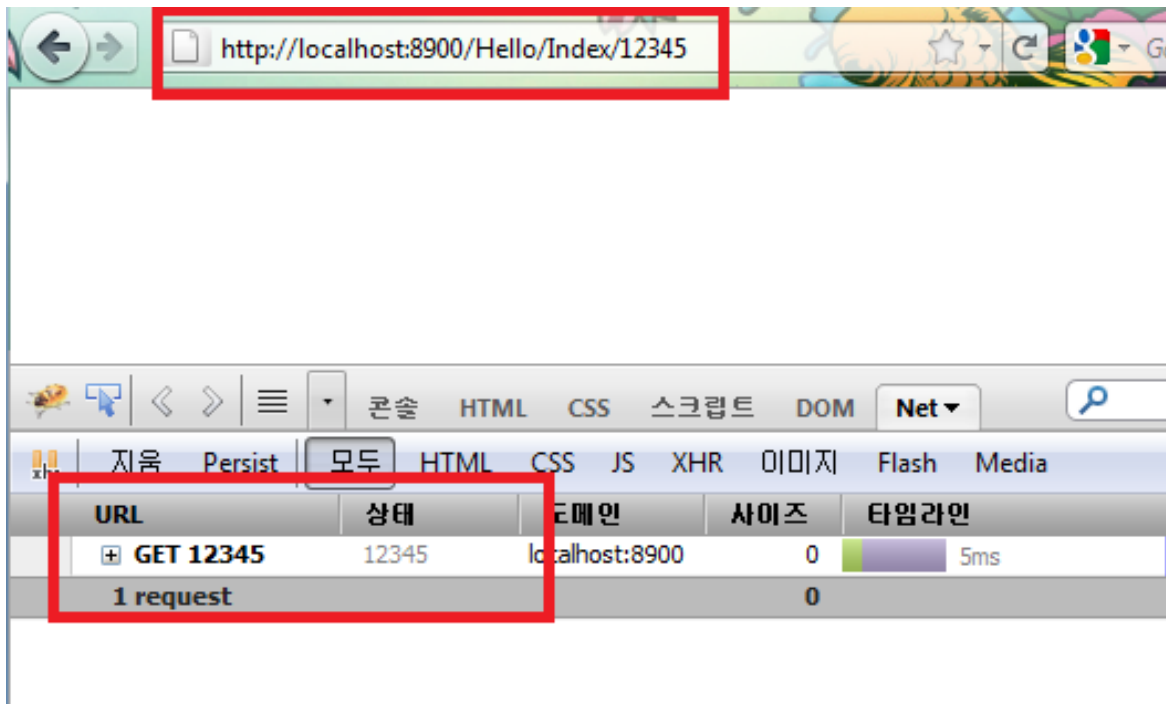
```
public ActionResult Index(Int32 id )
{
    if (id == 401)
    {
        return new HttpUnauthorizedResult();
    }
    return new HttpStatusCodeResult(id);
}
```

[코드 : UnauthorizedResult]

위의 코드는 각종 상태코드를 그대로 리턴하도록 만들어진 예제이다. 간단하게 파라미터에 int 값을 삽입하여 그것이 결과값으로 나오게 된다.



[그림 HttpUnauthorizedResult 를 호출하였을 때이다. 401 Error 이 발생한다.]



[그림 HttpStatusCode 를 호출하였을 때 이다. 실제 12345 라는 리턴값은 존재하지 않는다.]

Summary

이번 장에서는 MVC 에 대한 ActionResult 의 종류에 대해서 알아보았다. 각각의 ActionResult 는 주요한 웹 기능을 수행하는데 쓰인다. 예를 들면 `HttpUnauthorizedResult` 는 의도적으로 사용자에게 인증을 받을 것을 유도하여 로그인 페이지로 보낸다. `Json` 은 `Stream` 을 `Json` 포맷으로 포팅하여 `Json Stream` 을 호출한 것처럼 보이게 한다. `View` 를 통해 콘텐츠를 보내고 싶다면 간단하게 `ViewResult` 를 사용하면 된다. ActionResult 를 이해하는 것은 MVC 를 이용해 좀더 고급 콘텐츠를 만들기 위한 기본바탕이니 꼭 숙지 하기 바란다.

[ASP.NET MVC3 강좌] 10. MVC 처리 프로세스 & ActionFilter



ASP.NET 과 같이 MVC 도 내부적인 파이프라인이 존재한다. 그런데 MVC 는 ASP.NET 보다는 조금 더 단순하다. MVC 는 위와 같이 오직 4 단계로된 파이프라인이 존재하며, 이 사이에 개발자가 원하는 코드를 삽입할 수 있다. 이 처리과정은 Controller 와 View 의 처리 프로세스가 분리되어 있기 때문인데 이로써 코드를 더 단순화 시킬 수 있고, 사용자가 원하는 처리를 삽입하기도 더 쉬워진다.

OnActionExecuting : 컨트롤러에서 해당 액션이 호출되기 전에 호출되는 이벤트이다. 해당 이벤트에서는 사용자가 브라우저를 통해서 호출하는 명령값을 확인할 수 있고, 원하지 않는 코드가 삽입되었을 때에 올바른 처리를 할 수 있다. 이곳에서 주로 처리가 이루어지는 부분은, 사용자 권한 감시 등이 이루어 질 수 있다.

OnActionExecuted : 해당 이벤트는 Action 메서드가 처리된 이후에 호출되는 이벤트이다. 이 이후에는 View 를 렌더링 하기 전에 해야 할 일들을 처리할 수 있다. Action 이 처리된 후, 결과값에 대한 캐싱값을 만들거나 Action 에서의 Error 핸들링 등을 할 수 있다.

OnResultExecuting : 해당 이벤트는 View 가 렌더링 되면서 발생하는 이벤트이다.

OnResultExecuted : 해당 이벤트는 View 가 렌더링 되고 난 후에 발생하는 이벤트이다. 만약 에러처리나 권한 검사 등을 이곳에서 실행하게 된다고 하더라도, 이미 결과 값이 만들어 졌기 때문에 해당 페이지가 사용자에게 노출될 수 있다.

그런데 이 파이프라인에서 우리가 이벤트를 가져오기는 약간 곤란한 면이 있다. Action 은 Method 이기 때문에 ASP.NET 처럼 이벤트를 하나하나 삽입할 수 없다는 문제가 그것인데, 그렇기 때문에 MVC 에서는 ActionFilter 를 삽입해서 해당 이벤트를 핸들링 할 수 있다. 이번 예제에서는 실제 ActionFilter 를 작성해보면서 실질적인 파이프라인을 살펴보도록 하겠다.

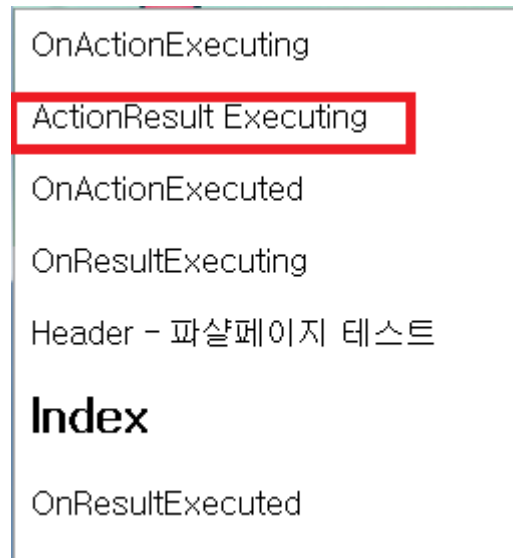
```
public class PipelineFilterAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        filterContext.HttpContext.Response.Write("OnActionExecuting<br/><br/>");
        base.OnActionExecuting(filterContext);
    }
    public override void OnActionExecuted(ActionExecutedContext filterContext)
    {
        filterContext.HttpContext.Response.Write("OnActionExecuted<br/><br/>");
        base.OnActionExecuted(filterContext);
    }
    public override void OnResultExecuting(ResultExecutingContext filterContext)
    {
        filterContext.HttpContext.Response.Write("OnResultExecuting<br/><br/>");
        base.OnResultExecuting(filterContext);
    }
    public override void OnResultExecuted(ResultExecutedContext filterContext)
    {
        filterContext.HttpContext.Response.Write("OnResultExecuted<br/><br/>");
        base.OnResultExecuted(filterContext);
    }
}
```

[코드 ActionResult 에 코드 삽입]

```
[PipelineFilter]
public ActionResult Index()
{
    HttpContext.Response.Write("ActionResult Executing<br/><br/>");
    return View();
}
```

[코드 Index ActionResult 작성]

위의 코드에서는 각각의 처리과정의 끝에 해당 이벤트에 대한 단계를 보여주는 String 을 삽입하였다. 마찬가지로 Index 에서도 같은 코드를 삽입하였는데, 이로써 처리 프로세스에 대한 결과를 볼 수 있을 것이다.



[그림 : 실행화면]

해당 그림과 같이 MVC 파이프라인이 정상적으로 실행된 것을 알 수 있다.

Summary

이번 장에서는 MVC 의 Controller 와 View 의 처리프로세스를 살펴보고 이를 핸들링 할 수 있는 강력한 장치인 ActionFilter 에 대해서 알아보았다. 이 정도까지만 하더라도 MVC 를 다루는 데는 큰 어려움이 없을 것이다. 다음 장에서는 이 ActionFilter 를 좀 더 널리 사용할 수 있는 방법과 다른 메서드를 전역적으로 다루기 위한 Global.asax 에 대해서 알아보도록 하겠다.

[ASP.NET MVC3 강좌] 11. Global.asax

Globalasax 는 asp.net classic 에서 사용했던 기능을 모두 사용할 수 있다. 또한 MVC 만의 추가적인 기능을 사용할 수 있는데, 이번 시간에는 그 기능에 대해 살펴보도록 하겠다.

MVC 의 Global.asax 는 모든 MVC 의 기능들을 등록하는 기능도 포함하고 있다.

이것은 Application_Start() 에서 이루어지게 되는데 이것이 기존 ASP.NET 과 구분되는 가장 큰 차이점일 것이다. 이 외에도 기존 ASP.NET 의 이벤트 들을 자유롭게 등록해서 사용할 수 있다.

백문이 불여일타라고 하니, global asax 에서 가장 빈도 높게 사용하는 이벤트인 Application_BeginRequest 를 등록해 보도록 하겠다.

```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_BeginRequest(object sender, EventArgs e)
    {
        HttpContext.Current.Response.Write("begin request");
    }
    ...
}
```

[코드 : Application_BeginRequest]



[그림 : 코드 실행 결과]

기존 ASP.NET 에서 사용하는 그대로 사용하면 되는데 , 문제는 이런 이벤트 들은 인텔리센스의 도움을 받지 못한다는 것이다. 여기에 대한 해답은 다음 링크를 참조할 수 있다.

[Global.asax 링크](#)

<GlobalFilter>

조금 전의 글에서 Application_Start 부분에 모든 MVC 기능의 초기화가 이루어진다고 하였다. 여기에는 Area 나, RoutingTable 을 등록하는 것도 포함되는데, 이번 MVC3 에서는 GlobalFilter 라는 기능이 새로 소개가 되었다. ActionFilter 를 사용하다 보면, 모든 페이지에 적용해야 하는 ActionFilter 가 있을 수 있다. 주로 캐싱이나, 인증 혹은 로깅 등이 그런 예가 될 수 있는데 이것은 GlobalFilter 라는 기능을 사용해서 모든 Action 에 등록할 수 있다. 이번 장에서는 초간단 예제를 통해 GlobalFilter 를 사용하는 법을 살펴해보도록 하겠다.

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    filters.Add(new HandleErrorAttribute());
}
```

[코드 : GlobalFilter]

이곳에 우리가 지난번 장에서 만든 필터를 삽입해보도록 하겠다.

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    filters.Add(new PipelineFilterAttribute());
}
```

[코드 : PipelineFilter 를 global.asax 에 추가하는 코드]

이것으로 모든 MVC 페이지에 해당 Attribute 가 추가되어 구동됨을 알 수 있다.

OnActionExecuting

OnActionExecuted

OnResultExecuting

Header - 파살페이지 테스트

안녕하세요? ViewBag 을 통한 Model 전 송입니다.

2011-06-13 오후 4:45:14

OnResultExecuted



[그림 : GlobalFilter]

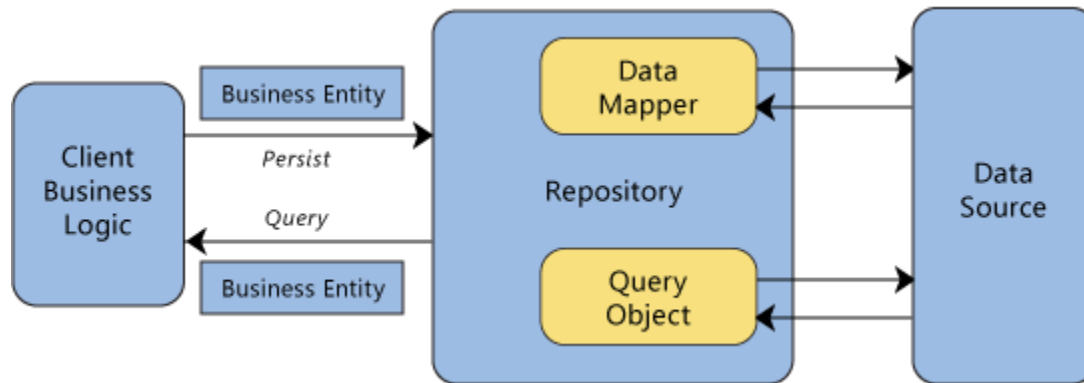
Summary

이 외에도 앞으로 언급할 MEF in MVC 등에서 이 global.asax 이 폭넓게 언급될 것이다. MVC 에서 어떤 추가 기능을 사용한다고 하면, 모두 이곳에 등록된다고 보면 될 것 같다.

다음 장에서는 MVC 의 직접적인 기능과는 관계가 없지만 많은 예제에서 폭넓게 다뤄지는 Repository 패턴에 대해 살펴보도록 하겠다.

[ASP.NET MVC3 강좌] 12. (번외편) Repository Pattern

Repository 는 저장소 라는 용어로 데이터베이스에 직접적으로 접근하는 것이 아닌 이것을 하나의 저장소 개체로 보고 이곳에 질의 하는 패턴이다.



이전 ADO 1.0 이나 그 이하의 세대에서 DB 를 다룰 때 가장 큰 문제점은 DB 데이터 형식의 매핑 문제였을 것이다. 사이즈나, 형태가 다르기 때문에 파생하는 문제는 가장 모호한 에러인데다, 프로그램 자체를 망가뜨리는 가장 핵심적인 원인이기도 하다. 이러한 이유로 DB 의 데이터를 직접 가공해 쓰는 형태보다는 가공하는 매개체를 두고, 비즈니스 레이어에서 이를 가공해 쓰는 형태의 데이터 매퍼가 유행하게 된다. 리파지터리 패턴은 이 즈음에 나온 개념으로써 마틴파울러가 그의 책 PoEAA 에서 처음 제시한 개념이다. Repository 패턴은 밑의 링크를 참조 할 수 있다.

<http://vandbt.tistory.com/27>

<http://msdn.microsoft.com/en-us/library/ff649690.aspx>

[Repository 패턴에 대해]

이러한 Repository 개념은 MVC 의 nertdinner 와 music store 등에서 폭넓게 사용되게 됩니다. 이번 장에서는 간단하게 Repository 패턴을 구현하여 보도록 하겠다.

먼저 MVC 어플리케이션에 Repository 폴더를 만들고 그 안에 HelloRepository.cs 파일을 만든다.

```

public class HelloRepository
{
    private TestEntities db = new TestEntities();
    public List<HelloworldApp3.Models.HelloModel> GetList()
    {
        List<HelloworldApp3.Models.HelloModel> retList = new List<Models.HelloModel>( );
        foreach(var t in db.TestTable)
        {
            retList.Add(new Models.HelloModel{
                HelloMsg = t.SomeData,
                TodayDate = t.RegDate.ToString()
            });
        }
        return retList;
    }

    public HelloworldApp3.Models.HelloModel GetDetail(Int32 idx)
    {
        TestTable testtable = db.TestTable.Single(t => t.Idx == idx);
        HelloworldApp3.Models.HelloModel retModel = new HelloModel( );
        if (testtable != null)
        {
            retModel.HelloMsg = testtable.SomeData;
            retModel.TodayDate = testtable.RegDate.ToString();
        }
        return retModel;
    }

    public Int32 Delete(Int32 idx)
    {
        TestTable testtable = db.TestTable.Single(t => t.Idx == idx);
        db.TestTable.DeleteObject(testtable);
        return db.SaveChanges();
    }

    public Int32 Update(HelloworldApp3.Models.HelloModel model)
    {
        TestTable tb = new TestTable
        {
            Idx = 1,
            RegDate = DateTime.Parse(model.TodayDate),
            SomeData = model.HelloMsg
        };
        db.TestTable.Attach(tb);
        db.ObjectStateManager.ChangeObjectState(tb, EntityState.Modified);

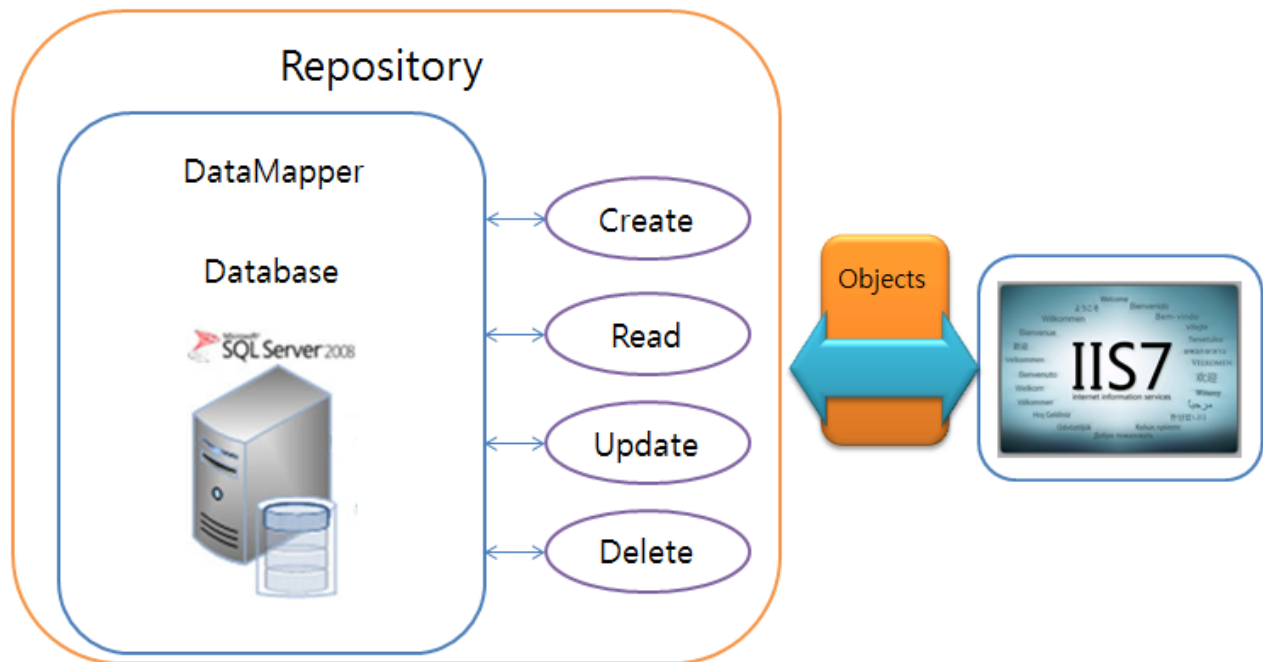
        return db.SaveChanges();
    }

    public Int32 Insert(HelloworldApp3.Models.HelloModel model)
    {
        db.TestTable.AddObject(new TestTable
        {
            Idx = 1,
            RegDate = DateTime.Parse(model.TodayDate),
            SomeData = model.HelloMsg
        });
        return db.SaveChanges();
    }
}

```

[코드 : Repository 패턴을 이용한 EF 코드 작성]

다음의 코드는 다음과 같은 도식도를 가진다



[그림 Repository 도식도]

* 해당 코드 자체가 일종의 DataMapping 이 되어 있는 상태이기 때문에 약간은 불필요한 코드가 되긴 했지만, 이렇게 C# 객체화 하는 것이 Repository 의 기본적인 구도라는 것을 명심하기 바란다.

Summary

요즈음에 있어서 Repository 패턴이 강조되는 이유는 이것이 IoC 와 결합이 되었을 때 많은 다른 종류의 데이터 저장소를 지원하기 때문이다. 실제로 필자는 이전에 DataBase 에서 직접적으로 호출하는 코드를 웹 서비스에서 호출하는 코드로 직접 변환한적이 있다. (그때 처음으로 IT 의 생산성에 대해 고민을 하기 시작했다. OTL 단순 작업코드로 3 일을 고생했었다는...) 그때 만약 IoC 와 Repository 를 알고 있었다면 3 일이나 되는 긴 시간이 소요되진 않았을 것이다. 또한 데이터 형태가 변하더라도 Repository 의 객체를 변경하게 되면 컴파일 타임에 그 오류를 잡는 것이 모두 가능해진다.

다음 장에서는 IoC 를 이용해서 실질적으로 이 Repository 를 활용하는 예제를 살펴보도록 하겠다.

이 모든 기능은 MVC 에 포함되어 있는 확장 기능이다.

[ASP.NET MVC3 강좌] 13. MEF(Managed Extensibility Framework) in MVC

지난 시간에는 Repository Pattern 에 대한 간단한 개요에 대해서 알아보았다. 이것은 MVC 에서 유행처럼 쓰이며 모든 MVC 의 Model 에 대한 개념은 이곳에서 나온다고 할 수 있다. 이번 장에서는 지난 시간에 언급되었던 IoC 를 구현하는 방법에 대해 살펴보도록 한다.

<Dependency Injection>

의존성 주입과 Inversion of Container 라는 개념은 모듈을 직접 참조하지 않고 그 인터페이스만을 참조하여서, 참조하는 모듈과 참조하는 어플리케이션간의 결합도를 줄이는 개발 방법론이다. 이로써 사용자는 좀더 static 하고 서로간의 결합도가 높은 - 단순하게 유지보수 하기 힘든 - 어플리케이션 개발에서 탈피해서 좀 더 유연한 프로그래밍을 할 수 있다. 이는 특히 유지보수에서 빛을 발하게 되는데, 내부 모듈의 내용이 교체되는 것이 실제 어플리케이션에 영향을 주지 않기 때문이다.

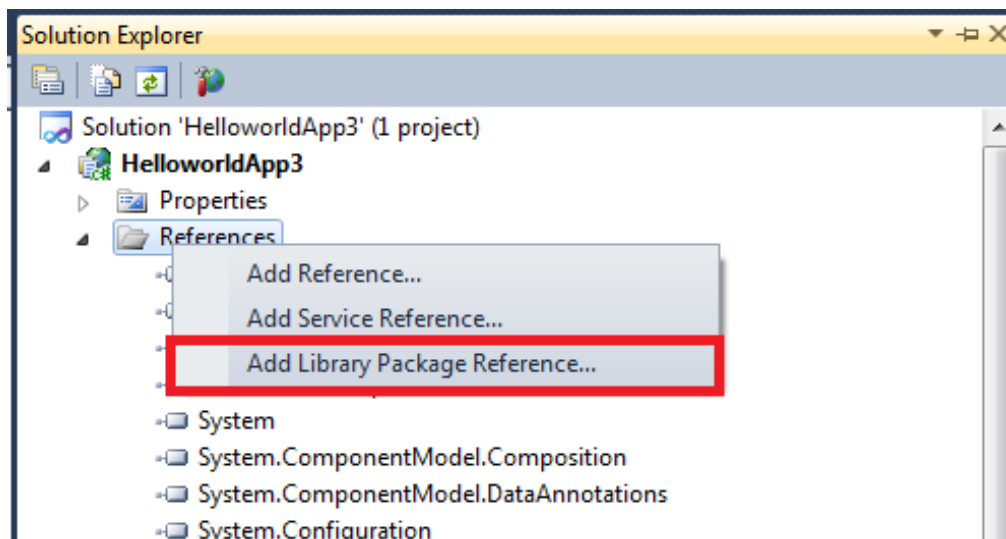
<MEF(Managed Extensibility Framework)>

MEF (Managed Extensibility Framework) 또한 DI 와 IoC 의 개념을 구현하기 위해서 개발된 프레임워크의 일종이다. 사실 이 MEF 는 IoC 보다는 좀더 깊은 개념의 프레임워크이지만, MVC 에서 IoC 를 구현하는 하나의 방법으로도 사용되고 있다. MEF 에 대한 자세한 설명은 Visual Studio2010 Team Blog 에서 자세 히 소개 하고 있다 . (<http://vsts2010.net/13>)

Nuget 은 Microsoft 에서 새로 선보이는 모듈 다운로드 엔진으로써 이번 MVC 가 도입되면서 같이 선을 보이게 되었습니다. 이 Nuget 에 대한 포스팅은 MVC 강좌가 끝난 후 다시 언급하도록 하겠다.

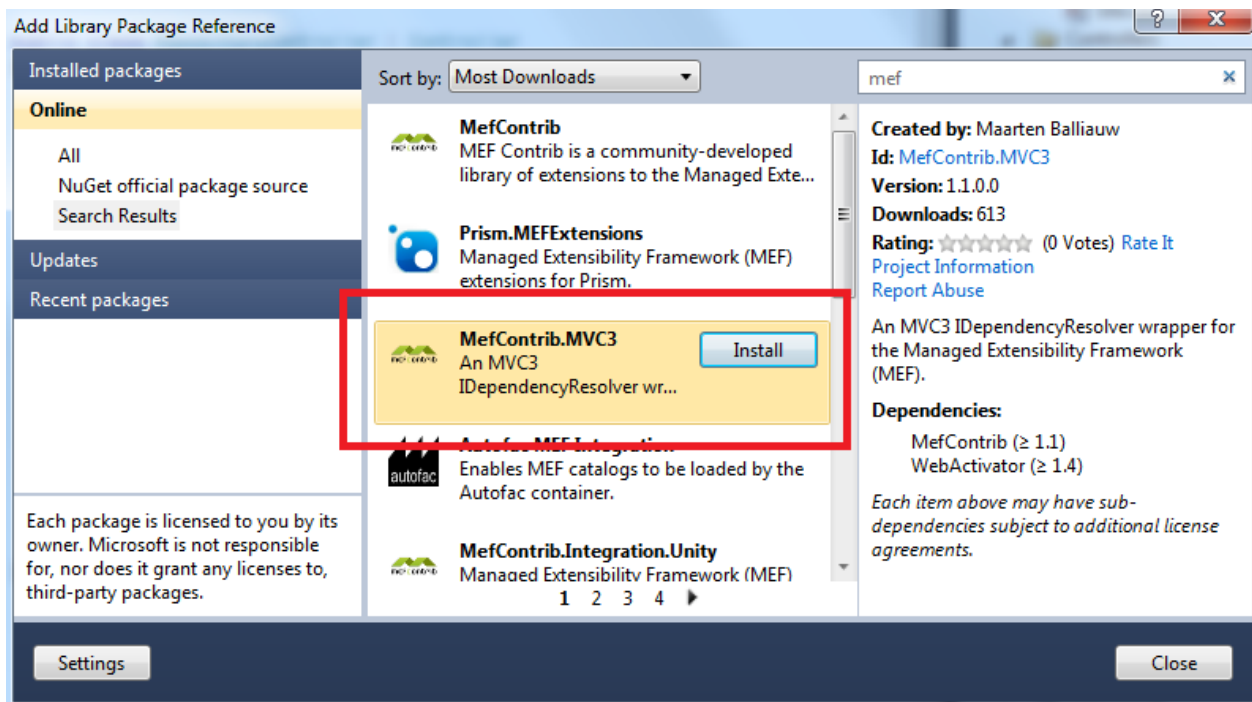
그러면 이제 MEF 를 이용해서 IoC 를 구현하는 예제를 살펴보도록 하겠다.

먼저 Reference 에서 우클릭 한 후 그림과 같이 Add Library Package Reference 를 선택한다.



[그림 Add Library Package Reference]

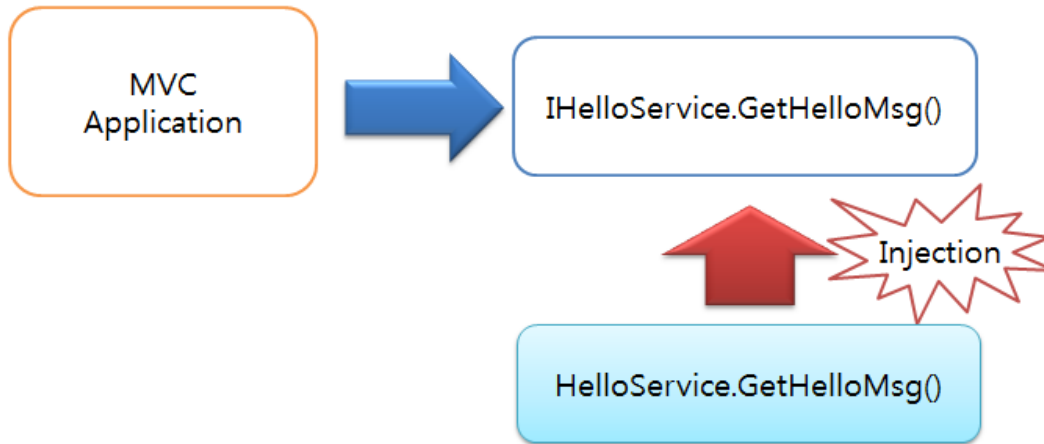
그러면 다음과 같이 Nuget 이 동작하면서, Microsoft 가 공급하는 여러 패키지를 인터넷에서 가져와서 보여줍니다. 이곳에서 MefContribMVC3 을 설치 한다.



[그림 : Nuget 에서 MefContrib 설치]

이제 MVC 에서 MEF 를 사용할 준비가 다 되었다.

IoC 의 도식도는 다음과 같다.



이제 IoC 에 대해서는 충분히 이해하였으니, 예제를 통해서 구현해 보도록 하겠다.

먼저 컨트롤러에서 사용할 인터페이스를 정의하고 컨트롤러의 생성자를 구현한다.

```
private IHelloService Service { get; set; }

public RepositoryController( IHelloService serv)
{
    this.Service = serv;
}
```

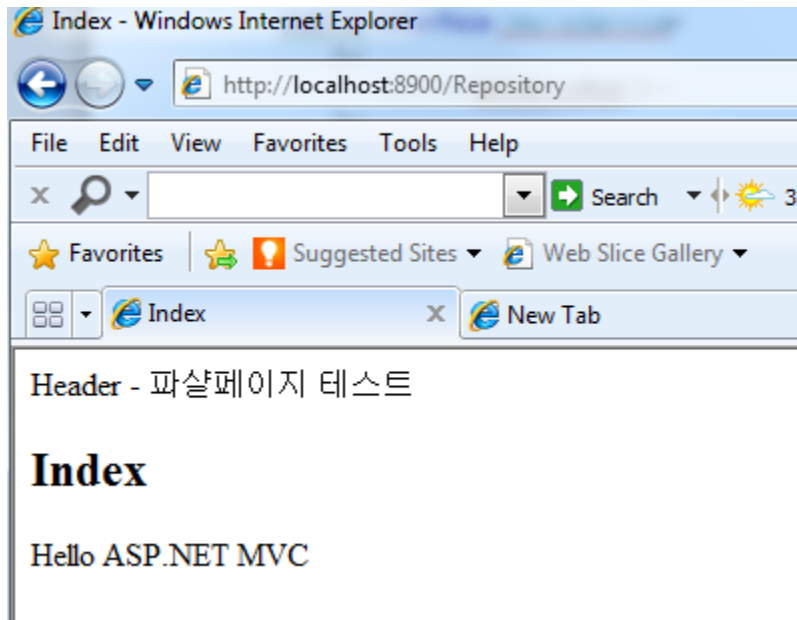
이제 컨트롤러는 만들었으니 컨트롤러가 만들 객체의 인터페이스를 만든다.

```
public interface IHelloService
{
    String GetHelloMsg( );
}
```

이제 IoC 로 사용할 인터페이스를 구성 했다. 이제 이걸 상속받는 HelloService 를 구현해보도록 한다.

```
[Export(typeof(IHelloService))]
public class HelloService : IHelloService
{
    public String GetHelloMsg( )
    {
        return "Hello ASP.NET MVC";
    }
}
```

아래는 실행시킨 결과이다.



[그림]

정상적으로 MEF 가 구동되어 HelloWorld class 의 GetHelloMsg 를 실행한 것을 확인할 수 있다.

Summary

IoC 는 요새의 프로그래밍에서는 유행을 탄다는 이야기가 나올 정도로 급속도로 전파되고 있는 개념이다. MVC 도 다른 여러 IoC 컨테이너를 구현하고 있지만, MEF 와 함께하는 IoC 는 상당히 간단하게 강력한 기능을 구현할 수 있다. 다음 장에서는 웹개발에 필수적인 자바스크립트를 MVC 에서 활용하는 방법에 대해서 살펴보도록 하겠다.

[ASP.NET MVC3 강좌] 14. Javascript in MVC

우리는 지난 시간에 MVC 에 적용된 Validation 에 대해서 알아보았다. 그런데 사실 이 Validation 은 초기에는 form 방식으로 동작하지만 이후에는 자동으로 적용된 자바스크립트를 이용해서 동작하게 된다. 사실 이 기능은 MVC2 에서 처음 소개가 되었으나, 여러 가지 문제가 있었고, 사실 정상적으로 사용하기에 약간 무리가 있는 정도로 퀄리티가 좋지 않았다. MVC3 에서는 이런 문제들을 해결하고, 새롭게 Unobtrusive Client Validation 이란 이름으로 새로운 개념을 도입하게 된다.

MVC3 을 처음 세팅하게 되면 다음과 같은 값들이 appSetting 에 있는 것을 확인할 수 있다.

```
<appSettings>
  <add key="webpages:Version" value="1.0.0.0" />
  <add key="ClientValidationEnabled" value="true" />
  <add key="UnobtrusiveJavaScriptEnabled" value="true" />
</appSettings>
```

이 옵션이 있고 없는 차이에 따라 렌더링 방식이 변하게 된다. 다음을 살펴보자.

이 예제는 기존 8 장에서 사용하였던 Validation 예제의 일부이다.

```
<div class="editor-field">
  <input class="text-box single-line" id="RegDate" name="RegDate" type="text" value="" />
</div>
```

[코드 UnobtrusiveJavaScriptEnabled 옵션이 설정되지 않은 상태]

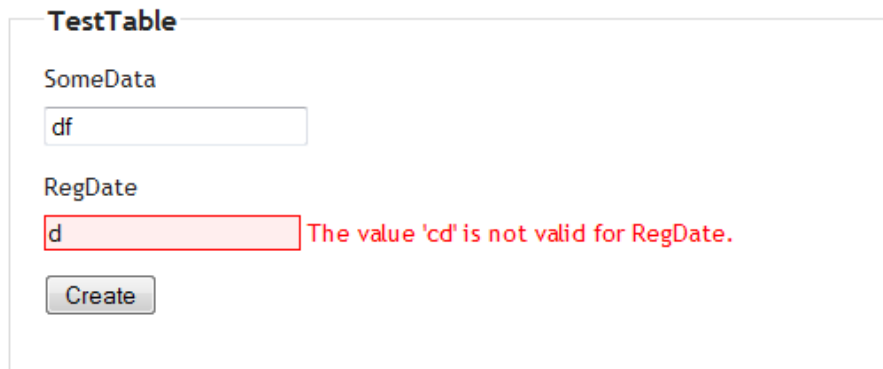
```
<div class="editor-field">
  <input class="text-box single-line" data-val="true" data-val-required="The RegDate field is required." id="RegDate" name="RegDate"
  type="text" value="" />
  <span class="field-validation-valid" data-valmsg-for="RegDate" data-valmsg-replace="true"></span>
</div>
```

[코드 UnobtrusiveJavaScriptEnabled 옵션을 설정한 상태]

이 어트리뷰트는 jquery 에서 해당 필드의 validation 을 검사하기 위해 사용된다. 그러기 위해 MVC3 에서는 디폴트로 해당 페이지를 생성하게 되면 다음의 두 개의 Javascript 가 참조 된다

```
<script src="/Scripts/jquery.validate.min.js" type="text/javascript"></script>  
<script src="/Scripts/jquery.validate.unobtrusive.min.js" type="text/javascript"></script>
```

Create



TestTable

SomeData
df

RegDate
d The value 'cd' is not valid for RegDate.

Create

[그림 : 자바스크립트가 참조되지 않은 상태]

해당 필드에서는 DateTime 의 타입을 체크하고 있는데 자바스크립트가 참조되지 않은 상태에서는 모든 요청을 서버에서 처리하기 때문에 해당 필드의 유효성 검증이 폼 전송을 통해서만 이루어지게 된다.

Create



TestTable

SomeData
df

RegDate
2010/01/01

Create

[그림 : unobtrusive 옵션을 설정하고 자바스크립트를 참조한 상태]

이제 정상적으로 Client Validation 이 설정되며 , DateTime 의 타입을 설정하는 모습을 볼 수 있다.

Summary

Client Validation 은 그렇게 큰 주목을 받고 있진 못하지만 MVC 팀에서는 상당히 신경쓰고 있는 기능 중에 하나입니다. 점점 더 좋은 모습을 보여주는 validation 인데 앞으로 MVC4 에서는 좀 더 향상된 기능이 될 것을 기대해 본다.

References

<http://bradwilson.typepad.com/blog/2010/10/mvc3-unobtrusive-ajax.html>

[ASP.NET MVC3 강좌] 15. Javascript Intellicense in Visual Studio

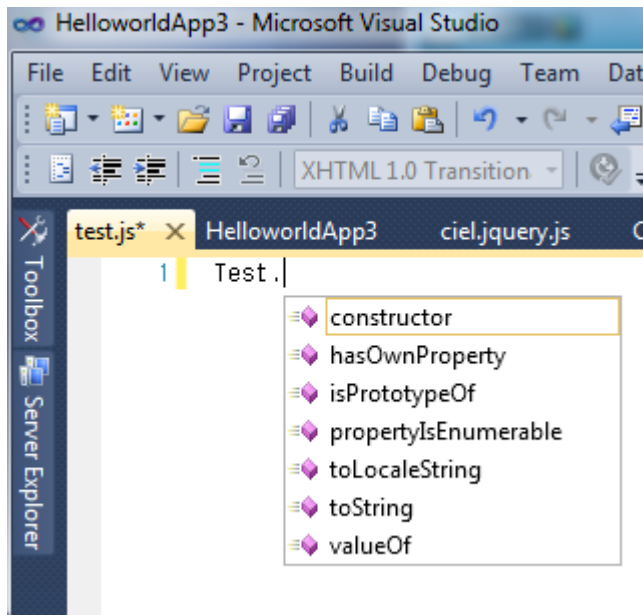
보통 자바스크립트로 어떠한 모듈을 만들 때, 사용자들은 대부분 부모 클래스를 만들고 그것을 참조해서 사용하는 형태로 많이들 사용하고 있다. Microsoft 에서는 Visual Studio 2008 부터 자바스크립트 인텔리센스를 적용해 폭넓은 편의성을 제공하고 있다.

그런데 이러한 자바스크립트 인텔리센스는 외부 자바스크립트를 연동할 때 동작하지 않는 문제점이 있습니다.

```
var Test = {  
    TestAlert: function () {  
        alert("Test");  
    }  
}
```

[코드 : ciel.jquery.js]

위의 코드를 다른 자바스크립트에서 참조해서 사용한다고 가정해보겠습니다.



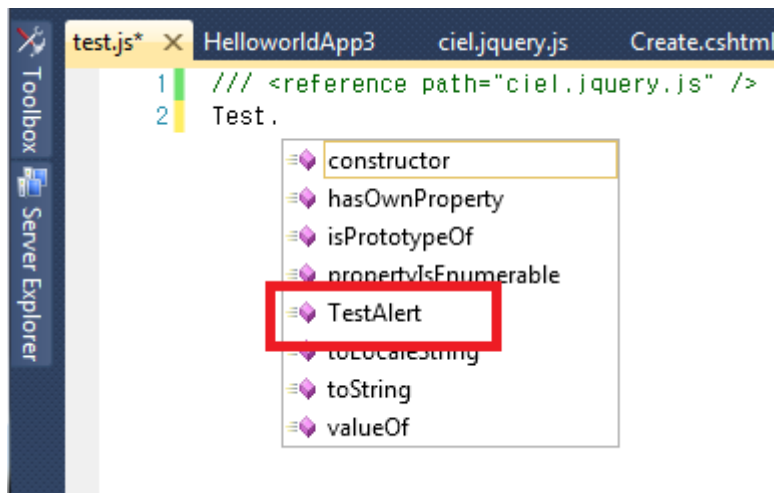
[그림 : 인텔리센스]

Test 개체에 있는 TestAlert 을 호출하고 싶었으나, test.js 에서는 ciel.jquery.js 와 아무런 연관성이 없으므로 인텔리센스에서 나타나지 않습니다.

Visual Studio 에서는 이 문제를 해결하기 위해 아래와 같은 xml 을 삽입하는 방법으로 인텔리센스를 지원한다.

```
/// <reference path="ciel.jquery.js" />
```

여기서 주의할건 주석이 항상 자바스크립트 파일의 최 상단으로 와야 한다는 것이며, 주석은 항상 '///' 으로 구성되어야 한다는 것 이다.



[그림 인텔리센스가 정상적으로 적용된 모습]

Summary

이번 장에서는 개발시에 고충을 많이 토로하시는 자바스크립트 인텔리센스의 지원에 대한 것을 알아보았다.

다음 장에서는 Ajax 와 Json 포맷 그리고 MVC 와의 연동에 대해 살펴해보도록 하겠다.

[ASP.NET MVC3 강좌] 16. JSON in MVC - 1

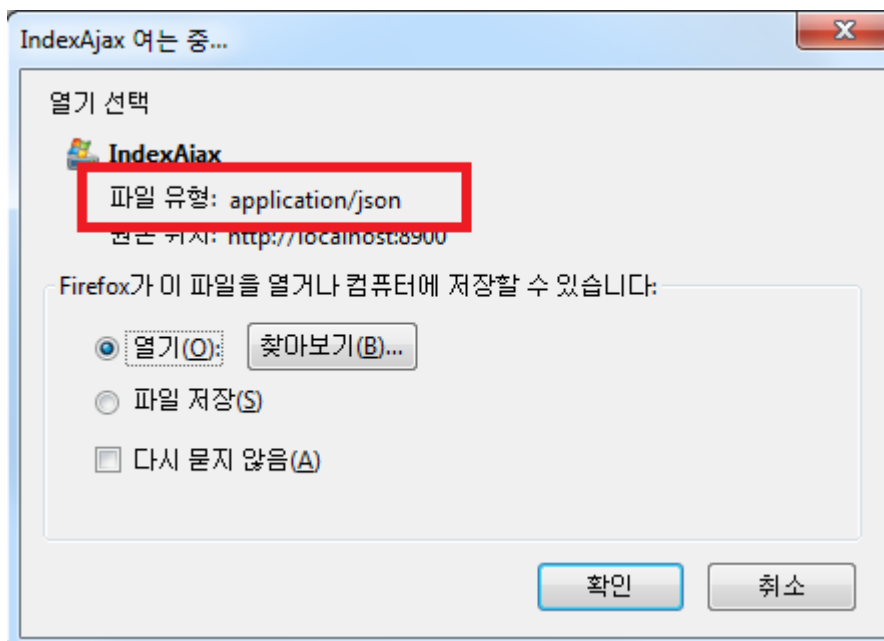
요즈음 웹 개발에서 가장 중요한 부분을 차지하는 부분은 Ajax 일 것이다. 이로 인해 더 적은 양의 데이터를 보내고, 화면의 깜박임 없이 사용자의 요청을 처리하는 것은 고급 웹 솔루션의 경우에는 거의 필수적인 기능으로 자리 잡아 가게 되었다. 이런 Ajax 의 가장 중요한 요소는 비동기로 원하는 데이터를 전송하는 것인데, 이를 위한 규격 중에 가장 가벼운 JSON(JavaScript Object Notation)을 많이 선호하게 되었다. 이번 장에서는 이러한 JSON 을 MVC 에서 사용하는 방법에 대해 살펴보도록 하겠다.

먼저 아래의 컨트롤러 액션을 살펴 보면 다음과 같다.

```
public ActionResult IndexAjax()  
{  
    return Json("test" , JsonRequestBehavior.AllowGet);  
}
```

[코드 : JSon 데이터 전달]

단지 하나의 메서드로, MVC 는 간편하게 Json 데이터를 전송할 수 있다.



[그림 JSon 데이터 전송]

<Json Serialization>

때로는 동적으로 게시판 등에서 바인딩을 위해 MVC 에서 사용하는 데이터를 JSON 형식으로 전송해야 하는 경우도 있습니다. 이런 경우에 MVC 는 간단하게 MVC 객체를 Serialize 해 준다.

이전 7 장에서 사용되었던 EF 코드를 JSON 으로 노출시키고, 바인딩을 해보도록 하겠다.

먼저 두 개의 ActionResult 를 작성한다.

```
public ActionResult TestView()  
{  
    return View();  
}  
  
//Json 을 호출하기 위한 뷰  
public ActionResult IndexAjax()  
{  
    return Json(db.TestTable.ToList(), JsonRequestBehavior.AllowGet);  
}
```

이제 실제 TestView 페이지를 작성해보도록 하겠습니다.

```
<div id="container"></div>  
<script type="text/javascript">  
    $.ajax({  
        url: '/Test/IndexAjax',  
        type: 'POST',  
        dataType: 'json',  
        contentType: "application/json; charset=utf-8",  
        success: function (result) {  
            if (result != null) {  
                var divHtml = "";  
                for(var i = 0 ; i < result.length ; i++)  
                {  
                    divHtml+= result[i].SomeData + "<br />";  
                }  
                $("#container").html(divHtml);  
            }  
        }  
    });  
</script>
```

[코드 : TestView 에서 작성된 Ajax 로 리스트를 호출하는 코드]



[그림 : 뷰에서 List 형태로 호출된 Json 뷰]

Summary

JSON 을 이용해서 DB 의 내용을 웹페이지와 통신하는 것은 이제 웹 개발에 있어서 기본적인 스펙 중에 하나로 인식이 되고 있다. MVC 에서는 이러한 JSON 통신에 대한 지원을 폭넓게 함으로써 개발에 대한 선택의 폭을 더 넓혀주고 있다. 다음 장에서는 이제 이러한 JSON 이 어떻게 MVC 의 고급 기능과 통합할 수 있는지 알아보도록 하겠다.

[ASP.NET MVC3 강좌] 17. JSON in MVC - 2

지난 시간에 우리는 JSON 으로 MVC 에서 어떻게 통신하는지에 대해서 간략하게 설명하려 한다.

```
<div id="container"></div>
<script type="text/javascript">
    $.ajax({
        url: '/Test/IndexAjax',
        type: 'POST',
        dataType: 'json',
        contentType: "application/json; charset=utf-8",
        success: function (result) {
            if (result != null) {
                var divHtml = "";
                for(var i = 0 ; i < result.length ; i++)
                {
                    divHtml+= result[i].SomeData + "<br />";
                }
                $("#container").html(divHtml);
            }
        }
    });
</script>
```

이전의 코드에서 단지 IEnumerable<T> 형태로 넘겨준 JSON 을 객체에 접근 하는 방법을 보면 이것은 다차원 배열에도 적용 가능하며 , '.' 을 이용해서 객체에 접근이 가능하다고 한다.

```
public class Test2
{
    public String element1 { get; set; }
    public String element2 { get; set; }
    public String element3 { get; set; }
}
public class Test
{
    Test2 t = new Test2();
}
```

위와 같이 선언되어 있는 경우 JSON 에서 Test.t.element1 과 같은 식으로 객체에 접근이 가능하다.

<Strong View 에 값 넘기기>

이제 실질적으로 Ajax 를 이용해서 값을 넣어 보도록 하겠습니다. 먼저 기존의 EF 코드에 Ajax 용 Create 를 작성한다.

```
[HttpPost]
public ActionResult CreateAjax( TestTable testtable)
{
    db.TestTable.AddObject(testtable);
    db.SaveChanges();
    return Json("true", JsonRequestBehavior.DenyGet);
}
```

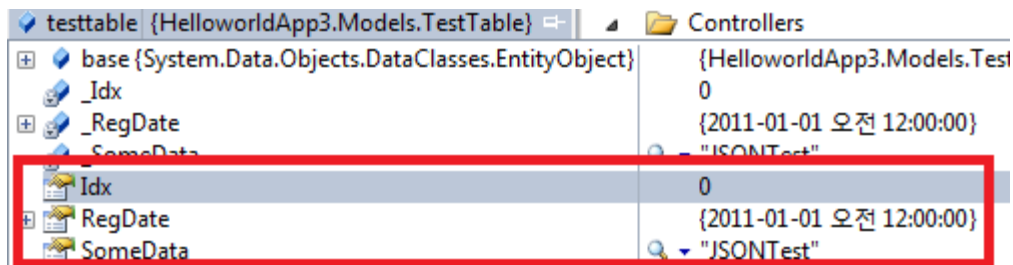
[코드 Ajax 용 Create ActionResult]

이제 자바스크립트를 이용해서 TestTable 객체를 전달 한다.

```
$.ajax({
    url: '/Test/CreateAjax',
    type: 'POST',
    dataType: 'json',
    data: JSON.stringify({
        SomeData: "JSONTest",
        RegDate: "2011/01/01"
    }),
    contentType: "application/json; charset=utf-8",
});
```

[코드 : 자바스크립트]

여기서 한가지 주의할 점은 JSON 형태로 객체를 전달할 때 MVC 가 인식하기 위해서는 반드시 직렬화를 해야 한다는 사실인데 이를 지원하기 위해서 JSON.stringify 메서드를 사용해서 JSON 을 직렬화 한다.



testtable (HelloworldApp3.Models.TestTable)	Controllers
base {System.Data.Objects.DataClasses.EntityObject}	{HelloworldApp3.Models.Test
_Idx	0
_RegDate	{2011-01-01 오전 12:00:00}
SomeData	"JSONTest"
Idx	0
RegDate	{2011-01-01 오전 12:00:00}
SomeData	"JSONTest"

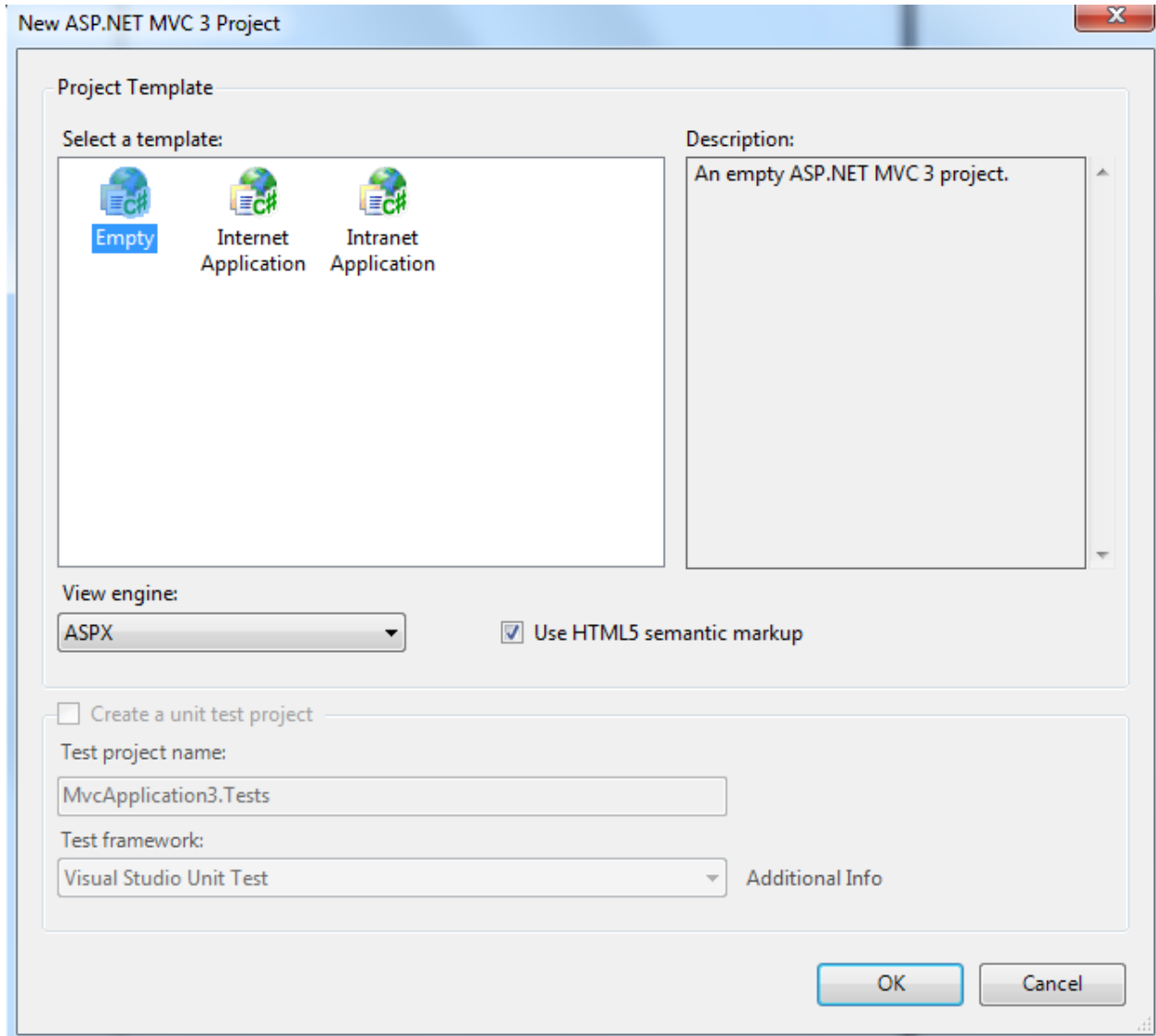
[그림 : 직렬화된 JSON 이 MVC 로 정상적으로 넘어온 모습]

Summary

지금까지 JSON 과 자바스크립트를 이용한 MVC 와의 통신방법에 대해서 짧게 알아보았다. 이 부분은 Javascript 와 MVC 를 다른 개념으로 생각하지 않고 개념을 일원화 시킨다는 점에서, 상당히 효율적인 개발 방법론을 제시하고 있다고 생각 한다. 다음 장에서는 MVC 이 제공하는 html5 지원 기능들에 대해서 살펴해보도록 하겠다.

[ASP.NET MVC3 강좌] 18. HTML5 in MVC

MVC3에서는 최근 추세에 발맞춰서 html5를 지원하기 위한 장치를 선보이고 있는데요, 이번 장에서는 이러한 기능들에 대해서 살펴 본다.



[그림 MVC3에서의 html 마크업 지원]

Mvc3에서는 기본적으로 html5 마크업을 선택해서 작성할 수 있는 기능이 있다. 아쉽게도 모든 인텔리센스가 html5로 맞춰지는 것은 아니지만, 시작하는 페이지가 html5로 작성되어 있다는 것만으로도 많은 편의성이 제공된다.

Html5 마크업을 기본적으로 선택하게 되면 스캐폴딩으로 생성되는 모든 페이지의 마크업이 html5 로 작성되게 된다.

```
<!DOCTYPE html>

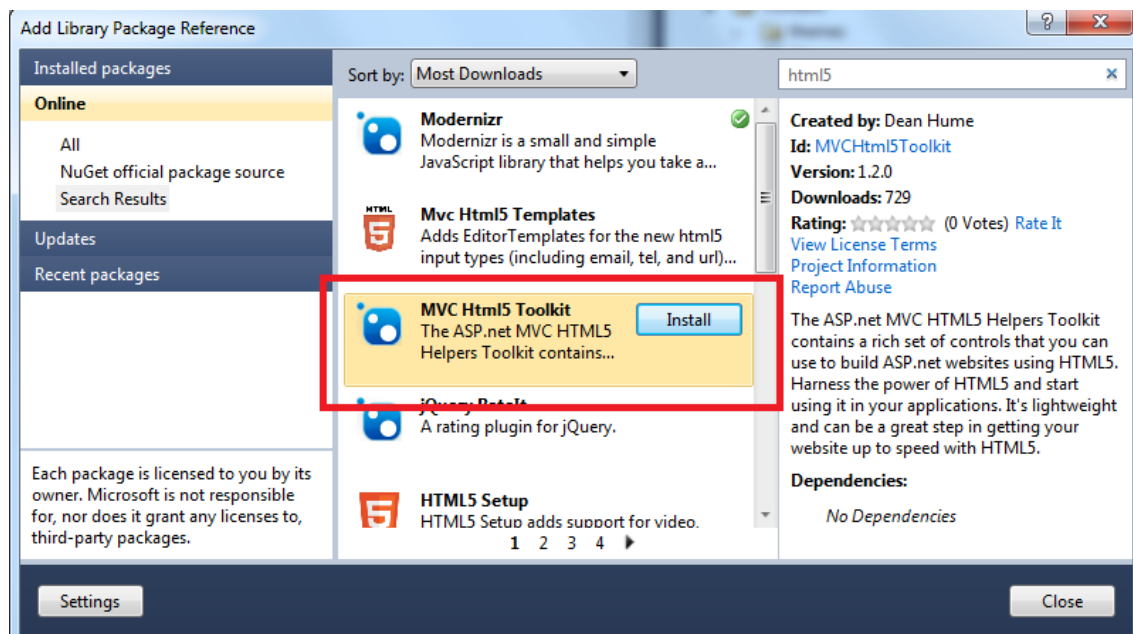
<html>
<head runat="server">
  <title>Index</title>
</head>
<body>
  <div>

    </div>
</body>
</html>
```

[코드 : html5 마크업이 적용된 MVC View 페이지]

<HTML5 컨트롤>

MVC 의 기본기능은 아니지만, Nuget 에서는 HTML5 컨트롤을 사용할 수 있는 패키지를 제공해 준다. 이는 MVC Html5 Toolkit 이름으로 배포되고 있다.



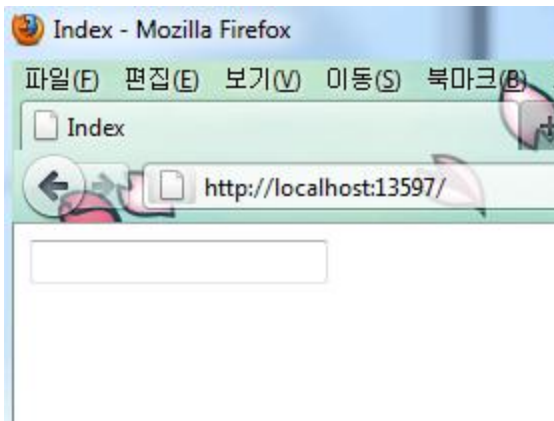
[그림 MVC Html5 Toolkit]

아쉽게도 현재는 Html5 가 표준으로 정해진 것이 아니기 때문에 브라우저에 따라 지원하는 컨트롤들이 한계가 있다. 이번 예제에서는 Email 유효성을 자동으로 체크해주는 유효성 검사 텍스트 박스를 작성해보도록 하겠다.

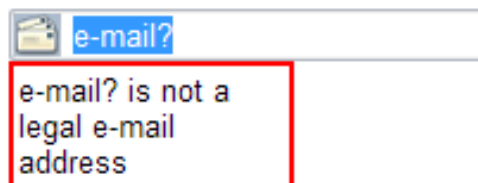
```
@using System.Web.Mvc.Html5;
@{
    ViewBag.Title = "Html5Test";
    Layout = "~/Views/Shared/_LayoutMaster.cshtml";
}
<h2>Html5Test</h2>
@Html.Html5TextBox("userEmail", InputTypes.InputType.Email, plah@plah.com);
```

[코드 : html5 텍스트 박스]

해당 기능은 Nuget 애드온 형태이기 때문에 반드시 코드의 최상단에 @using System.Web.Mvc.Html5;를 삽입해야 한다.



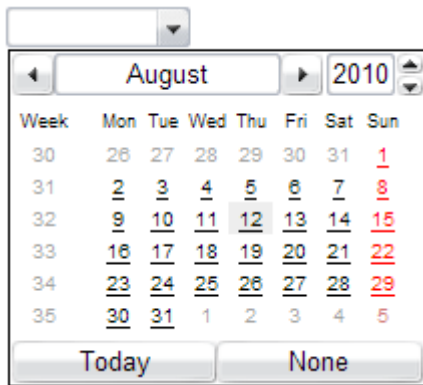
아쉽게도 파이어폭스에서는 일반적인 텍스트 박스로 나타난다. 이 텍스트 박스는 오페라에서 정상적으로 보여진다.



아래와 같은 html 컨트롤 들을 MVC 에 적용할 수 있다.



Html.Html5Range(0 , 100,1,50,null);



Html.Html5TextBox("deliveryDate", InputTypes.InputType.Date)

Summary

Html5 를 컨트롤 형태로 제공하는 것은 MVC 로 간편하게 개발할 수 있는 또 다른 편의성의 척도가 될 것이다. 단순한 부가기능으로 치부될 수도 있지만, 이러한 세세한 지원은 최신 트렌드를 따라간다는 의미에서 MVC 를 높게 평가할 수 있겠다.

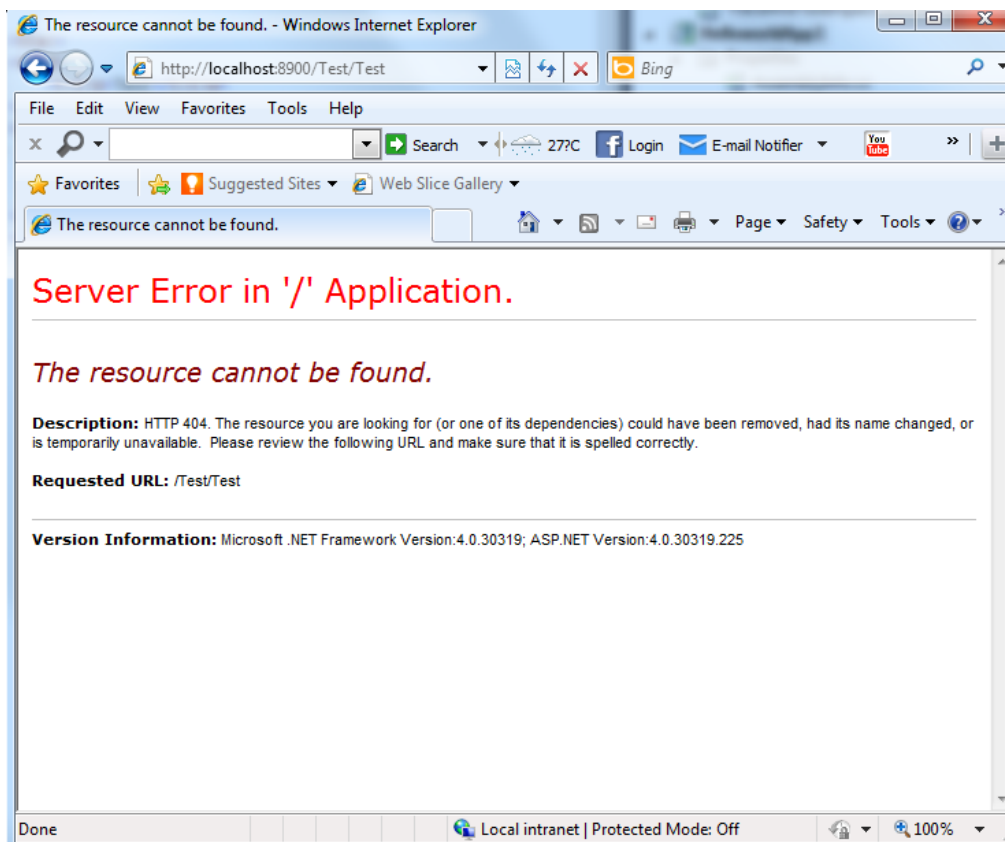
[ASP.NET MVC3 강좌] 19. MVC Tips 1 - HandleUnknownAction

이번 시간부터는 MVC 의 기본기능 보다는 트러블 슈팅이나, 팁에 대한 내용을 주로 다루어 보도록 하겠다.

MVC 를 사용할 때 몇몇 개의 기능은 기존 ASP.NET Classic 보다 부족하다고 느껴질 수 있다. 특히나 사용자가 어떠한 페이지를 생성해야 할 때 반드시 그에 매칭되는 Action 을 만들어야 한다는 것이 디자이너와의 협업에서 상당한 불편함을 가져올 수 있다. 이는 MVC 초반부터 항상 제기되어 오던 문제이며, MVC 의 개발상의 약점으로 손꼽히곤 했다. 이번 시간에는 간단하게나마 이러한 문제점들을 해결하기 위한 방법을 공유하고자 한다.

1. HandleUnknownAction?

HandleUnknownAction 은 컨트롤러에서 Action 에 대한 요청을 받았을 때 해당 액션이 없으면 호출되는 메서드 이다. 이 메서드가 구현되지 않았을 때 사용자는 다음과 같은 에러를 보게 된다.

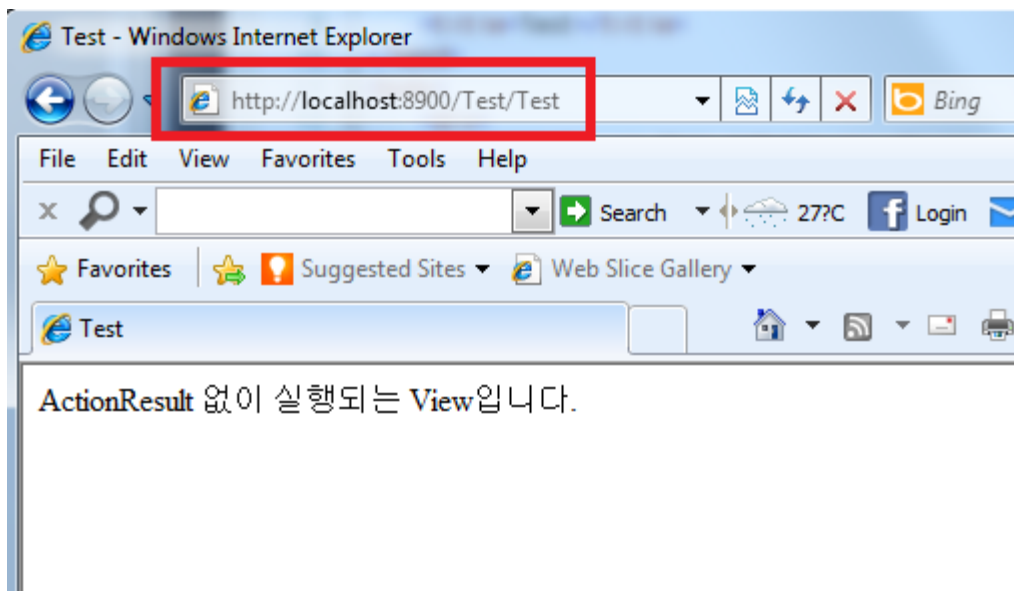


[그림 : ActionResult 없기 때문에 404 에러를 반환하는 모습]

흔히 볼 수 있는 404 Error 입니다. 하지만 Test 폴더에는 여전히 Test.cshtml 이 존재하는 상태이다. 이제 이것을 HandleUnknownAction 을 이용해서 나타나게 하겠다.

```
protected override void HandleUnknownAction(string actionName)
{
    this.View(actionName).ExecuteResult(this.ControllerContext);
}
```

해당 메서드를 실행하게 되면 ActionName 을 받았을 때 해당 컨트롤러를 이용해서 View 를 즉 ActionResult 의 실행 없이 실행되게 된다.



[그림 : ActionResult 없이 View 를 호출]

Summary

웹 개발에서 협업이라는 이슈는 가장 오래되고 진부하지만 중요한 이슈이다. 그 중심에는 컴파일이라는 복병이 숨어 있게 되는데, 이 문제 때문에 디자이너가 자신이 만든 파일을 볼 수 없다면 이 또한 답답한 일일 것이다. HandleUnknownAction 은 이러한 문제를 해결하는데 아주 유용하며, 또한 다른 에러를 처리하기도 용이하다. 다음 장에서는 ActionFilter 를 응용한 사례를 소개하도록 하겠다.

[ASP.NET MVC3 강좌] 20. MVC Tips 2 - Cache ActionFilter

웹 페이지에서 가져오는 데이터가 많거나 데이터베이스의 접근량이 많을수록 데이터베이스는 많은 요청을 받게 되고 그로 인해 DB 는 많은 부하를 받게 된다. 특히나, 포털사이트의 메인페이지 같은 경우 이러한 요청으로 인해 문제가 많이 발생하게 되는데 대부분의 경우 이러한 문제를 캐싱을 통해서 해결하게 된다. 캐싱은 크게 클라이언트 캐싱과 서버 측 캐싱이 있는데, 서버 쪽 캐싱은 서버에 데이터베이스에서 가져온 객체를 저장하는 방식으로, MVC 뿐 아니라 많은 ASP.NET 솔루션에서 구현이 가능하다. 그에 비해 클라이언트 캐싱은 주로 header 값에 caching 을 추가하는 형태로 구현을 하게 되는데 여기서는 클라이언트 캐싱을 ActionFilter 로 구현하는 예제를 소개할 예정이다.

```
public class CacheAttribute : ActionFilterAttribute
{
    /// <summary>
    /// Gets or sets the cache duration in seconds. The default is 10 seconds.
    /// </summary>
    /// <value>The cache duration in seconds.</value>
    public int Duration
    {
        get;
        set;
    }

    public CacheAttribute()
    {
        Duration = 10;
    }

    public override void OnActionExecuted(ActionExecutedContext filterContext)
    {
        if (Duration <= 0) return;

        HttpCachePolicyBase cache = filterContext.HttpContext.Response.Cache;
        TimeSpan cacheDuration = TimeSpan.FromSeconds(Duration);

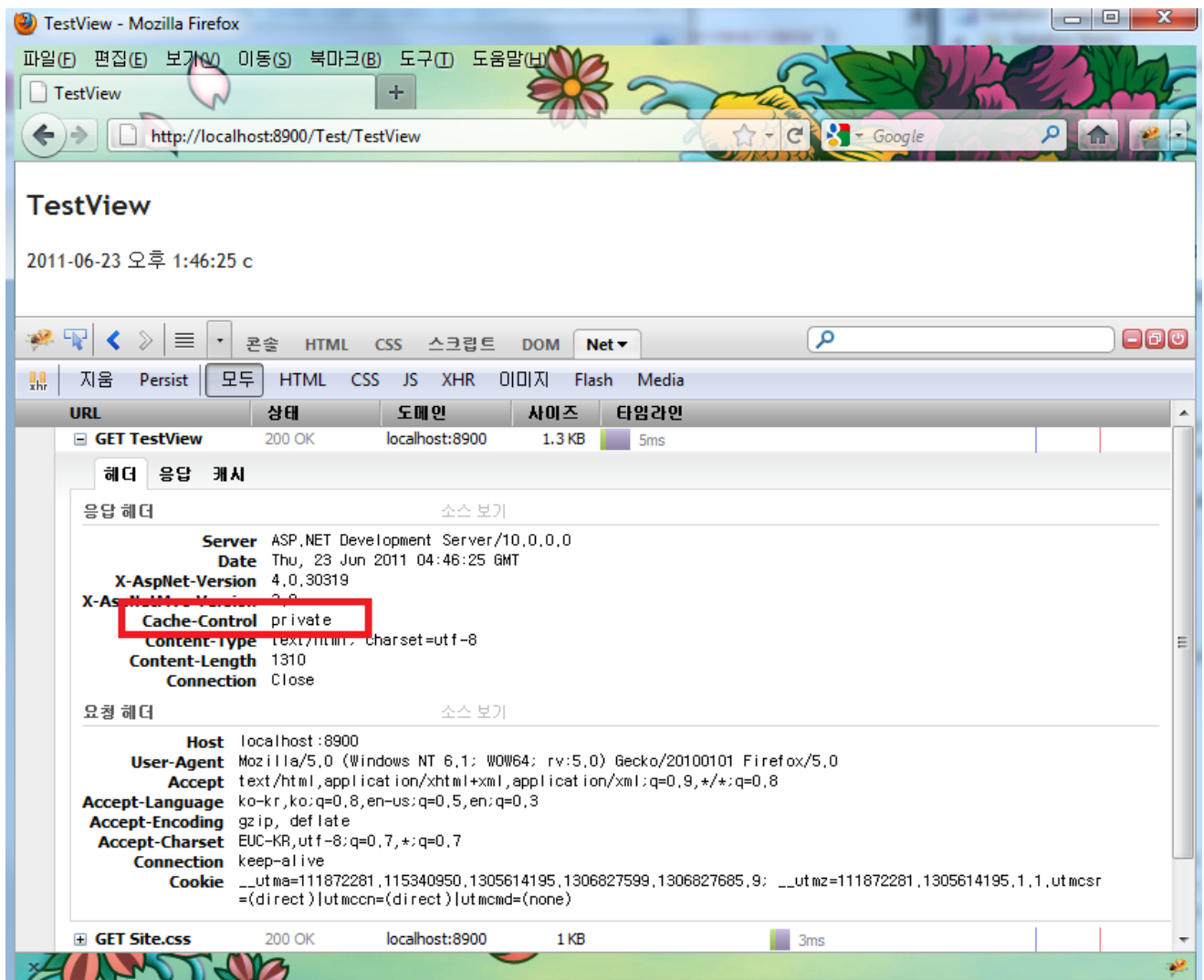
        cache.SetCacheability(HttpCacheability.Public);
        cache.SetExpires(DateTime.Now.Add(cacheDuration));
        cache.SetMaxAge(cacheDuration);
        cache.AppendCacheExtension("must-revalidate, proxy-revalidate");
    }
}
```

ActionFilter 는 ActionFilterAttribute 를 상속받아 구현되게 된다. 이 또한 Attribute 이기 기존 C#에서 구현할 수 있는 모든 Attribute 의 기능이 사용 가능 하다. 이번 캐싱 예제에서는 Duration 을 이용해서, 최대 캐싱될수 있는 초 범위를 설정을 하였다.

이제 예제를 위한 Action 을 생성한다.

```
public ActionResult TestView()  
{  
    ViewBag.Test = DateTime.Now.ToString();  
    return View();  
}
```

간단하게 ViewBag.Test 값에 id 를 넘겨주는 예제이다.

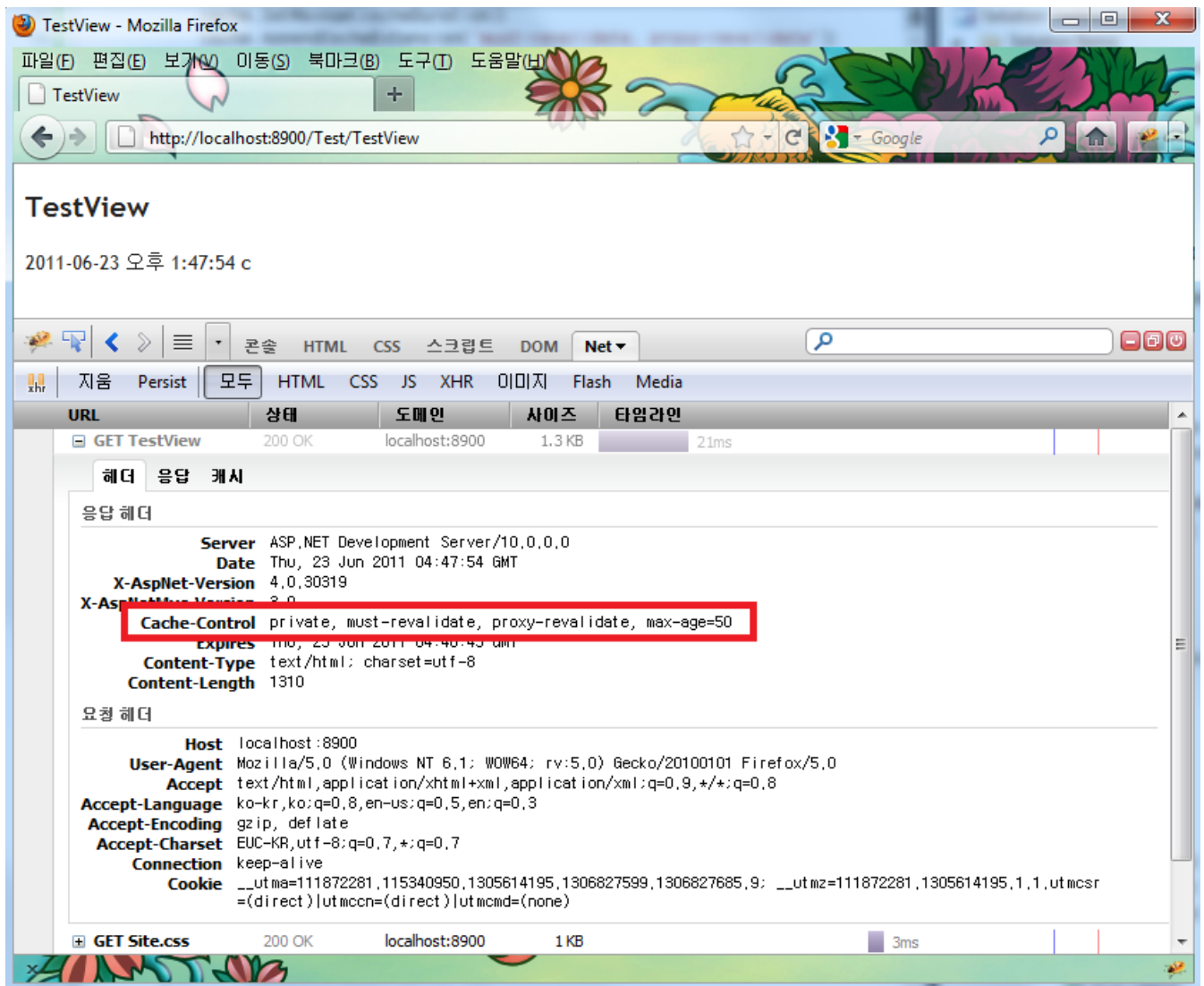


현재 캐시가 설정되어 있지 않은 화면이다.

이제 ActionFilter 를 적용한다.

```
[Cache(Duration=50)]
public ActionResult TestView()
{
    ViewBag.Test = DateTime.Now.ToString();
    return View();
}
```

Attribute 에서 Duration 을 설정할 수 있다. 이번 예제에서는 50 초로 설정하도록 하겠다.



헤더에서 정상적으로 캐싱이 반영된 것을 확인할 수 있다.

Summary

MVC 는 언뜻 보면 허술하고 할 것이 많이 보이는 프레임워크로 치부될 수 있다. 그러나 이곳에 있는 기능을 속속들이 살펴보게 되면, 기존 ASP.NET Classic 보다 더 쉽게 개발이 가능하며 또한 유동적 프레임워크라는 것을 발견 하실 수 있을 것이다.