HTTP는 비연결 프로토콜입니다. 클라이언트(정확하게는 웹 브라우저)가 서버에 특정 처리를 요청하고 나면 서버에서 그에 맞는 응답을 해주는 것으로 끝납니다. 연결이 계속해서 유지되어야 하는 FTP와는 다른것입니다.

이러한 HTTP통신 방식으로 인해 클라이언트가 어떤 데이터를 서버에 요청을 할때마다 서버는 어떻게 해서든 이 사용자가 누구인가 혹은 어떤 데이터를 다루고자 하는가를 구분해야할 필요가 있습니다. 이럴 때 세션(Session)이 사용될 수 있습니다. 다만 세션은 사용자를 구분하기 위한 유일한 방법이 아닙니다. 그저 사용자를 구분하는 방법중 '하나'에 해당할 뿐입니다.

세션은 쿠키와 비교될 수 있습니다. 쿠키는 클라이언트에 저장되는 반면 세션은 서버에서 저장되고 관리됩니다. 쿠키보다 훨씬 보안적으로 우수하지만(쿠키는 원한다면 얼마든지 사용자에 의해 변조될 수 있습니다.) 세션은 서버의 자원을 할당해야 한다는 측면에서 성능상으로 단점이 존재합니다.

1. InProc

세션은 기본적으로 InProc모드로 관리됩니다. In-Process라는 것으로 HttpRuntime 개체(프로세스)의 내부 캐시에 세션을 저장하고 관리합니다. 이 때문에 실행속도가 가장 빠른 모드이지만 세션을 계속 참조상태에 유지하고 있고 이에대한 가비지 컬렉션은 별도로 이루어지는게 없어서 메모리를 계속 점유하고 있다는 단점이 존재합니다. 물론 그렇다 하더라도 메모리 점유는 만료시간이 20분이 경과하거나 별도의 구성파일에 지정된 시간이 지나면 자동으로 만료되므로 평생 세션을 저장해두지는 않습니다.

참고로 InProc모드에서의 세션은 dll이 빌드되어 새로 배포되거나 Web.config와 같은 구성파일의 변경등과 같은 이유로 프로세스가 다시 시작하면 기존 세션은 모두 손실됩니다.

ASP.NET에서 세션은 다음과 같이 읽고 쓸 수 있습니다.

```
Session["aaa"] = "aaabbb";
Response.Write(Session["aaa"].ToString());
```

세션은 ASP.NET의 모든 페이지에서 위와 같은 방법으로 엑세스할 수 있습니다. 만일 "aaa"라는 인덱스의 세션을 사용하는 특정페이지가 호출된다면 이 요청이 완료될때까지 같은 세션을 사용하는 다른 페이지는 대기상태에 들어가게 됩니다.(사용하는 세션의 인덱스가 다르면 영향이 없습니다.)

필요하다면 특정 페이지마다 EnableSessionState 속성을 조정함으로서 세션접근에 대한 권한을 조정할 수 있습니다.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
 Inherits="web_form_test.Default" EnableSessionState="True"  %>
```

True값은 세션에 대한 읽기/쓰기 권한을 모두 부여합니다. 같은 세션을 사용하는 다른 페이지에서는 세션이 잠기게 됩니다.

False값은 세션에 대한 권한을 주지 않습니다. 만약 False설정이된 페이지에서는 세션이 접근할 수 없습니다.

ReadOnly는 세션에 대해 오로지 읽기권한만을 부여합니다. 이 경우에는 세션잠금이 설정되지 않기 때문에 다른 페이지에서 같은 세션의 값을 읽을 수 있습니다. 하지만 세션에 쓰기가 시도되는 경우 모든 읽기또한 차단됩니다.

재미있는 점은 EnableSessionState가 False로 설정된 페이지는 설정이 없는 페이지요청보다 우선적으로 처리하게 된다는 것입니다. 따라서 계획적으로 잘만 설정된다면 성능적으로 이익을 볼 수 있습니다. 극단적으로 Web.config를 아래와 같이 처리하면

```
<sessionState mode="Off"></sessionState>
```

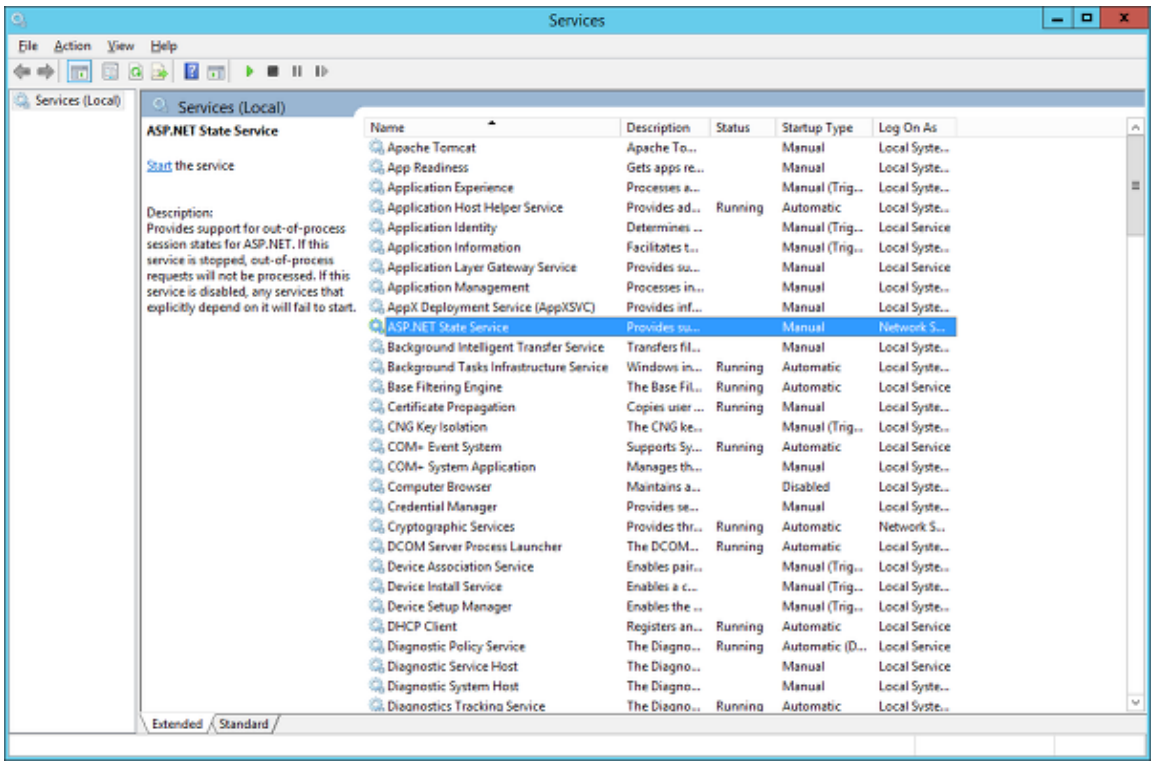
웹애플리케이션 전체에 대해 세션사용을 고려하지 않게 되므로 성능면에서 훨씬 이득이지만 세션은 사용할 수 없습니다.

2. Out-of-Process

Out-of-Process는 세션을 aspnet_state.exe라는 프로세스에 저장하는 방식입니다. 다른 물리적 서버에 해당 프로세스를 띄어놓고 세션을 관리하는 형태(세션을 처리하는 서버를 별도로 두는것)이지만 원한다면 로컬에 프로세스를 실행하는것 또한 가능합니다. aspnet_state.exe는 자동으로 시작되는 프로세스가 아니므로 서비스를 수동으로 시작해 줘야 합니다.

```
net start aspnet_state
```

위와 같은 명령줄 대신 Services 관리자에서 시작할 수도 있습니다.



이 프로세스는 TCP통신을 하는데 기본포트가 42424입니다. 이 설정은 아래 레지스트리경로에서 바꿀 수 있습니다.

[HKEY_LOCAL_MACHINE]₩SYSTEM₩CurrentControlSet₩Services₩aspnet_state₩Parameters₩Port

세션관리를 Out-of-Process로 하기로 했다면 Web.config에서 세션설정을 바꿔줍니다.

```
<sessionState mode="StateServer" stateConnectionString="tcpip=127.0.0.1:42424">
</sessionState>
```

mode를 StateServer로 설정하여 Out-of-Process 사용을 명시하고 stateConnectionString을 통해서 연결될 서버의 IP와 Port를 설정합니다. 예제에서는 로컬에서 사용할 것이므로 127.0.0.1:42424로 설정하였습니다. 다른 서버라면 그에 맞는 IP를 지정해 주시면 됩니다.

이 방식은 일단 이름에 걸맞게 프로세스를 벗어난 세션관리형태의 방식입니다. 그러니까 ASP.NET애플리케이션(코드)은 w3wp.exe프로세스에서 동작하는데 세션관리는 aspnet_state.exe 프로세스에 관리되므로 코드에서 세션을 참조할 수 있는 형태가 아닙니다.

그래서 세션에 단순한 문자열을 담는 경우는 상관없지만 클래스나 구조체등을 초기화 하고 세션에 담으려 하면 오류가 발생합니다. 코드의 동작과 세션관리가 다른 프로세스로 분리되어 있기 때문입니다. 이들 프로세스간 통신을 수행해서 세션처리를 정상적으로 처리하려면 사용하려는 클래스에 [Serializable] 특성을 선언해야 합니다.

```
[Serializable]
public class testClass
```

```
{
    public string myname;
}
```

클래스를 위와같이 작성하고

```
testClass tc = new testClass();
protected void Page_Load(object sender, EventArgs e)
{
    tc.myname = "cliel";
    Session["aaa"] = tc;
}

protected void Button1_Click(object sender, EventArgs e)
{
    Response.Write((Session["aaa"] as testClass).myname);
}
```

클래스를 세션에 담으면 정상적으로 코드가 수행될 것입니다. 다만 클래스(개체)를 [Serializable] 특성을 사용해 직렬화 하면 직렬화되는 과정에서 크기가 다소 커질 수 있습니다. 그 만큼 aspnet_state.exe 프로세스는 관리해야 하는 세션의 크기가 커질 수 있다는 뜻인데 sessionState 요소의 compressionEnabled속성을 true로 하면 개체를 압축해서 저장하므로 크기에 대한 부담이 다소 줄어들 수 있습니다.

```
<sessionState mode="StateServer" stateConnectionString="tcpip=127.0.0.1:42424"
 compressionEnabled="true"></sessionState>
```

하지만 실제 개체를 사용하려면 어떻게 해서든 압축된걸 푸는 과정이 필요하므로 InProc보다는 성능에서 불리할 수 있습니다.

3. SQL 서버 세션

세션은 필요하다면 SQL 서버에서 저장하고 관리할 수도 있습니다. 이 방식은 이전에 말씀드린 방식에 비해 가장 느리지만 반면 가장 안전하게 동작하는 방식입니다. 심지어는 IIS전체가 재시작되는 경우라 하더라도 서버에 세션정보가 저장되므로 사용자와의 세션은 계속 유지될 수 있습니다.

SQL서버를 통해서 세션을 관리하려면 특정 DB안에 세션저장을 위한 구성이 추가되어야 합니다. htt이 작업은 aspnet_regsql.exe 를 통해서 구현할 수 있습니다.

아래는 세션관리를 위한 aspnet_regsql.exe 실행 예시입니다.

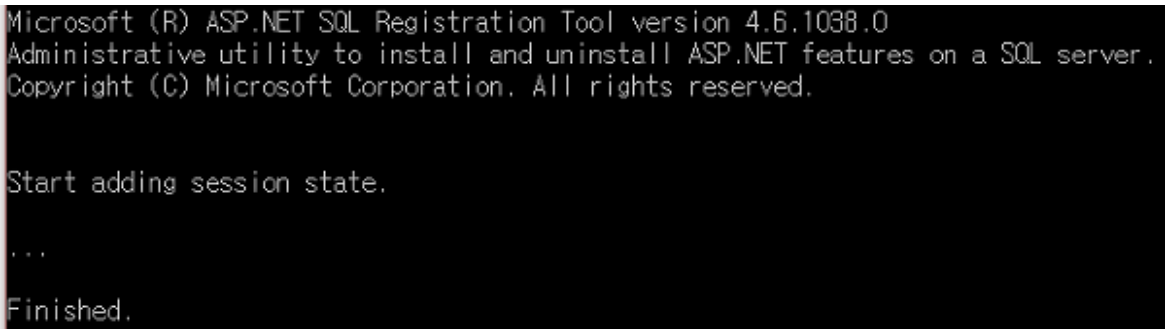
```
aspnet_regsql.exe -S localhost -U sa -P 1234 -ssadd -sstype p
```

위와 같이 명령줄을 입력하면 해당 서버에 ASPState DB를 자동으로 생성하고 세션저장을 위한 구성을 완료하게 됩니다.

참고로 -ssadd는 지정된 서버에 세션상태저장을 위한 구성을 추가하는 옵션이며 반대로 -ssremove는 추가된 구성을 제거하도록 하는 옵션입니다.

-sstype으로 지정된 p는 데이터와 프로시저를 ASPState DB에 저장하도록 합니다. 물론 DB도 알아서 만들어 주므로 일단 편리함은 최고입니다. p외에 t라는 옵션도 있는데 이 옵션은 세션을 tempdb에 저장하고 프로시저는 ASPState에 생성하도록 합니다. 이는 세션을 임시테이블에 저장하는 형태다 보니 SQL서버가 다시 시작되면 세션데이터는 사라진다는 특징이 있습니다.

만일 ASPState DB에 세션저장과 프로시저가 생성되는걸 원하지 않는다면 c옵션으로 직접 세션이 저장되고 프로시저가 생성되는 DB를 지정해 줄 수도 있습니다.



aspnet_regsql.exe를 실행시켜 위와 같이 정상적으로 처리되었다면 이제 web.config를 수정하여 세션을 SQL Server에 저장하도록 해주면 됩니다.

```
<sessionState mode="SQLServer" sqlConnectionString="data source=127.0.0.1;user
 id=sa;password=12345"></sessionState>
```

참고로 ASPState DB가 아닌 다른 이름의 DB를 사용한다면 위 Web.config설정에서 database=mydb형식으로 내용을 추가해 줘야 합니다.