Sessions et Application

ASP.NET MVC

Sessions et Application

Il existe des événements particuliers en ASP.NET permettant de détecter les lancement et les arrêt des sessions et de l'application.

Ces événements se trouvent dans le fichier Global.asax

1. **Application**

Evénement	**Description**
Application_Start	Exécuté après le premier appel à une page du site depuis le démarrage de IIS
Application_End	Appelé lorsque l'application se termine, cela ne signifie pas que IIS s'arrête mais est d'office appelé si, pour une raison quelconque IIS est arrêté

2. **Session**

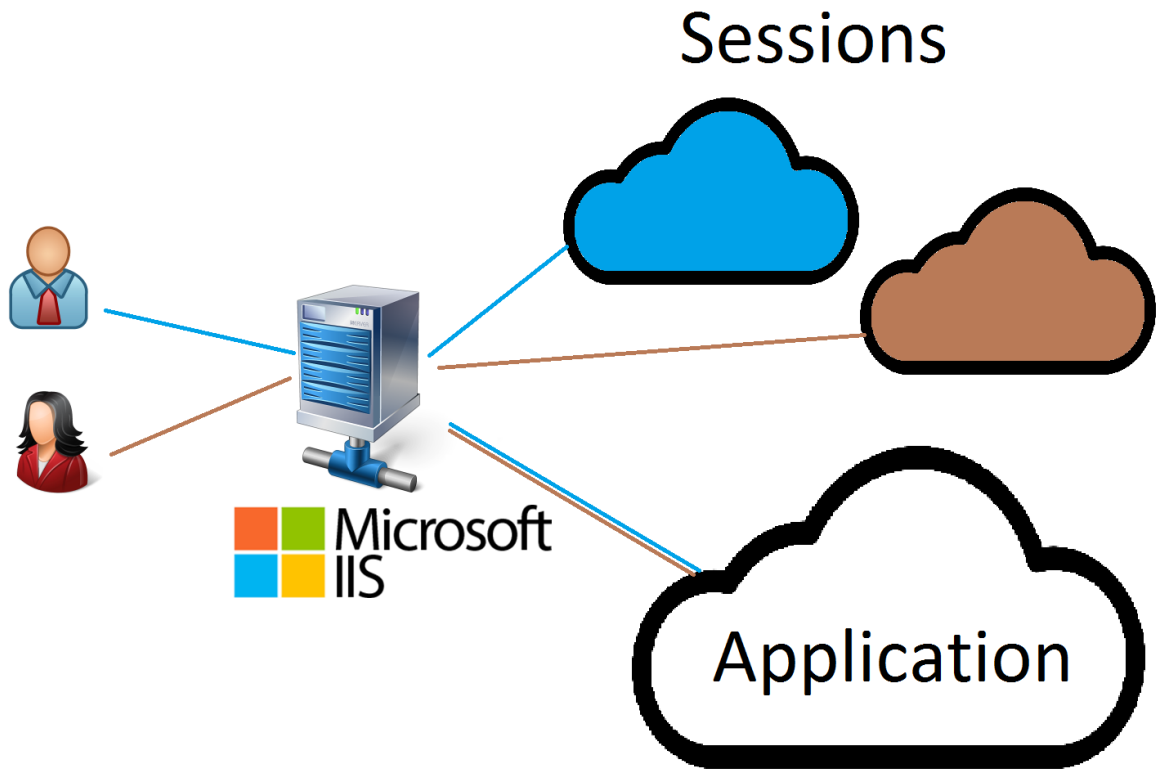
Evénement	**Description**
Session_Start	Appelé lors de chaque nouvelle session d'un navigateur client
Session_End	Fin de session soit explicite (Session.Abandon()) ou suite à un time out

Sessions et Application

Soit deux utilisateurs qui envoient une requête vers le serveur IIS.

Il y aura :

- **un seul objet "Application" commun** à tous les utilisateurs du site
- **deux objets "Session"** correspondant chacun à un utilisateur précis.



Sessions et Application

Pourquoi des Session ?

Les session ASP.NET vous permet de stocker et de récupérer des valeurs pour un utilisateur à mesure que ce dernier navigue dans les différentes pages ASP.NET qui composent une application Web. HTTP est un protocole sans état, ce qui signifie que votre serveur Web traite chaque demande de page HTTP comme une demande indépendante

Sessions et Application

- **<u>Comment utiliser les session ?</u>**

  La session ASP.NET est activé par défaut pour toutes les applications ASP.NET. Les variables d'état de session ASP.NET sont définies et récupérées facilement à l'aide de la propriété Session qui stocke les valeurs de variables de session comme une collection indexée par nom.

  Par exemple, l'exemple de code suivant crée les variables de session ***FirstName*** et ***LastName*** pour représenter le prénom et le nom d'un utilisateur et les définit sur les valeurs récupérées des contrôles TextBox.

```
Session["FirstName"] = FirstNameTextBox.Text

Session["LastName" ] = LastNameTextBox.Text
```

ASP.NET stocke par défaut les informations sur la session dans l'espace mémoire de l'application ASP.NET. Vous pouvez stocker éventuellement les informations sur la session à l'aide d'un service autonome afin qu'elles soient conservées, en cas de redémarrage de l'application ASP.NET, dans un serveur SQL Server

Sessions et Application

- **Comment changer l'endroit de stockage des sessions ?**

  Vous pouvez spécifier le mode que l'état de session ASP.NET doit utiliser en assignant une valeur d'énumération **SessionStateMode** à l'attribut **mode** de l'élément sessionState dans le fichier ***Web.config*** de votre application.

➔ **Mode InProc**

Il s'agit du mode par défaut : les variables de sessions sont stockées en mémoire sur le serveur web

➔ **Mode Sql server**

- Le mode **SQLServer** stocke l'état de session dans une base de données SQL Server.
- L'utilisation de ce mode garantit que l'état de session est conservé en cas de redémarrage de l'application Web et qu'il est disponible pour plusieurs serveurs Web dans une batterie de serveurs Web.

```
<configuration>
        <system.web>
                <sessionState mode="SQLServer" sqlConnectionString="Integrated Security=SSPI;data source=SampleSqlServer;" />
        </system.web>
</configuration>
```

Sessions et Application

Pour utiliser le mode SQLServer, vous devez d'abord être sûr que la base de données d'état de session ASP.NET est installée sur SQL Server.

> **Remarque :**
> Vous pouvez installer la base de données d'état de session ASP.NET à l'aide de l'outil Aspnet_regsql.exe

Pour configurer une application ASP.NET afin qu'elle utilise le mode SQLServer, procédez comme suit dans le fichier Web.config de l'application :

- Définissez l'attribut mode de l'élément **sessionState** avec la valeur SQLServer.
- Définissez l'attribut sqlConnectionString avec une chaîne de connexion pour votre base de données SQL Server.

Plus d'info : https://support.microsoft.com/en-us/help/317604/how-to-configure-sql-server-to-store-asp-net-session-state

Sessions et Application

- **Installation de la base de données d'état de session à l'aide de l'outil Aspnet_regsql.exe**

Pour installer la base de données d'état de session sur SQL Server, exécutez l'outil

Aspnet_regsql.exe présent dans le dossier ***systemroot*\Microsoft.NET\Framework*numéroversion*** de votre serveur Web.

Fournissez les informations suivantes avec la commande :

- Nom de l'instance du serveur SQL Server à l'aide de l'option -S.
- Informations d'identification de connexion pour un compte ayant l'autorisation de créer une base de données sur SQL Server.
  Utilisez l'option -E pour utiliser l'utilisateur actuellement connecté, ou l'option -U pour spécifier un ID utilisateur avec l'option -P permettant de spécifier un mot de passe.
- Option de ligne de commande -ssadd pour ajouter la base de données d'état de session.
- Par défaut, vous ne pouvez pas utiliser l'outil Aspnet_regsql.exe pour installer la base de données d'état de session sur SQL Server Express. Pour exécuter l'outil Aspnet_regsql.exe et installer une base de données SQL Server Express, vous devez d'abord activer l'option SQL Server Agent XPs à l'aide des commandes T-SQL, comme illustré ci-après :

Sessions et Application

- **<u>Bonne pratique pour l'utilisation des sessions</u>**

  Pour placer un objet dans une Session d'un utilisateur, nous procédons ainsi :

```
Session["NomUtilisateur"] = "Richard"
```

  - Il y a plusieurs inconvénients :
    - Si la variable NomUtilisateur est utilisé dans plusieurs pages, il faut bien savoir qu'elle s'appele NomUtilisateur, il ne faut pas faire de fautes de frappes et surtout être sur que cette variable n'existe pas déjà dans une autre page avec une utilisation complètement différente. De plus, si vous voulez changer le nom de la variable en LeNomDelUtilisateur, vous devez revoir tout votre code de toutes vos pages,
    - La variable est non typé (c'est un Object dans Session) donc quand vous voulez y accéder, vous êtes obligés d'effectuer des opérations de casting :

```
String sM = (String)Session["NomUtilisateur“]
```

Qui dit casting dit possibilité d'erreur de casting.

ASP.NET: les sessions – Bonne pratique

Donc, l'idée est de créer une classe tampon qui accédera aux variables sessions

```
public static class UserSession
  {
     public static  string NomUtilisateur
     {
        get{return HttpContext.Current.Session["NomUtilisateur"];}
        set{HttpContext.Current.Session["NomUtilisateur"] = value;}
     }
  }
```

Dans nos pages nous pouvons désormais écrire :

```
String sM  = UserSession.NomUtilisateur
```

Pas de casting, un seul endroit pour gérer le nom des variables sessions, etc.

AuthorizeAttribute : Attributs personnalisés

Maintenant que nous savons mettre en place une session, il serait bien de protéger les accès à certaines pages tant que l'utilisateur n'est pas connecté à une session valide!

Grâce à la class AuthorizeAttribute, nous aurons la possibilité de redéfinir via un Attribut quelles méthodes, voir controllers nous limiterons les actions utilisateurs.

C'est plus précisément deux méthodes de cet attribut qu'il nous faudra redéfinir par un override de celle-ci:

- **AuthorizeCore** →

  Nous permet d'établir une vérification et ainsi définir si nous sommes autorisé ou non à accéder au bloc de code soumis à l'attribut; Il nous retourne donc un booléen, mais demande un paramètre HttpContextBase (ce qui tombe bien, notre controller en contient justement un en propriété).

- **HandleUnauthorizedRequest** → Définit une action en cas de refus. Ne retourne rien, mais demande un paramètre AuthorizationContext.

AuthorizeAttribute : Attributs personnalisés

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method)]
// nous permet de définir que notre attribut pourra affecter des classes et des méthodes

    public class AuthRequiredAttribute : AuthorizeAttribute

    {

        protected override bool AuthorizeCore(HttpContextBase httpContext)
            //nous vérifions si le UserSession est différent de null

        {

            return UserSession.CurrentUser != null;

        }


        protected override void HandleUnauthorizedRequest(AuthorizationContext filterContext)
            //Redirige vers une autre action si non-autorisé

        {

            filterContext.Result = new RedirectToRouteResult(new RouteValueDictionary(new { Controller = "Auth", Action = "Login" }));

        }

    }
```