1) Жизненный цикл ASP.NET страницы // Веб-службы Web API

Жизненный цикл ASP.NET страницы

В жизненном цикле страницы можно выделить три главных этапа: подготовительный, этап обработки информации, связанной с возвратом формы, и завершающий. Каждый из них делится на несколько меньших шагов, на которых генерируются события.

1. Подготовка страницы к выполнению

Когда исполняющая среда HTTP создает экземпляр страницы для обслуживания запроса, конструктор страницы формирует дерево её элементов управления. Подготавливаются и инициализируются все дочерние элементы управления и внутренние объекты страницы, такие как объекты запроса и ответа.

В процессе обработки запроса первым делом страница выясняет причину своего запуска. Этой причиной может быть получение обычного запроса, возврат формы, межстраничный возврат формы или обратный вызов сценария.

Объект страницы конфигурирует своё внутреннее состояние с учетом этой причины и подготавливает коллекцию параметров запроса. После выполнения этого первого шага страница готова генерировать события для пользовательского кода.

1.1. Событие `PreInit`

Это событие является точкой входа жизненного цикла страницы. Когда оно генерируется, ни эталонная страница, ни тема ещё не связаны с текущей страницей. Однако позиция прокрутки страницы уже восстановлена, доступны возвращенные клиентом данные, экземпляры всех элементов управления страницы созданы и инициированы значениями свойств по умолчанию, которые определяются в исходном файле `.aspx`. Это единственный момент, когда можно программным способом задать для текущей страницы эталонную страницу и тему. Данное событие доступно только для страницы. Свойства `IsCallback`, `IsCrossPagePostback` и `IsPostback` к этому моменту уже установлены.

1.2. Событие `Init`

Теперь эталонная страница и тема уже связаны с текущей страницей, и изменить их невозможно. Метод `ProcessRequest()` класса `Page` перебирает в цикле все дочерние элементы управления, давая каждому из них шанс инициализировать свое состояние с учетом текущего контекста. У каждого дочернего элемента управления есть метод `OnInit()`, который и вызывается. Каждому дочернему элементу управления присваивается контейнер имен и идентификатор, если они не указаны в файле `.aspx`.

Событие `Init` вначале достигает дочерних элементов управления, а уж потом самой страницы. На этом этапе страница и ее элементы управления обычно начинают загрузку определенных составляющих своего состояния.

1.3. Событие `InitComplete`

Данное событие сигнализирует об окончании стадии инициализации. Между событиями `Init` и `InitComplete` страница производит только одну операцию: включает режим отслеживания изменений в состоянии представления. Эта функция

обеспечивает элементам управления возможность сохранить те значения, которые программно добавляются в коллекцию ViewState. Если для какого-то из элементов управления данная функция отключена, любые добавленные им в коллекцию ViewState значения между возвратами формы утрачиваются.

1.4. Восстановление состояния представления

Если страница обрабатывается по причине возврата формы (её свойство IsPostBack содержит значение true), восстанавливается содержимое скрытого поля __VIEWSTATE клиентской страницы. В этом поле хранится состояние представления всех элементов управления, записанное туда в конце выполнения предыдущего запроса. На данном этапе каждый элемент управления получает возможность обновить свое текущее состояние, сделав его таким, каким оно было во время выполнения предыдущего запроса. События не генерируются.

1.5. Обработка данных, принятых в результате возврата формы

На этом этапе обрабатываются все клиентские данные, содержащиеся в запросе HTTP, то есть содержимое всех полей ввода, определенных в тэге <form>. Обычно данные пересылаются в такой форме:

```
TextBox1=text&DropDownList1=selectedItem&Button1=Submit
```

Это разделенная символами & строка пар «*имя=значение*», которые загружаются в специальную коллекцию, предназначенную для внутреннего использования. Процессор страницы ищет соответствие между именами из полученной коллекции и идентификаторами элементов управления страницы. Найдя его, он проверяет, реализует ли соответствующий серверный элемент управления интерфейс IPostBackDataHandler. Если да, вызываются методы этого интерфейса, чтобы дать элементу управления возможность обновить свое состояние с использованием полученных данных. В частности, процессор страницы вызывает метод LoadPostData().

1.6. Событие PreLoad

Данное событие указывает, что страница завершила этап инициализации системного уровня и переходит к тому этапу, на котором пользовательский код страницы имеет возможность сконфигурировать ее для дальнейшего выполнения и рендеринга. Это событие генерируется только для страниц.

1.7. Событие Load

Это событие генерируется сначала для страницы, а потом рекурсивно для всех её дочерних элементов управления. Страница готова к выполнению инициализационного кода, связанного с её логикой и поведением. На этом этапе доступ к свойствам элемента управления и состоянию представления можно осуществлять без каких-либо опасений.

1.8. Обработка динамически созданных элементов управления

Когда все элементы управления страницы получили возможность завершить перед отображением свою инициализацию, процессор страницы предпринимает вторую

попытку идентифицировать полученные от клиента значения, для которых не нашлось соответствий среди существующих элементов управления. Процессор рассчитывает на то, что недостающие элементы управления могли быть созданы динамически.

Динамическое добавление элемента управления в дерево страницы может осуществляться, например, в ответ на определенное действие пользователя.

2. Обработка возврата формы

Механизм возврата формы является главной движущей силой любого приложения ASP.NET. Суть операции возврата формы заключается в том, что данные формы клиентской страницы передаются серверной странице - той самой, которая эту клиентскую страницу сгенерировала, и серверная страница восстанавливает контекст вызова, используя сохраненное ранее состояние представления и текущие данные формы.

После того как страница выполнила инициализацию и обработала полученные от клиента значения, приходит время двух групп серверных событий: события первой группы сигнализируют об изменении состояния определенных серверных элементов управления, а события второй группы генерируются в ответ на действие клиента, вызвавшее возврат формы.

2.1. Обнаружение изменений в состоянии элементов управления

Система ASP.NET действует, исходя из предположения о наличии взаимно-однозначного соответствия между HTML-тэгами ввода, используемыми в браузере, и элементами управления ASP.NET, функционирующими на сервере. Примером может служить соответствие между тэгом `<input type="text">` и элементом управления `TextBox`. Технически связь между этими двумя элементами устанавливается посредством их идентификаторов, которые должны быть одинаковыми.

Для всех элементов управления, вернувших из метода `LoadPostData()` значение `true`, пришло время выполнить второй метод интерфейса `IPostBackDataHandler` – `RaisePostDataChangedEvent()`. Его вызов сигнализирует элементу управления, что пора уведомить приложение ASP.NET об изменении своего состояния. Большинство элементов делают в нем одно и то же: генерируют серверное событие и предоставляют разработчику страницы возможность включиться в игру и выполнить код, обрабатывающий данное событие. Например, если после возврата формы содержимое свойства `Text` элемента управления `TextBox` оказалось измененным, элемент управления `TextBox` генерирует событие `TextChanged`.

2.2. Обработка серверного события возврата формы

Операция возврата формы начинается с того, что на клиенте осуществляется некоторое действие, требующее реакции сервера. Например, пользователь щёлкает кнопку, предназначенную для отправки содержимого формы серверу. Такая клиентская кнопка, обычно реализованная как гиперссылка или кнопка `submit`-типа, связана с серверным элементом управления, реализующим интерфейс `IPostBackEventHandler`. Процессор страницы просматривает полученные от клиента

данные и определяет, какой элемент управления инициировал возврат формы. Если этот элемент реализует интерфейс `IPostBackEventHandler`, процессор вызывает его метод `RaisePostBackEvent()`.

2.3. Событие `LoadComplete`

Поддерживаемое только для страниц, событие `LoadComplete` сигнализирует об окончании этапа подготовки страницы. Обратите внимание, что дочерние элементы управления этого события не получают. Сгенерировав событие `LoadComplete`, страница вступает в фазу рендеринга.

3. Завершающий этап выполнения страницы

После обработки события возврата формы страница готова сгенерировать вывод для браузера. Этап рендеринга делится на две стадии: предрендеринг и генерирование разметки. Стадия предрендеринга также делится на две части, которым соответствуют события предобработки и постобработки.

3.1. Событие `PreRender`

Обрабатывая событие `PreRender`, страница и элементы управления могут выполнять изменения, которые необходимо внести до того, как начнется рендеринг страницы. Данное событие сначала достигает страницы, а затем рекурсивно всех ее элементов управления.

3.2. Событие `PreRenderComplete`

Поскольку событие `PreRender` рекурсивно генерируется для всех дочерних элементов управления страницы, ее автор не знает, когда завершится фаза предрендеринга. Поэтому введено событие, `PreRenderComplete`, генерируемое только для страницы и уведомляющее об этом моменте.

3.3. Событие `SaveStateComplete`

Следующим шагом является сохранение текущего состояния клиентской страницы. Сохранение состояния страницы - рекурсивный процесс, при выполнении которого процессор страницы проходит по всему ее дереву, вызывая метод `SaveViewState()`элементов управления и самой страницы. `SaveViewState()` - это защищенный и виртуальный метод, отвечающий за сохранение содержимого словаря `ViewState` текущего элемента управления. Событие `SaveStateComplete` генерируется после полного сохранения состояния элементов управления страницы.

3.4. Генерирование разметки

Генерирование разметки для браузера осуществляется путем вызова каждого элемента управления страницы, с тем, чтобы он сгенерировал собственную разметку и вывел ее в буфер, где накапливается код формируемой клиентской страницы. Несколько переопределяемых методов позволяют разработчику вмешиваться в процесс на разных этапах генерирования разметки: когда выводится начальный тэг, тело и конечный тэг.

3.5. Событие `Unload`

По окончании этапа рендеринга для каждого элемента управления, а затем и для самой страницы генерируется событие `Unload`. Это событие позволяет элементам выполнить перед освобождением объекта страницы заключительные операции, такие как закрытие файлов и подключений к базам данных. Для освобождения памяти, занимаемой объектом страницы, её процессор вызывает метод `Dispose()`. Это происходит сразу после того, как завершится выполнение всех рекурсивно вызванных обработчиков события `Unload`.

Веб-службы Web API

Веб-службы (или веб-сервисы) – это один из способов создания распределённых приложений, то есть приложений, компоненты которых размещены на различных компьютерах и взаимодействуют через локальную или глобальную сеть. Протоколы веб-служб стандартизированы, а, значит, не зависят от языка и платформы реализации.

Начнём с некоторых терминов, используемых при работе с веб-службами:

1. SOAP – протокол для взаимодействия с веб-службой, основанный на XML. Сообщение SOAP - это обычный XML-документ особой схемы.
2. WSDL (Web Services Description Language) - язык описания веб-служб, основанный на XML. WSDL-документ используется клиентами службы, чтобы узнать, какие методы предлагает служба, каков тип и число параметров этих методов.
3. DISCO (сокр. Discovery) и UDDI (Universal Description, Discovery, and Integration) - стандарты для поиска и публикации веб-служб. При использовании DISCO на сайте размещается файл, который содержит список имеющихся веб-служб. UDDI - это централизованный каталог, в котором веб-службы опубликованы по именам компаний.

В платформе .NET веб-службы – часть инфраструктуры ASP.NET, или, иначе говоря, веб-службы являются элементами веб-приложений. Для каждой веб-службы необходимо создать класс с методами службы и связанную с классом точку входа – файл с расширением asmx. Каждый asmx-файл начинается с директивы @WebService, которая должна объявить серверный язык и класс веб-службы. Директива @WebService может задавать и другую информацию, например, имя файла Code Behind. Приведём пример файла EmployeeService.asmx:

```
<%@ WebService Language="C#" Class="EmployeeService"
        CodeBehind="~/App_Code/EmployeeService.cs" %>
```

Класс веб-службы не содержит полей, то есть, не поддерживает состояние. Дело в том, что объекты классов веб-служб, как и страницы ASP.NET, создаются для каждого вызова клиента. Методы класса веб-службы, доступные клиенту, помечаются атрибутом [WebMethod].

Чтобы использовать веб-службу в коде клиента, необходим специальный прокси-класс. Прокси-класс обладает методами, сигнатура которых идентична методам класса веб-службы. Методы прокси-класса выполняют сетевое соединение с сервером веб-

службы, формирование правильных SOAP-пакетов для передачи серверу, интерпретацию данных, принятых от веб-службы.

Существуют два основных способа создания прокси-класса:

- Применение утилиты командной строки wsdl.exe.
- Использование сервисных ссылок в Visual Studio.

Для генерации прокси-класса с применением wsdl.exe требуется указать адрес службы в качестве параметра этой утилиты:

```
wsdl http://localhost:49253/WebSite/EmployeeService.asmx
```

Класс веб-службы может наследоваться от класса System.WebServices.WebService. Наследование позволяет получить доступ к встроенным объектам ASP.NET - Application, Context, Server, Session, Site, User - посредством соответствующих свойств. К классу веб-службы можно применить атрибут [WebService]. Свойства этого атрибута позволяют указать описание и имя службы, а также пространство имен XML, используемое службой.

Web API –функция платформы ASP.NET, позволяющую легко и быстро создавать веб-службы, которые предоставляют API для HTTP-клиентов. Web API является частью ядра платформы ASP.NET и может использоваться в других типах(не только MVC) веб-приложений или работать как автономный движок веб-служб.

Функциональность Web API реализуют контроллеры. Но определение контроллера Web API отличается от обычного контроллера. Во-первых, он образован от класса ApiController, который не связан с базовым классом обычных контроллеров - Controller

Во-вторых, контроллеры Web API применяют стиль REST (Representation State Transfer или "передача состояния представления").

Для взаимодействия с сервером в REST-архитектуре используются методы HTTP:

- GET
- POST
- PUT
- DELETE

У нас тут нет обычных методов действий, как в традиционных контроллерах, которые возвращают ActionResult. При создании методов контроллера Web API действует некоторые условности. Имена методов по умолчанию должны начинаться с имени, предназначенного для них метода HTTP. В случае с контроллером по умолчанию все просто: все методы действий носят названия методов HTTP.

Однако мы можем использовать и любые другие имена без префиксов, но в этом случае нам надо будет явно указать метод HTTP в виде атрибута, например:

```
[HttpPost]
public void CreateItem([FromBody]string value)  {  }
```

Поскольку в Web API методы контроллера не являются прямыми ресурсами и сопоставляются с методами HTTP, то и весь механизм маршрутизации действует не как при

определении обычных маршрутов. Все определения маршрутов для Web API находятся в файле *WebApiConfig.cs* (в папке *App_Start*):

```
public static class WebApiConfig
{
   public static void Register(HttpConfiguration config)
   {
      config.Routes.MapHttpRoute(
         name: "DefaultApi",
         routeTemplate: "api/{controller}/{id}",
         defaults: new { id = RouteParameter.Optional }
      );
   }
}
```

В данном случае определен один маршрут, где в качестве второго параметра выступает контроллер, а третий необязательный параметр представляет некоторый идентификатор. Таким образом, в отличие от маршрутов обычных контроллеров у нас здесь нет действия, только контроллер и дополнительный необязательный параметр.

В итоге обращение *api/values* будет соответствовать обращению к контроллеру ValuesCotroller, причем почти ко всем действиям без параметров.

Вызов методов API-контроллера осуществляется в представлении с помощью JS-скрипта, а именно ajax-запроса по определенному URL (например: *http://localhost:2512/api/values/*).