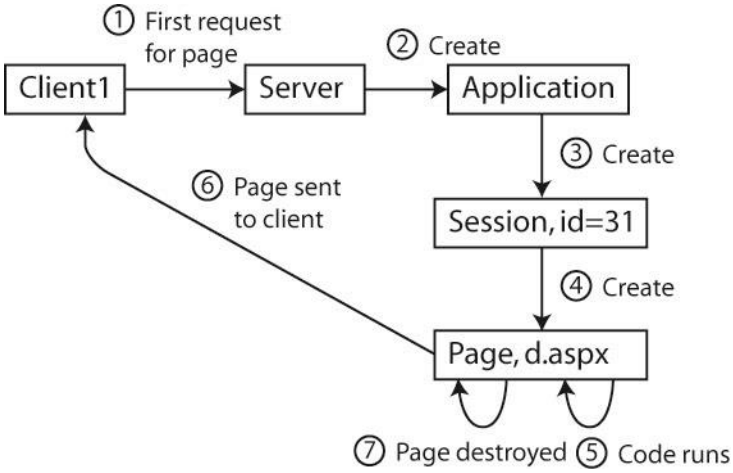
Page, Session, & Application Lifecycles

A brief discussion of Application, Session, and Page lifecycles.

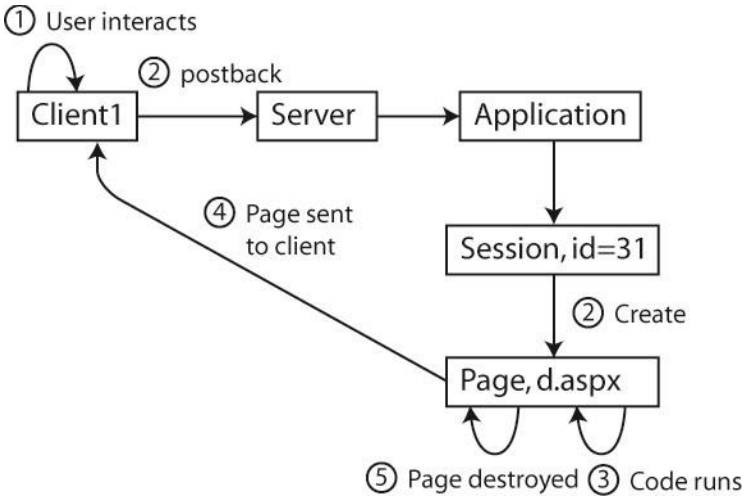
Application & Session Lifecycle

1. Example of Application and Session lifecycle.

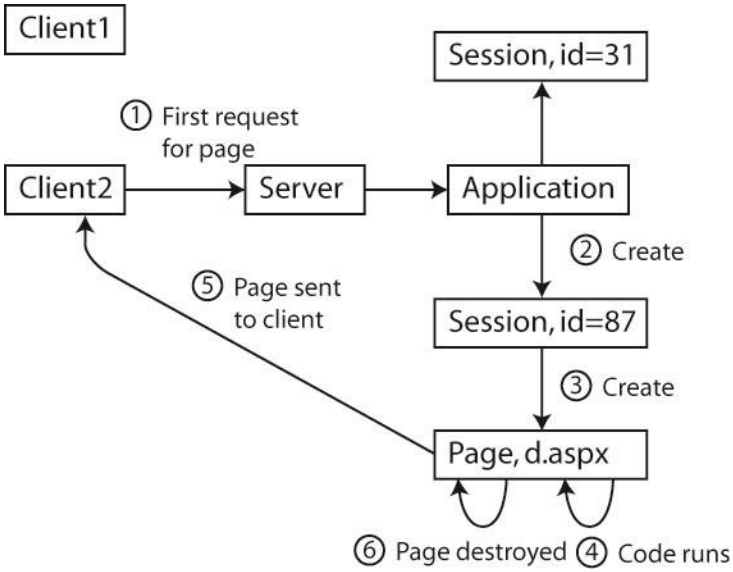
   a. Suppose a web application is loaded onto a server. The very first request for a page in this application begins the Application lifecycle as well as the Session lifecycle for that particular client



   b. Client1 interacts with the page and postsback. The application and session objects already exist.



c. Client2 makes its first request for a page in the application. The application already exists. The new session object is created.



2. A *web app* is only *alive* (accessible) when it is loaded into memory by the web server. At some later point a web app may be removed from memory. Thus, we say that an application has a *lifetime* and goes through a series of stages, an *application lifecycle*[1].

3. Suppose we put a web app in a web folder on a web server. The very first request for a page in the app creates an *Application* object on the server. The developer of the app can write code to store things in the Application object. For example: number of people logged in, number of guests, number of hits on main page*, etc*.

4. This first request also creates a *Session* object and the *Application* object maintains a link to it. A *Session* represents a period of time that a user is using an application. For instance, as long as the client continues to interact with a page (posting-back), generally, the session will remain. When they stop interacting for some time, the session will *timeout* and end. Other times, the client might explicitly log-out, thus ending the session.

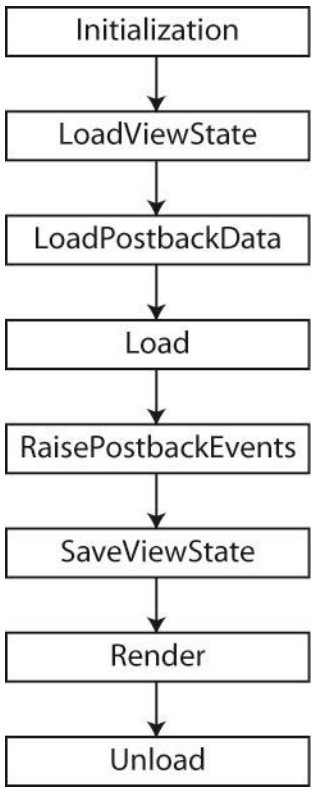
5. Suppose another user requests a page from the web app. The Application object creates another Session, one for this particular client. Each Session has a unique *Session ID* which is sent to the browser and returned when posting. Thus, when requests come into the server, the server's *ApplicationManager* can direct it to the correct Session. A *Session* stays alive until it is programmatically terminated, or it *times-out*.

6. An a*pplication* stays alive until it is removed from memory programmatically or the server is turned off. If an application has been removed from memory and a subsequent request comes in, this process is repeated, starting from step 2 above.

7. A *Session* has an *HttpRequest* object that handles requests from a client. When a request is received a processing pipeline is excecuted. 26 methods are called with the culmination being an HTTP Response is sent to the client.[2] All of these methods fire events which we can handle if we want. The 15th step (method), *ProcessRequest* is the one we are particularly interested in. It creates the new page and executes the code. This process is called the *Page Lifecycle* and is described below.

[1] http://msdn.microsoft.com/en-us/library/ie/bb470252.aspx
[2] http://msdn.microsoft.com/en-us/library/ie/bb470252.aspx

ASP.NET Page Lifecycle



1. When a page is requested the server goes through a number of steps that relate to this page. This is referred to as the *page lifecycle*. The figure on the right is a summary of the *page lifecycle*[3]:

2. A description of these general steps follows[4]:

    1. **Initialization** – The controls are created.
    2. **LoadViewState** – The *ViewState* is read and set into the server controls.
    3. **LoadPostBackData** – The data that has been posted back is loaded into the server controls.
    4. **Load** – The *Page_Load* method is executed. Next, recursively, all controls on the page have their *Load* event fired and go through a *control lifecycle* similar to the page lifecycle.
    5. **RaisePostBackEvent** – The *control events* defined by the programmer are executed.
    6. **SaveViewState** – After all the code is run, the ViewState is saved in the page as a hidden HTML field.
    7. **Render** – Next, the page is rendered. Each control has its *Render* method called which outputs the appropriate HTML to display itself.
    8. **Unload** – Finally, the page is unloaded. The page object is destroyed and any variables that were defined in the page are destroyed.

3. At each of the steps above a number of methods can be overridden[5] [6]: PreInit, Init, InitComplete, PreLoad, Load, LoadComplete, PreRender, PreRenderComplete, SaveStateComplete, Render, Unload.

    https://msdn.microsoft.com/en-us/library/ms178472.aspx

[3] http://www.4guysfromrolla.com/articles/050504-1.aspx
[4] http://msdn.microsoft.com/en-us/library/ms178472.aspx
[5] https://msdn.microsoft.com/en-us/library/ms178472(v=vs.100).aspx#lifecycle_events
[6] http://www.c-sharpcorner.com/uploadfile/61b832/Asp-Net-page-life-cycle-events/