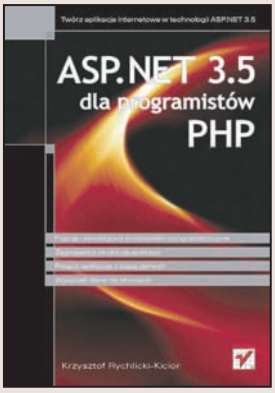
ASP.NET 3.5 dla programistów PHP

Autor: Krzysztof Rychlicki-Kicior
ISBN: 978-83-246-1916-0
Format: 158x235, stron: 160



Stwórz aplikacje internetowe w technologii ASP.NET 3.5

- Poznaj i skonfiguruj środowisko programistyczne
- Zaprojektuj strukturę aplikacji
- Połącz aplikację z bazą danych
- Wyświetl dane na stronach

Technologie PHP i ASP.NET mają kilka cech wspólnych – obie służą do tworzenia aplikacji internetowych, uruchamianych za pośrednictwem przeglądarki internetowej, i generują wynik swojego działania w postaci dokumentów XHTML. Jednak istnieje między nimi jedna istotna różnica – PHP wykorzystuje rozwiązania open source, a ASP.NET rozwiązania komercyjne, rozwijane przez firmę Microsoft. ASP.NET opiera się na platformie .NET – uniwersalnym frameworku, pozwalającym na korzystanie z różnych języków programowania w celu tworzenia skalowalnych, szybkich i łatwych do rozwijania aplikacji sieciowych. Nie bez znaczenia jest także środowisko programistyczne – Visual Studio to doskonałe narzędzie, znacznie przyspieszające i usprawniające proces tworzenia oprogramowania.

„ASP.NET 3.5 dla programistów PHP" to wprowadzenie do technologii ASP dla wszystkich twórców aplikacji internetowych, którzy dotychczas korzystali z PHP. Czytając tę książkę, zrozumiesz główne zasady korzystania z platformy .NET i techniki budowania aplikacji sieciowych. Dowiesz się, jak skonfigurować środowisko pracy i zaprojektować strukturę aplikacji. Nauczysz się korzystać z najważniejszych elementów platformy .NET – kontrolek będących gotowymi „cegiełkami" do budowy stron internetowych ASP. Poznasz podstawowe kontrolki, tworzące interfejs użytkownika i zarządzające sesją, połączysz aplikację z bazą danych i zastosujesz kontrolki do wyświetlenia danych na stronie. Przeczytasz o sprawdzaniu poprawności danych wprowadzanych do formularzy i przypisywaniu ról użytkownikom aplikacji. Znajdziesz tu także informacje o tym, w jaki sposób wykorzystać technologię AJAX do budowy interfejsów użytkownika aplikacji ASP.NET.

- Porównanie technologii służących do tworzenia aplikacji sieciowych
- Konfiguracja bazy MS SQL i instalacja narzędzi programistycznych
- Kontrolki interfejsu użytkownika
- Zarządzanie sesją i plikami cookie
- Komunikacja z bazami danych za pomocą ADO.NET
- Wyświetlanie danych na stronach
- Weryfikacja poprawności danych z formularzy
- Obsługa kont użytkowników
- AJAX w aplikacjach ASP.NET

Spróbuj czegoś nowego. Może w ASP.NET 3.5 odnajdziesz to, czego brakowało Ci w PHP?

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Rozdział 5.

Obsługa strony

Strony internetowe to nie tylko kontrolki graficzne. Zanim ASP.NET zajmie się obsługą kontrolek, musi zostać wysłane przez klienta żądanie, a po wszystkim powinna zostać odesłana do niego odpowiedź. Ponadto często (nawet w poprzednim rozdziale) konieczne jest uzyskanie pewnych informacji o serwerze lub samej aplikacji niemożliwych do odczytania z poziomu kontrolek. Większość tych informacji można pobrać za pośrednictwem klasy `Page` i jej właściwości.

Wszystkie drogi prowadzą do strony — Page

Z klasą `Page` mieliśmy do tej pory kontakt wielokrotny, gdyż wszystkie tworzone przez nas proste aplikacje złożone z jednej strony (formularza) były w istocie klasami wywiedzionymi bezpośrednio z klasy `Page`. Co za tym idzie, wiele elementów wykorzystywanych uprzednio bezpośrednio (jak właściwości `Response` czy `Server`) stanowiło właściwości klasy `Page`.

Można powiedzieć, że klasa `Page` sama w sobie nie oferuje rozbudowanej funkcjonalności; głównym jej zadaniem jest zebranie w jedną całość. Właściwości tej klasy można podzielić na dwie nieformalne grupy — zwykłe, pomocne w określeniu działania strony, i rozbudowane, których działanie nie jest blisko związane z samą klasą `Page`. W pewnym sensie klasa `Page` stanowi jedynie zbiór odwołań do kluczowych obiektów, dzięki czemu nie trzeba ich szukać w innych miejscach. Oczywiście wszystkie te obiekty nie biorą się znikąd — instancje znajdujące się we właściwościach danej strony są powiązane właśnie z nią. Do „dużych" właściwości, które będą tematem osobnych podrozdziałów i rozdziałów, zaliczają się:

- `HttpApplicationState Application` — określa obiekt klasy, której zadaniem jest wymiana informacji między różnymi żądaniami i stronami, czyli zarządzanie stanem aplikacji.
- `HttpRequest Request` — zwraca obiekt aktualnie obsługiwanego żądania.

- `HttpResponse Response` — określa obiekt aktualnie generowanej odpowiedzi na żądanie.
- `HttpServerUtility Server` — określa obiekt pomocny przy obsługiwaniu żądań, udostępniając pewne mechanizmy serwera — w przeciwieństwie do właściwości `Request`, której zadaniem nie jest interakcja ze środowiskiem serwera.
- `HttpSessionState Session` — określa obiekt sesji, w którym można przechowywać wartości między kolejnymi żądaniami HTTP.
- `IPrincipal User` — określa dane użytkownika, który wysłał żądanie (przydatne zwłaszcza po wprowadzeniu do aplikacji mechanizmu autentyfikacji — uwierzytelnienia).

Jak widać, klasa `Page` gromadzi niemal wszystkie istotne elementy przydatne w trakcie projektowania aplikacji WWW. Oczywiście można wykorzystywać również inne właściwości; z niektórymi już mieliśmy kontakt (np. `IsPostBack`), niektóre poznamy przy okazji konkretnych rozwiązań zamieszczonych w dalszej części książki.

Z pewnością w tym miejscu warto zapoznać się z modelem zdarzeń klasy `Page`. Opiera się on rzecz jasna na modelu zdarzeń klasy `Control`. Wytłuszczonym drukiem zostały zaznaczone zdarzenia wprowadzone dopiero w klasie `Page`:

- **`PreInit`** — na najwcześniejszym możliwym etapie ładowania strony można określić, czy strona jest ładowana w wyniku żądania zwrotnego (postback). Należy jednak pamiętać, że kontrolki nie są jeszcze w pełni załadowane. Z tego względu ustawianie ich wartości na tym etapie nie gwarantuje ich zachowania później.
- `Init` — w tym momencie kontrolki są zainicjalizowane (strona może wywołać zdarzenie dopiero wtedy, gdy wszystkie kontrolki zgłosiły to zdarzenie). Można więc dowolnie korzystać z ich właściwości.
- **`InitComplete`** — zgłaszane, gdy cały proces inicjalizacji (włącznie ze stroną) został zakończony. Jeśli jakaś czynność wymaga do działania zakończenia inicjalizacji, powinna znaleźć się w tej metodzie zdarzenia.
- **`PreLoad`** — podczas tego zdarzenia jest ładowany stan widoku strony i kontrolek oraz dane przesłane w żądaniu zwrotnym.
- `Load` — ładuje stronę, a następnie rekursywnie wszystkie kontrolki na stronie. W tym momencie można utworzyć połączenia z bazą danych. Po zdarzeniu `Load` następują inne zdarzenia kontrolek, takie jak kliknięcie przycisku.
- **`LoadComplete`** — podobnie jak `InitComplete`, to zdarzenie zachodzi po zakończeniu ładowania wszystkich kontrolek.
- `PreRender` — zachodzi tuż przed renderowaniem, czyli generowaniem kodu HTML na podstawie kontrolek ASP.NET. Przed zajściem tego zdarzenia kontrolki są wiązane ze źródłami danych, więc to jest najlepszy moment na wykonanie ewentualnych zmian związanych z danymi w takich kontrolkach. Bezpośrednio po tym zdarzeniu zachodzi renderowanie, po którym nie można zmodyfikować już w żaden sposób kontrolek.

- `Unload` — ma miejsce po zajściu zdarzenia dla wszystkich innych kontrolek. Można wtedy zwolnić zasoby przydzielone w ramach całej strony.

Wszystkie te zdarzenia są częścią większego procesu, jakim jest realizacja żądania HTTP. Można powiedzieć, że proces ten składa się z trzech części:

1. Wysłanie żądania do serwera.
2. Analiza treści żądania i przetworzenie go (ten etap jest pomijany, w przypadku gdy wysyłane są statyczne strony HTML i nie są używane żadne technologie *server-side*) — powstanie odpowiedzi na podstawie plików znajdujących się na serwerze i danych zawartych w żądaniu.
3. Odesłanie odpowiedzi do klienta.

W przeciwieństwie do innych protokołów sieciowych, takich jak FTP, protokół HTTP jest protokołem bezstanowych. Oznacza to, że każde nowe żądanie wysłane przez klienta nie ma związku z poprzednim. Gdyby nie zastosowanie mechanizmów, takich jak sesje czy ciasteczka (ang. *cookies*), praktycznie nie byłoby możliwe korzystanie np. z poczty elektronicznej za pośrednictwem przeglądarki — wysyłając każde żądanie, trzeba byłoby za każdym razem podawać login i hasło — protokół sam w sobie nie umożliwia zapamiętania takich danych.

Etap nr 1 stanowi zadanie przeglądarki. Na podstawie decyzji podejmowanych przez użytkownika (wpisanie adresu URL, kliknięcie odnośnika, wysłanie formularza) generowane są odpowiednie żądania (ang. *request*) HTTP, które następnie są wysyłane do odpowiedniego serwera. Nasz kontakt z tym etapem sprowadza się do możliwości pobrania informacji na temat żądania — zajmuje się tym klasa `HttpRequest`, która za pośrednictwem właściwości `Request` jest obecna w klasie `Page`.

Etap nr 2 stanowi meritum niniejszej publikacji. Niezależnie od użytej technologii server-side schemat działania jest zawsze taki sam — serwer WWW sprawdza typ pliku (na podstawie rozszerzenia lub typu MIME), jaki jest zawarty w żądaniu. Na tej podstawie może po prostu odesłać zawartość pliku do klienta (jeśli jest to zwykły plik danych) lub poddać treść pliku działaniu którejś z dostępnych technologii *server-side*. Oczywiście w takiej sytuacji do modułu *server-side* musi przekazać także szereg innych danych — informacje przesłane przez klienta (np. z formularzy albo adresu URL), dane żądania, ciasteczka, etc. Gdy to wszystko zostanie razem połączone i przetworzone przez moduł *server-side*, w wyniku renderowania (używając terminologii ASP.NET, choć termin ten pasuje również do innych technologii *server-side*) powstaje końcowy dokument HTML.

Etap nr 3 polega na przesłaniu danych od serwera do klienta. W przypadku pominięcia etapu 2. przesyłane dane pochodzą po prostu z wczytanego pliku znajdującego się na dysku serwera. Gdy klient odbierze cały plik, połączenie jest zamykane, a operacja wykonana przez klienta i serwer przechodzi do historii. Co ważne, z poziomu ASP.NET możemy mieć wpływ również i na ten etap (np. przez ustawienie kodowania przesyłanego tekstu) za pośrednictwem właściwości `Response` klasy `Page` (analogicznie jak w etapie 1.).

Ponieważ drugi etap omawiamy w zasadzie w znamienitej większości tej książki, przyjrzymy się teraz pozostałym etapom procesu.

Request — tak wszystko się zaczyna...

Główną zaletą tej właściwości klasy `Page` jest możliwość odczytania wielu przydatnych informacji zawartych w żądaniu. Chociaż samo żądanie jest jedynie łańcuchem znaków (może, co prawda, zawierać pliki w przypadku użycia kontrolki `FileUpload`, ale nie wpływa to na samą formę), zawarta w nim ilość informacji z pewnością zadziwi każdego, kto nie zna dokładnej struktury żądania. Przeglądarka przesyła wraz z adresem URL więcej informacji, niż można by się spodziewać.

Zacznijmy jednak od tego, co najprostsze, czyli właśnie adresu URL. Wbrew pozorom nie jest on reprezentowany tylko przez jedną właściwość. Dostajemy aż trzy właściwości, które umożliwiają nam dostęp do wybranych elementów całego adresu:

- `string Url` — określa pełny adres URL żądania (zawierający adres serwera, ścieżkę do pliku oraz parametry dodatkowe).
- `string RawUrl` — określa adres URL bez nazwy serwera, czyli np. */katalog/plik.aspx?parametr=wartosc*.
- `NameValueCollection QueryString` — określa kolekcję parametrów przekazanych w żądaniu (po znaku zapytania).

Na uwagę zasługuje zwłaszcza ostatnia właściwość. Łańcuch zapytania pozwala na przekazywanie parametrów do strony WWW przez adres URL. W innych technologiach *server-side* jest to jeden z dwóch głównych sposobów na przekazywanie informacji między kolejnymi żądaniami. W przypadku ASP.NET przekazywanie danych odbywa się bez ingerencji programisty — po wysłaniu żądania wystarczy po prostu odczytać wybrane wartości tak, jak zostało to opisane w rozdziale 4.

Mimo to istnieje możliwość wykorzystania mechanizmów podstawowych, takich jak opisane wyżej przekazywanie parametrów przez adres URL, nazywane metodą `GET`, i przesyłanie danych przez formularz, jednak bez jawnej reprezentacji w ramach adresu, a jedynie w obrębie żądania, czyli metoda `POST`. Każda z tych metod ma swoje wady i zalety. Metoda `GET` umożliwia przechowanie danych w jednym łańcuchu znaków, ów dodatku widocznym w pasku przeglądarki.

Z drugiej strony, istnieje ograniczenie długości przesyłanych danych (z reguły do kilku kilobajtów). Tych ograniczeń nie posiada metoda `POST`. Należy jednak pamiętać, że o ile żądanie `GET` można swobodnie przechowywać i przetwarzać (np. przesyłać pocztą elektroniczną, umieszczać jako odnośniki na innych stronach WWW, etc.), to żądanie `POST` nie daje tej możliwości — raz wygenerowane żądanie przesłane do serwera nie może być wyświetlone w podobny sposób. Co za tym idzie, proste parametry są z reguły przesyłane za pośrednictwem metody `GET`, a bardziej skomplikowane — metodą `POST`.

Chociaż dane w ASP.NET są niejawnie (tj. w sposób, który programista nie ingeruje) przekazywane za pomocą widoku stanu, można korzystać również z metody `GET`. Wystarczy odpowiednio sformułować np. adresy zawarte w odnośnikach (kontrolka `HyperLink`):

```
Strona.aspx?parametr1=wartosc1&parametr2=innawartosc
```

W ten sposób mimo normalnego działania, zgodnego z ASP.NET, można odczytać wartości żądania GET. Wystarczy skorzystać z opisanej powyżej właściwości QueryString. Jest to typowa kolekcja-słownik, w której do żądanych elementów-wartości uzyskuje się dostęp przez podanie klucza-nazwy. Jeśli dla powyższego adresu chcemy pobrać wartość pierwszego parametru, wystarczy wywołać instrukcję:

```
string s = Request.QueryString["parametr1"];
```

Drugą grupą właściwości, niejako spokrewnionych z adresem URL, są fizyczne ścieżki dostępu, czyli ścieżki określające położenie katalogu strony bądź samej strony na dysku serwera.

- string PhysicalApplicationPath — określa ścieżkę do katalogu na dysku serwera, w którym znajduje się dana strona.
- string PhysicalPath — określa ścieżkę do danej strony na dysku serwera.

Można używać też właściwości bez słowa Physical w nazwie, które określają ścieżkę wirtualną, np. */WebSite* (ApplicationPath) lub */WebSite1/Strona.aspx* (Path). Taki zestaw właściwości (łącznie siedem) pozwala na swobodne korzystanie z wszelkiego rodzaju ścieżek.

Istotnym elementem każdego żądania HTTP są nagłówki. Są one reprezentowane przez właściwość NameValueCollection Headers. Jak widać, jest to kolejny słownik-kolekcja. Odpowiada to faktycznej strukturze nagłówków. Przykładowy nagłówek ma następującą postać:

```
Content-type: text/html
```

Przed dwukropkiem znajduje się klucz, a za dwukropkiem — wartość. Oczywiście zawartość nagłówków zależy od przeglądarki i konkretnego żądania — mogą one ulegać zmianie.

Ciasteczka

Inną istotną kolekcją są ciasteczka — specjalne pliki tekstowe o niedużym rozmiarze, zapisywane na komputerze użytkownika. Dzięki nim można zapisać informacje dotyczące na przykład danych logowania do poczty elektronicznej. Nie trzeba wtedy wprowadzać ich za każdym razem. Ciasteczka, w odróżnieniu od wykorzystywanego już typu NameValueCollection, mają nieco bardziej skomplikowaną strukturę, są więc reprezentowane za pomocą właściwości HttpCookieCollection. Klasa ciasteczek (jak można się domyślić — HttpCookie) składa się z kilku ważnych właściwości, które charakteryzują każde ciasteczko. Oto najważniejsze:

- string Name — określa nazwę ciasteczka.
- string Value — określa wartość ciasteczka.
- DateTime Expires — określa datę wygaśnięcia ciasteczka.

- `bool HasKeys` — określa, czy wartość ciasteczka jest pojedyncza, czy stanowi kolekcję.
- `NameValueCollection Values` — określa kolekcję wartości w przypadku, gdy stwierdzono jej istnienie.

Dwie pierwsze właściwości muszą być wypełnione, inaczej istnienie ciasteczka nie ma sensu. Pominięcie właściwości `Expires` skutkuje usunięciem ciasteczka z komputera użytkownika po zakończeniu sesji na danej stronie, czyli najczęściej po zamknięciu przeglądarki. Reszta parametrów jest opcjonalna i z reguły nie wymaga stosowania.

Oczywiście można również tworzyć ciasteczka. Jak większość podobnych, ale odwrotnych procesów (np. czytanie żądania HTTP — tworzenie odpowiedzi) tak i ten należy do odpowiedzi, czyli klasy `HttpResponse`.

Niewinne szpiegowanie

Żądania HTTP zawierają sporo informacji na temat użytkownika, a konkretnie jego przeglądarki czy komputera. W tym celu można wykorzystać właściwości zaczynające się w większości od słów `User`:

- `string UserAgent` — określa nazwę i informacje o przeglądarce.
- `string UserHostAddress` — określa adres IP komputera, z którego zostało nawiązane połączenie.
- `string[] UserLanguages` — określa preferowane języki użytkownika (ustawione w przeglądarce; nie wszystkie mają taką możliwość).
- `bool IsAuthenticated` — określa, czy użytkownik jest zalogowany (o ile mechanizm ten został zaimplementowany).
- `bool IsLocal` — określa, czy żądanie pochodzi z lokalnego komputera (tego samego, na którym działa serwer) — właściwość przydatna zwłaszcza do celów testowych, umożliwiająca rozgraniczenie testów, dokonywanych często z lokalnego komputera, i innych żądań.
- `bool IsSecure` — określa, czy nawiązywane połączenie jest bezpieczne (z użyciem protokołu SSL).

Na tym kończymy omawianie klasy `HttpRequest`; kilka ciekawych właściwości i metod, oprócz już poznanych, zostanie wykorzystanych przy okazji konkretnych przykładów.

Response — ...a tak się kończy

W tej klasie odnajdziemy wiele podobieństw łączących ją z klasą `HttpRequest`. Przede wszystkim niektóre procesy, działające niejako „w jedną stronę” w klasie `HttpRequest`, zostaną odwrócone w klasie `HttpResponse`. Na przykład, wspomniane ciasteczka mogły

być odczytywane za pośrednictwem klasy `HttpRequest`; teraz interesować nas będzie tworzenie ciasteczek i dodawanie ich do odpowiedzi — klasy `HttpResponse`.

Ogólnie rzecz ujmując, w zdecydowanej większości przypadków, jeśli trzeba samemu zająć się właściwością `Response` klasy `Page`, polega to na dodawaniu/tworzeniu różnych obiektów.

Zaczniemy przewrotnie od faktu pozwalającego na zlekceważenie aktualnie przetwarzanej strony. Dysponując właściwością Response, możemy pozbyć się treści HTML generowanej przez ASP.NET, a następnie wysłać do strumienia dowolną zawartość w bajtach. Zasada działania opiera się na kilku prostych elementach klasy `HttpResponse`:

- `string ContentType` — określa typ MIME odpowiedzi. W przeciwieństwie do klasy `Request`, gdzie właściwość ta pełniła rolę jedynie informacyjną, dzięki niej możemy określić typ odpowiedzi, np. na obrazek.
- `void Clear()` — czyści zawartość bufora z danych. Dzięki temu można pozbyć się danych, które już znalazły się w strumieniu odpowiedzi.
- `bool BufferOutput` — określa, czy odpowiedź zostanie wysłana dopiero po całkowitym przetworzeniu strony.
- `Stream OutputStream` — określa strumień, z którego dane zostaną wysłane do klienta.

Ponadto klasa `Response` definiuje właściwości `Cookies` i `Headers`, które zostały opisane przy okazji klasy `Request`. Tym razem można do nich dodawać własne ciasteczka bądź nagłówki — wszystko dla lepszej kontroli wysyłanych do użytkownika danych. Wystarczy wykorzystać do tego celu metodę `Add()`, która przyjmuje odpowiednio obiekty klas `HttpCookie` i parę danych: klucz-wartość.

Z ciekawszych elementów tej klasy należy wymienić na pewno dwie metody, które są często wykorzystywane w praktyce. Są to:

- `void Redirect(string url)` — przerywa wykonywanie strony i przekierowuje klienta pod podany adres. Uwaga! Należy podawać absolutne adresy URL wraz z protokołem (np. *http://www.helion.pl*), gdyż w przeciwnym razie serwer potraktuje adres URL jako relatywny.
- `void TransmitFile(string sciezka)` — przekazuje plik o podanym adresie (należy podawać relatywny URL) do klienta, pomijając w odpowiedzi żądaną stronę.

Zwłaszcza pierwsza z metod jest stosowana w wielu sytuacjach; najczęstszym przykładem jest przekierowanie przy logowaniu/wylogowaniu, o ile nie skorzystano z domyślnych mechanizmów. Poniżej znajduje się przykładowy kod, który umożliwia skierowanie do klienta obrazka, wyświetlając przekazany w polu tekstowym tekst:

Default.aspx

```
<asp:Label ID="Label1" runat="server" Text="Podaj tekst do wyswietlenia:"
➥></asp:Label><br />
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="OK" />
```

Default.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
 if (IsPostBack)
 {
  System.Drawing.Image img = System.Drawing.Image.
  ↪FromFile(Server.MapPath("Puch.jpg"));
  System.Drawing.Graphics g = System.Drawing.Graphics.FromImage(img);
  g.DrawString(TextBox1.Text, new System.Drawing.Font("Arial",20.0F),
  ↪System.Drawing.Brushes.White, new System.Drawing.PointF(10, 10));
  Response.ContentType = "image/bmp";
  Response.Clear();
  img.Save(Response.OutputStream, System.Drawing.Imaging.ImageFormat.Jpeg);
  g.Dispose();
  img.Dispose();
 }
}
```

Po kliknięciu przycisku (może być ono zastąpione utratą fokusa w połączeniu z właściwością `AutoPostBack`) dochodzi do otwarcia i edycji obrazka. Wyświetlamy na nim tekst wysłany przez użytkownika. Następnie wykorzystujemy poznane wcześniej właściwości i metody, aby przygotować strumień i zapisać do niego zawartość obrazka. Oczywiście powyższe rozwiązanie mogłoby zostać pominięte, gdyby chodziło jedynie o wyświetlenie obrazka — wystarczyłoby użyć poznanej metody `TransmitFile()`. Takie rozwiązanie jest jednak bardziej uniwersalne, pozwala na modyfikację danych, a nawet pobieranie ich z innych źródeł — wszystko, co można przekazać przez strumień, może być wykorzystane.

Nie tylko dla studentów — potyczki z sesjami

Sesja jest jednym z podstawowych pojęć, które spędza sen z powiek nie tylko studentom, ale i programistom aplikacji sieciowych. Uwzględniając to, że protokół HTTP nie umożliwia zachowania stanu (tak jak np. FTP), technologie *server-side* muszą umożliwiać przechowywanie danych pomiędzy poszczególnymi żądaniami. Jedną z metod są ciasteczka, ale użytkownik może wyłączyć ich obsługę w przeglądarce. Z tego względu powszechnie wykorzystywanym mechanizmem są sesje.

Sesja, w technologiach *server-side* najczęściej reprezentowana jako kolekcja-słownik, umożliwia zapis i odczyt wartości niczym zwykła kolekcja. Dzięki specjalnym mechanizmom słownik ów ma taki sam stan; przechowuje te same wartości pomiędzy różnymi żądaniami. Typowymi zastosowaniami sesji są przechowywanie obiektów kluczowych dla obsługi użytkownika, takich jak koszyk z zakupami w sklepie internetowym, dane logowania lub ostatnie operacje wykonane przez użytkownika. Niektóre z informacji, które w innych technologiach programista musi zapisywać w sesji sam, są automatycznie przechowywane przez ASP.NET.

Klasa `HttpSessionState`, udostępniająca funkcjonalność sesji, jest najzwyklejszą kolekcją. Z tego względu zapis i odczyt danych w kolekcji przebiega znacznie prościej niż w PHP. Wystarczy po prostu zapisać dane w kolekcji. Prosty i krótki przykład prezentuje dwie pułapki, na jakie muszą uważać programiści korzystający z mechanizmu sesji:

Default.aspx

```
<asp:Label ID="Label1" runat="server" Text="Podaj login:"></asp:Label><br />
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Rejestruj" />
<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="Default.aspx">
↪Przeładuj stronę od nowa</asp:HyperLink>
```

Default.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
  if (!IsPostBack && Session["login"] != null)
    TextBox1.Text = Session["login"].ToString();
}
protected void Button1_Click(object sender, EventArgs e)
{
  Session["login"] = TextBox1.Text;
}
```

Zadanie strony jest proste — jest to zwykła symulacja zapamiętania wartości pola tekstowego w sesji i odtwarzania go w razie potrzeby. Zapis do właściwości (przypominam — `Session` jest właściwością klasy `Page`, więc możemy się do niej odwoływać bez dodatkowych zapisów). Interesujący jest również odczyt. Na pewno musimy sprawdzić, czy wartość loginu istnieje w sesji. Ciekawsze jest jednak sprawdzanie postbacku — to warunek konieczny wynikający z kolejności wywoływania zdarzeń. Gdybyśmy pominęli tę część warunku, przypisanie wartości z sesji do pola tekstowego odbywałoby się za każdym razem po pierwszym określeniu loginu. Ze względu na kolejność wywoływania zdarzeń (`Page.Load` zachodzi przed innymi zdarzeniami kontrolek potomnych) każda kolejna próba zmiany wartości w sesji (metoda `Button1_Click`) nie mogłaby się powieść. Nowa wartość pola tekstowego byłaby zastępowana starą wartością z sesji. Dzięki dodatkowemu warunkowi wartość z sesji pobieramy tylko wtedy, gdy nie podjęto próby przypisania nowej wartości. Oczywiście przy większej liczbie kontrolek, które również mogłyby wygenerować postback, należałoby po prostu dokonać przypisania *po* ewentualnym wywołaniu metody `Button1_Click`.

Drugi haczyk tkwi w kontrolce, która pojawiła się po raz pierwszy — `HyperLink`. Została ona przedstawiona w prostym celu — aby zademonstrować, że sesja działa rzeczywiście. Można powiedzieć, że przycisk również powoduje przeładowanie strony. Problem w tym, że działają wtedy również mechanizmy widoku stanu i wartość pola tekstowego zostałaby zapamiętana bez względu na wykorzystanie sesji. Jeśli jednak przeładujemy stronę całkowicie, przy zakomentowanej treści metody `Page_Load`, wartość pola tekstowego nie zostanie zapamiętana.

Pozostałe istotne klasy, które znalazły swoje odzwierciedlenie w klasie `Page`, zostaną omówione w praktyce, w drugiej części książki. Teraz możemy zająć się tym, co tygrysy webowe lubią najbardziej — obsługą danych.