Kommunikáció „fű alatt” – AJAX (1. rész)

Melyik webfejlesztő nem ábrándozott még arról, hogy anélkül kommunikáljon a szerverrel, hogy az oldalt újra kellene hívni? Az AJAX , mint a JavaScript és az XML nyerő párosa váltja valóra ezt az ábrándot. Ismerjük meg közelebbről ezt a megoldást, melynek segítségével talán hamarosan csökkenni fog a honlap és alkalmazás kifejezések közötti különbség.

Egyszer volt , hol nem volt …

Hogy igazán megértsük, miért is fontos és nagy áttörés az ***AJAX***, valamint , hogy mi is ez valójában, érdemes áttekinteni a ***World Wide Web (WWW)*** történetét napjainkig. Az első név, melyet meg kell említenünk ***Tim Berners-Lee***, aki lefektette a ***WWW*** alapjait a 1990-ben azzal, hogy megálmodott egy hálózatot. Egy olyan hálózatot, melyben az adatok rendszerezetten ***hypertext*** formátumban érhetők el, úgynevezett ***URI-k (Uniform Resources Identifiers)*** segítségével. Ekkor bontakozott ki a jó öreg ***HTML (Hypertext Markup Language)***, melynek segítségével formázni lehetett az adatokat, valamint linkek alkalmazásával összekapcsolni az egyes dokumentumokat. Ezeken felül nem sok tudása volt a ***HTML***-nek, az adatok statikus megjelenítése volt a jellemző.

Az üzlet azonban hamarosan közbeszólt. A reklámok, hirdetések, egyéb üzleti elképzelések vezettek oda, hogy fellépet az igény arra, hogy a felhasználók szabadabban kezelhessék a megjelenített adatokat. A ***HTML*** azonban nem támogatott eddig semmilyen eszközt, mellyel a megjelenített adatok manipulációját eszközölni lehetett volna. Ekkor a semmiből bukkant elő egy vállalat, aki igen csak nagyot lendített a ***WWW*** fejlődésén, megoldást találva a fent leírt problémára és melyet úgy hívtak, hogy ***Netscape***.

Az első igazi böngésző, mely igazán elterjedt a ***Netscape Navigator*** volt. A ***Netscape*** ebben a böngészőjében tette lehetővé először a ***JavaScript*** használatát, melynek eredeti neve ***LiveScript*** volt. Fejlesztőjét ***Brendan Eichnek*** hívták, az első verzió pedig az 1995-ben kiadott ***Netscape Navigator 2.0***-ban látott napvilágot. Ezekben az időkben ha valaki 28.8 Kbps-os modemmel rendelkezett az nagy jelentőséggel bírt. A lényeg, hogy nem kellett minden interakció után a szerver felé kérést intézni, hanem a ***JavaScript*** segítségével helyben is lehetett módosítani az adatokat, vagy ugyancsak a szkripten keresztül csak bizonyos adatokat fogadni illetve küldeni, melynek segítségével csökkent az adatforgalom és nőtt a kiszolgálás sebessége. Ez kulcsfontosságú volt az ***AJAX*** történetében.

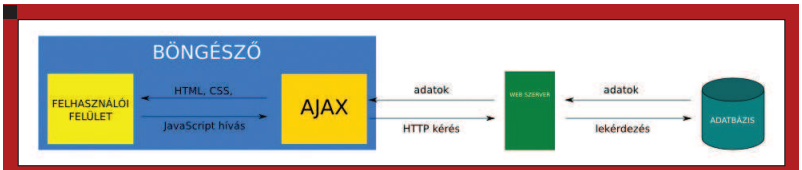
A ***HTML*** „ősi” verziója csak „egyke” dokumentumokat tudott megjeleníteni. A ***frame-ek*** bevezetése, ugyancsak nagy áttörés volt. Segítségükkel egy dokumentumot, több független részre lehetett osztani, így megint csak külön lehetett választani a statikus elemeket, és változó, dinamikus adatok másik ***frame***-be kerülhettek. Ez azért jelentős, mert nem az egész dokumentummal kellett dolgozni, csak egy bizonyos részével, megint csak csökkentve a feldolgozott adatok mennyiségét és a feldolgozás idejét. A ***Netscape*** megelőzve a ***HTML 4.0*** szabvány bevezetését, saját kezébe vette az irányítást, és a már említett 2.0-ás verziójú böngészőjét frame-es támogatással dobta piacra. A ***JavaScript*** és a ***frame***-ek használata többféle kliens-szerver kommunikációra adott lehetőséget, főleg azután, hogy a 90-es évek végén a „böngésző háborúban” (***Netscape Navigator – Internet Explorer***) a ***frame*** és ***JavaScript*** lehetőségek a versenyhelyzetben egyre jobban csiszolódtak, fejlődtek.

Az ***AJAX***-nak még be kellett várnia az ***XML*** forradalmát is, hogy végre úttörő lehessen a kliens-szerver kommunikációban a böngészők és a webfejlesztés terén. Végül 2005-ben látott napvilágot a kifejezés: ***AJAX***, mely nem egy új nyelv, mint majd látni fogjuk, hanem több alkalmazás együttesét takarja.

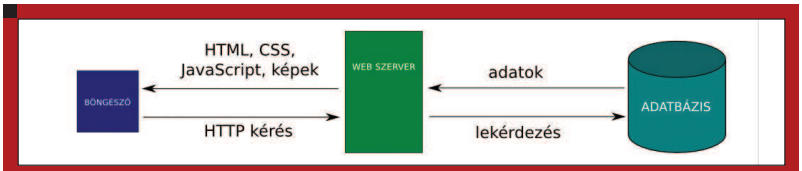
Az AJAX előtti technikák

Még mindig húzzuk az időt egy kicsit, olyan technikák megismerésével amelyekkel megvalósítható az oldal újratöltése nélküli kommunikáció a szerverrel. ***Frame*** és ***JavaScript*** alkalmazásával könnyen tudunk a háttérben kommunikálni a szerverrel. Ezt nevezzük „rejtett frame” módszernek. Az alapelv a következő: az oldalunk legalább két ***frame***-ből áll, melyek közül az egyik rejtett és ezen a ***frame***-en keresztül kommunikálunk a szerverrel. A rejtett ***frame*** annyit takar, hogy a szélessége vagy magassága 0 pixel. Az adatáramlást a két frame között a ***JavaScript*** fogja biztosítani.

A rejtett ***frame*** tartalmaz egy ***formot***, amit a látható ***frame***-ből származó adatok valamint a ***JavaScript*** segítségével kitöltünk és elküldünk. Amikor az információ megérkezik a rejtett ***frame***-be akkor azt kimásoljuk a látható területre.



1. ábra Az AJAX



2. ábra Általában így kommunikálnak a honlapok a szerverrel

A következő változást a ***DHTML (Dynamic HTML)*** és a ***DOM (Document Object Model)*** megjelenése hozta. Ezen két technológia segítségével lehetővé vált, hogy ***JavaScript*** programozással a kliens oldalon lehessen módosítani a ***HTML*** dokumentum szerkezetét, bármely apró részének tulajdonságát, tartalmát. Ez megint egy lépés volt az ***AJAX*** fejlődésében.
Bár a *„**rejtett frame**”* módszer nagyon elterjedt, hamarosan mégis egy új közkedvelt megoldás érkezett.
Az ***iframe*** elem 1997-ben futott be a ***HTML 4.0*** részeként. Használata sokkal egyszerűbb volt, mint a ***frame***-eké. Az oldalban bárhol el lehetett helyezni ezt az elemet, és ***CSS*** segítségével ***(Cascading Style Sheets***) láthatatlanná tenni. Amikor pedig az ***Explorer 5***-ös és a ***Netscape 6***-os böngészők hozták a ***DOM***-ot, a *„**rejtett frame**”* módszernek végleg befellegzett.
A ***DOM*** segítségével a fejlesztők ***JavaScripttel*** bármikor be tudtak szúrni a dokumentumba egy ***iframe*** elemet a kliens oldalon, anélkül hogy eleve a dokumentumban létezett volna ilyen elem egyáltalán. Azaz a szerverrel való kommunikáció végleg a ***JavaScript***-re maradt, nem kellett már kombinálni, előre megírt ***HTML*** elemekkel, hiszen azokat is létre tudja már hozni futási időben.
Az utolsó fejlemény, ami talán végleg megpecsételte az ***AJAX*** sorsát – hiszen valamilyen szempontból innen is ered maga a kifejezés: ***AJAX*** – az a ***Microsoft XMLHttp ActiveX*** objektumának megjelenése volt 2001 környékén. Ezen az objektumon keresztül lehetővé vált, hogy kérést küldjön a ***JavaScript*** a szerver felé, valamint hogy kezelje a ***HTML*** státusz kódokat, fejléceket is. Lehetővé vált végre, hogy ***frame***-ek illetve ***iframe***-ek használata nélkül valósuljon meg a kommunikáció. Persze a ***Mozilla*** projekt később sajátot fejlesztett ki ebből az objektumból, ami hordozza az ***XMLHttp*** főbb jellemzőit: ***objectXMLHttpRequest***. Még fogunk vele találkozni a cikk során.

Végre: AJAX

Ennyi történet után már remélhetőleg valami körvonalazódik az olvasó fejében, hogy mi is lehet az az ***AJAX***.
Az ***AJAX*** tehát annak a technológiának a neve, mely a ***háttérben*** valósítja meg a kommunikációt a szerverrel és lehetővé teszi a kapott információk által a felhasználói felület módosítását. Lényeges eleme, hogy a háttérben várja meg az adatok megérkezését, tehát nem áll meg az oldal működése, amíg a szerver felé a kérés le nem zárul. Az ***adatáramlásban bármilyen adat szerepelhet***: sima szöveg, ***XML*** vagy amire éppen a fejlesztőnek szüksége lehet. Végeredményben egy ***kommunikációs rétegről*** beszélhetünk: *1. ábra*. Az általában megszokott kommunikációs forma: *2. ábra.*
Ennek a megoldásnak nagyon sok előnye van. Mivel csak a szükséges adatokat mozgatja a szerver és a kliens között ezért minimalizálódik az áramlott adatok mennyisége, nincs felesleges kommunikáció. Felesleges lehet például mindig újratölteni egy oldalon a ***CSS***-t, képeket, ***HTML*** elemeket, ha csak a tartalom változik.
A kevesebb adatforgalom gyorsabb kommunikációt eredményez, ezáltal gyorsabb a felhasználók igényeinek kiszolgálása is. Mivel nincs szükség az oldal újratöltésére, szinte egy alkalmazás látszatát kelthetjük a felhasználóban, miközben egy ***HTML*** dokumentumot kezel.
Természetesen hátrányai is vannak ennek a módszernek. A programozás során elég bonyolult kód keletkezhet, ha elég jól értünk a ***JavaScripthez*** akkor nem lehet akadály az ***AJAX*** programozás. Persze az ***AJAX*** technológia nem csak kizárólag ***XML***-re és ***JavaScriptre*** épít, ahhoz, hogy megfelelően kezelhessük az ***AJAX***-ot ismernünk kell az ***XHTML***, ***HTML*** nyelveket a ***CSS***-t, ***DOM*** ismeretekre fokozottan szükségünk van, nem árt ha képben vagyunk az ***XML*** alapjaival, az ***XMLHttp*** és az ***objectXMLHttpRequest*** kezelése is szükséges a ***JavaScripten*** felül.
Már ezekből is látszik, hogy egy bonyolult rendszer keletkezhet egyszerűbb ***AJAX*** oldalból is, de majd látni fogjuk, hogy az eredmény kárpótol mindenért, és később már nem is fog olyan nehéznek tűnni.

AJAX akcióban

Lássunk végre példát a módszer alkalmazására. Mi is lehetne más a példa, mint a jó öreg *„**Hello World**”* alkalmazás. Kicsit persze alakítunk rajta, hogy két irányú adatáramlásról legyen szó. Készítünk egy egyszerű ***HTML*** oldalt, melyben szerepelni fog egy input mező valamint egy gomb. A gomb megnyomásakor az input mező tartalma elküldjük a szervernek ott feldolgozzuk egy ***PHP*** szkript segítségével és visszaadjuk a böngészőnek úgy hogy elé fűzzük a *„**Hello**”* szót, ezt pedig egy mondjuk egy ***SPAN***-ban fogjuk megjeleníteni. Az alkalmazás három részből fog állni: a ***HTML*** felület, ***JavaScript***, szerver oldali szkript (ez most ***PHP*** lesz).
Először készítsük el a ***HTML*** felületet, ami mivel most szemléltetési eszköz lesz, igen egyszerűen fog kinézni: *1. lista.* A gombnak és a beviteli mezőnek itt szerintem érdemesebb ***id***-t adni mint nevet (***name***). Véleményem szerint ez alapján könnyebb őket visszakeresni a ***DOM*** segítségével (`document.getElementById` függvény). A gomb ***onclick*** metódusa a ***Kuld*** eljárás lesz.

1. lista HTML felület

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C
➥//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/
➥xhtml1/DTD/xhtml1-
➥strict.dtd">
<html xmlns="http://
➥www.w3.org/1999/xhtml">
  <head>
    <title>AJAX -
    ➥LINUXVILÁG</title>
    <script language=
    ➥"javascript"
type="text/javascript">

        // ide jön majd a
➥JavaScript kódunk: 2. lista

    </script>
  </head>
  <body>
     Név:<input type="text"
     ➥id="nev" />
    <input type="button"
    ➥onclick="Kuld()"
    ➥value="Ok" />
    <br />
    Ide fog kerülni a
    ➥szerverről visszaérkező
    ➥adat: <span id="box">
    ➥</span>
   </body>
</html>
```

A kommunikációs réteg programozása *JavaScript* segítségével: 2. *lista*. Ahogyan nagyon sok *JavaScript* alkalmazás működik, itt is figyelnünk kell, hogy melyik alternatívát használó böngészővel van dolgunk, és ez alapján hozzuk létre az `XMLHttpRequest` vagy az `XMLHttp` objektumunkat. Ezen az objektumon keresztül fog történni a kommunikáció. A `Kuld` eljárás végzi a kommunikációs feladatokat. Kivesszük a beviteli mezőből a beírt nevet, értéket. Az `Ajax.onreadystatechange` egy olyan függvény vagy eljárás, mely akkor fut le ha az adatok megérkeztek, ez lesz nekünk most a `Kesz` eljárás. Az `open` tag fogja nyitni a csatornát a szerver felé. Most *GET* módszerrel fogunk küldeni adatot a *hello.php*-nak (a küldendő anyagot a *URL*-hez fűzzük), a harmadik paraméterben pedig megadjuk hogy a futás ne szakadjon meg amíg az adatok érkeznek (`false` érték esetén pedig a futás a kommunikáció idejére megszakad). A `send` tag paramétere csak *POST* típusú küldés esetén érdekes, ezért ezt most null értékre állítjuk. A `Kesz` függvényben a feltétel vizsgálja a kommunikáció eredményességét. A `readyState` több értéket is felvehet, amivel az adatforgalom feletti ellenőrzést lehet megvalósítani:

0 – még nem indult el az adatcsere,
1 – töltés,
2 – töltés kész,
3 – feldolgozás,
4 – kész.

A feltételünk persze az *ActiveX* objektummal is számol, ezért vizsgálja a `complete` egyezést is. Amint megjött az adat a szerverről,amit a `responseText` tag tartalmaz, elhelyezzük a *SPAN* tagon belül. A szerver oldalon pedig annyira egyszerű ebben az esetben a kód, hogy külön listát kár lenne rá pazarolni. A *hello.php* tartalma a szerveren: `<?php echo 'Hello '.$_GET['nev'].'!'; ?>`. Láthatjuk, hogy egy egyszerűbb példa is kicsit bonyolultra sikerülhet, de az eredmény véleményem szerint nagyszerűen hat. Nem kell az oldalt újratölteni, csak a lényeges elemek változnak. Kisebb adatok esetén, szinte egy helyben futó alkalmazás érzetét kelti a honlap. A jó öreg *Google* a *Gmail* nevű szolgáltatásánál is előszeretettel használja ezt a technológiát. Még külön *API*-t is le tudunk tölteni a honlapjukról (➲ *code.google.com*), ami az *AJAX*-ra épít és felhasználhatjuk honlapok fejlesztéséhez.

Ami következik

A következő alkalommal, természetesen *AJAX*-ot használva felépítünk egy honlapot, hogy komolyabb alkalmazásra is lássunk példát. Addig is ha valakit elkapott az ihlet, akkor kellemes időtöltés kívánok az *AJAX*-hoz. Véleményem szerint hamarosan eljön az idő, amikor alkalmazás és honlap egybe fog olvadni. Ez a technológia az első lépések egyike. A *Google* már ilyen úton jár a *Docs & Spreadsheets* nevű szolgáltatásával is.

2. lista AJAX JavaScriptben

```
      var Ajax = null;

        // Mozilla
        if (window.XMLHttp
        ➥Request) { Ajax =
        ➥new XMLHttpRequest
        ➥(); }

        // Explorer
        if
(window.ActiveXObject)
        {
           Ajax = new
➥window.ActiveXObject("Micro
➥soft.XMLHTTP");
        }

        function Kuld()
        {
          var nev    =
➥document.getElementById
➥('nev').value;

          Ajax.onready
➥statechange = Kesz;
          Ajax.open('GET',
➥'hello.php?nev='+nev,true);
          Ajax.send(null);
        }

        function Kesz()
        {
          if (Ajax.ready
➥State == 4 || Ajax.ready
➥State == "complete")
          {
            document.get
➥ElementById('box').inner
➥HTML = Ajax.responseText;
          }
        }
```

Radics Péter
(peter.radics@gmail.com)
Az ELTE-n tanulok programtervező matematikus szakon. Hobbim a kosárlabda, autóvezetés, web-design, programozás. Főleg webes alkalmazások fejlesztése érdekel. 4 éve megrögzött Linux felhasználó vagyok.