Ajax

Jiří Zacpal

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

KMIWEBA Webové aplikace

Ajax

- AJAX (Asynchronous JavaScript and XML) je revoluční použití asnychronního JavaScriptu (JS) pro změnu části webové stránky bez nutnosti jejího znovunačtení.
- Přestože se to dnes může zdát jako triviální záležitost, právě AJAX stojí za současnou podobou webových stránek a aplikací.
- Princip, na kterém je AJAX založen je poměrně jednoduchý.
  - JS kontaktuje webový server a asynchroně čeká až mu server odpoví.
  - Jakmile obdrží data provede příslušnou změnu v DOM.
- Jelikož je vše řízené JS, není nutné provádět znovunačtení stránky.
- Dodejme že JS ve výchozím stavu běží v jednom vlákně, vykonávání instrukcí je tedy synchronní.
- X v názvu AJAX představuje XML, které je využíváno pro přenos dat ze serveru na klienta v asynchronní komunikaci.
- Přestože je možné XML použít, dnes se běžněji využívá formát JSON případně prostý text.

Realizace

- Asynchronní požadavek je možné poslad dvěmi způsoby. Starší způsob využívá XMLHttpRequest objekt. Tento způsob je velmi univerzální, široce podporovaný a umožňuje sledovat postup zpracování asynchroního dotazu na server.

```
function ajaxTest() {
            const xhr = new XMLHttpRequest();  // xhr.readyState === xhr.UNSENT
            xhr.open("GET",
"https://www.cnb.cz/cs/financni_trhy/devizovy_trh/kurzy_devizoveho_trhu/denni_kur
z.txt");  // xhr.readyState === xhr.OPENED

            // očekávaný typ odpovědi, pro JSON: json
            xhr.responseType = 'text';
```

Realizace

```
            // obsluha zpracování odpovědi, // xhr.readyState === xhr.LOADING
(stahování dat)
            xhr.onload = () => {
                 //všechna data stažena, můžeme je zpracovat
                if (xhr.readyState === xhr.DONE) {
                    if (xhr.status === 200) {
                        //console.log(xhr.response);

                        document.getElementById("demo").innerHTML = xhr.response;
                    }
                }
            }
            xhr.send(); // xhr.readyState === xhr.HEADERS_RECEIVED
        }
    </script>
```

Realizace

- Novější, ale mírně omezenější způsob, využívá funkci fetch(), která je integrovaná přímo v JS. Zjevnou výhodou je snadnost použití.

```
// fetch vytvoří promis, pomocí, then() zpracujeme výsledek až je k dospozici
fetch("http://localhost:8888/ajax/server.php").then(r =>
r.text()).then(console.log);

// + ošetření chyb
fetch("http://localhost:8888/ajax/server.php").then(r =>
r.text()).then(console.log).catch(console.error);

// poznámka: v případě zpracování JSON, stačí změnit na .then(r =>
r.json()).then(j => j.results).then(console.log)
```

Realizace

Promis můžeme vytvořit i explicitně pomocí Async a Await. Jedná se o alternativní zápis výše uvedeného.

```
const ajax = async () => {
    try {
        let res = await fetch("http://localhost:8888/ajax/server.php");
        let results = await res.text();
        console.log(results);
       } catch (error) {
        console.error(error);
         }
       }

ajax();

// poznámka: při asynchronní zpracování je třeba vždy ošetřovat výjimky
```

Realizace

- Přirozeně se nabízí možnost spojit přísliby a použití XMLHttpRequest objektu a docílit robustního asynchronního volání s možností sledování stavu.

```
const ajax = new Promise((resolve, reject) => {
  const api = "http://localhost:8888/ajax/server.php";
  const xhr = new XMLHttpRequest();
  xhr.open("GET", api);
  xhr.onload = () =>
    xhr.status === 200
      ? resolve(xhr.response)
      : reject(xhr.status);
  xhr.onerror = err => reject(err);
  xhr.send();
});

ajax.then(r => console.log(r)).catch(error => console.error(error));

// poznámka: ? :  je ternární operátor
// poznámka: v případě zpracování JSON, stačí změnit na
resolve(JSON.parse(xhr.response).results)
```

CORS

- Z bezpečnostních důvodů je v prohlížčích imlementován machanizmus Cross-Origin Resource Sharing (CORS), který brání načítání zdrojů pomocí AJAX z jiného domény, pokud zdroj neobsahuje správnou HTTP hlavičku (Access-Control-Allow-Origin: *).
- Vetšina veřejných služeb povoluje přístup, ale je zapotřebí HTTP protokolu. To znamená, že obvykle nelze testovat asynchronní dotaz na jiné servery bez použití lokálního či jiného webového serveru.

Kam dál

- V principu žádná alernativa k AJAX neexistuje. Pro zajímavost uveďme ještě dvě zajímavé technologie, které s AJAX spolupracují (vyžadují asynchronní zpracování):
  - Server sent events
  - WebSocket.

- Server sent events - umožňuje serveru, bez vyžádání klienta, posílat data klientovi. Toto je velmi zajímavé, jelikož to odbourává základní premisu v klient-server komunikaci (klient žádá server). Takto je možné například řešit live-feed. Klient se nemusí neustále dotazovat na nová data. Ty jsou mo poslána hned, jak jsou dostupná.

- WebSocket - implemetuje soketovou komunikaci v prostředí Internetu. Díky tomu je možné zprovoznit rychlou obousměrnou komunikaci mezi klientem a serverem. Takto je možné řešit například chatovací místnosti nebo real-time přenos dat.

Úkoly

ÚKOL 1

- Upravte mazání uživatelů tak, aby bylo provedeno asynchroně (při mazání nedojde k znovunačtení stránky).

ÚKOL 2

- Upravte REST API z pátého semináře tak, aby vracelo data ve formátu JSON. Pomocí AJAX načtěte tato data do administračního rozhraní (například do sekce others).