AJAX, REST

Klasická a asynchronní komunikace, AJAX, promise, fetch API, API, metody http, GET, POST, DELETE, PUT, PATCH, OPTION, REST

Joseph Soukup – P4

AJAX

- Asynchronní Javascript A XML
- Technologie pro tvorbu interaktivních aplikací v prostředí webu
- Umožňuje měnit obsah stránky bez nutnosti jejího znovunačtení

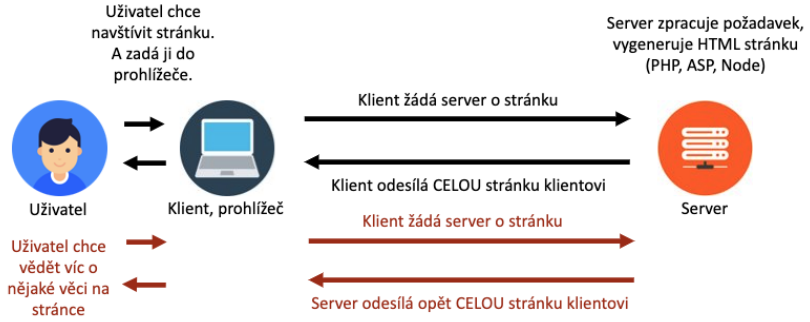
Použité technologie

- HTML a CSS - vzhled a struktura stránky
- DOM a Javascript – modifikování obsahu stránky na straně klienta
- XMLHttpRequest Object – vytváření dotazů na dodatečná data
- XML, JSON, … - formát dat pro výměny informací

Využití

- Našeptávače
- Interaktivní formuláře
- Validace dat na straně klienta
- Komplexní aplikace (Gmail, GAapps, Office365)
- Chat, online komunikace
- Agregátory (kombinace dat z více zdrojů)

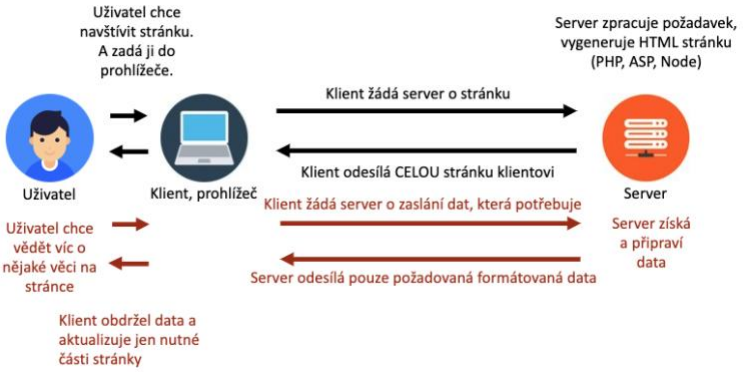
Schéma komunikace bez AJAXu



Výhody a nevýhody

- Pokaždé se zpracovává a přenáší kompletní webová stránka
- Přenáší se veškerá grafika, CSS, HTML, Javascript (ale ty se nezměnily)
- = náročné na data
- Je pokaždé nutné na stránku sestavit (server) a vyrenderovat (klient)
- = náročné na výkon
- (výhoda) Jednoduchá implementace

Schéma komunikace pomocí AJAXu



Výhody a nevýhody

- Přenáší se výrazně méně dat
- Ale vede to k tomu, že se přenášejí častěji
- Není potřeba sestavovat HTML stránku na straně serveru
- Je potřeba aby klient uměl překreslit části stránky

Implementace v Javascriptu

XML

- Formát pro výměnu dat
- Postavený na stejných principech jako je HTML
- ```
  <?xml version="1.0" encoding="UTF-8"?>
  <note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  ```

JSON

- Formát pro výměnu dat
- Založený na objektech v Javascriptu (Platný zápis v JS)
- Názvy „proměnných“ jsou v uvozovkách
- `{ "name":"John" }` – řetězec musí být uvozovkách
  `{ "age":30 }` - číslo
  `{ "sale":true }` - boolean
  `{ "middlename":null }` - null
  `{ "name":"John", "age":30, "car":null }` – objekt, asociativní pole
- `[ "Ford", "BMW", "Fiat" ]` - pole
  ```

XML vs JSON

```
{"employees":[
{ "firstName":"John", "lastName":"Doe" },
{ "firstName":"Anna", "lastName":"Smith" }
,
{ "firstName":"Peter", "lastName":"Jones"
}
]}
```

```
<employees>
  <employee>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName>
    <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName>
    <lastName>Jones</lastName>
  </employee>
</employees>
```

XMLHttprequest (XHR)

- JS object umožňující komunikaci se vzdáleným serverem
-

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "http://www.data.js/neco/nekde");
xhr.send();
```

Určení adresy a metody odeslání

Odeslání požadavku

Synchronní odeslání požadavku a získání dat

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "http://www.data.js/neco/nekde");
xhr.onLoad(function(){
  if (xhr.readyState === xhr.DONE) {
    if (xhr.status === 200) {
      console.log(xhr.response);
      console.log(xhr.responseText);
    }
  }
});
xhr.send();
```

Co se má stát po získání dat?
= xhr.load = function(){}

Aktuální stav akce

Návratový kód HTTP

Vrácená data

Textová podoba vrácených dat

Asynchronní odeslání a zpracování požadavku

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    document.getElementById("demo").innerHTML =
      this.responseText;
  }
};
xhr.open("GET", " http://www.data.js/neco/nekde",
true);
xhr.send();
```

Událost reagující na změnu stavu požadavku
= readystatechange

Toto je asynchronní požadavek

Parsování JSON řetězce na objekt

```
let result =
  '{ "name":"John", "age":30, "city":"New York"}‘;
let obj = JSON.parse(result);
console.log(obj.name);
```

Převod dat na JSON formátovaný řetězec

```
let obj = { name: "John", age: 30, city: "New York" };
let myJSON = JSON.stringify(obj);
```

- Lze převádět na objekty, pole, data, řetězce, čísla
- Nelze převádět funkce (budou odstraněny)
    - Je možné převést funkce na řetězec
    - Obj.funkce.toString()

AJAX s daty v JSON

```
let xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    let obj = JSON.parse(this.responseText);
    console.log(obj.name);
  }
};
xhr.open("GET", “http://something", true);
xhr.send();
```

Fetch API a Promise

Cesta k promise při asynchronních operacích

- Aktivní synchronní čekání
- Callback – funkce, zavolá se po dokončení operace a předá se jí výsledek
- Události
- Promise

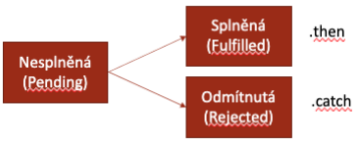
Promise

- Objekt reprezentující příslib dokončení asynchronní operace (=jednou dostaneme její výsledek nebo důvod zamítnutí)
- Asynchronní operace vrací objekt Promise
- Na objekt pověsíme obsluhu toho, až získáme výsledek této operace
- Využívá zápis přes řetězení
- doSomethingAsync(params, successCallback, failureCallback)
- doSomethingAsync(params).done(successCallback, failureCallback)
- Všechny callbacky se zavolají postupně, právě jednou, po dokončení asynchronní operace
- Snadno se řetězí, po skončení jedné asynchronní operace se zavolá další

Postup

1. Je zavolána asynchronní operace
2. Ta vytvoří a okamžitě vrátí promise (tak nedochází k blokování hlavního vlákna)
3. Promise má funkci **Promise.then(onFulfilled)**, která registruje callback onFulfilled, který se zavolá, jakmile slíbená hodnota dorazí

4. Promise má funkci **Promise.catch(onRejected)**, která registruje callback onRejected, který se zavolá, pokud dojde k chybě (například data nedorazí včas nebo server vrátí chybový kód)



Async…await

- Async function název(parametry) {kód}
  - V této funkci bude volána asynchronní metoda, která bude vracet Promise
- a = await getSomeAsyncResult(42);
  - operátor „čekáme, až bude mít promise výsledek“, sloučení vláken programu

```
function resolveAfter2Seconds() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}
```

```
async function asyncCall() {
  console.log('calling');
  var result = await resolveAfter2Seconds();
  console.log(result);
}

asyncCall();
```

Promise chaining

Pomocí callbacků

```
doSomething(function(result) {
  doSomethingElse(result, function(newResult) {
    doThirdThing(newResult, function(finalResult) {
      console.log('Got the final result: ' + finalResult);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);
```

Pomocí Promise

```
doSomething()
.then(function(result) {
  return doSomethingElse(result);
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log('Got the final result: ' + finalResult);
})
.catch(failureCallback);
```

Fetch API

- kompletně nahrazuje XMLHttpRequest globálně přístupnou metodou **fetch**
- Není funkční v Internet Exploreru a mobilních prohlížečích (podpora v Chrome, Firefox a Edge), nutné transkódovat přes Babel
- Používá mechanismus **Promise** (= vrací objekt Promise)
- Promise = fetch(adresa);
- Promise = fetch(adresa, inicializační data);

- Promise = fetch(Request_object, inicializační data)

Nejjednodušší kód (zápis fetch().then)

```
fetch('http://example.com/movies.json')
  .then(function(response) {
    return response.json();
  })
  .then(function(myJson) {
    console.log(JSON.stringify(myJson));
  });
```

Získej data z této adresy

Až dorazí, dekóduj je a vrať

A pak je ještě napiš do konzole

Kompletní příklad

```
fetch(`http://example.com/answer`, {
        method: "POST", // *GET, POST, PUT, DELETE, etc.
        mode: "cors", // no-cors, cors, *same-origin
        cache: "no-cache", // *default, no-cache, reload, force-cache,
                           only-if-cached
        credentials: "same-origin", // include, *same-origin, omit
        headers: {
            "Content-Type": "application/json",
            // nebo "Content-Type": "application/x-www-form-urlencoded",
        },
        redirect: "follow", // manual, *follow, error
        referrer: "no-referrer", // no-referrer, *client
        body: JSON.stringify({key: value}),
    },)
    .then(response => response.json());
    .catch(error => console.error(error));
}
```

Adresa

Parametry dotazu

Předávaná data:
= GET http://example.com/answer?key=value

Tohle udělej, pokud dojde k chybě

Tohle udělej až data přijdou úspěšně

Zajímavá inicializační data

- **Method –** metoda požadavku (GET, POST)
- **Headers –** hlavičky požadavku (typ vrácených dat, tokeny)
- **Body –** tělo požadavku (předávaná data)
- **Mode –** použití CORS

Fetch Request

```
const myImage = document.querySelector('img');
let myRequest = new Request('flowers.jpg');
fetch(myRequest)
  .then(function(response) {
    if (!response.ok) {
      throw new Error("ERR:" + response.status);
    }
    return response.blob();
  })
  .then(function(response) {
    let objectURL = URL.createObjectURL(response);
    myImage.src = objectURL;
  })
  .catch(function(error)) { … }
```

REST (Representational state transfer)

- Cesta, jak jednoduše vytvořit, číst, editovat nebo smazat informace ze serveru pomocí jednoduchých http volání
- Architektura rozhraní, navržená pro distribuované prostředí
- Distribuovaná architektura v tomto smyslu znamená, že části programu běží na různých strojích a pro svoji komunikaci využívají síť
- Webová aplikace -> webový server
    - GET – Číst data
    - POST – Vytvořit data
    - DELETE – Smazat data
    - PUT – Upravit/Nahradit data
    - PATCH – Upravit/Modifikovat data