Web and Database Computing

adelaide.edu.au

Client-Server Basics: Seamless Dynamic Content with AJAX

Dynamic Content the Old Way

Dynamic webpages

Dynamic webpages allow us to make complex web applications.

1. A user makes a request
2. The request is processed
3. Custom content is generated
4. The content is combined with HTML to create the webpages
5. The **whole** page is sent in response

Dynamic webpages

Dynamic webpages allow us to make complex web applications **but have drawbacks**

1. A user makes a request
2. The request is processed
3. Custom content is generated
4. The content is combined with HTML to create the webpages **Wastes server resources**
5. The **whole** page is sent in response **Wastes network resources**

The problem with Dynamic webpages

Demo

There's gotta be a better way!

AJAX

Asynchronous **J**avaScript **A**nd **X**ML allows us to make HTTP requests and process the responses in the backround.

- Save time
  - Load page content as we need it.
- Reduce network load
  - Send only the information we need.
  - Allows caching of other page resources.
- Reduce server load
  - Generate HTML on client
  - Less time spent processing and sending response.
- Better user experience
  - Whole page doesn't need to reload for user to view new content
  - Content can be loaded in the background - seamless

Better with AJAX

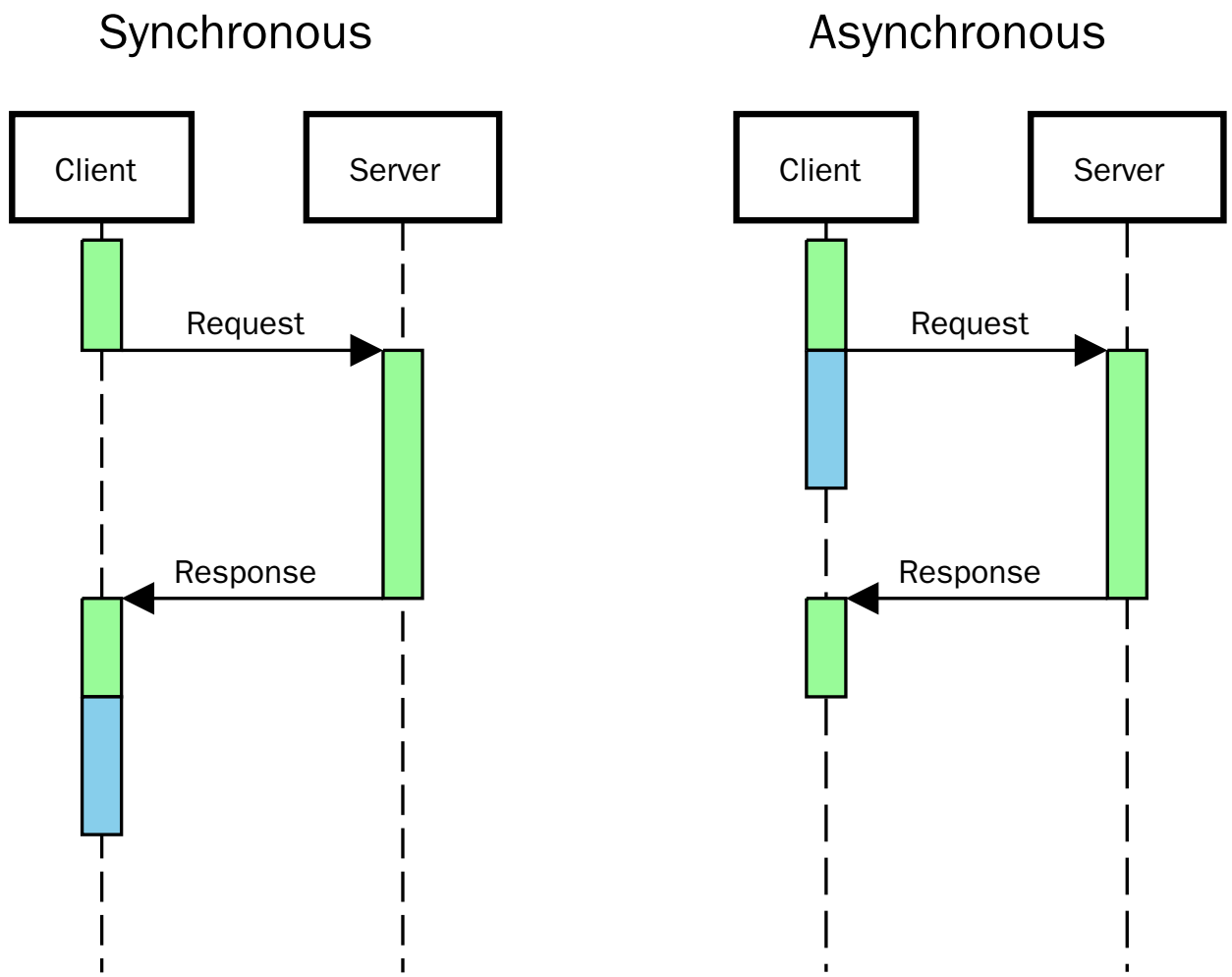
Demo

- AJAX page on server at /ajax.html

What's happening?

1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An AJAX request is created by JavaScript
3. The request is sent to the web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. JavaScript updates the page

Asynchronous vs Synchronous



Asynchronous vs Synchronous

Asynchronous code allows us to prevent the main event loop from blocking while we wait for a response. This allows the user to continue to interact with the page, which they would not be able to do if synchronous.

So what if we want the behaviour of synchronous code (a certain action to happen after another), but the advantages of asynchronous code (non-blocking)?

Callback functions

Use a callback function!

- A function passed as an argument to another function to be called after it completes

```
function a() {
    console.log("a");
}

function bAsync(callback) {
    console.log("b");
    callback();
}

bAsync(a); // Prints b then a
```

Do you need to use a callback?

- Make sure that code after asynchronous calls does not rely on the asynchronous call completing.
- If the code does rely on the asynchronous call completing, make it a callback.

Making an AJAX Call

```
/* 1. Create new AJAX request */
var xhttp = new XMLHttpRequest();

/* 4. Handle response (callback function) */
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        console.log(this.responseText);
    }
};

/* 2. Open connection */
xhttp.open("GET", "/request/uri", true);

/* 3. Send request */
xhttp.send();
```

The XMLHttpRequest object

```
/* 1. Create new AJAX request */
var xhttp = new XMLHttpRequest();

/* 2. Open connection */
xhttp.open("GET", "/request/uri", true);
```

The XMLHttpRequest object is a standard javascript browser object that can be used to send requests to servers.

- Initialise the request with the open method:
  `open(Method, URL, Asynchronous?)`
- Note that the call can be synchronous, by setting last argument to false, but then the browser will not do any further action on the page until the server responds, and this method is deprecated.

The XMLHttpRequest object

```
/* 3. Send request */
xhttp.send();
```

- Send the request to the server with the send method
  `send(Data)`
- Data is optional
- If you have data to send to the server, you send it here
  - Used for a POST request where there is form data or JSON to send to the server.

The XMLHttpRequest object

```
/* 4. Handle response (callback function) */
xhttp.onreadystatechange = function() {
    // do this code when the request state changes
};
```

- Sets up a listener (`onreadystatechange`) to listen for XMLHttpRequest events.

What triggers the onreadystatechange event of an XMLHttpRequest?

- When the request is opened, `request.readyState == 1`
- When the response headers are received, `request.readyState == 2`
- When the body starts to load, `request.readyState == 3`
- When the response is finished loading, `request.readyState == 4`

The XMLHttpRequest object

The `onreadystatechange` function will be called every time the state changes

```
/* 4. Handle response (callback function) */
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        // the response is done downloading
        // and the status code was 200 OK
        // this.response will give us the response
    } else {
        // Do I want to do something when the headers arrive
        // or one of the other states,
        // or the status code is 404 or another error
    }
};
```

Handling the AJAX response

The response can be accessed using

```
xhttp.responseText
```

An AJAX response can be anything!

- Plain text
  - Usually encoded as UTF-8
- HTML
  - Ready to insert into our webpage
- XML
  - Method of storing data using DOM
- JSON
  - Text representation of JavaScript Object
- Images
- Anything!

JSON

Java**S**cript **O**bject **N**otation is a way of representing JavaScript objects as text.

```
{
  "myarray": [
            { "name":"Max", "age":24 },
            { "name":"Ji", "age":23 }
            ],
  "count": 2
}
```

Using JSON allows us to send objects directly between the client & server!

THE UNIVERSITY
of ADELAIDE