|  |
| --- |
| PES UNIVERSITY  (Established under Karnataka Act No.16 of 2013) |
| 100-ft Ring Road, BSK III Stage, Bangalore – 560 085 |
| **Department of Computer Science & Engineering** |
| **Web Technologies-II(UE18CS325) UNIT I Reading Material** |
|  |

|  |
| --- |
|  |

**Unit I Course Information:**

* JavaScript Objects, OO Concepts
* Prototype Inheritance
* Hidden Frames Technique - GET
* Hidden Frames Technique - POST
* Image-based AJAX
* JSON vs XML
* XMLHttpRequest – GET, POST
* Fetching binary data with XHR
* Maintaining history in AJAX calls
* Cross-domain access (CORS)

**Javascript Objects and OO Concepts:**

**JavaScript is**

* OBJECT BASED (not object oriented).It is started as a scripting language.Everything in JS except the primitive types are objects

Javascript objects

var person1={fname:"abc",lname:"xyz",age:23}

* Object literals
  + keys can be properties/methods
  + Method as key

var person1={fname:"abc",lname:"xyz",age:23

sayname=function(){....}}

* Constructor functions

function Person(fname,lname,age){

this.fname=lname;

this.lname=lname;

this.age=age;

this.sayname=function(){

console.log("Hi");

}

}

person1=new Person("abc","xyz",23);

* this-> for object in current scope
* "new" keyword
  + constructor actually returns nothing
  + but new keyword indicates that an object is created and a reference to it is returned

var person1=Person("abc","xyz",23);

*//undefined*

* Every function is an object
* using new with non constructors makes no difference
* return statement in constructor
  + if constructor returns primitive type, value is ignored (return 5;)
  + if constructor returns an object
    - this replaces the object created

function Person(fname,lname,age){

....

return new Array("1","2");

}

person1=new Person("abc","xyz",23);

*//person1->Array*

* Issue with constructors
  + each object has copy of fn
    - But OOP languages : fns are common, each object does not have a copy
* Solution is : Prototypes

**Prototype Object**

* + Shared among all objects
  + When creating objects with literals prototype points to Object.prototype
  + Irrespective of whether object has functions, a prototype is created
  + Every prototype object has constructor property
* Object literals --> Singletons
* Created person1 using object literal
  + prototype points to Object.prototype

<prototype>: {…}

constructor: function Object()

* Created person1 using Constructor (Person)
  + prototype points to Person.prototype

<prototype>: {…}​​

constructor: function Person()

* To share methods (instead of each object having a copy) -> use protoype property of the constructor
  + Adding functions to prototype of constructor is accessed by all objects

function Person(fname,lname,age){

this.fname=lname;

this.lname=lname;

this.age=age;

}

Person.prototype.sayname=function(){

console.log("Hi "+this.fname);

}

<prototype>: {…}

​​ constructor: function Person()

​​ sayname: function sayname()

var person1= new Person("abc","xyz",23);

var person2= new Person("a","x",3);

* person1 **proto** points to Person.prototype
* person2 **proto** points to Person.prototype
* Person.prototype contains sayname

Add properties to prototype to make default ones for all rather than directly adding new properties to the object itself. It wastes lot of memory space.

Person.prototype.sem=5;

var person1= new Person("abc","xyz",23);

person1.sem=5;

*//person1 sem =5, person2 sem=3*

* Every function has a property prototype
* **Person = constructor**
  + **Property : prototype --points to--> Person.prototype**
    - **can add common methods here**
* **Creating a new instance**
  + **Property : proto --points to --> Person.prototype (parent)**
* Person.prototype
  + has common methods
  + common properties for default values
* JS now supports inheritance and overwriting methods

**Q : Create a constructor function for a rectangle which has len and breadth. To its prototype add a fn to calculate its area. Also override toString method to print len breadth. Create an instance of rectangle and invoke both**

function Rectangle(len,breadth){

this.len=len;

this.breadth=breadth;

}

Rectangle.prototype.CalculateArea=function(){

return this.len\*this.breadth;

}

Rectangle.prototype.toString=function(){

console.log("len "+this.len+" breadth"+this.breadth);

}

var rect=new Rectangle(2,3);

console.log(rect.CalculateArea()); *//6*

rect.toString(); *//len 2 breadth3*

**Prototype inheritance**

* calling constructor without parameters is allowed in JS, but should not throw error
* Square -> inherit from rectangle

function Square(size){

this.len=size;

this.breadth=size;

}

Square.prototype=new Rectangle();

Square.prototype.constructor=Square;

var sq=new Square(4);

console.log(sq.CalculateArea()); *//16*

sq.toString(); *//len 4 breadth 4*

Square.prototype.toString=function(){

console.log("side "+this.len);

}

sq.toString(); *//side 4*

* If we say : Square.prototype=Rectangle.prototype()
  + Now changing any method of Square will affect Rectangle
* **call**
  + "call" takes object and other parameters

var person1= new Person("abc","xyz",23);

function sayhi(){

console.log(this.fname);

}

sayhi(); *//undefined*

sayhi.call(person1); *//abc*

* call -> call(object,arg1,arg2,...)
  + arguments[0]=arg1

function sayGreeting(){

console.log( arguments[0]+this.fname);

}

sayGreeting.call(person1,"SayHi"); *//SayHi abc*

*//object=person1*

*//arguments[0]=SayHi*

* **apply**
  + Difference between call and apply is
    - call -> arguments separated by comma
    - apply-> arguments in an array

function Person(name){

this.name=name;

}

var person1=new Person("a");

var Hello3=function(){

console.log("hello "+arguments[0]);

}

Hello3(); *//hello undefined*

Hello3(person1); *//hello [object Object]*

Hello3.apply(person1); *//hello undefined*

Hello3.apply(person1,["abc"]); *//hello abc*

var Hello4=function(){

console.log("hello "+this.name);

}

Hello4(); *//hello*

Hello4(person1); *//hello*

Hello4.apply(person1); *//hello a*

Hello4.apply(person1,["abc"]); *//hello a*

Hello4.apply(person1,[person1]); *//hello a*

* **CONSTRUTOR STEALING**
  + A subclass' constructor can use a parent's constructor to initialize
  + In JS : use call /apply for constructor stealing
  + Inside Square's constructor, call Rectangle's constructor

function Square(size){

Rectangle.call(this,size,size);

}

* Whenever a function is invoked, an array called **arguments** is created
  + All arguments passed in the function call are a part of arguments
  + If function has no parameters in definition and in the call you pass arguments, arguments array is created

function hello(){

console.log(arguments[0]);

}

hello(1,2,3); *//1*

[More on prototypes :](https://googleweblight.com/i?u=https://javascriptweblog.wordpress.com/2010/06/07/understanding-javascript-prototypes/&hl=en-IN)

* Javascript object
  + unordered collection of key-value pairs
* Protoytype : an object from which other objects inherit properties
  + objects can be prototypes
  + every object has prototype by default
  + real power of prototype is
    - Multiple instances share a common prototype.
    - Properties of the prototype object are defined once but inherited by all instances which reference it.

var a ={};

console.log(a);

*// Object {*

*//<prototype>: {*

*//constructor: function Object()*

*// .......*

*//}*

*//}*

console.log(typeof a); *//object*

console.log(Object.getPrototypeOf(a)); *//Object { … }*

console.log(a.constructor.prototype); *//Object { … }*

* Every function gets a prototype property (built-in function excepted)

function test(){

}

console.log(test);

*//test()*

*// arguments: null*

*// caller: null*

*// length: 0*

*// name: "test"*

*// prototype: Object { … }*

*// <prototype>: function ()*

console.log(test.prototype); *//Object { … }*

* A function’s prototype property
  + object that will be assigned as the prototype to all instances created when this function is used as a constructor.
  + nothing to do with it’s actual prototype
* Adding properties to prototype

function A(name) {

this.name = name;

}

var a = new A('alpha');

A.prototype.val1=2;

console.log(a);

*//name: "alpha"*

*// <prototype>: {…}*

*// constructor: function A()*

*// val1: 2*

Changing the prototype

A.prototype={x:1};

console.log(a);

*//a.\_\_proto\_\_ still references the original object*

*//name: "alpha"*

*// <prototype>: {…}*

*// constructor: function A()*

*// val1: 2*

[More on call and apply :](http://hangar.runway7.net/javascript/difference-call-apply)

function Person(name){

this.name=name;

}

var person1=new Person("a");

var person2=new Person("b");

var Hello=function(){

console.log(this);

console.log("hello "+this.name);

}

Hello(); *// this=Window, o/p=hello*

Hello(person1); *// this=Window, o/p=hello*

Hello.call(person1); *//this=Object { name: "a" } o/p=hello a*

var Hello2=function(x){

console.log("hello "+this.name+" x= "+x);

}

Hello2(); *// hello x= undefined*

Hello2(person1); *// hello x= [object Object]*

Hello2.call(person1); *//hello a x= undefined*

Hello2.call(person1,"4"); *//hello a x= 4*

**AJAX:**

AJAX is not a technology but group of inter-related technologies. AJAX technologies includes:

* HTML/XHTML and CSS
* DOM
* XML or JSON
* XMLHttpRequest
* JavaScript

The major break-through in web applications are solving the “waiting” problem.

* Make asynchronous calls to the server, without making the user wait for a response.
* Refresh a part of the page only.
* Download contents of a page in stages.
* Refresh page content periodically without having to re-render the page.

**AJAX Major applications are:**

* Login forms (no need to navigate to a specific login page before returning to where we want to be)
* Auto-complete enabled apps
* Voting
* Chat rooms and instant messaging
* Alternative to popups (which are usually blocked)
* Gmail, FB, twitter, youtube, etc,…. uses ajax

**AJAX Mechanisms:**

* Using hidden frames
* Using XMLHttpRequest
* Using images
* Using Javascript
* Using Stylesheet

**Hidden Frames technique:**

The hidden frame technique involved setting up a frameset where one frame was set to a width or height of 0 pixels, and its sole purpose was to initiate communication with the server. The hidden frame would contain an HTML form with specific form fields that could be dynamically filled out by JavaScript and submitted back to the server. When the frame returned, it would call another JavaScript function to notify the original that data had been returned. The hidden frame technique represented the first asynchronous request/response model for web applications

* Hidden IFrames – An iframe, reserved for the server data, is hidden initially. The main window which houses the iframe, is visible.
* On an event in the main window, the ‘src’ of the hidden iframe is changed to point to a server resource.
* The server returns data to the hidden iframe. The main window then makes the frame visible (if need be) or the hidden frame updates the main window with the data it received.

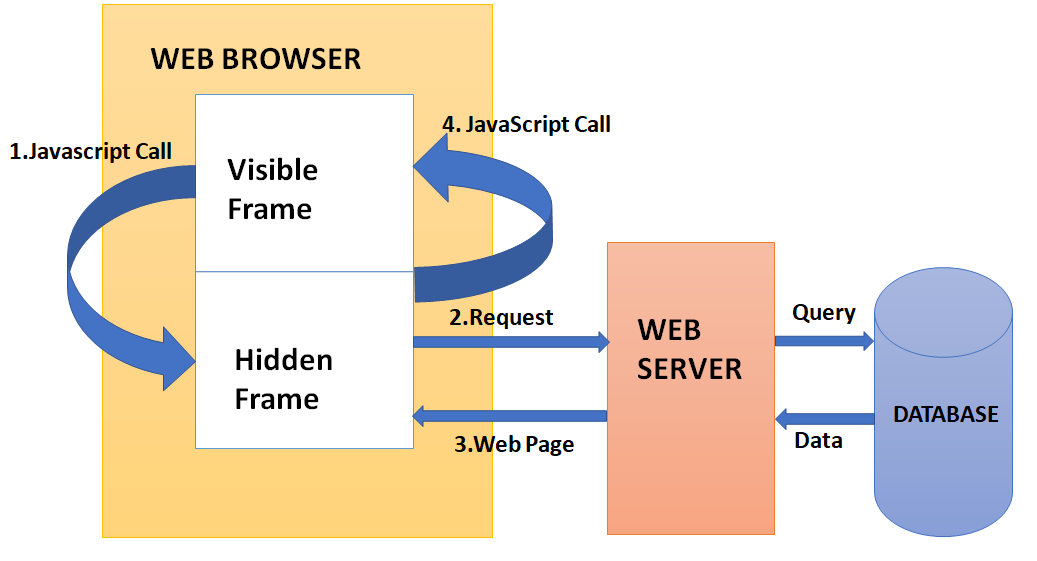


Fig: 4 step Hidden Frame Technique

**Advantages**

* Can store browser history. So the back and forward buttons work.
* Can make GET and POST requests.
* GET Query string can take 512 KB of info
* POST – 2 GB of information
* A page is a collection of frame sets
  + iframe tag
    - Set src
    - when src changes, only that part of page changes
  + Hidden frame can access any frame using parent

<body>

<form>

SRN <input type="text" name="srn" id="srn" onblur="getDetails()"/><br/>

NAME <input type="text" name="name" id="name" disabled/><br/>

CGPA <input type="text" name="cgpa" id="cgpa" disabled/><br/>

<input type="button" name="Save" onclick="submitForm()"/>

<iframe id="hframe" name="hframe"/>

</form>

</body>

function getDetails(){

var srn=document.getElementById("srn");

*//get access to frame*

var hfr=document.getElementById("hframe");

*//for async commn : change src of hframe*

hfr.src="http://localhost/dev/fourth\_year/sem\_7/WT2/11thAug.php?srn="+srn.value;

}

function showDetails(res){

var resArr=res.split(";")

var name=document.getElementById("name");

var cgpa=document.getElementById("cgpa");

name.value=resArr[0];

cgpa.value=resArr[1];

}

PHP File:

<head><script>

<?php

extract($\_GET);

if($srn=="SRN1"){

$ret="abc;9.8";

echo("parent.showDetails('$ret')");

}

if($srn=="SRN2"){

$ret="abcdddd;9.4";

echo("parent.showDetails('$ret')");

}

if($srn=="SRN3"){

$ret="xyz;6.7";

echo("parent.showDetails('$ret')");

}

?>

</script></head>

* echo statement is what is sent back to client
  + Response from PHP file -> parent.showDetails('abc;9.8')
    - inside script tag
* "parent.showDetails('$ret')"
  + go to parent of frame : i.e main page
  + find showDetails function and execute it
* Advantage of hidden frames
  + easier to code
  + both GET and POST request
  + easy to maintain history (compared to XMLHttpRequest)
  + click on back button shows the previous values entered

*//php file*

$ret="SRN1;abc;9.8";

*//html file*

function showDetails(){

....

srn.value=resArr[0];

name.value=resArr[1]

cgpa.value=resArr[2];

}

**Disadvantage of hidden frames**

* Domain restrictions. (Even sub-domains create problems)
* Poor error handling.If there are errors to request to server, parent cannot figure it out
* No control over operation. If the iframe fails to load, then there will be no updates.

**Summary**

* + Hidden frames -> small parts of the page; async commn
  + embedded in a page is a iframe (hidden)
    - height & width=0
    - display none
    - visibility hidden
  + If src of frame is changed => async request triggered
  + onblur event handler
    - get access to iframe object
    - change src attr (http:localhost....)
    - pass query string along with the URL (default is GET request)
  + Server side
    - extract input value & process
    - send back complete html page
    - loaded into iframe
    - Actually sending a JS code (echo statement)
      * response
  + history is maintained in frame's history object
* form
  + action -> file that is invoked
  + method ->
    - GET (default if method is not specified)
    - POST parameters not visible in the URL
* Hidden frames and POST method

<<form id="ipform" action="11thAug\_post.php" method="POST" target="hframe">

SRN <input type="text" name="srn" id="srn" onblur="getDetails()"/><br/>

NAME <input type="text" name="name" id="name" disabled/><br/>

CGPA <input type="text" name="cgpa" id="cgpa" disabled/><br/>

<input type="button" value="Save" onclick="submitForm()"/>

<iframe id="hframe" name="hframe" height="0" width="0"/>

</form>

* target: (name of hidden frame)
  + when it is submitted, go to the server, finish processing
  + response should come back to hframe (not the page)
* when blur action happens -> form should be submitted

function getDetails(){

var frm=document.getElementById("ipform");

frm.submit();

}

* when the Save button is clicked->submitForm()
  + change action to different php file

function submitForm(){

var frm=document.getElementById("ipform");

frm.action="11thAug\_post\_submit.php";

frm.target="\_self";

frm.submit();

}

**Image Based Ajax:**

* Instead of having hidden frame, we have images where
  + calls to fetch images are asynchronous
  + src of image can be request to php page
  + php file processes and send back image (create and send)
    - either check for name of image (yes/no)

Async communication using Images

* Images in a page are loaded asynchronously
* When src of image is changed, asynchronous request is sent
  + Image src can be any file (.php, etc)
* PHP side : createImage, colorallocate, sendImage
  + create 1 pixel image and send it ( small images )

**Advantages**

* + Error handling
    - If image load is success ->Image.onload event is triggered
    - If image load is not success ->Image.onerror event is triggered
    - Both cases we can handle the error
  + Cross origin access (resource sharing)
    - image src can be from any url
    - (hidden frames can't be this way)
  + No change on webpage
    - Image is not added to the webpage
  + Much more simpler and easier in cases like yes/no response
  + Good when bandwidth constraints are severe

**Q : Check if username exists**

<form>

Username <input type="text" name="uname" id="uname" onblur="checkUname()"/><br/>

Password <input type="text" id="pwd" disabled/><br/>

Confirm password <input type="text" id="conf" onblur="checkPass()"disabled/><br/>

<input type="button" id="sub" value="SAVE"/>

</form>

Create image (html file) :

function checkUname(){

console.log("checking username");

uname=document.getElementById("uname");

*//creating image (global variable)*

img=document.createElement("img");

img.id="image";

img.src="http://localhost/ test.php?uname="+uname.value;

img.onload=success; *//if loaded successfully, call fn success*

img.onerror=failure;

}

img.onload=success

* No parenthesis
  + Here : only setting the attribute to a function "success"
  + Inline mode uses parentheses

PHP file : (graphics designer library for these functions)

*//Username not available*

$im=imagecreate(1,1);

imagecolorallocate($im, 255, 255, 255);

imagejpeg($im); *// send image*

* When sending an image from the server to the html file :
  + browser only accepts text
  + But to send back an image :
    - **RESPONSE HEADER : mention that it is an image ( and not text)**

<?php

header("Content-type:image/jpeg");

extract($\_GET);

.......

Success and failure (html file) :

function success(){

console.log("Image loaded");

if(img.width=1){

document.getElementById("res").innerHTML="<p style='color:red'>Username not available</p>";

}

}

function failure(){

console.log("failure");

}

* Disadvantages
  + Can use only GET request
    - POST : can't send data via URL
  + Issue when cookies are disabled on the browser
  + It fails when images are disabled on the browser
    - But images are rarely disabled these days

**XML** *(use slides)*

* eXtensible Markup language
* carry data, not display
* self descriptive
* **well-formed**
  + Unlinke HTML, it is very strict
  + documents must have root element
  + you cannot forget to close tags
    - should have opening&closing tags
  + tags case sensitive
  + attr vals must have double quotes
  + proper nesting
* **validation**
  + Valid XML : XML document that has an associated document type declaration or schema and complies with the constraints expressed in it
  + XML schema
    - XML schema is written in XML
    - XML doc -> A, XML scheme -> S
    - in A you refer that its schema is S
      * S will have things like datatypes
  + XML DTD - another way of validation
    - Document Type Definition (DTD)
    - not written in XML (unlike XML schema)
* **processing**
  + API for processing
    - Programming languages and SAX API
    - Programming Languages and DOM API
* XML stylesheets

**ADVANTAGES**

* provides a basic syntax that are used to share information between different kinds of computers,different applications, and different organizations.
* data can be available to all kinds of "reading machines"  (Handheld computers, voice machines, news feeds, etc)
* It Provides a gateway for communication between applications
* Even applications on wildly different systems. As long as applications can share data (through HTTP, file sharing, or another mechanism)
* Supports Unicode allowing almost any information in any written human language to be communicated.
* Internationalisation & Globalisation facilitated
* It can represent common data structures like records, lists and trees.
* self-documenting format describes structure, field names, specificvalues.
* Schema support

i.e. the ability for party A to specify the format of a document, and the ability for party B to check that they are supplying something that matches this format.

* Crucial when passing data between separate systems, where a deviation from the expected format might mean that the data cannot be processed (or worse, is processed incorrectly).
* Namespace support i.e. the ability to mix data intended to be read by multiple sources (or written by multiple sources) in the same document.
* **XSLT**
* Easy to take XML and apply XSLT to make XHTML.
* XSL : eXtensible Stylesheet Language
* styling language for XML.
* XSLT : XSL Transformations.
* use XSLT to transform XML documents into other formats (like transforming XML into HTML).

**DISADVANTAGES**

* Difficult for the end-user to understand its capabilities.
* XML syntax is redundant that may affect application efficiency through higher storage, transmission and processing costs
* large ( relative to binary representations of similar data especially with tabular data)
* verbose (especially for human readers) relative to other alternative 'text-based' data transmission formats.
* hierarchical model for representation limited in comparison to an object oriented graph

**JSON** (use slides)

* Javascript Object Notation
* It is text format that is completely language independent
* **methods to parse JSON :**
  + JSON.parse
  + eval fn in Javascript
* Example : If we had to send student data to see what electives are chosen, XML might be better as we can validate the data (Not allow sending data without the electives field)
* data oriented -> JSON
* server side -> XML

**XML vs JSON**

* Human readability : JSON
  + json is less verbose
* Speed : JSON
  + parsing
    - XML : slow ,large amt of memory
    - JSON : part of JS code, less time to parse
* Security : XML
  + JS eval does not check the json code
    - code injection attack
* Data interpretation : JSON
  + JSON : map like, easy
  + XML : tree like

**AJAX XHR (XML Http Request)**

* The standard way to make an AJAX call.
* New XMLHttpRequest object is created.
* The onreadystatechange event is registered.
* The open() call is made with 3 parameters – the method, the resource and the async value.
* The send(data) call is made to actually send data.
* The readyState and status properties are checked before using the server data.

**Characteristics:**

* API in the form of an object
* Provided by the browser’s JS environment
* Methods transfer data between client and server
* Can be used with other protocols than HTTP
* Can work with data other than XML(JSON,plain text)

**XMLHTTPREQUEST**

* Methods
  + open
  + send(params)
    - if POST is used, send using params
  + abort()
  + setRequestHeader
* Properties
  + xhr = XMLHttpRequest() object
  + readyState
    - 0 when xhr is created
    - 1 when xhr.open()
    - 2 when xhr.send()
    - 3 part of response is coming
    - 4 when response is recieved

Everytime readyState changes, an event readyStateChange happens. From when the object is created to when response is got, readyState keeps changing

* 5 Http methods : GET, POST, PUT, HEAD, DELETE

**Properties and Methods of XHR:**

* open

open(method, url, username, password)

* readyState

readyState=4 -> completed

* Send()
* opening a file with file: and localhost
  + localhost --> an HTTP req is sent to server, that file is fetched and is run on browser
  + file: --> double clicking on .html file is as good as opening it on Word

**Getting JSON data as response:**

HTML file

<body>

Food <input type="text" name="fname" id="fname" onblur="obj.getDetails()"/>

Cuisine <input type="text" name="cs" id="cs"/>

Price<input type="text" name="price" id="price"/>

Veg <input type="text" name="veg" id="veg"/>

</body>

Script :

<script>

var obj={

xhr: new XMLHttpRequest(),

getDetails:function(){

*//when readyState changes, call fn showDetails*

this.xhr.onreadystatechange=this.showDetails;

var fname=document.getElementById("fname").value;

this.xhr.open("GET","test.php?fname="+fname,true);

this.xhr.send();

}

.........

}

</script>

* Sending json from php file
  + convert php array to json using json\_encode  
    PHP file

<?php

*//send json data*

$res=array();

extract($\_GET);

if($fname=="pizza"){

$res["cs"]="Italian";

$res["price"]="200";

$res["veg"]="non-veg";

}

.....

*//json\_encode*

$ret=json\_encode($res);

echo $ret;

?>

HTML file (script continuation)

var obj= {

.........

showFile:function showFile(){

*//in showfile current object is obj.xhr*

if(this.readyState==4 && this.status==200){

var res=this.responseText; *//as text string*

*//convert to json object*

var resJson=JSON.parse(res);

document.getElementById("cs").value=resJson.cs;

document.getElementById("price").value=resJson.price;

document.getElementById("veg").value=resJson.veg;

}

}

}

* Getting json data in html file
  + No separate function
  + Have to parse it
* Every time an event handler is invoked, the browser binds it to which object on which it occurred
  + this.xhr.onreadystatechange = this.showDetails;
  + onreadystatechange -> event handler
  + bound to this.xhr
* POST method (instead of get)
  + cannot use query string
  + data is sent as part of body of the request
  + parameters are sent in xhr.send() method
  + Ctr+Shift+I -> Network -> Request -> Request Payload : fname=pizza (see the data being sent here)
  + For the server to know where to look for data we have to set header. Multiple values : separate by &

this.xhr.open("POST","29thAug\_xhr\_post.php",true);

this.xhr.setRequestHeader("Content-type","application/x-www-form-urlencoded");

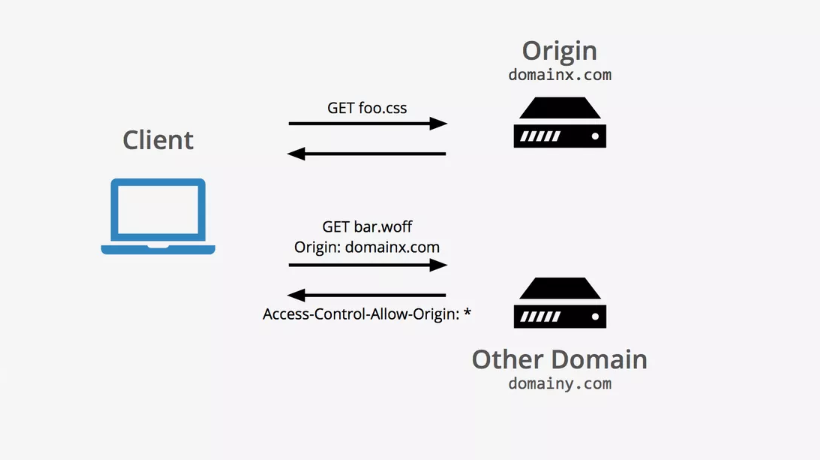
this.xhr.send("fname="+fname)

**Advantages of XMLHttp:**

* cleaner code compared to hidden frames
* more control over AJAX operation

**Cross origin resource sharing (CORS)**

* Images allow cross origin request
* Frames do not allow cross origin requests



* Disadvantages of XMLHttp is
  + Does not maintain history
  + Same Origin Policy. Solution is to add header on server side to allow cross origin resource sharing.The domain whose resources are being requested can respond to the first domain

**Different Headers:**

* Access-Control-Allow-Origin
* Access-Control-Allow-Credentials
* Access-Control-Expose-Headers
* Access-Control-Max-Age
* Access-Control-Allow-Methods
* Access-Control-Allow-Headers

header("Access-Control-Allow-Origin:\*")

*// \* -> allow from all (otherwise specify sites)*

* Way to simulate this
  + open html file by double clicking (file:...)
  + send req to localhost php file
    - this.xhr.open("GET","http://localhost/test.php?fname="+fname,true);

**Preflighted Requests**

* Simple requests : GET HEAD POST
* preflighted requests
  + PUT or DELETE (complex requests)
    - Browser first sends pre-flight requests with method as options
  + Headers are sent back
    - browser checks headers (eg does it allow PUT)
  + If allowed, PUT requests is sent.

**client side**

this.xhr.open("PUT","http://localhost/test.php?fname="+fname,true)

**Server side (php file)**

Add header

header("Access-Control-Allow-Methods:PUT")

Two requests : first OPTIONS then PUT

**OPTIONS** /test.php?fname=pizza HTTP/1.1  
Host: localhost  
.............................

**PUT** /test.php?fname=pizza HTTP/1.1  
Host: localhost  
..................................

Note : to see CORS working open the html file without localhost

Preflighted requests:

* request methods other than GET POST HEAD
* Have custom headers
* Have request bodies with Content-Type other than
  + text/plain
  + application/x-www-form-urlencoded
  + multipart/form-data

**XML response**

* PHP file
  + send xml data
  + set header

header("Content-type:text/xml");

echo"<movies><movie><name>BL</name><lead>C</lead></movie><movies>";

* browser by default accepts string
  + var res=this.responseText will work but will make the entire thing a string

**CORS**

* Simple request methods = GET, HEAD and POST (x-www-form-urlencoded, multipart/form-data, text/plain). These are not preflighted.
* The browser blindly makes a request with the "origin" header field set to the ip address of the client.
* The server responds with the "data" and also the **Access-Control-Allow-Origin** header.
* The browser compares the "origin" with this header and if they match, the data is shown to the user. Otherwise, blocks it.
* The other methods - PUT, DELETE and also POST with other content-types are preflighted with the "OPTIONS" method.
* Browser sends the following headers (first line is the "kind of" request anyway)

OPTIONS http:*//somecrossdserver.com HTTP/1.1*

Origin: ip address

Access-Control-Request-Method: POST (the method type to be used during the ACTUAL request)

Access-Control-Request-Headers: Content-type, X-SOME-OTHER

* Server responds with whatever headers the code has set (plus default of its own)
* Now the browser checks the **"Access-Control-Allow-Origin**", "**Access-Control-Allow-Methods**"
* To make sure it "CAN" make the ACTUAL request.

**Maintaining history in AJAX calls**

History can be maintained using the AJAX calls. We have to use window.history.pushState() to save state and then use window.onpopstate event (which fires when we click the back or front buttons in the browser

**Fetching binary data with XHR**

Binary data can be fetched as response by setting the responseType = “blob” and xhr.response

**File Uploading using XHR**

* You can also upload files using XHR.
* Use the <input type=“file”/> to select the file.
* You can create forms in XHR using the FormData() constructor.
* You can append variables to this by doing

form = new FormData();

And form.append(“varname’, somevalue).

* For files, it can be

file = file\_elem.files[0];

form.append(“myfile”, file);

**COMPARISON OF VARIOUS AJAX TECHNIQUES**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Technique | Cross domain requests | Code intent and clarity | Error handling | History of Requests | REQUEST TYPE | Other points |
| 1. | Image based | YES | GOOD | GOOD | NO | GET only. | See \*\* below |
| 2. | Hidden Frames | NO and YES\* | AVG | POOR | YES | GET and POST |  |
| 3. | XHR based | NO and YES\* | VERY GOOD@ | VERY GOOD | NOβ | GET and POST |  |
| 4. | Script based | YES | GOOD# | GOOD$ | NO | GET only. |  |

* \* - This needs to accomplished by using the “Access-control-allow-origin” header field in the server script that is supposed to allow the request to go through. Otherwise cross-domain calls are not possible with XHR.
* # - This technique can only be used to dynamically load scripts at run time. When bandwidth is a constraint, then scripts may also be downloaded on a “need-basis”.
* $ - onload and onerror events are available with script downloads also. Note that with script based AJAX, the script element should actually be added to the DOM to trigger the AJAX call. With image based AJAX, the image need not be part of the DOM. Merely changing the “src” attribute is enough.
* @ - XHR can be used to download XML (as a DOM), JSON, text, video and images. It is the most versatile technique. Asynchronous file uploads are also easily accomplished using the “FormData” constructor.
* \*\* - Can be very good when bandwidth constraints are severe. However, if cookies are disabled on the browser, this method can cause a lot of headache. If images are disabled on the browser, this method fails completely. Fortunately with today’s network speeds, images are rarely disabled. If we want to get plain text as a response, then cookies are the only way. We have to then read document.cookie on the client side, split on “;” and then split on “=” to read each cookie. Since these are part of the response header, we can only send limited text.
* β – History can be maintained here too (by writing code), but not straight-forward like the Hidden Frames technique where nothing needs to be done. We have to use window.history.pushState() to save state and then use window.onpopstate event (which fires when we click the back or front buttons in the browser)