中山大學
SUN YAT-SEN UNIVERSITY

Lecture 10a.
Asynchronous Programming I

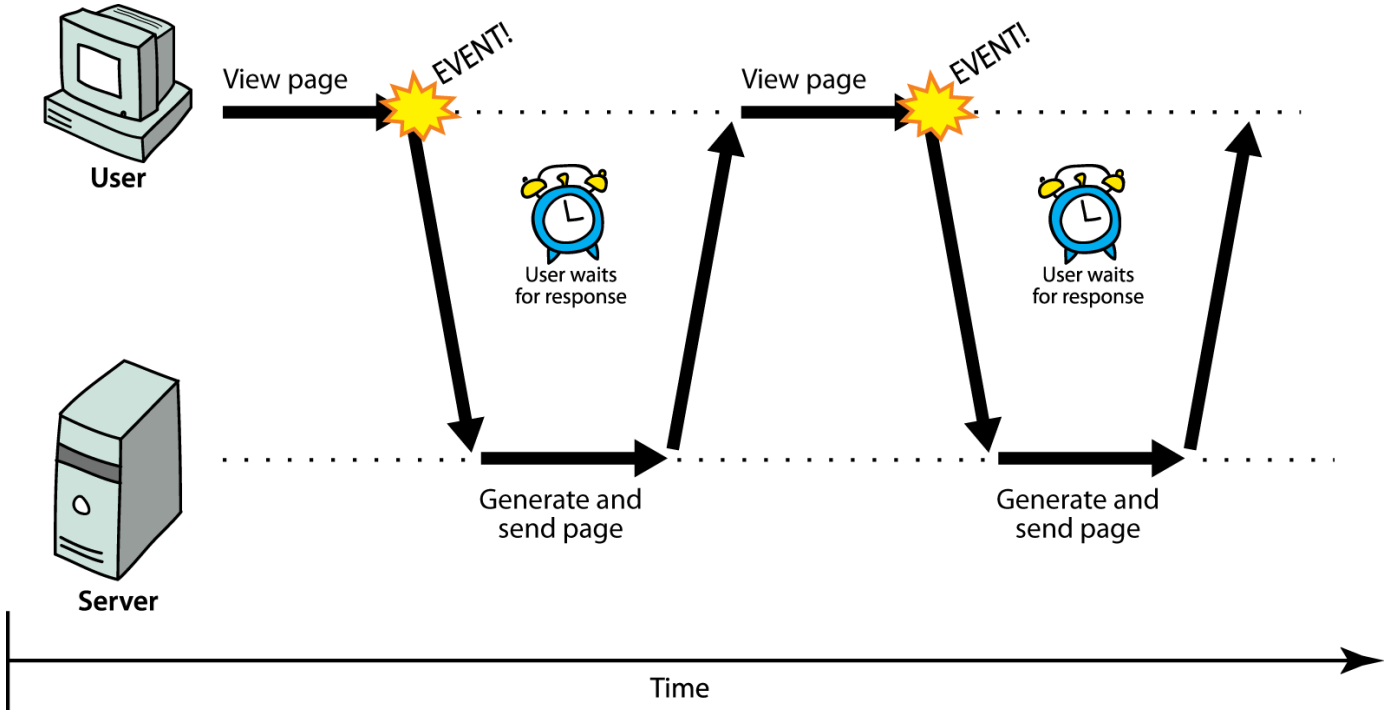
Modern Web Programming
(http://my.ss.sysu.edu.cn/wiki/display/WEB/ **supported by Deep Focus**)

School of Data and Computer Science, Sun Yat-sen University

Outline

- **Ajax, Web services, REST**
- Async in JavaScript

Synchronous web communication

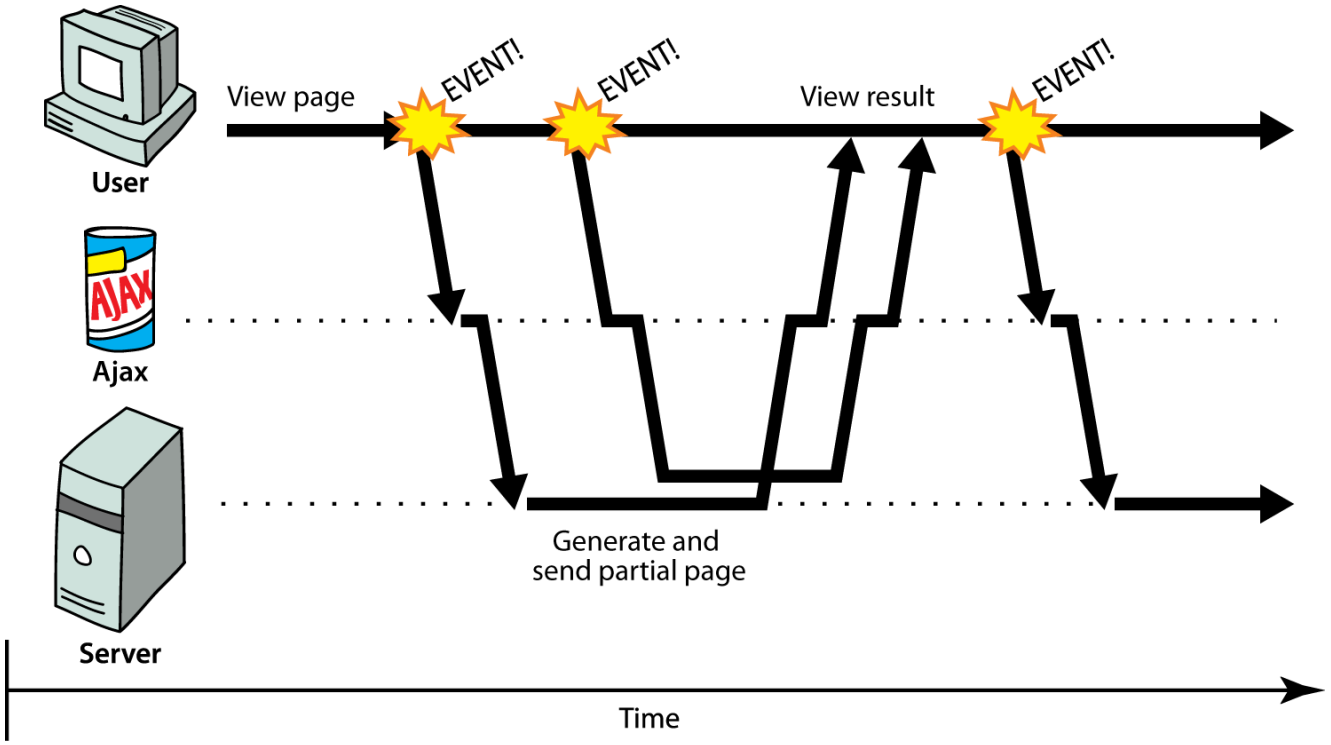


- **synchronous**: user must wait while new pages load
  - the typical communication pattern used in web pages (click, wait, refresh)
- almost all changes with new data lead to page refresh

Web applications and Ajax

- **web application**: a dynamic web site that mimics the feel of a desktop app
  - presents a continuous user experience rather than disjoint pages
  - examples: Gmail, Google Maps, Google Docs and Spreadsheets, Flickr
- **Ajax**: Asynchronous JavaScript and XML
  - not a programming language; a particular way of using JavaScript
  - downloads data from a server in the background
  - allows dynamically updating a page
  - avoids the "click-wait-refresh" pattern
  - examples: Google Suggest

Asynchronous web communication



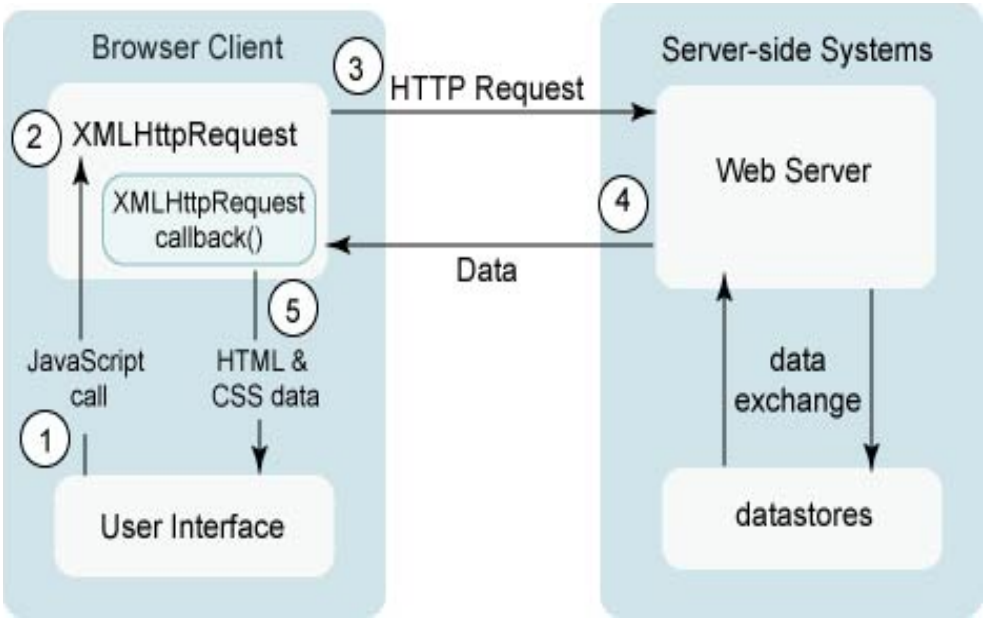
- **asynchronous**: user can keep interacting with page while data loads
  - communication pattern made possible by Ajax
- Changing with new data but without page refresh

XMLHttpRequest

- **JavaScript** includes an `XMLHttpRequest` object that can fetch files from a web server
  - supported in IE5+, Safari, Firefox, Opera, Chrome, etc. (with minor compatibilities)
- it can do this **asynchronously** (in the background, transparent to user)
- the contents of the fetched file can be put into current web page using the **DOM**
- `$.ajax $.get $.post`

A typical Ajax request

- user clicks, invoking an event handler
- handler's code creates an `XMLHttpRequest` object
- `XMLHttpRequest` object requests page from server



- server retrieves appropriate data, sends it back
- `XMLHttpRequest` fires an `event` when data arrives
  - this is often called a **callback**
  - you can attach a **handler** function to this event
- your callback event handler processes the data and displays it

Web services

```
1  var http = require('http');
2  var port = 3000;
3
4  http.createServer(function(req,res){
5    var random_time = 1000 + getRandomNumber(2000);
6    var random_num  = 1 + getRandomNumber(9);
7    setTimeout(function(){
8      res.writeHead(200,{'Content-Type':'text/plain'});
9      res.end("" + random_num);
10   }, random_time);
11 }).listen(port, function(){
12   console.log('server listen on ', port);
13 });
14
15 function getRandomNumber(limit) {
16   return Math.round(Math.random() * limit);
17 }
```

REST

```
http.createServer(function(req, res){
  switch(req.url){
    case '/validator.js':
    case '/signup.js':
    case '/style.css':
    case '/api/validate-unique':
      api.validateUnique(users, req, res);
      break;
```

```
module.exports = {
  validateUnique: function(users, req, res){
    req.on('data', function(chunk){
      var params, field, result, user;
      params = chunk.toString().match(/field=(.+)&value=(.+)/);
      user = {};
      user[field = params[1]] = decodeURIComponent(params[2]);
      result = validator.isAttrValueUnique(users, user, field) ?
        {isUnique: true} : {isUnique: false}
      res.writeHead(200, {"Content-Type": "application/json"})
      res.end(JSON.stringify(result));
    });
  }
}
```

REST

```
$.post('/api/validate-unique', {field: this.id, value: $(this).val() },
  function(data, status){
    if (status == 'success'){
      if (data.isUnique){
        $(self).parent().find('.error').text('').hide();
      } else {
        $(self).parent().find('.error').text("value is not unique").show();
        validator.form[self.id].status = false;
      }
    }
  });
```

Outline

- Ajax, Web services, REST
- **Async in JavaScript**

Async in JavaScript

- I/O
  - event handlers
  - ajax calls
  - File I/O
- setTimeout

```
$('button').click(function(event){...});

$.get('/api/get-person?name=eric', function(data, status){...

fs.readFile('./data.txt', function(err, data){...});


setTimeout(function(){...}, 500);
```

Asynchronous Programming

- Style of AP
  - callAysnc(param1, param2, callback)
  - callback(error, result)

- Traps of AP
  - callback is the return of AP
  - order of AP calls
    - when need line-up calls, you have to wrap them in callbacks
  - Once a async, all async
    - when a call in a calling stack is async, all its caller ancestors are asyncs
  - don’t throw error, callback error
  - this will be lost, use closure instead

Async. example

```
1  // synchronous version
2  function add(a, b){ return a + b;}
3
4  function sum(){
5    var amount = 0;
6    for(var i = 0; i < arguments.length; i++) amount = add(amount, arguments[i]);
7    return amount;
8  }
9
10 function divide(divident, divisor){
11   if (divisor === 0) {
12     throw new Error("can't divide by zero");
13   } else {
14     return divident / divisor;
15   }
16 }
17
18 function average(numbers){
19   return divide(sum(numbers), numbers.length);
20 }
```

Async. example

```
1  // synchronous version
2  function add(a, b){ return a + b;}
3
4  function sum(){
5    var amount = 0;
6    for(var i = 0; i < arguments.length; i++) amount = add(amount, arguments[i]);
7    return amount;
8  }
9
10 function divide(divident, divisor){
11   if (divisor === 0) {
12     throw new Error("can't divide by zero");
13   } else {
14     return divident / divisor;
15   }
16 }
```

```
1  function decrypt(ciphertext, callback) {
2    // find plaintext via some I/O operations .....
3    var plaintext = ciphertext;
4    callback(null, plaintext);
5  }
```

Async. example

```
function add(a, b){ return a + b;}

function add(a, b, callback){
  decrypt(a, function(err, p_a){
    decrypt(b, function(err, p_b){
      callback(null, parseFloat(p_a) + parseFloat(p_b));
    });
  });
}
```

callback is the return of AP

line up async calls in nested callbacks

once aysnc, all callstacks above aysnc

Async. example

```
function divide(divident, divisor){
  if (divisor === 0) {
    throw new Error("can't divide by zero");
  } else {
    return divident / divisor;
  }
}
```

```
function divide(divident, divisor, callback) {
  if (divisor === 0) {
    callback(new Error("can't divide by zero"));
  } else {
    callback(divident / divisor);
  }
}
```

Async can be tricky

```
function average(numbers){
  return divide(sum(numbers), numbers.length);
}

function average(numbers, callback) {
  sum(numbers, function(err, amount){
    if (err) {
      callback(err);
    } else {
      callback(amount/numbers.length);
    }
  });
}
```

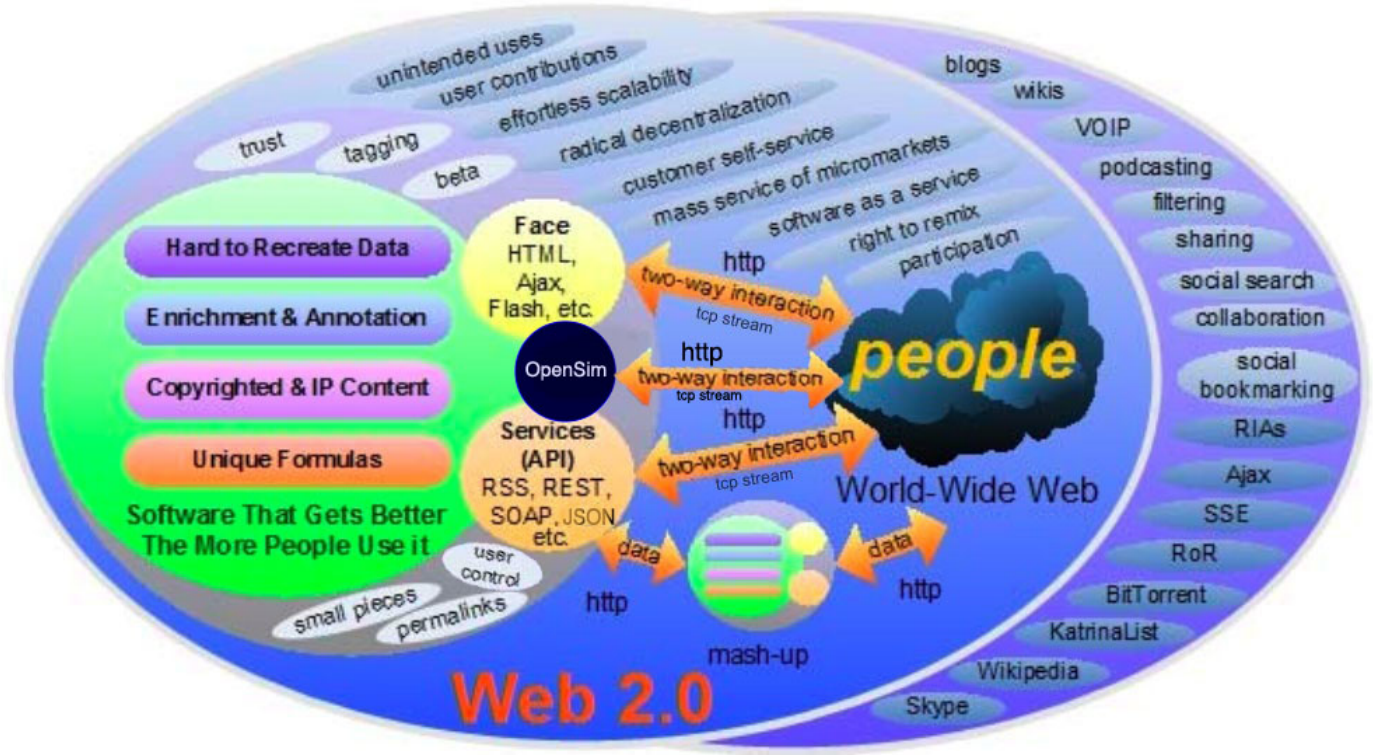
don’t throw error, callback error

Async can be tricky

```
function sum(){
  var amount = 0;
  for(var i = 0; i < arguments.length; i++) amount = add(amount, arguments[i]);
  return amount;
}

function sum() {
  var addents = Array.prototype.slice.call(arguments, 0, arguments.length - 1);
  var done = arguments[arguments.length - 1];
  var callbacks = [];
  for (var i = 1; i < addents.length - 1; i++){
    (function(i){
      callbacks[i] = function(err, sum){
        console.log('i: ', i, ", sum: ", sum);
        add(sum, addents[i + 1], callbacks[i + 1]);
      }
    })(i);
  }
  callbacks[addents.length - 1] = done;
  add(addents[0], addents[1], callbacks[1]);
}
```

Thank you!


unintended uses
user contributions
effortless scalability
radical decentralization
customer self-service
mass service of micromarkets
software as a service
right to remix
participation
trust
tagging
beta
blogs
wikis
VOIP
podcasting
filtering
sharing
social search
collaboration
social bookmarking
RIAs
Ajax
SSE
RoR
BitTorrent
KatrinaList
Wikipedia
Skype
Hard to Recreate Data
Enrichment & Annotation
Copyrighted & IP Content
Unique Formulas
Software That Gets Better The More People Use it
Face
HTML, Ajax, Flash, etc.
OpenSim
Services (API)
RSS, REST, SOAP, JSON etc.
http
two-way interaction
tcp stream
people
World-Wide Web
data
user control
small pieces
permalinks
mash-up
Web 2.0