

TDDD97 - Web Programming

Client-Server Communication

Sahand Sadjadee

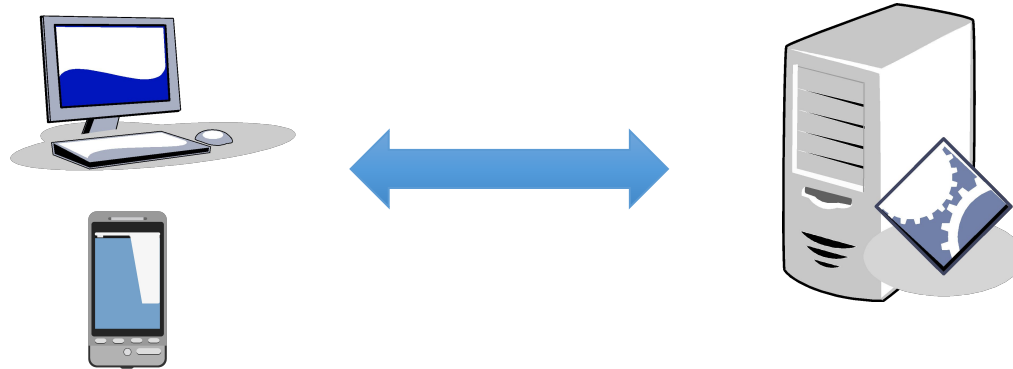
Dept. of Computer and Information Science

Linköping University

Outline

- Ajax
 - Web-sockets
 - JSON
 - Project Overview
-

Asynchronous Javascript And XML



- Browser clients get HTML documents from the web server when you load a web page (through the HTTP protocol)
- How can a JavaScript program transfer information to and from the web server?
 - Several solutions to this problem
 - In the labs: XMLHttpRequest (XHR)

XMLHttpRequest

- Exchange data between client and server using AJAX
 - Exchange data with a server behind the scenes
- Originally designed by Microsoft, currently being standardized by W3C
- Supported by all modern browsers
 - IE7+, Firefox, Chrome, Safari, Opera
- Not just for receiving XML, useful for text, JSON, etc.
- The XMLHttpRequest JavaScript object
 - Creating an XMLHttpRequest object:

```
variable = new XMLHttpRequest();
```

XMLHttpRequest – send request

- The `open()` and `send()` methods of the `XMLHttpRequest` object send a request to the server.
- Simple GET request:

```
var xmlhttp=new XMLHttpRequest();  
xmlhttp.open("GET", "course_info.txt", true);  
xmlhttp.send();
```

- Simple POST request:

```
xmlhttp.open("POST", "course_post.asp", true);  
xmlhttp.send();
```

- The `send()` method can take an optional single parameter with the data to send; that is, `send(myText)`



XMLHttpRequest – asynchronous calls

- Avoids blocking during the call execution
 - Do not wait for the server response
- Set the third parameter of `open()` to `true` to enable asynchronous calls
- Specify a function to execute when the response is ready in the `onreadystatechange` event:

```
xmlhttp.onreadystatechange=function()  
{  
    if (xmlhttp.readyState==4 && xmlhttp.status==200) {  
        document.getElementById("myDiv").innerHTML=xmlhttp.responseText;  
    }  
}  
xmlhttp.open("GET", "course_info.txt", true);  
xmlhttp.send();
```

XMLHttpRequest – onreadystatechange event

- The `readyState` property holds the status of the XMLHttpRequest

Property	Description
<code>onreadystatechange</code>	Stores a function (or the name of a function) to be called automatically each time the <code>readyState</code> property changes
<code>readyState</code>	Holds the status of the XMLHttpRequest. Changes from 0 to 4: 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready 
<code>status</code>	200: "OK" 404: Page not found 


XMLHttpRequest – response

- Getting the response from the server
 - Use the `responseText` or `responseXML` property of the `XMLHttpRequest` object
- The `responseText` property:



```
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```

- The `responseXML` property:

```
xmlDoc=xmlhttp.responseXML;   
txt="";  
x=xmlDoc.getElementsByTagName("COURSES");  
for (i=0;i<x.length;i++)  
{  
    txt=txt + "<div>" + x[i].childNodes[0].nodeValue + "</div>";  
}  
document.getElementById("myDiv").innerHTML=txt;
```


XMLHttpRequest – more things

- The `setRequestHeader` method
 - Example: tell the server that this call is made for ajax purposes

```
xmlhttp.setRequestHeader('X-Requested-With', 'XMLHttpRequest');
```

- Aborting requests: The `abort()` method
 - Aborts the request if the `readyState` of the `XMLHttpRequest` object has not yet become 4
 - Ensures that the callback handler does not get invoked in an asynchronous request

Flask repetition

- Flask routing
- Flask Template Rendering
- SQL and Flask
- Sample Flask Server

Flask Routing

- The `route()` decorator binds a function to a URL
- Examples:

```
@app.route('/')
def index():
    return 'Index Page'

@app.route('/hello')
def hello():
    return 'Hello World'
```



- Variable URLs:

```
@app.route('/user/<username>')
def show_user_profile(username):
    # show the name of a user
    return 'User %s' % username
```

Flask Template Rendering

- Based on the Jinja2 template language/engine for Python
- HTML templates should be located in the `templates` directory
- Template rendering function: `render_template()`
- Example:

```
from flask import render_template

@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

Flask Template Rendering (cont.)

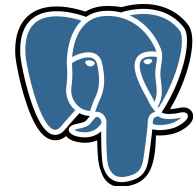
- In the file templates/hello.html

```
<!doctype html>
<title>Hello from Flask</title>
{% if name %}
    <h1>Hello {{ name }}!</h1>
{% else %}
    <h1>Hello World!</h1>
{% endif %}
```

SQL and Flask

Two possible methods

- SQLite3
 - Light-weight
 - Will be used in the labs
- Any other relational database like MySQL, PostgreSQL and Oracle.
 - Not light-weight
 - A separate process



ORMs like SQLAlchemy can be used combined.

- Python SQL toolkit and Object Relational Mapper
- More powerful and flexible—suitable for larger applications



Sample database implementation

```
import sqlite3
from flask import g

def connect_db():
    return sqlite3.connect("mydatabase.db")

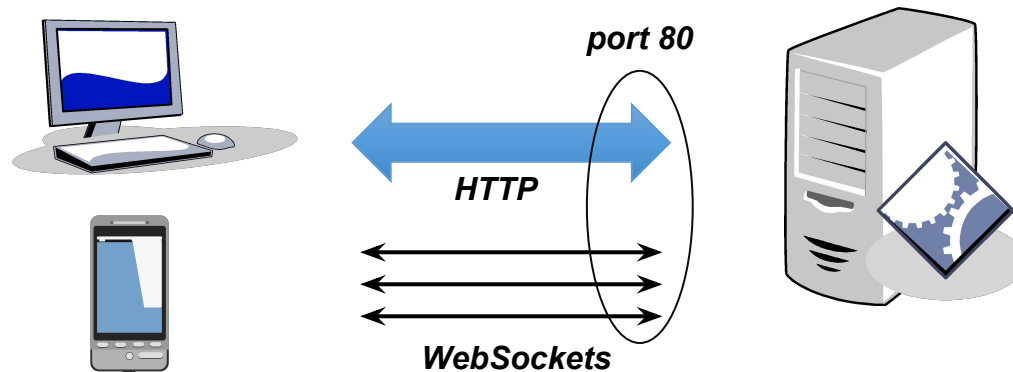
def get_db():
    db = getattr(g, 'db', None)
    if db is None:
        db = g.db = connect_db()
    return db

def init():
    c = get_db()
    c.execute("drop table if exists entries")
    c.execute("create table entries (id integer primary key, name text, message text)")
    c.commit()

def add_message(name,message):
    c = get_db()
    c.execute("insert into entries (name,message) values (?,?)", (name,message))
    c.commit()

def close():
    get_db().close()
```

WebSockets



- Overcome (historic) limitations with the HTTP protocol
 - Backward compatible with HTTP while providing new features
- Full duplex communication
 - Additional channels

WebSockets

- Protocol providing full-duplex communications channels over a single TCP connection
 - Part of the HTML5 initiative
 - Unlike HTTP, WebSockets provide for full-duplex communication
- Designed to be implemented in *web browsers* and *web servers*
- Enables more interaction between browsers and web sites
 - Normally, communications are performed over TCP port 80
- Supported by common web browsers
 - Google Chrome, Internet Explorer, Firefox, Safari, Opera

Other alternatives/ Interval Polling

The client asks for any updates every X interval of time.

Example:

```
var askForUpdates = function () {  
    // Ajax call to the server  
};  
setInterval(askForUpdates, 1000);
```

Disadvantages:

1. A request shall be sent even if there are no updates.
2. The updates at the client-side are not instantaneous.
3. High traffic.

Other alternatives/ long polling(Comet)

The server does hold the connection with the client made by a http request sent from the client. The server does respond when a new update is available. Once responded, the client sends another empty http request for the next update.

Disadvantages:

1. Complex and messy code
2. More resources at the server-side are required.

HTML5/ Server-Sent Events(SSE)

The server can send updates to the client.


Disadvantages:

1. The client cannot send data to the server (Half-duplex).
2. More implementation issues like disconnect detection and overriding headers.

WebSocket protocol handshake


- WebSockets use “upgraded” HTTP connections
- Client request:

```
GET /mychat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHmBDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 12
Origin: http://example.com
```



- Server response:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```



Once the connection is established, the client and server can send WebSocket data or text frames back and forth in full-duplex mode.

WebSocket – client side

- Open a WebSocket connection

```
var connection = new WebSocket('ws://html5rocks.websocket.org/echo');
```

- Attach some event handlers

```
// When the connection is open, send some data to the server  
connection.onopen = function () {  
    connection.send('Ping'); // Send the message 'Ping' to the server  
};  
// Log errors  
connection.onerror = function (error) {  
    console.log('WebSocket Error ' + error);  
};  
// Log messages from the server  
connection.onmessage = function (e) {  
    console.log('Server: ' + e.data);  
};
```

WebSocket – client side

- Communicating with the server

```
// Sending String  
connection.send('your message');
```

Flask WebSocket library – server side

- Straightforward in Flask
- Install:
`pip install Flask-Sockets`
- Import websockets
- Add routing decorations

```
from flask import Flask
from flask_sockets import Sockets

app = Flask(__name__)
sockets = Sockets(app)

@sockets.route('/echo')
def echo_socket(ws):
    while True:
        message = ws.receive()
        ws.send(message)

@app.route('/')
def hello():
    return 'Hello World!'
```


Gevent WebSocket library – server side

- WebSocket library for the gevent Python networking library
- Gevent
 - Based on greenlet for lightweight concurrency
 - Provides event handling and event loop
 - Needs to be installed with pip

Gevent WebSocket echo service

```
from geventwebsocket.handler import WebSocketHandler
from gevent.pywsgi import WSGIServer
from flask import Flask, request, render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/api')
def api():
    if request.environ.get('wsgi.websocket'):
        ws = request.environ['wsgi.websocket']
        while True:
            message = ws.receive()
            ws.send(message)
    return

if __name__ == '__main__':
    http_server = WSGIServer(('', 5000), app, handler_class=WebSocketHandler)
    http_server.serve_forever()
```

Running the server

- Initialize and start server

```
@run_with_reloader
def run_server():
    app.debug = True
    http_server = WSGIServer(('',5000), app, handler_class=WebSocketHandler)
    http_server.serve_forever()

if __name__ == "__main__":
    run_server()
```

Sample standard service

- Server method for logging in (e.g., in Lab 2)

```
def signIn(email, password):  
    c = get_db()  
    res = c.execute("SELECT * FROM users WHERE email='"+email+"' AND password='"+password+"' LIMIT 1")  
    res = res.fetchone()  
    if not res:  
        # Not logged in  
        return json.dumps({"success": False, "message": "Invalid email or password"})  
    else:  
        # Logged in  
        return json.dumps({"success": True, "message": "You are now signed in", "data": token})  
    return None
```

- Routing

```
@app.route("/signin", methods=["POST"])  
def sign_in():  
    return signIn(request.json["email"], request.json["password"])
```

Gevent WebSocket under Linux (IDA)

```
> python -V
Python 2.7.6
> virtualenv test
New python executable in test/bin/python
Installing setuptools.....done.
Installing pip.....done.

> cd test
> bin/pip install flask
Downloading/unpacking flask...

> pip install gevent
...

> pip install gevent-websocket
...
```



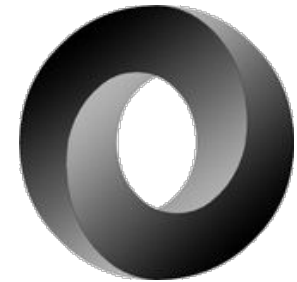
Resources

- Flask-Sockets library
 - <http://kennethreitz.org/introducing-flask-sockets/>
 - <https://github.com/kennethreitz/flask-sockets>
- Gevent-WebSocket library
 - <http://www.gevent.org>

Formatting Data for Transfer

- Several different formats possible
 - Varying complexity
 - Varying library/language support
 - Varying efficiency
- Examples of formats
 - CSV
 - XML
 - JSON
- For the labs: JSON

JSON



- JavaScript Object Notation — JSON
- Compact, text-based format for data exchange
- Easy to read and write (for humans)
- Easy to parse and generate (for machines)
- Language independent
- Code for parsing and generating available in many programming languages (e.g., Java, C++, and JavaScript)
 - Maps well to many programming languages
 - Example: Matches well to a Python dictionary
- MIME type for JSON text: “application/json”

JSON VS XML

JSON is Like XML Because

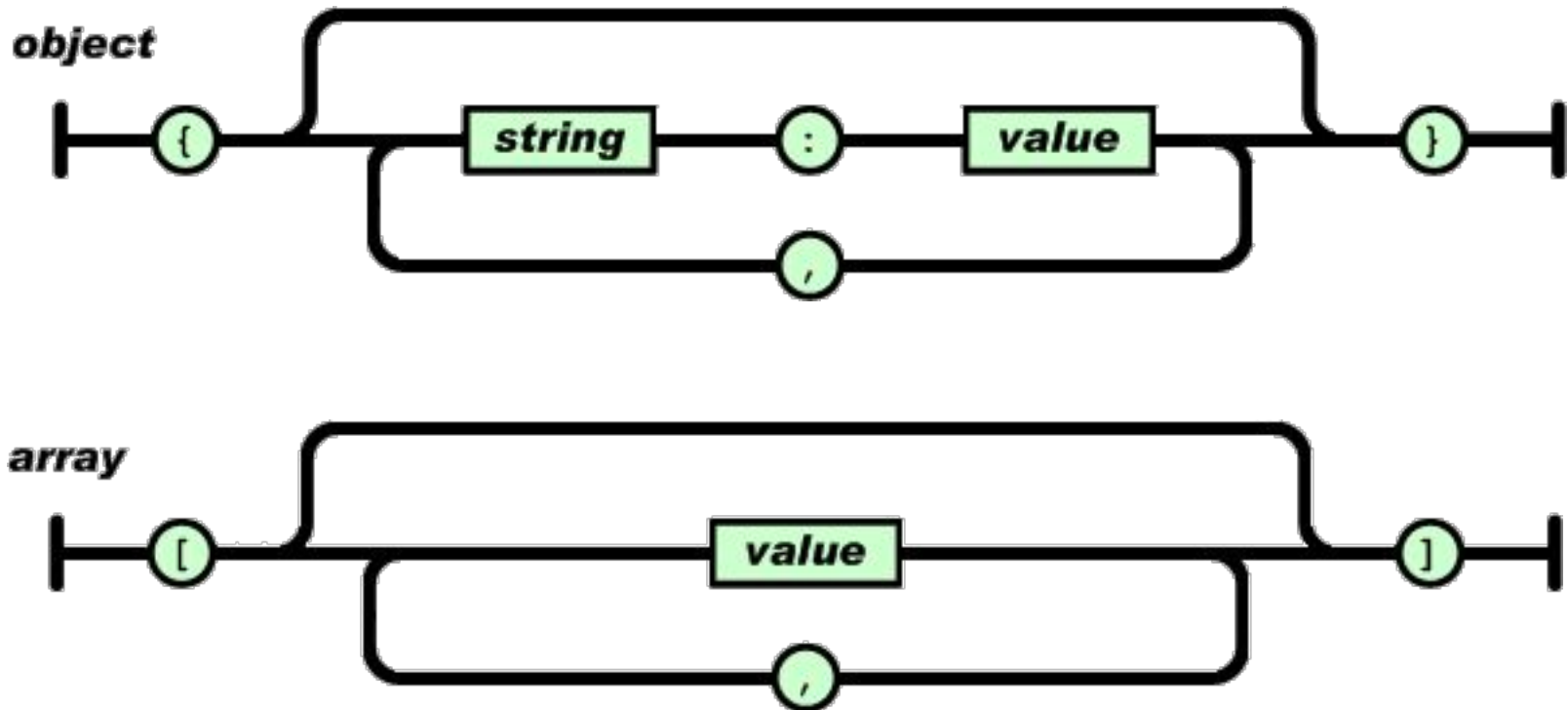
- Both JSON and XML are "self describing" (human readable)
- Both JSON and XML are hierarchical (values within values)
- Both JSON and XML can be parsed and used by lots of programming languages
- Both JSON and XML can be fetched with an XMLHttpRequest

JSON is Unlike XML Because

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays

https://www.w3schools.com/js/js_json_xml.asp

Basic JSON syntax: Objects and arrays



JSON Examples

- Object

```
{  
  "code": "TDDD97",  
  "title": "Web programming",  
  "credits": 6  
}
```

- Array

```
{  
  "courses": [  
    {"code": "TDDD24" , "credits": 4 },  
    {"code": "TDDD97" , "credits": 6 }  
  ]  
}
```

JSON Examples (cont.)

- JSON describing a person

```
{
  "firstName": "John",
  "lastName": "Doe",
  "age": 22,
  "address": {
    "streetAddress": "Drottninggatan 1",
    "city": "Linköping",
    "postalCode": "58183"
  },
  "phoneNumber": [
    { "type": "home", "number": "013-123456" },
    { "type": "mobile", "number": "070-123456" }
  ],
  "newSubscription": false,
  "companyName": null
}
```

JSON in Python

- Sample interactive Python session

```
>>> import json
>>> data = [ { 'a':'A', 'b':(2, 4), 'c':3.0 } ]
>>> print 'DATA:', repr(data)
DATA: [{ 'a': 'A', 'c': 3.0, 'b': (2, 4)}]
>>>
>>> data_string = json.dumps(data) #jsonify(data)
>>> print 'JSON:', data_string
JSON: [{"a": "A", "c": 3.0, "b": [2, 4]}]
>>>
```

JSON functions in Python

- Use JSON
 - `import json`
- Serialize *obj* as JSON formatted stream to *fp* (file)
 - `json.dump(obj, fp, <options>)`
- Serialize *obj* to a JSON formatted string
 - `json.dumps(obj, <options>)`
- Deserialize *fp* to a Python object.
 - `json.load(fp, <options>)`
- Deserialize the string *s* to a Python object
 - `json.loads(s, <options>)`

Examples of JSON applications

- Import and export of data files
- Configure menus and tools
- Create attribute mapping files
- Store any key–value data

JSON validation

- Helpful for validating JSON syntax
- Several on-line and off-line validators available
- JSONLint: <http://jsonlint.com>

JSONLint

The JSON Validator

Want more from JSONLint? Try [JSONLint Pro](#)



A Tool from the Arc90 Lab. [Source is on GitHub.](#)
Props to [Douglas Crockford](#) of JSON and JS Lint and
[Zach Carter](#), who provided the pure JS implementation of jsonlint.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

Enter JSON to validate, or a URL to JSON to validate.

Validate

JSON Lint is an idea from Arc90's Kindling

Kindling

FAQ

JSON and XMLHttpRequest

- Putting JSON and XMLHttpRequest (AJAX) together

```
var xml = new XMLHttpRequest();

xml.onreadystatechange = function() {
    if (xml.readyState==4 && xml.status==200) {
        var serverResponse = JSON.parse(xml.responseText);
        ...
    }
};

xml.open("GET", "test", true);
xml.send(null);
```

The Project(lab 4)

Overview

- Project – learn more concepts, techniques, and technologies
 - Independently search, assess, apprehend, and apply information about new technologies and third-party resources
 - Download, install, configure, and troubleshoot relevant libraries and frameworks
- Extend your Twidder application (from labs 1–3) by implementing different functionality (from a list of alternatives)
- Several alternatives
 - Providing Live Data Presentation
 - Use of HTML5 for Drag and Drop
 - Performing Client-side Routing + Overriding Back/Forward buttons using the History API
 - Third-Party Authentication Using OpenID/OAuth 2.0
 - Applying Further Security Measures
 - Testing Using Selenium
 - Client-side Templating Using a Third-Party API
 - Media Streaming
 - Styling and Responsive Design

Grading

- The course is graded based on lab 4.
- Each successfully implemented criteria gives points, which are added together and translated to a grade.

Total number of points	Grade
3	3
6	4
9 + well-documented code	5

Grading example

- Let us say your project implements
 - Live Data Presentation [3 points]
 - Drag and Drop using HTML5 [1 point]
 - Testing using Selenium [2 points]
- Total points: $3 + 1 + 2 = 6$
- Resulting grade: 4

Important Dates (Deadlines)

Gentle reminder:

- March 19, 2019 – Deadline for *demonstrating* all labs and project
 - Demonstrate on one of the final lab sessions
- March 23, 2019 – Deadline for submitting the *source code* for all labs and project
 - Including making corrections/fixing bugs

Alternative 1

Providing Live Data Presentation

Use case:

Stock market apps, Analytics and in general where the data set is produced by a third party and it needs to be observed visually instantaneously.

Example:

<http://www.jscharts.com/examples>

<https://www.dailyfx.com/usd-sek>

Alternative 2

Use of HTML5 for Drag and Drop

Use case:

Almost can be used in any Graphical User Interface.

Example:

<https://html5demos.com/drag/#>

Alternative 3

Performing Client-side Routing + Overriding Back/Forward buttons using the History API

Use case:

The most common usage is in Single Page Applications where the application is composed of one web-page but multiple views.

Example:

<http://www.camedin.com>

Alternative 4

Third--Party Authentication Using OpenID/OAuth 2.0

Advantages:

Decreasing risk, cost and complexity.

Example:

<https://www.camedin.com>

Alternative 5

Applying Further Security Measures

Use case:

Banking apps and generally where security is a high priority.

Example:

Token hashing and salting with other data in the request.

Alternative 6

Testing Using Selenium

Advantages:

<http://alvinalexander.com/testing/automated-gui-testing-benefits-tools>

<https://www.youtube.com/watch?v=juKRybHPMwE>

Alternative 7

Client--side Templating Using a Third-Party API

Advantages:

Code reusability by defining a view structure for showing changing information. Mostly used in SPAs while being combined with asynchronous calls.

Example:

www.camedin.com

Alternative 8

Media Streaming

Advantages:

No need to download the whole media completely before being able to view it.

Example:

www.youtube.com

www.facebook.com

Alternative 9

Styling and Responsive Design

Advantages:

One GUI which adapts itself to different screen resolutions instead of having different GUI layouts for different screen resolutions.

Example:

www.bbc.co.uk