
CSE 331

Software Design & Implementation

James Wilcox

Autumn 2021

HTTP Servers

HTTP SERVERS

Server Frameworks

- How do we write a modular HTTP server?
 - need to split up the code into multiple classes
- Usual technique is to route requests using the **path**
 - use path to choose class that handles the request
 - used in Java, C++, Python, JavaScript, ...
 - pass data to class using:
 - query string
 - POST body
 - (part of) path

Spark Java

- Simple library for writing HTTP servers in Java
 - not to be confused with “Apache Spark” — very different!
- Give Spark paths and corresponding classes
 - latter are called “routes” in this library
 - server will read the request path and invoke appropriate class
 - info about the request passed in request object
 - response can be written to response object or returned
- Library handles the event loop

Spark Java

```
Spark.get("/path", new MyRoute());
```

- GET request with this path are sent to this object
- **Second argument must implement Route interface**
 - single required method `handle(Request, Response)`
 - that means it can also be implemented with a Lambda

```
Spark.get("/ready", (request, response) -> {  
    return "Nah, I'm busy";  
});
```

Example: Hello Server

`HelloServer.java`

Example: To-Do Server

- Stores a To-Do list
- Clients can retrieve the current list
- Clients can update the list
 - check off an item
 - add a new item

Example: To-Do Server

`ToDoServer.java`

Spark Java

- Many more features
 - simple things are simple
 - complex things are possible
- Simple version is single threaded
 - makes life much easier
 - medium scale would use threads
 - high scale would not use them (see lecture 16)
- Documentation at <http://sparkjava.com/documentation>

HTTP CLIENTS

Client / Server communication

- Original JavaScript API: `XmlHttpRequest`
- Create object call `open` to configure
 - pass in GET / POST, path, and `async = true`
- Listen for response event
 - `onload` invoked when done
 - `responseText` contains the response body string
- Call `send` to start the request
 - for a POST, pass in the request body
 - for GET, pass `null`

Example: To-Do Client

TodoApp.tsx

Client / Server communication

- Original JavaScript API: `XMLHttpRequest`
- Newer APIs discussed in section
 - fetch API returns a Promise object
 - widely used in JS programming these days
 - works well for *sequential* reqs: start task 1, wait for result, start task 2, wait for result, start task 3, wait for result
 - works well for *parallel* reqs: start tasks 1–3, wait for all
 - `async / await` JS keywords automatically create promises
 - write sequential code in one block
 - compiler will split into separate pieces

Client / Server communication

- By default, client can only talk to the server from which the code was loaded
 - same machine and same port
 - “same origin” policy
- For development, we often want to split do this
 - `npm` runs a separate server that recompiles client code
 - can allow cross-domain requests in the Java server
 - example code does this
 - can set up recompiling server to forward these requests
 - (annoying but we’re stuck with it)

Debugging

- Network tab in Chrome shows every request
 - full details of request
 - path, headers, etc.
 - full details of response
 - status code, response body, etc.
 - timing information