

Application Layer (VI)

Node.js Example 3 (cont.)

- Client side AJAX

- send and retrieve data from a server asynchronously without interfering with the display and behavior of the existing page (open form3.html using browser to illustrate).
- It's common to use GET in AJAX unless sending a sensitive data or calling script processing data on the server (open form3.html using a text editor)

```

<script>
  $("form").submit(function(f) {
    f.preventDefault();
    $.ajax({
      url: "/request",
      data: "q="+$("#q").val(),
      type: "GET",
      success: function(data) {
        var results = JSON.parse(data);
        $("#result").html(results["result"]);
      },
      error: function(e) {
        alert("Error: "+e.status+": "+e.statusText);
      }
    });
  });
</script>

```

GET /request?q=test HTTP/1.1

{"result": "server response"}

- Server side

- Since we want to display the user entered string in the Results table, we need to fetch the string from the query part of the url, and then send it back with the HTTP response.
- Once the input is submitted, the pathname becomes "/request." So, we can use it as a trigger to parse the query. If the pathname is not "/request," send the required file back to the client. (open server3.js)

```

if (pathname == '/request') {
  var query = url.parse(req.url).query;
  console.log("query " + query + " received.");

  resp.writeHead(200, {'Content-Type': 'text/html'});
  resp.write('{"result":"' + query.substring(2) + '"}');
  resp.end();
}

```

- Run server3.js and open form3.html on web browser.

Node.js Example 3 Extension

- In this specific example, the server doesn't need to do anymore computation but send the query string back to the client. However, sometime, the server side needs to calculate the results and then send the results back. E.g., the server may need to search into a database or run some program to calculate the results. Then, we can let the server spawn a child process to do the calculation and then send the result back when the calculation is done.
- Using the same example, we can let the server executes an echo command that prints the stdin string to the stdout. The server echos the query, and once it's done, send the stdout string to the client side.

```
if (pathname == '/request') {  
  var query = url.parse(req.url).query;  
  console.log("query " + query + " received.");  
  
  // Way 2: generate a child process to produce data and send back  
  var spawn = require('child_process').spawn;  
  var cp = spawn('echo', ['{"result":"' + query.substring(2) + '"}']);  
  cp.stdout.on('data', function(data) {  
    console.log(`child process stdout:\n${data}`);  
    resp.writeHead(200, {'Content-Type': 'text/html'});  
    resp.write(data.toString());  
    resp.end();  
  });  
}
```

Disclaimer: Notes adapted from the textbook and online resources.