CIT 382
Web Dev II

Week 2

Winter 2017

Phil Colbert

Debugging

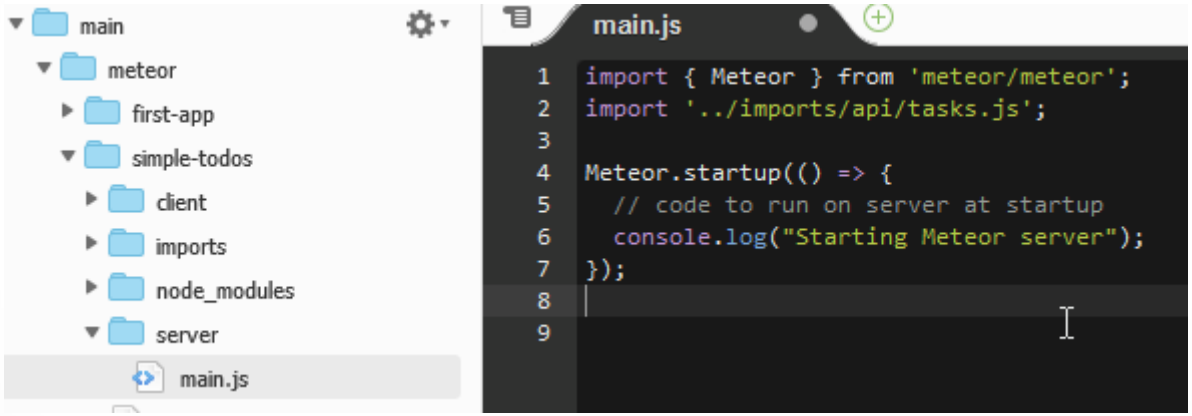
- The Meteor tutorial is an excellent opportunity to practice debugging your application
- Remember that our application is composed of a client components that run in the browser, and server components that run on the server
- When debugging you have a number of options depending on whether you're using the Cloud9 environment, or debugging from your own computer

Log to Console

- For the client or the server code, you can always use console.log to send output to the console
- For server-side code, the output will be displayed in the window where you launched your Meteor application
- For client-side code, the output will appear in the console of the browser

Cloud9 Server Console

- Below is the modified server code to output to the console and the output



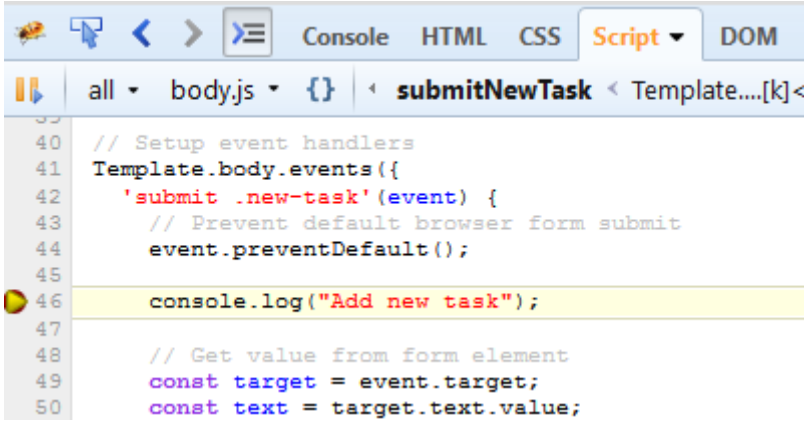

```
W20170118-17:20:23.652(0)? (STDERR) In the root director
I20170118-17:20:23.788(0)? Starting Meteor server
=> Meteor server restarted
```

Client-Side Debugging

- The easiest way to debug client-side code is to use debug features built into your browser in addition to logging information to the browser console
- The list of client-side JavaScript files can be extensive, but you should be able to find your original files in the list
- Let's debug the Meteor Todos tutorial application

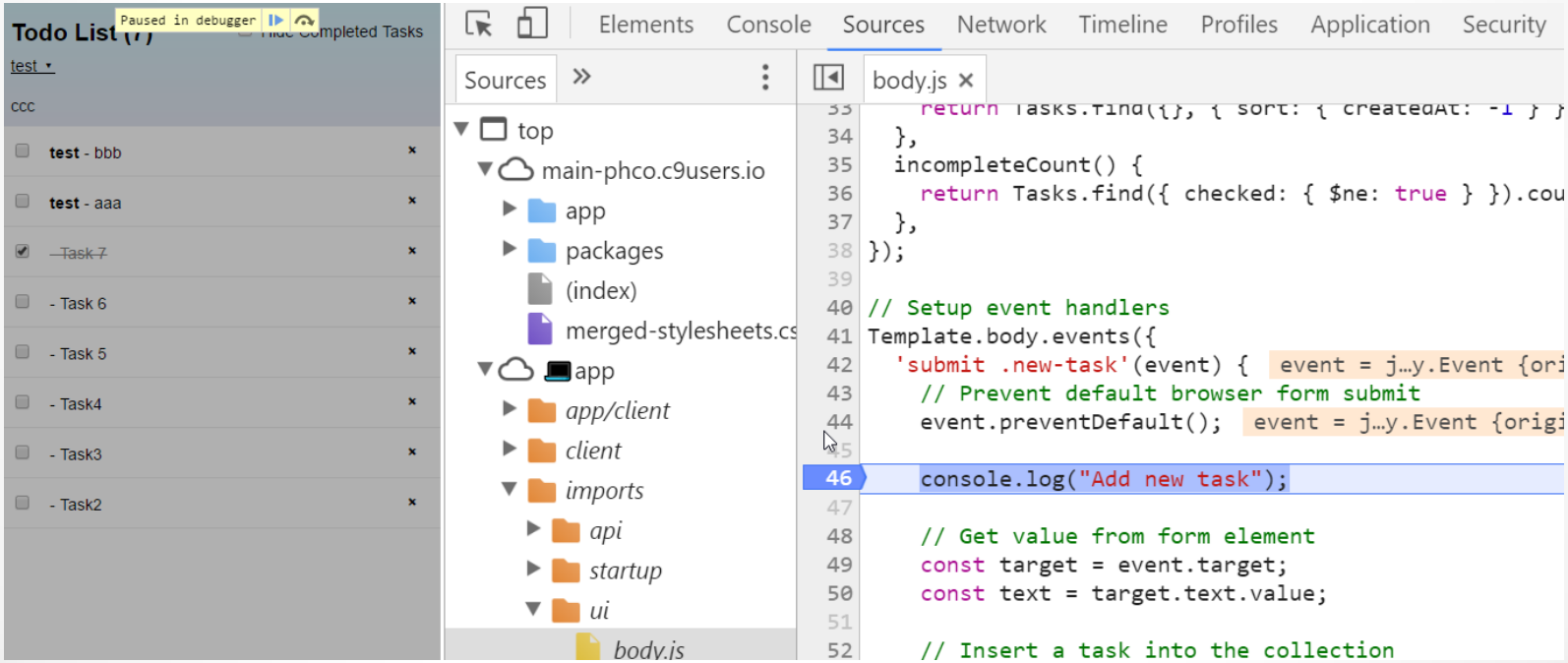
Breakpoints

- Below is an example of debugging using Firebug in Firefox



Breakpoints

- Below is an example of debugging using Developer Tools in Chrome



Debugger

- You can also insert the stand-alone command debugger into your client-side code to force a breakpoint
- The behavior of this command depends on your browser and whether or not you have a development console open
- With the browser console open, any debugger statements that are found while executing code will immediately halt execution at the debugger line of code like a breakpoint

Debugger

- Note the code halt in the code below

```
4
5   export const Tasks = new Mongo.Collection('tasks');
6
7   Meteor.methods({
8     'tasks.insert'(text) {
9       check(text, String);
10
11      debugger;
12
13      // Make sure the user is logged in before insertin
14      if (!this.userId) {
15        throw new Meteor.Error('not-authorized');
16      }
```

Server-Side Debugging

- Using Cloud9 you have access to an IDE (Integrated Development Environment) with an excellent GUI (Graphical User Interface) for the server components
- If you want more capability than logging statements to the console, Meteor comes with a couple of options
  - node-inspector
  - Repl
- The article *Easily Debugging Meteor.js* illustrates the debugging options, but please note that currently I have not had success getting any of the options to work

Making Changes

- You can make changes to a running Meteor application
- When a change is made you will see the running Meteor application indicate the software is refreshing
- When making changes as you work through the tutorial, some changes will require stopping the running application to prevent the application from crashing
- Recommendation: Stop the running application with each new step, and restart the application once the changes are completed for that step

Software Testing

- As part of the tutorial you were instructed to install the Mocha JavaScript test framework
- The ability to test your software is an import part of software development
- Testing software is commonly broken into the following categories:
  - Unit testing
  - Integration testing
  - Functional testing
  - Stress and load testing
  - Acceptance testing

Unit Testing

- Unit testing involves testing the smallest components of your software possible
- Unit testing typically involves creating methods or functions to essentially "exercise" the units of your software, and testing these components as isolated, single units
- Unit testing typically involves creating test scaffolds
- As a scaffold is a real-work device for supporting workers, a software scaffold is a structure to setup and implement code to support the testing of your software

Integrated Testing

- As the name implies, integrated testing relates to testing the interaction of software unit components
- Integrated testing is still performed in isolation to avoid inheriting any problems from other units
- Integrated testing attempts to determine if software components or units perform correctly as a group

Functional Testing

- Functional testing involves removing any concept of unit or group testing, and testing the applications functionality
- Functional testing attempts to test software use cases
  - Use cases are a list of actions, steps, or events that define the interaction between different components of a system that typically mimic real-world uses of software

Stress and Acceptance Testing

- Stress and load testing is exactly what the name implies: testing to determine if the software can continue to run despite certain stress or load conditions
  - Stress and load testing is typically considered non-functional testing
- Acceptance testing determines if the customer accepts the performance of the software, and that the software delivers the expected functionality

MongoDB

- The MongoDB (or just Mongo) is a NoSQL database that stores documents in collections (databases)
- Meteor by default uses Mongo
- Let's reference a number of online resources to work with either your own Mongo instance that's part of your sample app, or with an online Mongo database
  - If you elect to use your sample app Mongo on Cloud9, your sample application will have to be running before Mongo is available
  - Open a new terminal window in Cloud9 and issue the command meteor mongo to access the Mongo command line shell

MongoDB

- Helpful MongoDB (Mongo) web pages
  - Mongo Shell
  - Mongo Query
  - Mongo CRUD operations
  - Mongo Query and Projector Operators
- Online Mongo test database
  - MongoDB Terminal Online
  - Interact with online local database commands
    - db.help()
    - show dbs
    - db.local.count()
    - db.local.insert({test: 'abc'})
    - db.local.find()
    - db.local.find({a: "1"})

Sample Mongo Data

- The MongoDB website contains sample data that can be used to generate a users collection
- Unfortunately, the online terminal doesn't support cut and paste so the only option is to attempt to create a file and upload

Database Cursors

- Many databases (SQL and NoSQL) reference the concept of a cursor
- A database cursor is essentially a control structure that enables traversal over the records of the database
- Database cursors are usually the result of a query, so the specific records may be limited based on the nature of the query (SQL or NoSQL)

Mongo Queries

- The db.*collectionName*.find() method allows querying a Mongo collection and returns a cursor with the matching documents
- The .find() method has two optional fields
  - The query filter to specify which documents to return
  - The query projection to specify which fields in the matching documents to return
- The .find() method employs object literal name value pairs as part of the query

Query Operators

- Mongo includes a number of query operators

Name	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$in	Matches any of the values specified in an array.
$nin	Matches none of the values specified in an array.