PEACOCK

Jasvir Nagra / @jasvir
Marianna Bezler/@maribezler
Web Application: Perception

DELIVER A FASTER, SAFER, AND MORE PROFITABLE DIGITAL EXPERIENCE
Web Application: Realty

Servlets, databases etc  Routers, CDNs, load balances  Browser
xhr = new XMLHttpRequest();
xhr.open("GET", "/conf.json");
xhr.onreadystatechange = ...;
xhr.send();
Why Us?

Why are we the right people to talk to you about this problem?
Takeaway

Web applications consist of communicating event loops that connect asynchronously

Humans reason synchronously

We will show a series of tips and tricks to bridge this mismatch and a way to more systematically think about this problem
How People Debug: Reality

- add `console.log` statements everywhere and then grep
- singlestep through a program
- put `debugger` statements everywhere
setTimeout(function () {
    window.addEventListener('load', function () {
        console.log('onload');
    });
}, 10)
Better debugging with “controlled” debugger

```javascript
function SomeFunction() {
    return Math.random() > 0.5;
}

for (var i=0; i < 100; i++) {
    var rnd = SomeFunction() ? 1 : 2;
    debugger();
}

alert('open console');
alert('debugger');
alert('debugger');

for (var attr in window) {
    if (attr.indexOf("TogetherJSConfig_on") === 0) {
        attrName = attr.substr("TogetherJSConfig_on.", length);
        globals[attrName] = window[attr];
    } else if (attr.indexOf("TowTruckConfig_on") === 0) {
        attrName = attr.substr("TowTruckConfig_on.", length);
        TogetherJS.config[attrName, window[attr]];
    } else if (attr.indexOf("TowTruckConfig_on") === 0) {
        attrName = attr.substr("TowTruckConfig_on.", length);
    }
```
Special case of XHR
How people are debugging XHR

Browsers Network tab can provide visibility on:

- who requested
- request/response headers and content
- send/set cookies

But it’s missing state at the moment when the request was send.
Tips on debugging this case

Customize columns in network browser tab:

- Add cookies and setcookies
- Add custom headers with the information about the state
Special case of promises
What are Promises?

```javascript
new Promise((resolve, reject) => { ... })
  .then( result => { ... })
  .catch( error => { ... });
```
Unhandled promises

```javascript
function setupPromiseErrorHandler(p) {
    setTimeout(() => p.catch(error => console.log(error)), 0)
}

function asyncFunc(url) {
    return fetch(url);
}

let promise = asyncFunc('london.hello/sunshine')
    .then(encrypted => rot13(encrypted))
    .then(decrypted => console.log(decrypted));

// Setup catch handler
setupPromiseErrorHandler(promise);
```
So what if it fails?

Chrome and Firefox dev tools:

- log unhandled rejections as an exception with some stacktrace

Nodejs?

(node:98543) UnhandledPromiseRejectionWarning: Unhandled promise rejection (rejection id: 2):
ReferenceError: a is not defined

(node:98543) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In the future, promise rejections that are not handled will terminate the Node.js process with a non-zero exit code.
Tips on debugging this case

Unhandled rejection event listeners (only in Chrome from v49)

window.addEventListener("unhandledrejection", (event)=> {
    console.warn(`WARNING: Unhandled promise rejection. Reason: ${event.reason}`);
});

Use Bluebird promise library that has a richer API and was built with debugging in mind.
Case of Postmessages
What are Postmessages?

The `window.postMessage()` method safely (sort of) enables cross-origin communication.

```javascript
receiverWindow.postMessage(data, target);

window.addEventListener("message", (event) => {
  console.log(event.origin, event.target, event.data)
}, false);
```
One more problem with Postmessages

Sometimes you receive a message and want to know where it came from:

```javascript
window.addEventListener("message", function (event) {
    if (event.origin === expectedOrigin) {
        /* do something */
    } else {
        /* not interesting messages */
    }
});
```

You can check:

- Event.origin - but multiple iframes with the same origin
- Event.target - but you can’t see the target.location from cross-origin frames

Workaround: Compare Event.target to every contentWindow on the page!
Tips on debugging this case

Special debugging event listener.

```javascript
window.addEventListener("message", (event) => {
    console.log(`Received a message ${event}`);
});
```

If all you’re doing in a debugger is logging, you can also:

```javascript
monitorEvents(window, "message");
monitorEvents(window, "mouseup");
monitorEvents(window, "mouse");
```

This gets noisy even quicker!
Programmer’s Cockpit
Systematic Debugging of Async code

- Collect all APIs that are called during an event loop and their arguments
- Let a human augment this database with facts they have learnt while debugging
- Then be able to ask queries of this database to see what things are true or potential sources of ideas to explore
- Finally, be able to visualize asynchronous program state

We want a programmer’s cockpit where you can sit back see the execution space of your program
What To Collect: DOM Apis

- All state changes
- All “system calls”
- Component interactions
How To Collect: Virtualization

- Replace DOM apis with functions with:
  - Same signature
  - By default call the original function

- Allows us to:
  - Intercept all the calls an application makes
  - Optionally change the inputs and outputs
  - Optionally prevent the original from being called at all

- For debugging, a “poor man’s” version suffices!
How Do We Virtualize The DOM

- Simple - replace the original with a fake!

```javascript
var original = document.createElement;
document.createElement = function(type) {
    LOG(this, 'createElement', arguments);
    return original.apply(this, arguments);
};

var s = document.createElement('script');
```
Instart Logic’s Approach to Virtualization

- Reverse proxy
- Nanovisor virtualization layer is injected as content is proxied
- HTML, CSS, JS & other resources are analyzed and rewritten
“Poor Man’s” Challenges

- `document.location/top.location`
  - Frozen and cannot be modified on intercepted for security reasons

- `window.postMessage` - magical because:
  - It’s the only function callable cross-origin
  - Browser adds “event.target” which is the caller’s window

Unfortunately both these are commonly used
How Do You Query The Database

- SQL for querying can be complicated

```sql
SELECT COUNT(*), method, arguments FROM api
WHERE method = "postMessage"
AND arguments = (SELECT method, arguments FROM api
WHERE method = 'onmessage');
```

Need some better pattern matching system…
Prolog

Constraint satisfaction programming language

- Lets you create a database of facts; and
- Search over them efficiently

father(luke, vader).
father(miniluke, luke).

grandpa(Boy, Man) :- father(Boy, Dad), father(Dad, Man).

?- grandpa(miniluke, Who).
Who = vader.
Applying Prolog to the problem

- Virtualization lets you intercept all the DOM api calls
- Stream all these DOM api calls & meta-data to a prolog database
- If we collect everything that happened as a series of Prolog facts,
  - We can search over these database of facts in a very intuitive fashion

```
happened(postMessage, ["Hi", "*"], "http://g.com/", "frame#1",
startTime, endTime,
special(crosswindow)).
```
Example

Transform into even more readable way for simple querying:

```
postMessage(Args) :- happened(postMessage, Args, _, _, _, _, _, ...).
```

What are the all postmessages?

```
?- postMessage(X).
X = ["abcd", "*"] ;
X = ["hello", "*"] ;
X = ["sunshine", "http://www.receiver.com"].
```
Example

Do we have any postMessages that were sent but never received?
?- forall(postMessage([X, _]), not(onmessage([X]))).
  true

Lets list them all:
?- setof(X, postMessage([X, _]), AllPostMessages),
   setof(A, onmessage([A]), AllOnMessage),
   subtract(AllPostMessages, AllOnMessage, Missing).

Missing = ["abcd"] .
Other Queries

- Functional bugs and correctness
- Performance problems
- Security problems
Visualization

- It’s not enough to have a database of facts
- Humans are visual debuggers
- You want to get overall sense of the flow
Simple visualizer

We can bring data to life using D3.js. It’s well documented library with lots of capabilities and examples in a wild.

crossWindow.postMessage("hello", "");
setTimeOut(function () {
    sendXHR();
}, 100);
alert(document.location);
crossWindow.onmessage = function (evt) {
    el.innerHTML = (document.location +
    document.cookie);
}
Many different visualizers

- Opentracing.io and zipkin
  - This is about timing
- Found one for Promises
  - [http://shlomiassaf.github.io/PromiseTree/playground/](http://shlomiassaf.github.io/PromiseTree/playground/)
- Even more powerful things
  - Since you have causal relationships, you can (if you find the right visualizer), visualize causal relations - which is really the thing you want
More to come

Prolog

● Library of common functions
● Some common ideas aren’t clean expressible because prolog

Linq

● Runs in the browser and might allow interactive debugging

Client/Server

● Extending the idea to logging client and server
Holistic view of a distributed world

- Servlets, databases etc
- Routers, CDNs, load balances
- Streaming database of facts
- Visualizations

Browser