Debugging Mobile Web Applications with **weinre**

for OSCON 2011

Patrick Mueller
IBM Research Triangle Park, NC

@pmuellr

copy of this presentation available at:

http://muellerware.org/papers/

underlined text in this presentation is a link to a web page

resources

- documentation - http://phonegap.github.com/weinre/
- source / issue tracker - https://github.com/phonegap/weinre
- discussion - http://groups.google.com/group/weinre
- Chrome Developer Tools - http://code.google.com/chrome/devtools/docs/overview.html
  - this doc gives a good overview of Real Web Inspector

“weinre” pronunciation

- like “winery”
- not like “weiner” or “wiener”
- I don’t really care

deconstruct
project / presention name

- weinre === WEb INspector REmote
  - reuses WebKit's Web Inspector user interface
  - works remotely - debug a web page running on a device from your desktop
- mobile web applications - web pages running in a mobile browser or in a native mobile app using a browsing control (eg PhoneGap)
- debug those web applications

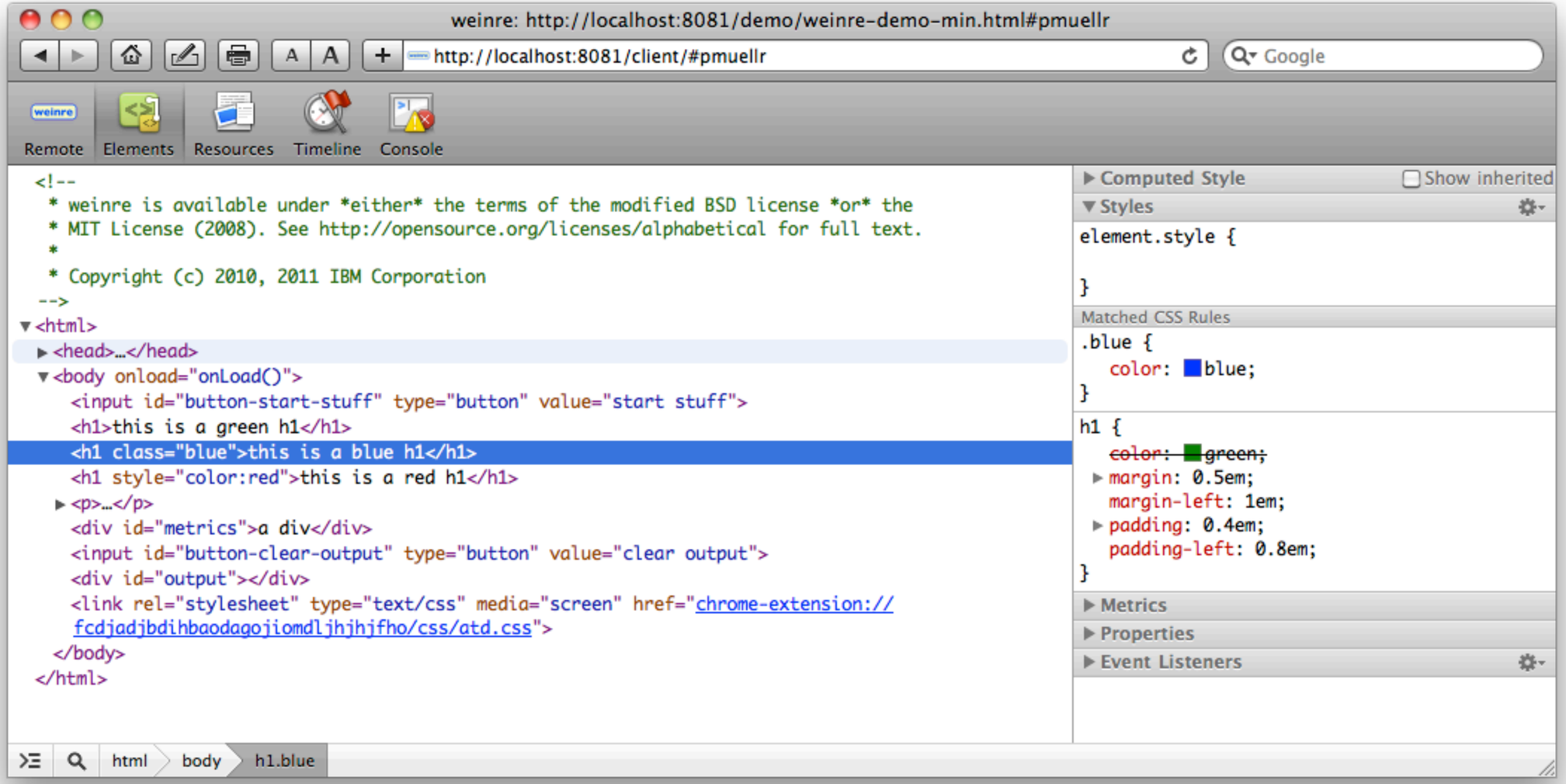
how does it work?

- access a public weinre server or run your own
- add a `<script src=>` into pages you want to debug; the src attribute points to a `.js` file provided by the weinre server
- run the web application on your device
- access the debugging user interface (a web page) from the weinre server from your desktop
- debug!

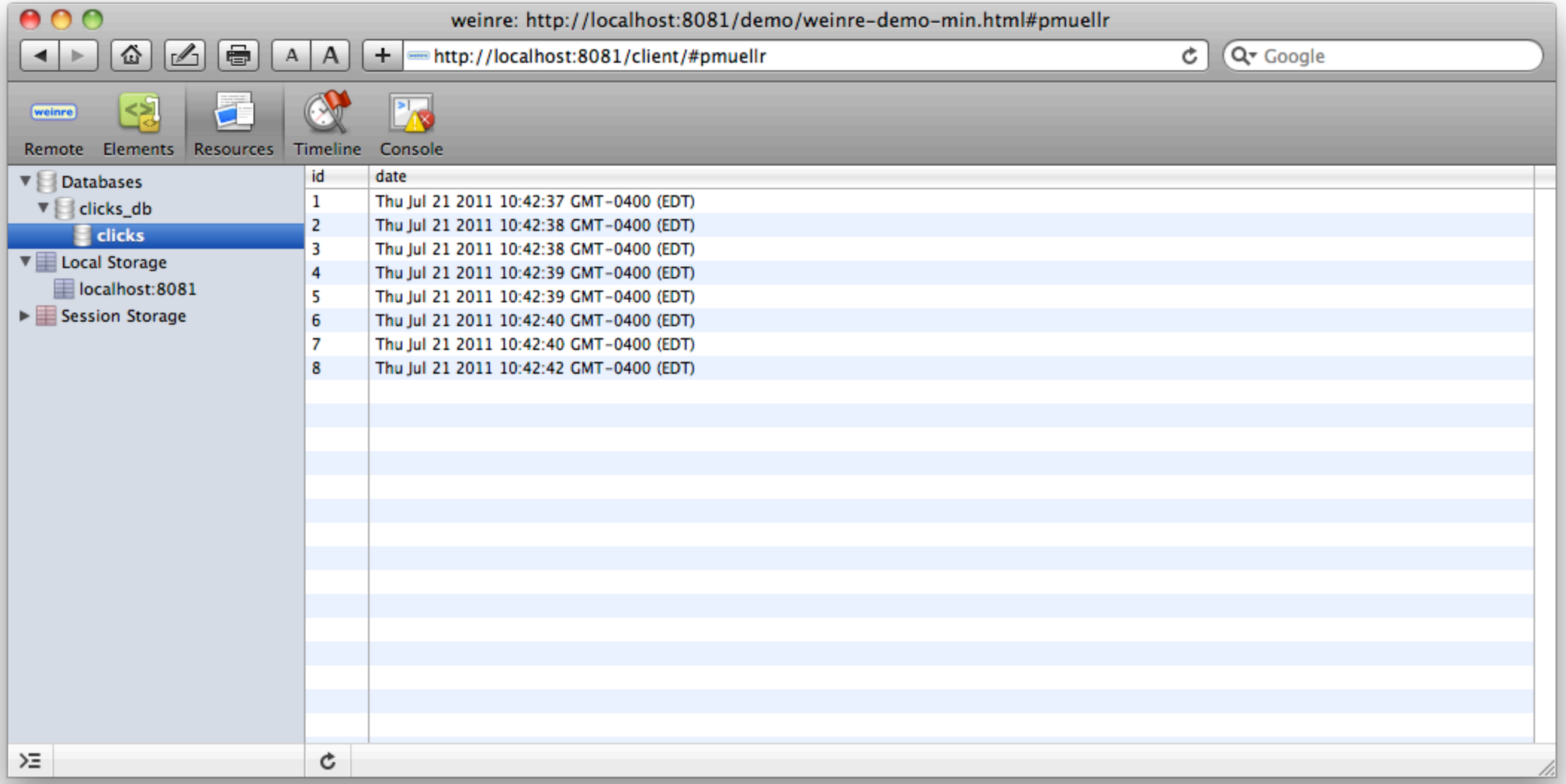
supported features

- DOM / CSS inspector
  - inspect / edit / delete DOM elements and CSS rules
- localStorage / WebSQL inspector
- event timeline
  - add your own events to the timeline
- console
  - run arbitrary JavaScript code in your web page

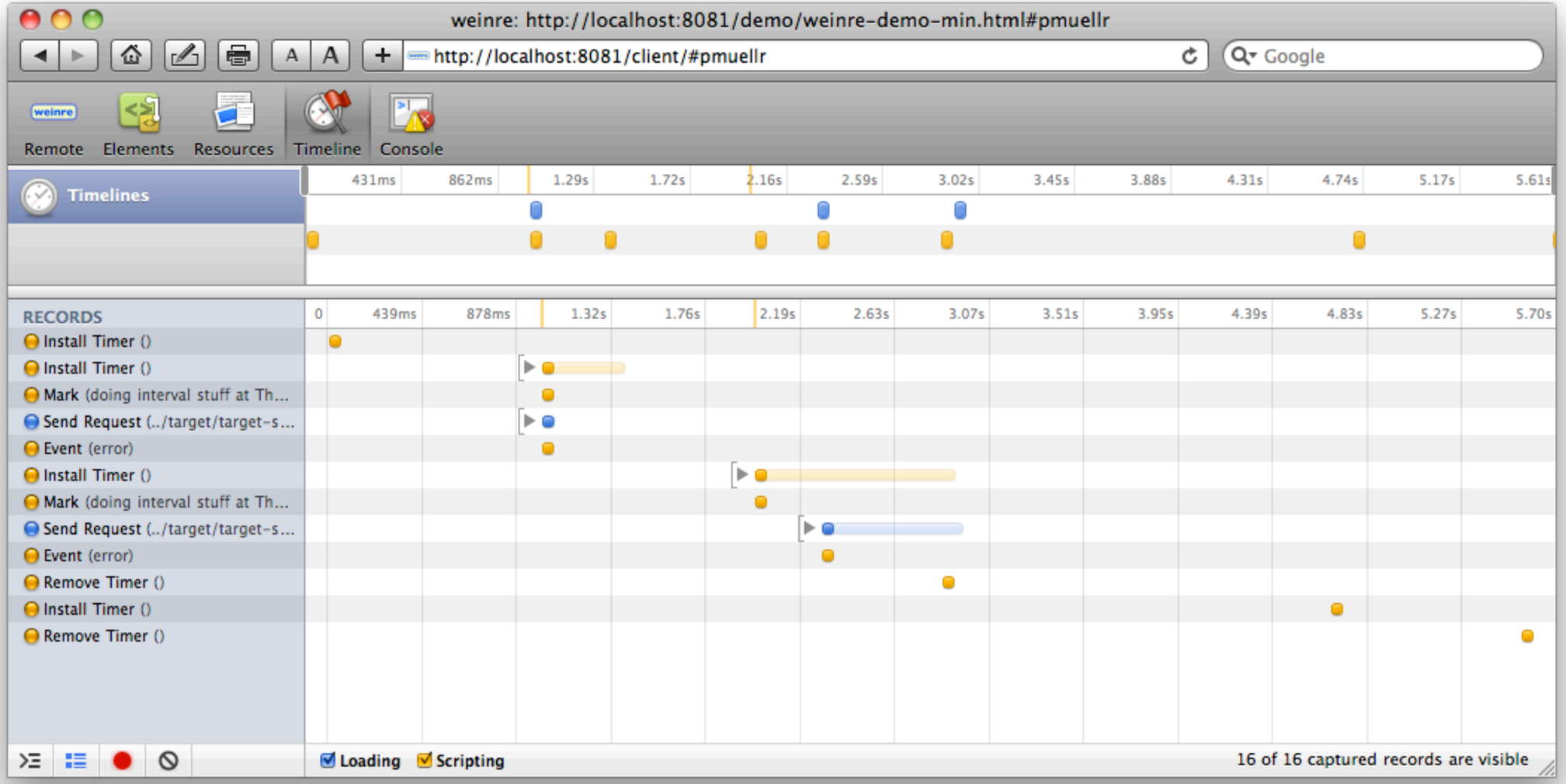
DOM / CSS inspector



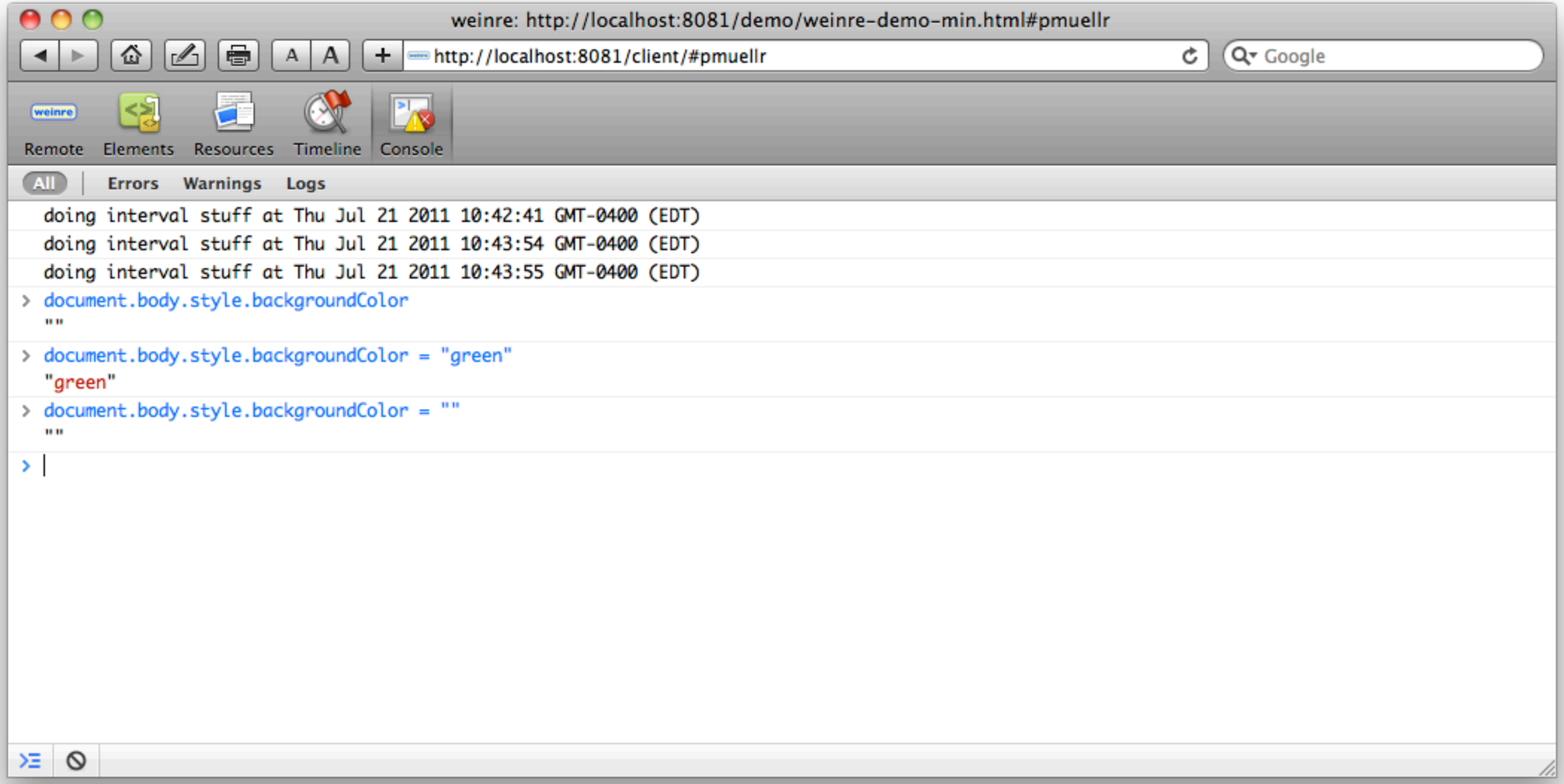
sql / localStorage inspector



event timeline



console


weinre: http://localhost:8081/demo/weinre-demo-min.html#pmuellr
http://localhost:8081/client/#pmuellr
Google
Remote
Elements
Resources
Timeline
Console
All
Errors
Warnings
Logs
doing interval stuff at Thu Jul 21 2011 10:42:41 GMT-0400 (EDT)
doing interval stuff at Thu Jul 21 2011 10:43:54 GMT-0400 (EDT)
doing interval stuff at Thu Jul 21 2011 10:43:55 GMT-0400 (EDT)
> document.body.style.backgroundColor
""
> document.body.style.backgroundColor = "green"
"green"
> document.body.style.backgroundColor = ""
""
>

not supported

- JavaScript debugging; no breakpoints / pausing / stepping
- most of the networking diagnostics
- most of the resource diagnostics
- profiling
- audits

why is <XYZ> not supported?

- in some cases, the work has not been done yet
- in other cases, not possible or very hard, usually because:
  - weinre is written using plain old JavaScript
  - no JavaScript APIs for breakpoints/stepping JavaScript code
  - no JavaScript APIs for low-level resource information

demo

find a demo on YouTube:

http://www.youtube.com/results?search_query=weinre

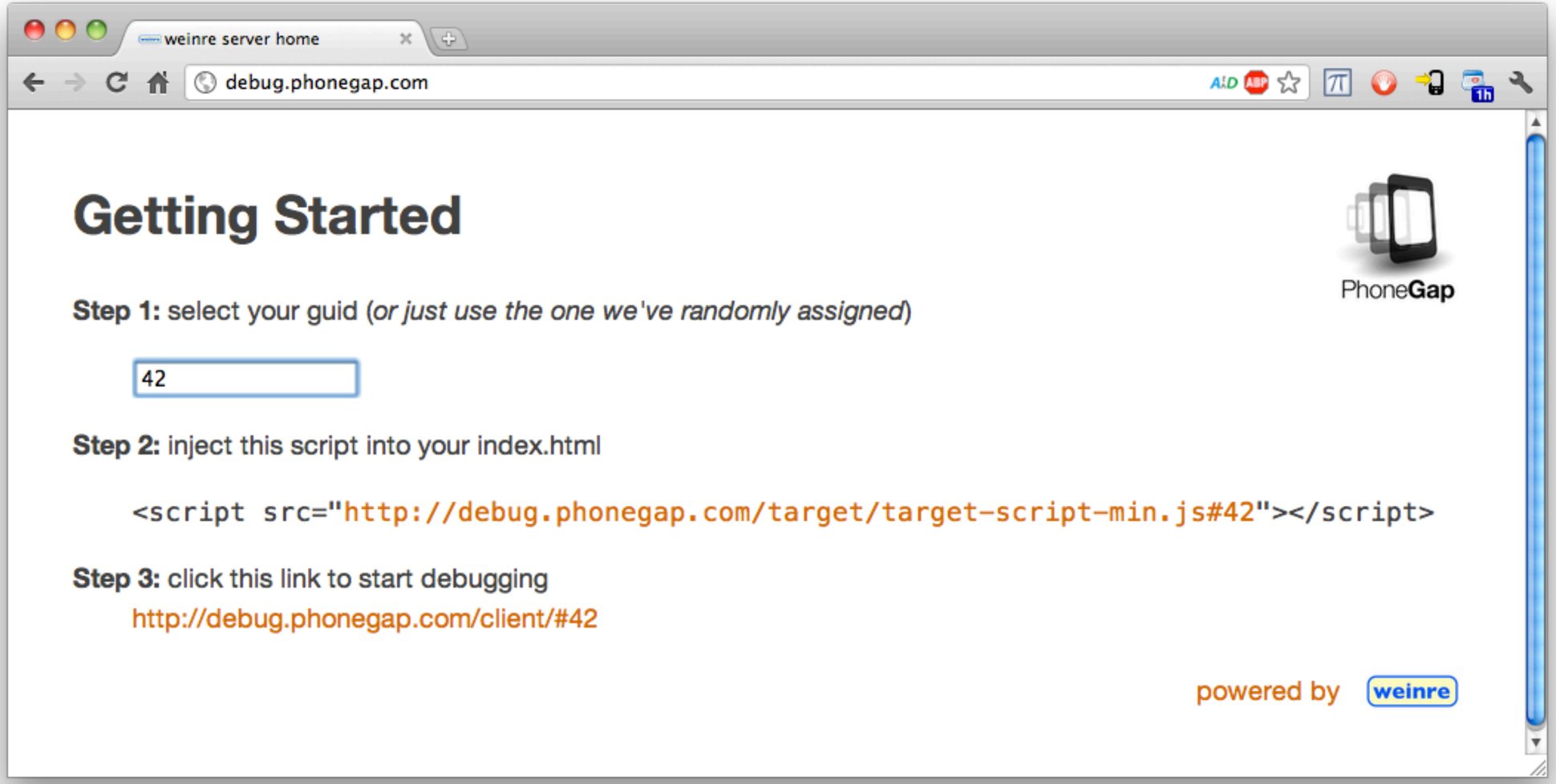
terminology

- debug target - the web page you want to debug
- debug client - the web page showing the Web Inspector user interface
- debug server - the HTTP server which services as a message switchboard between the debug target and debug client

using weinre

- Two options:
  - use debug.phonegap.com, hosted by Nitobi (thanks!)
  - download the server and run it yourself

debug.phonegap.com



pick a guid / unique id

- weinre does not use any kind of security between debug clients and targets
- the unique id keeps other people's debug clients from connecting to your debug target
- if you want to collaboratively debug, share your unique id with a colleague

run your own server

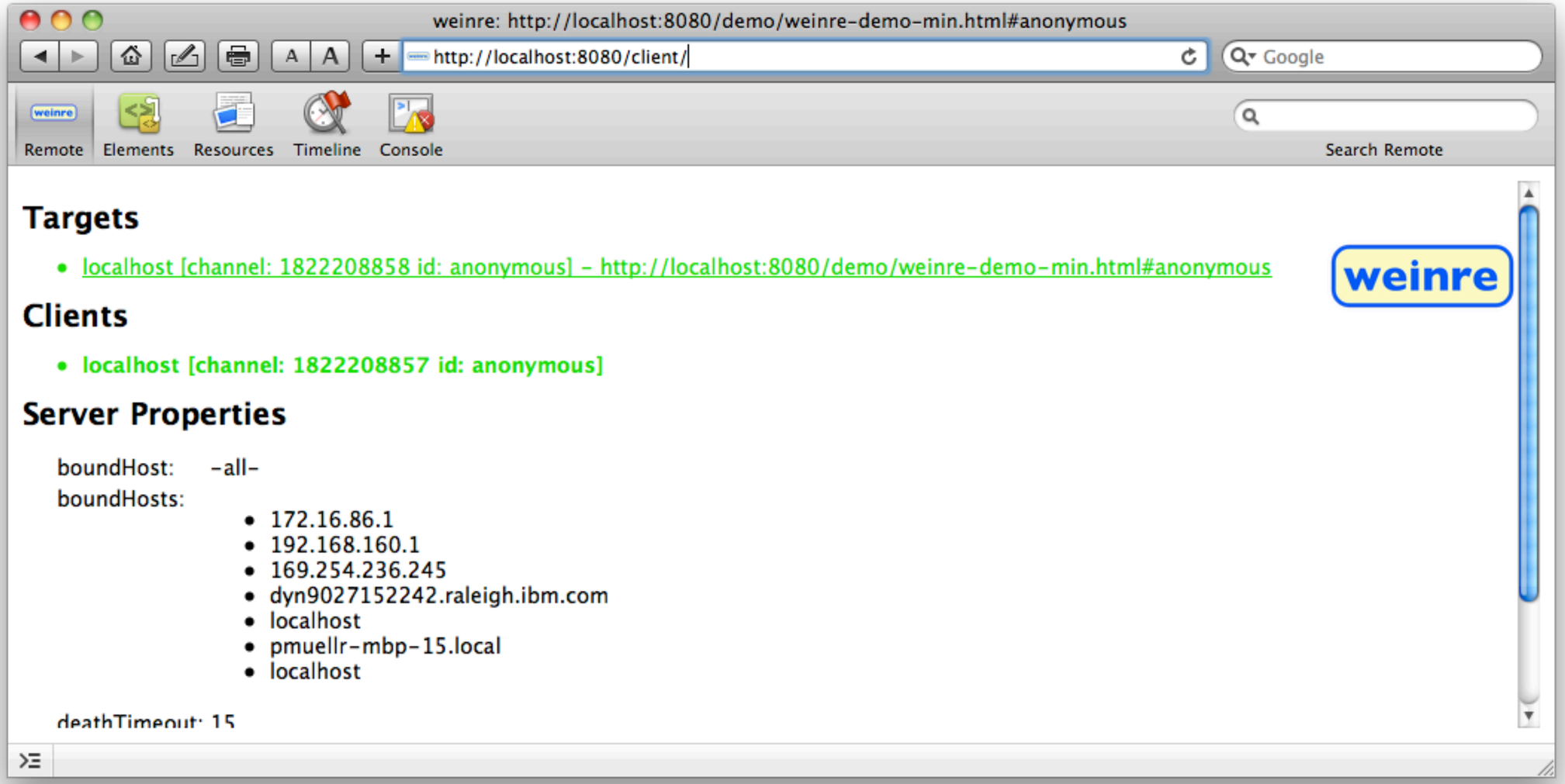
- download / unzip the “jar” build from https://github.com/phonegap/weinre/downloads
- run “`java -jar weinre.jar --boundHost -all-`”
  - requires Java
- add the following to your web page, and reload it:

```
<script src="http://[server-ip]:8081/target/target-script-min.js">
</script>
```

run your own server (continued)

- browse to http://localhost:8080
- the “Remote” panel should list your web application in green
- that means it’s connected
- start debugging!

connected to your server



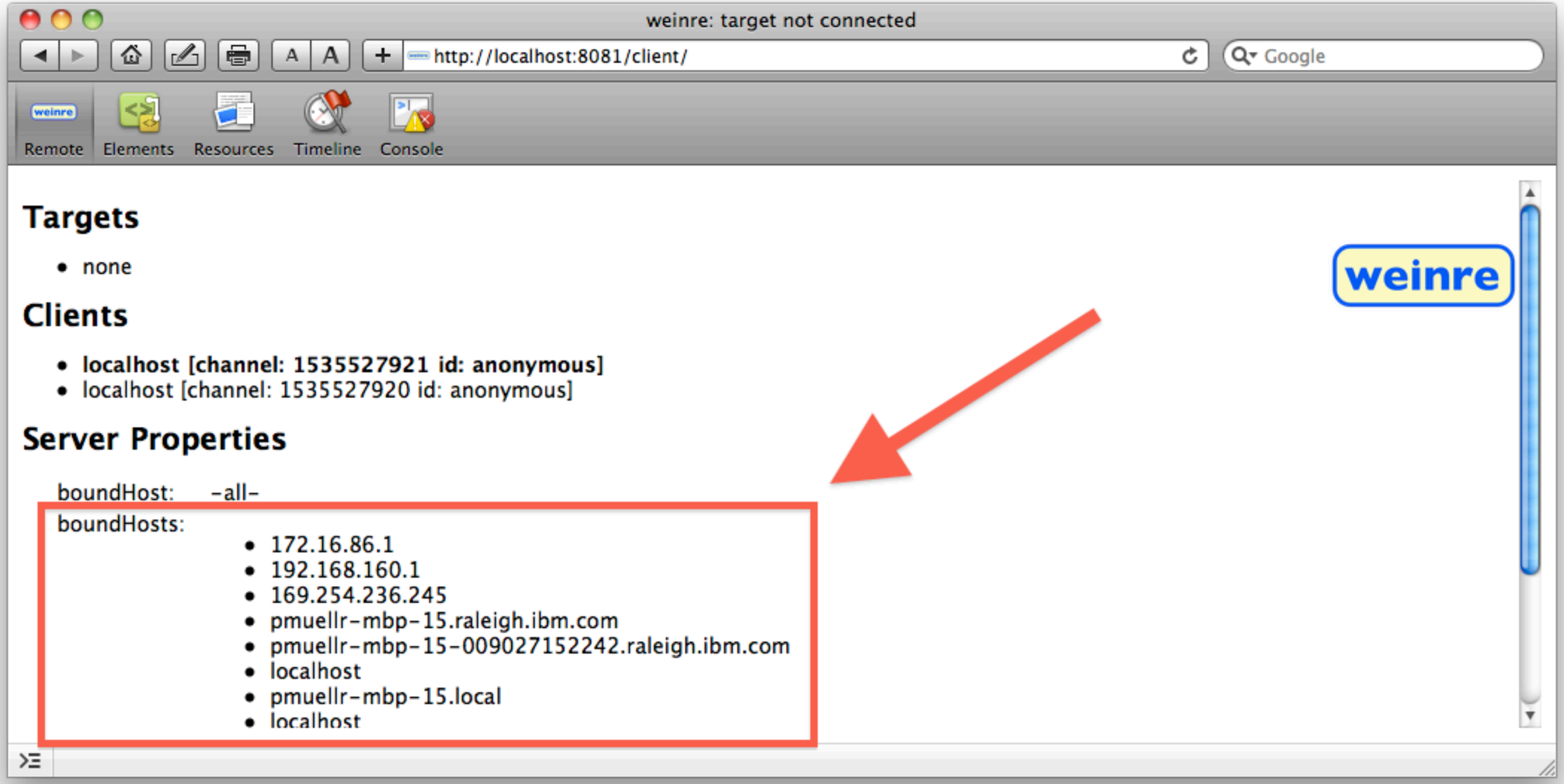
hard part using your own server: the server's ip address

- aka "bound host"
- `--boundHost -all-` option allows all ip address on the box to act as server; default is localhost
- server's ip address goes in the `<script src=>` element embedded in your web application
- that ip address must be reachable from your device to your server
- probably not `127.0.0.1` or `localhost` (maybe for emulator)

what's your server's ip address?

- Windows command line: `ipconfig`
- Mac/Linux command line: `ifconfig`
- weinre on your desktop: `http://localhost:8080/client`

weinre knows your bound hosts



problem: your ip address changes solution: dynamic dns service

- your ip address probably changes every day
  - meaning you need to change the URL in your web pages every day
- pro-tip: use a dynamic dns service with an update client
  - now you can use a host name that never changes

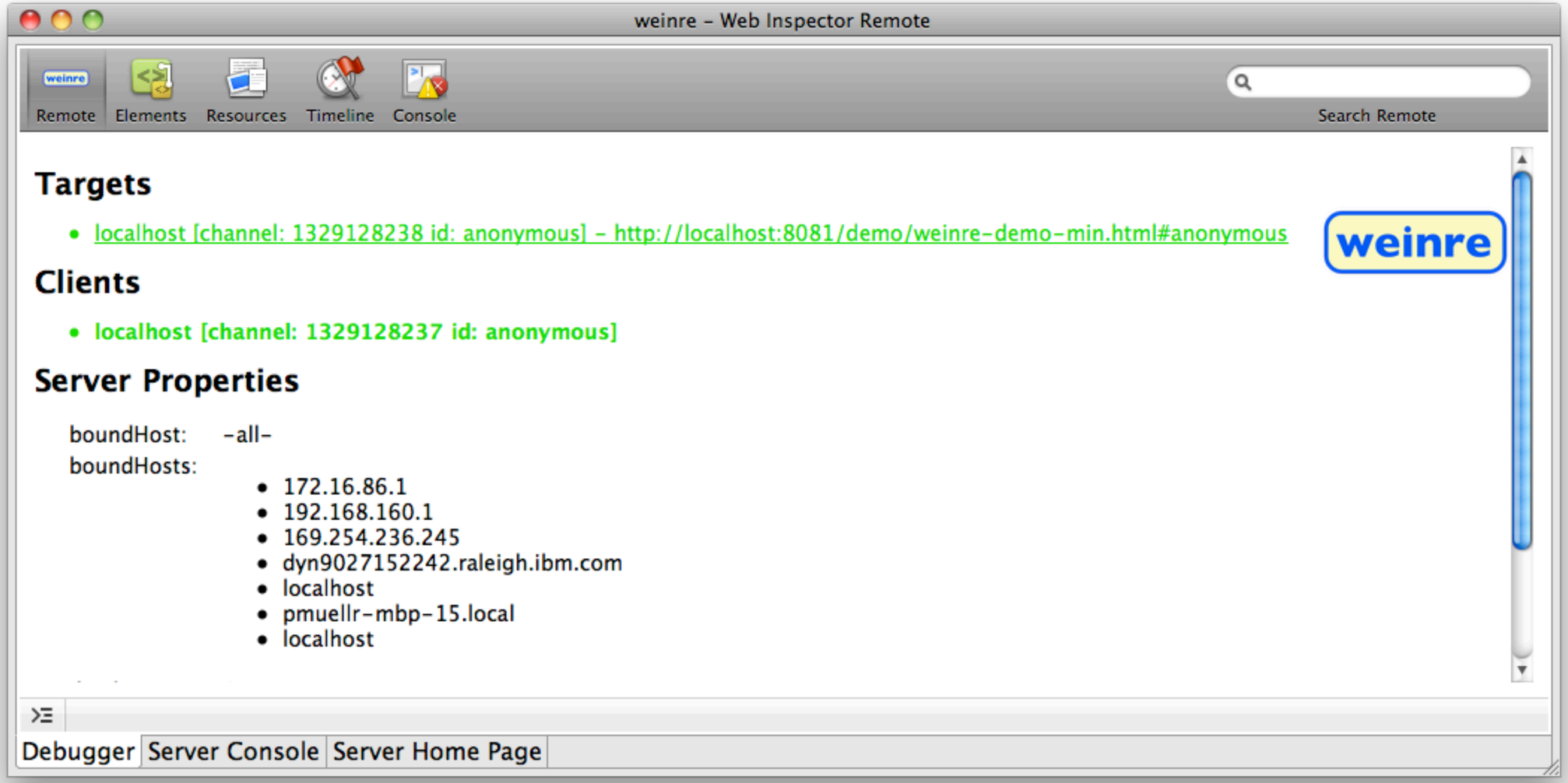
server command-line options

- see: http://phonegap.github.com/weinre/Running.html
- “--boundHost -all-” allows you to connect to the server from another machine (default only allows connections from same machine)
- “--httpPort <number>” allows you to change the server port

running the Mac application

- a Mac application is also available
- runs the server in a window which also displays the Web Inspector user interface
- built using Eclipse SWT - theoretically possible to port to Windows / Linux

mac app


weinre – Web Inspector Remote
Remote
Elements
Resources
Timeline
Console
Search Remote
Targets
localhost [channel: 1329128238 id: anonymous] – http://localhost:8081/demo/weinre-demo-min.html#anonymous
weinre
Clients
localhost [channel: 1329128237 id: anonymous]
Server Properties
boundHost: -all-
boundHosts:
172.16.86.1
192.168.160.1
169.254.236.245
dyn9027152242.raleigh.ibm.com
localhost
pmuellr-mbp-15.local
localhost
Debugger
Server Console
Server Home Page

bookmarklet

- possible to inject weinre target code into any web page with a bookmarket
- instructions available on weinre's main server page (when you run the server)
- not trivial to install on iOS or Android, and requires modern version of Android

collaborative debug

- multiple debug clients can connect to a single debug target
- must use a shared unique id
- not well tested
- not a design feature, just the way the web works

future

should be dim

- WebKit now has Remote Web Inspector baked in http://www.webkit.org/blog/1620/webkit-remote-debugging/
- RIM shipping Remote Web Inspector for Playbook
- Apple? Don't know, or if I did, I'd have to kill you, then myself.
- Google? "*I'm afraid we have no plans right now to enable this feature.*" http://bit.ly/r1clCt (webkit-dev mailing list)

easier / better PhoneGap integration

- examples:
  - auto-inject weinre JavaScript code into your app
  - diagnostics for PhoneGap-provided events
  - run weinre server **IN** your app

current issues logged

- **port the server to node.js, use socket.io for communications**
  - allows removal of the “message queue” code in Java and JavaScript
  - allows WebSocket usage, for better latency / less overhead (instead of XHR)
  - allows reuse of code between server and browser (wouldn’t be much though)

current issues logged

- **extension system that works**
  - there is an extension mechanism in place today, based on Web Inspector's extension mechanism
  - allows adding new panels, and any other hacking
  - hard/impossible to use; needs a re-write

current issues logged

- **provide better error handling**
  - error support not great for mobile devices - “onerror” not yet ubiquitous
  - can hook event handlers to provide try/catch with diagnostics for callbacks
  - catching errors at initial load time is still hard

until then

- if you need something fixed or added:
  - write a bug - https://github.com/phonegap/weinre/issues
  - ask a question - https://groups.google.com/group/weinre
  - DIY / fork it - https://github.com/phonegap

innards

target / server / communication

communication

- Web Inspector:
  - defines JSON-able messages sent between client and target
  - provides service framework to hook in message handlers
  - provides hooks at start-up time to start your own infrastructure

console.log(“hello, world”)

```
{
    interface: "ConsoleNotify",
    method:    "addConsoleMessage",
    args: [
        {
            message: "hello, world",
            level:   1,
            source:  3,
            type:    0,
            parameters: [
                {
                    hasChildren: false,
                    description: "hello, world",
                    type:        "string"
                }
            ],
        },
    ],
}
```

message interface / methods

- specified in WebKit via:
  - old: WebIDL-ish files (weinre currently using this version), converted to JSON in weinre build
  - new: JSON files
- data sent in messages not defined - “read the source”

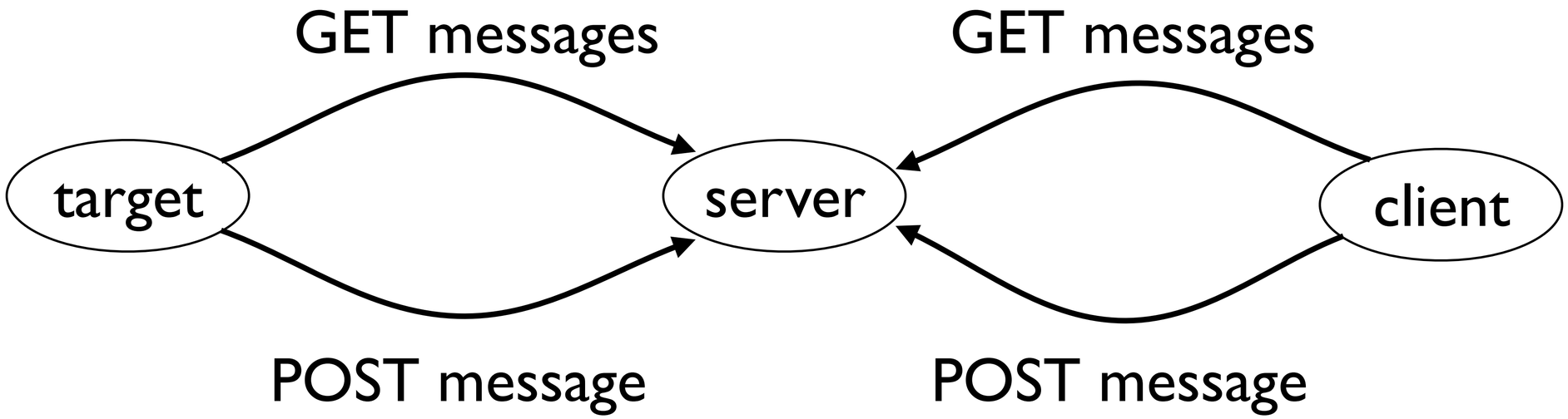
to implement weinre ...

- target must respond to client's messages correctly
- target must send events correctly
- basically, implement the target code - for Real Web Inspector, this is mainly C++, with some in JavaScript (we reuse their JavaScript)
- set up message queue and dispatch interface in target and client

HTTP usage

- client and target use the same JavaScript framework for message sending / receiving
- message queue-ish, REST-ish, implemented with XHR (not WebSocket)
- requires Cross-Origin Resource Sharing (CORS) to let target communicate, cross-origin, to weinre server
- target and client do not communicate directly, always through server

HTTP / XHR message flow



source

reused components

- Apache CLI - command line parser used by the server
- Eclipse Jetty - HTTP server used by the server (it's not a .war)
- Apache Wink JSON4J - JSON code for Java from used by server
- WebKit's Web Inspector - user interface used by the client

Web Inspector reuse

JavaScript:	116 files	48,000 lines	1.7 MB
CSS:	9 files	6,200 lines	140 KB
HTML:	1 file	175 lines	1.2 KB

- Almost all of this is for the “client” - the debugger user interface
- WebKit-specific, won’t run on FireFox, IE, Opera, etc

JavaScript modules

- almost all JavaScript:
  - written as CommonJS modules, using modjewel
  - written as in classical OO style using scooj preprocessor
  - looking at automagically porting from scooj to CoffeeScript using js2coffee

build script

- monster Ant script
  - downloads pre-reqs (if not already downloaded)
  - compile Java (optional, can also use Eclipse to do that)
  - bundle all the junk together
- if we port the server to node.js, will convert to a Makefile