

Exploring the Debugger Protocol for Test Authoring

Benjamin Gruenbaum

About me

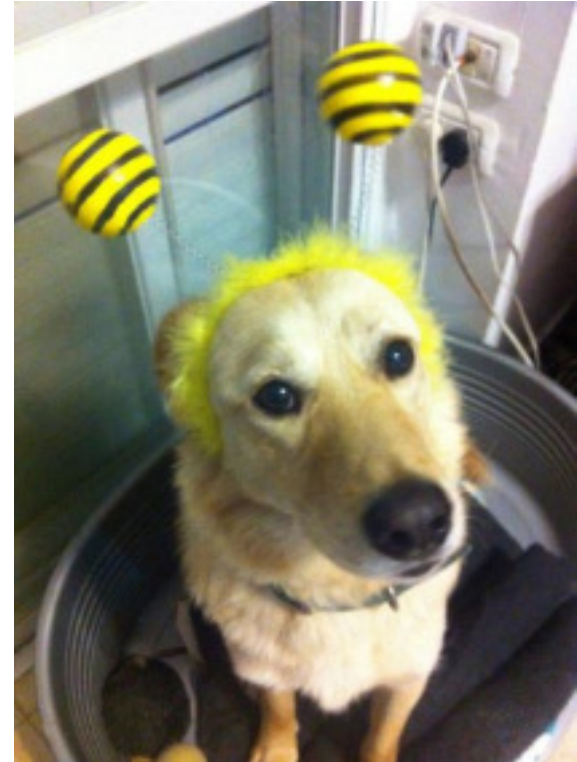


Sinon.JS



testim

מג ימים
תכנית הסייבר העלאומית



then



Disclaimer

C++ Ahead

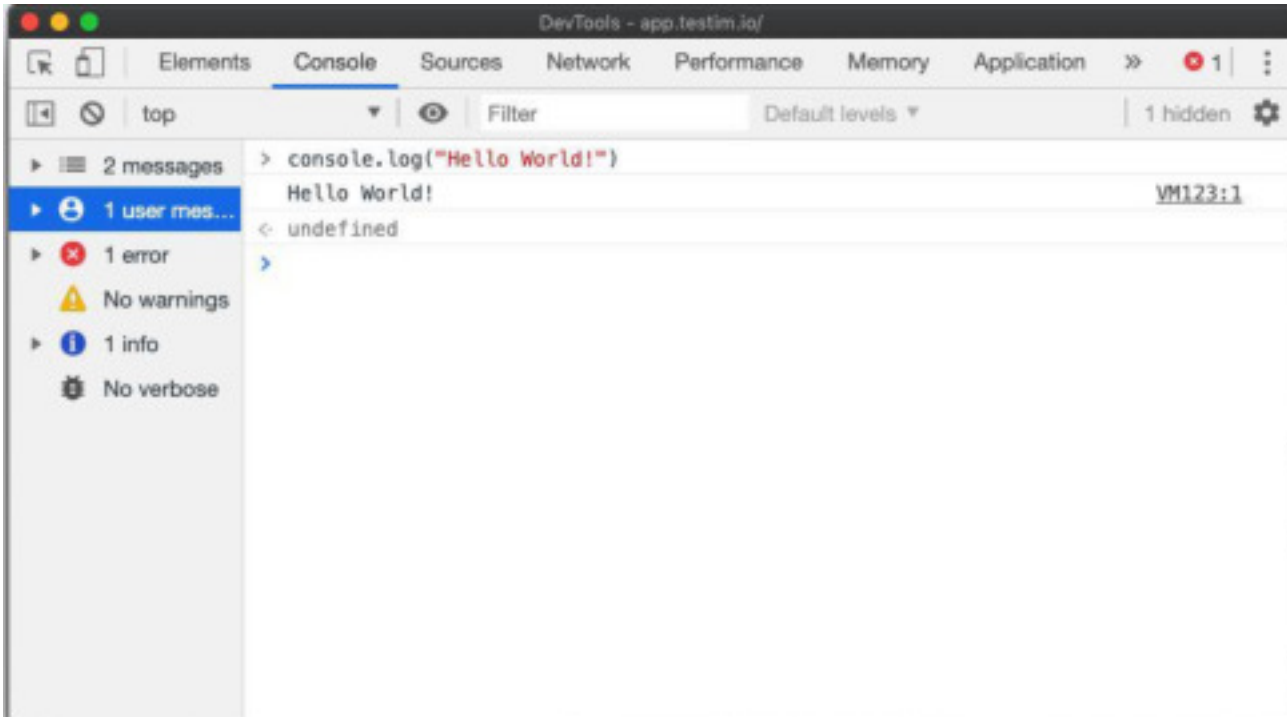
Not a lot, stay with me.



Disclaimer

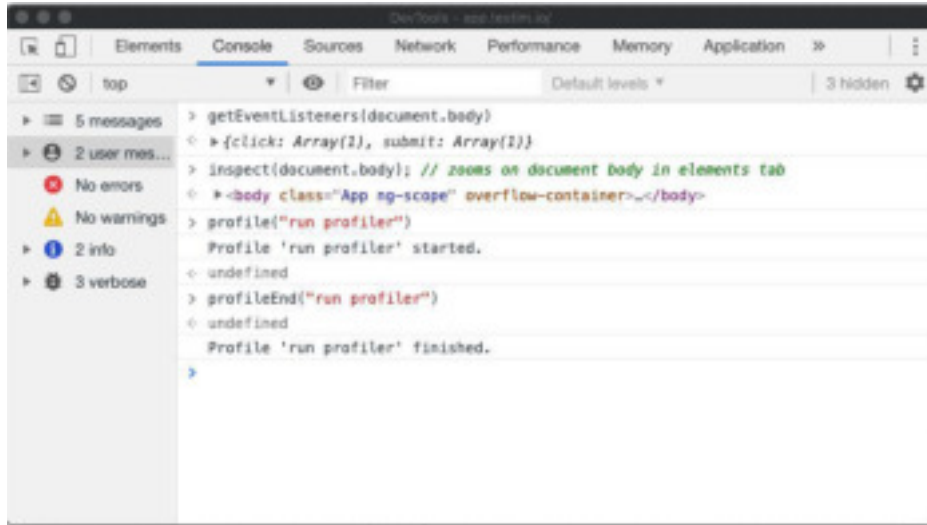
The code ahead has been modified for brevity
(logging removed etc)

The DevTools



What happens when you enter a command in the devtools?

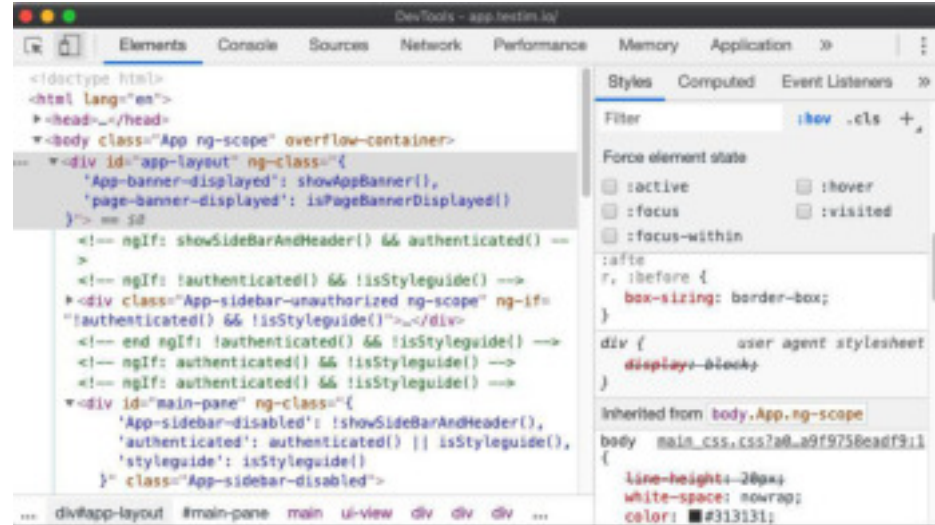
The Devtools has superpowers!



The screenshot shows the Chrome DevTools Console with the following content:

- 5 messages
- 2 user messages
- No errors
- No warnings
- 2 info
- 3 verbose

```
getEventListeners(document.body)
> {click: Array(1), submit: Array(1)}
inspect(document.body); // zooms on document body in elements tab
< *-body class="App ng-scope" overflow-container>...</body>
profile("run profiler")
Profile 'run profiler' started.
< undefined
profileEnd("run profiler")
< undefined
Profile 'run profiler' finished.
```



The screenshot shows the Chrome DevTools Elements panel with the following content:

```
<!doctype html>
<html lang="en">
  <head>...</head>
  <body class="App ng-scope" overflow-container>
    <div id="app-layout" ng-class="{
      'App-banner-displayed': showAppBanner(),
      'page-banner-displayed': isPageBannerDisplayed()
    }"> ... </div>
    <!-- ngIf: showSideBarAndHeader() && authenticated() -->
    <!-- ngIf: !authenticated() && !isStyleguide() -->
    <div class="App-sidebar-unauthorized ng-scope" ng-if="
      !authenticated() && !isStyleguide()">...</div>
    <!-- end ngIf: !authenticated() && !isStyleguide() -->
    <!-- ngIf: authenticated() && !isStyleguide() -->
    <!-- ngIf: authenticated() && !isStyleguide() -->
    <div id="main-pane" ng-class="{
      'App-sidebar-disabled': !showSideBarAndHeader(),
      'authenticated': authenticated() || isStyleguide(),
      'styleguide': isStyleguide()
    }" class="App-sidebar-disabled">
    ...
  </div>#app-layout #main-pane main ui-view div div div ...
```

The Styles panel on the right shows the following CSS rules:

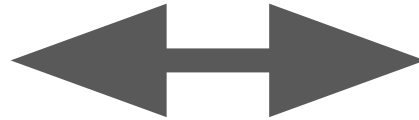
- Force element state
- Filter: `!hov .cls`
- :active
- :focus
- :focus-within
- :hover
- :visited
- after
- before {
- `border-box: border-box;`
- }
- div {
- `display: block;`
- inherited from `body.App.ng-scope`
- `body main_css.css?ab_a9f9756eadf9:1`
- {
- `line-height: 20px;`
- `white-space: nowrap;`
- `color: #313131;`
- }

How does it work?

Chrome Extension that lives in Chromium



WebSocket

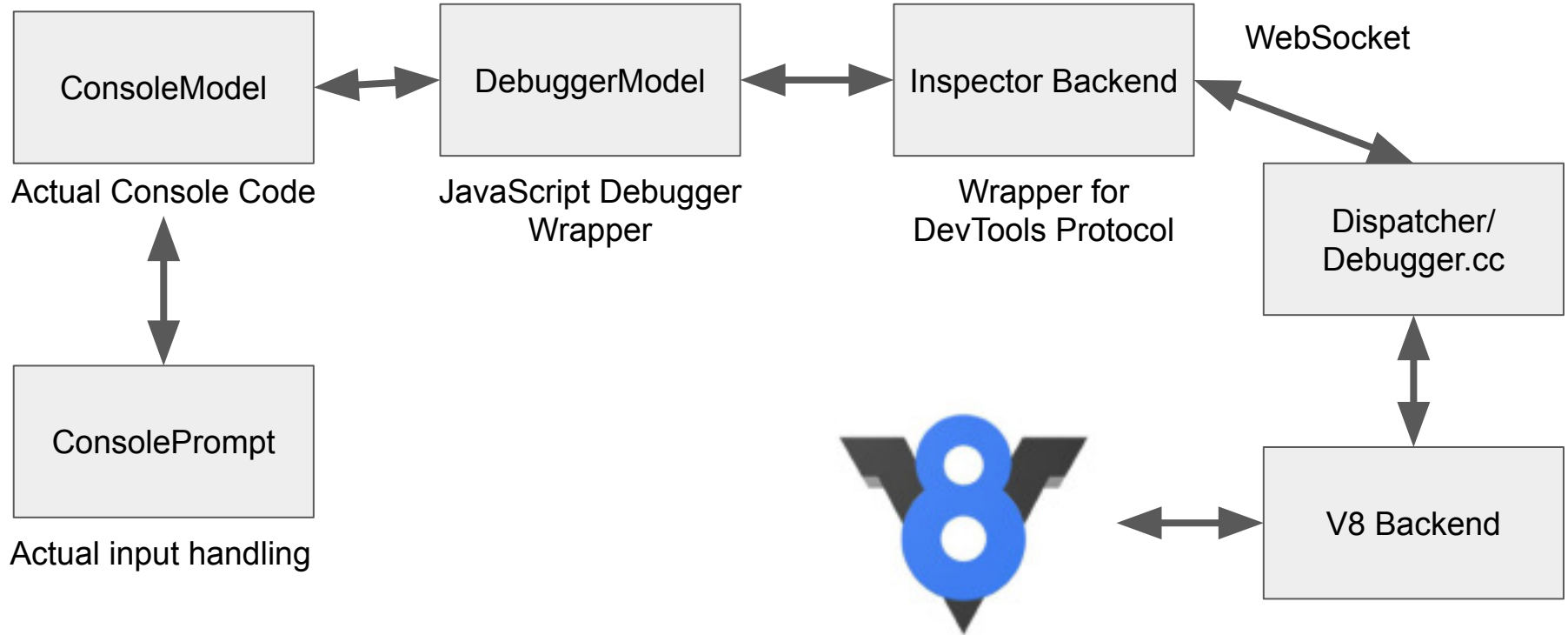


Page Context



So what actually happens?

How does it work?

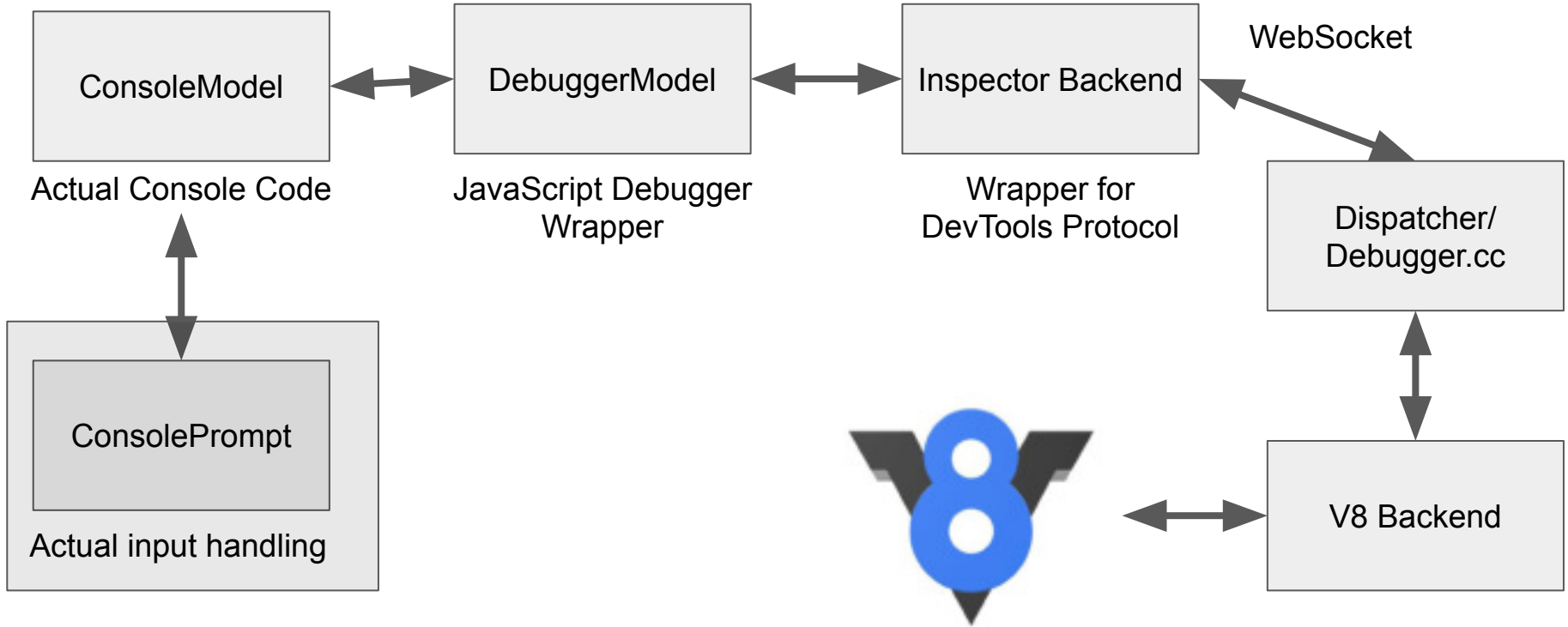


Why not “eval?”

You enter a command and press “enter”

- The devtools is itself a Chrome extension
- Its source lives in the Chromium repo like most of Chrome, which is open source
- Its source code is accessible, there are instructions for hacking on it in a google doc and it's easy to get started locally.
- There is a GitHub mirror for ease of usage since Chromium is a large and “scary” project

How does it work?



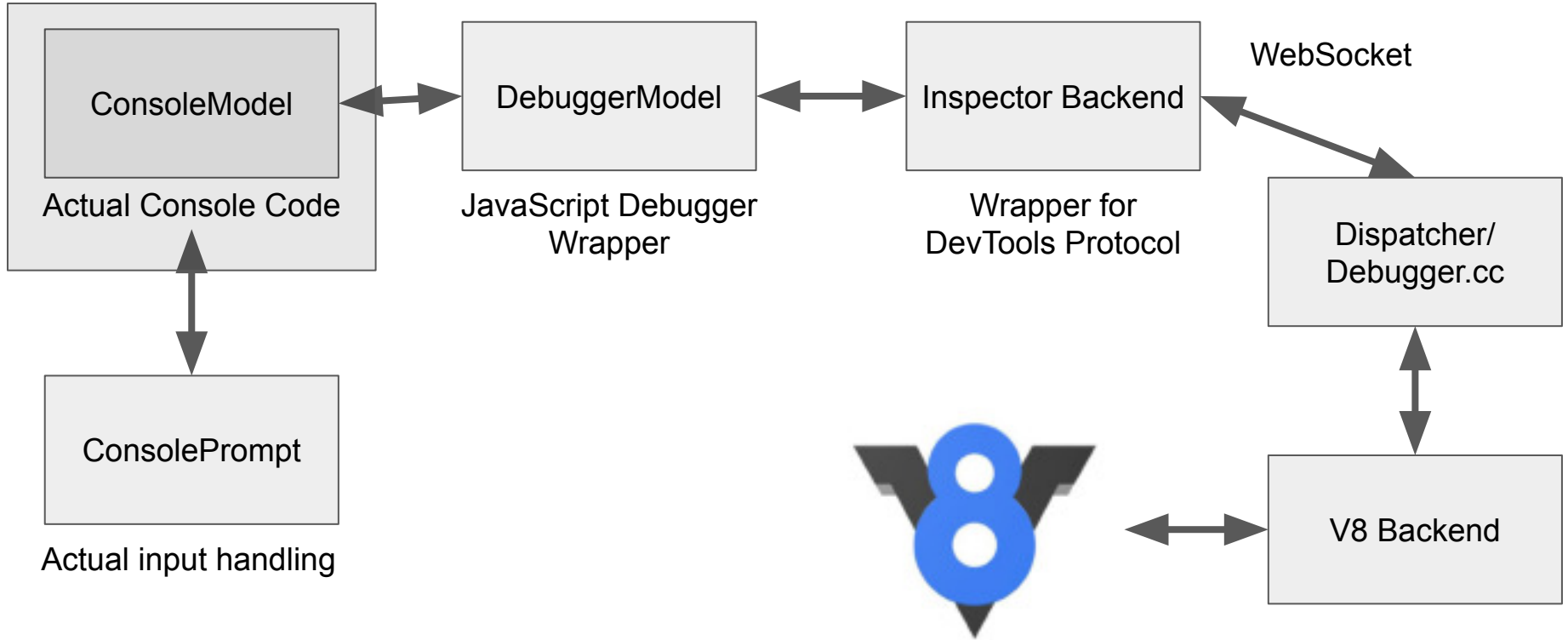
You enter a command and press “enter”

```
async _enterKeyPressed(event) {  
    ...  
    ...  
    if (await this._enterWillEvaluate())  
        await this._appendCommand(str, true);  
    else  
        this._editor.newlineAndIndent();  
    this._enterProcessedForTest();  
}
```

You enter a command and press “enter”

```
async _appendCommand(text, useCommandLineAPI) {
    this.setText('');
    const currentExecutionContext = flavor(SDK.ExecutionContext);
    const message = addCommandMessage(executionContext, text);
    const wrappedResult = await preprocessExpression(text);
    evaluateCommandInConsole(executionContext, message,
wrappedResult.text, useCommandLineAPI, /* awaitPromise */
wrappedResult.preprocessed);
}
```

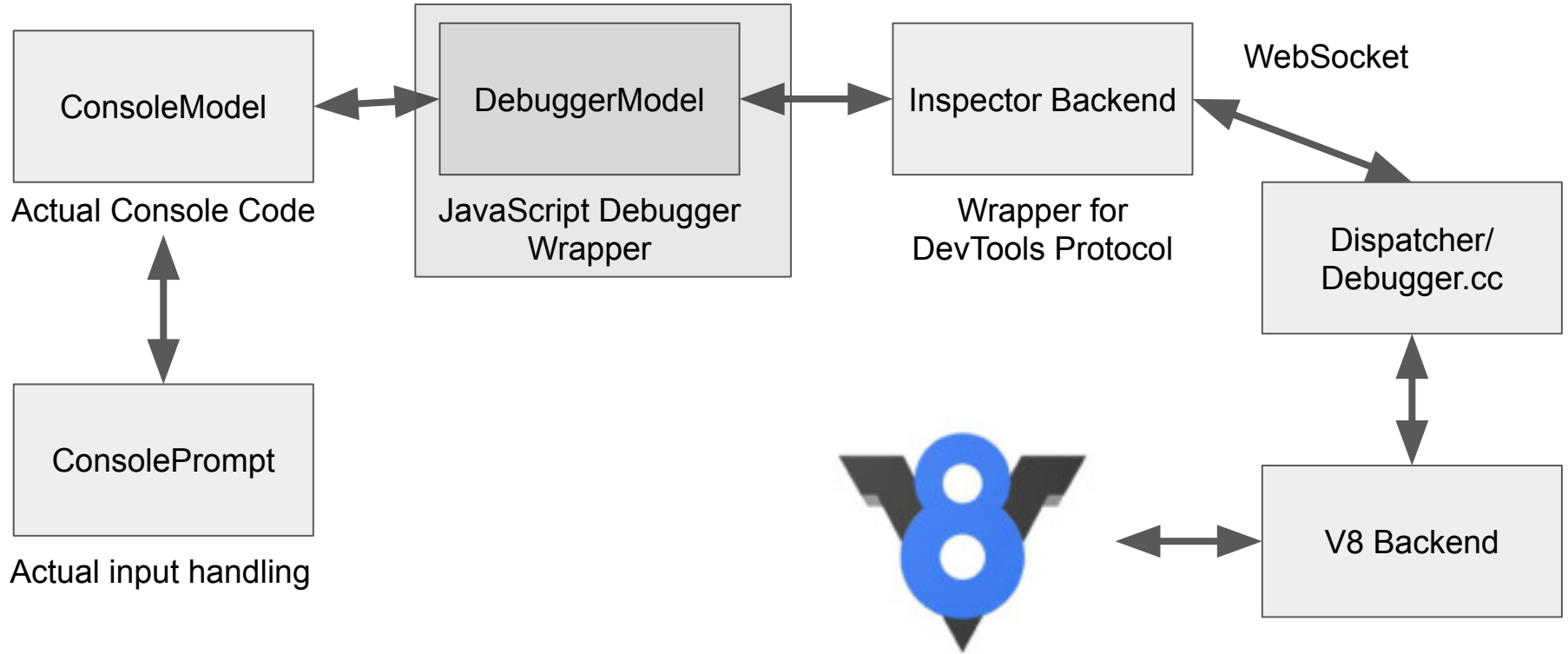
How does it work?



You enter a command and press “enter”

```
async evaluate(options) {  
    const response = await this.invoke_evaluateOnCallFrame({  
        expression: options.expression,  
    });  
    const error = response[Protocol.Error];  
    if (error) {  
        return {error: error};  
    }  
    return {object: runtimeModel.createRemoteObject(response.result),  
exceptionDetails: response.exceptionDetails};  
}
```

How does it work?



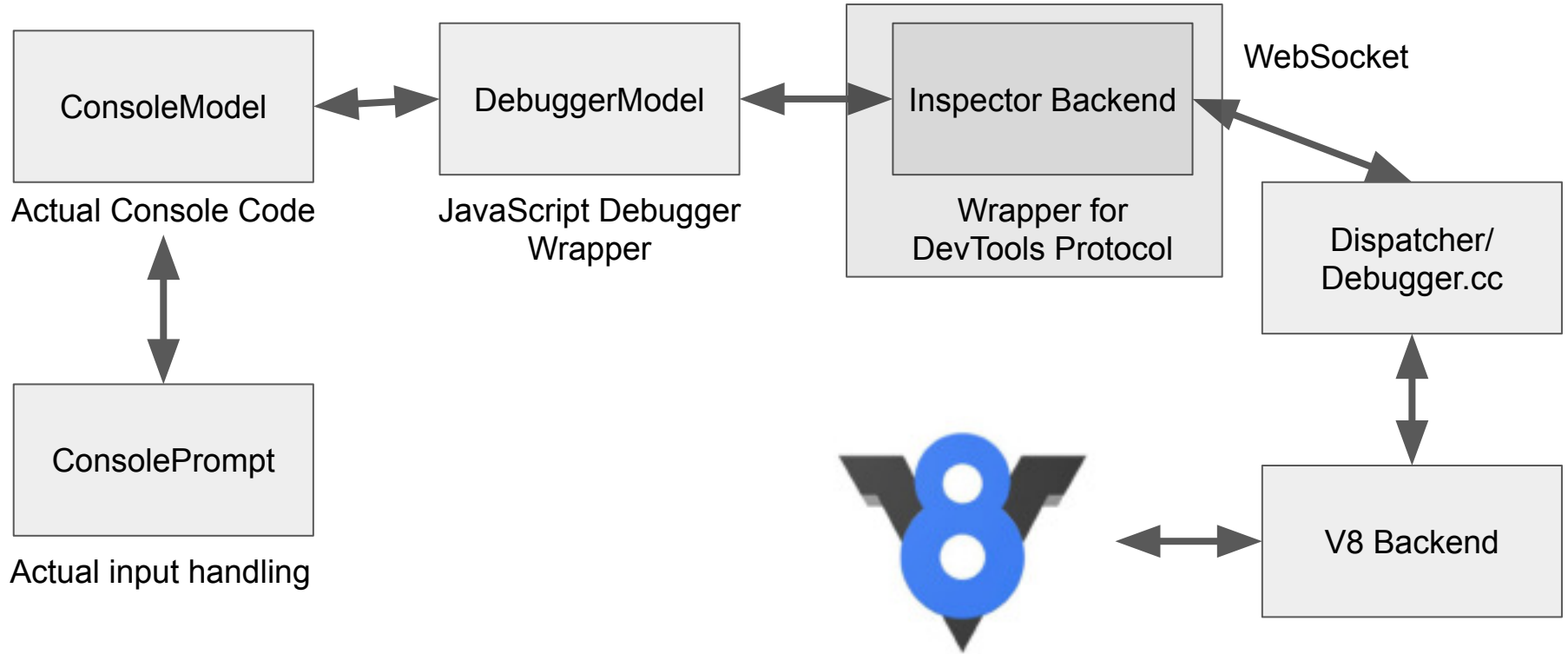
You enter a command and press “enter”

```
registerCommand(methodName, signature, replyArgs, hasErrorData) {  
    // ...  
  
    function invoke(request) {  
        return this._invoke(methodName, request);  
    }  
  
    this['invoke_' + methodName] = invoke;  
    this._replyArgs[domainAndMethod] = replyArgs;  
}
```

You enter a command and press “enter”

```
sendMessage(sessionId, domain, method, params, callback) {  
    const messageObject = {};  
    const messageId = this._nextMessageId();  
    messageObject.id = messageId;  
    messageObject.method = method;  
    messageObject.params = params;  
    this._connection.sendRawMessage(JSON.stringify(messageObject));  
}
```

How does it work?



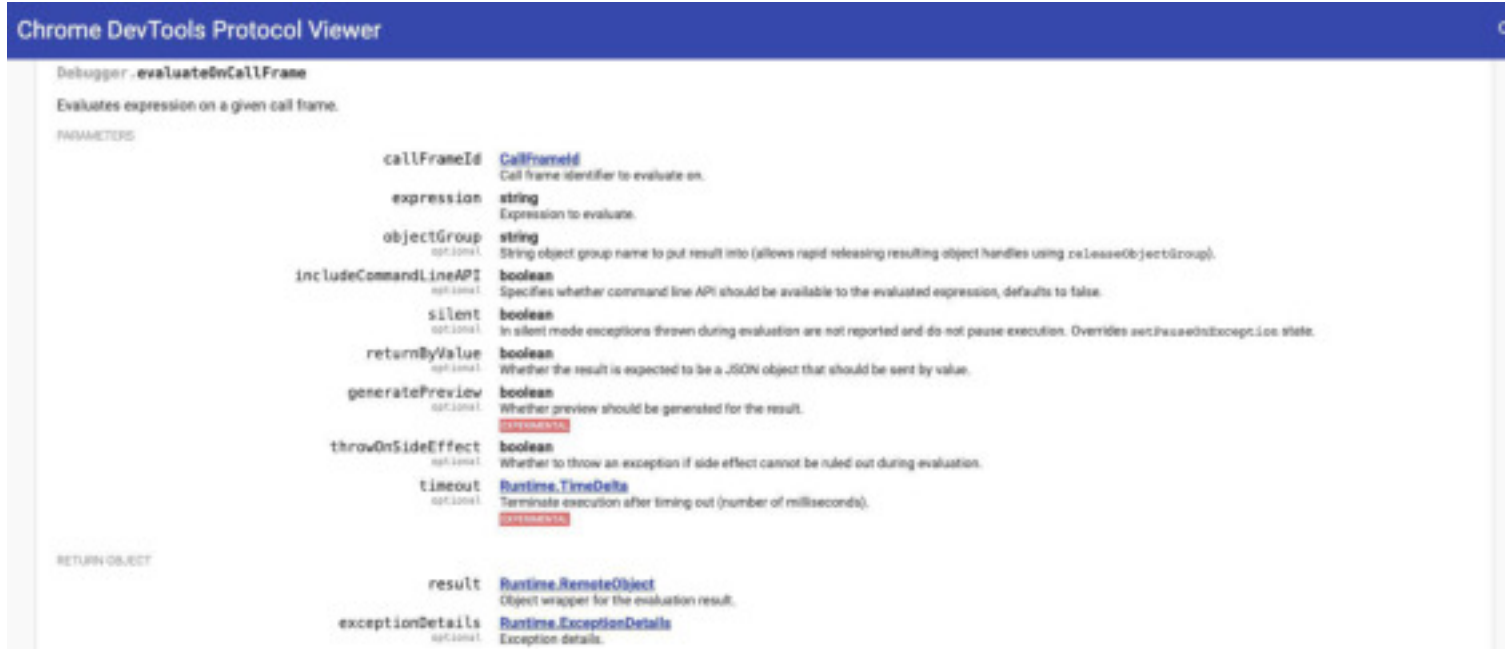
You enter a command and press “enter”

```
_invoke(method, request) {  
    return new Promise(fulfill => {  
        if (!this._target._router)  
            return dispatchConnectionError(callback);  
  
        this._target._router.sendMessage(sessionId, method, request,  
fulfill);  
    });  
}
```

You enter a command and press “enter”

```
    "name": "evaluateOnCallFrame",  
    "parameters": [  
...  
        {  
            "type": "string",  
            "name": "expression",  
            "description": "Expression to evaluate."  
        },  
...  
    ]  
}
```

You enter a command and press “enter”



Chrome DevTools Protocol Viewer

Debugger.`evaluateOnCallFrame`

Evaluates expression on a given call frame.

PARAMETERS

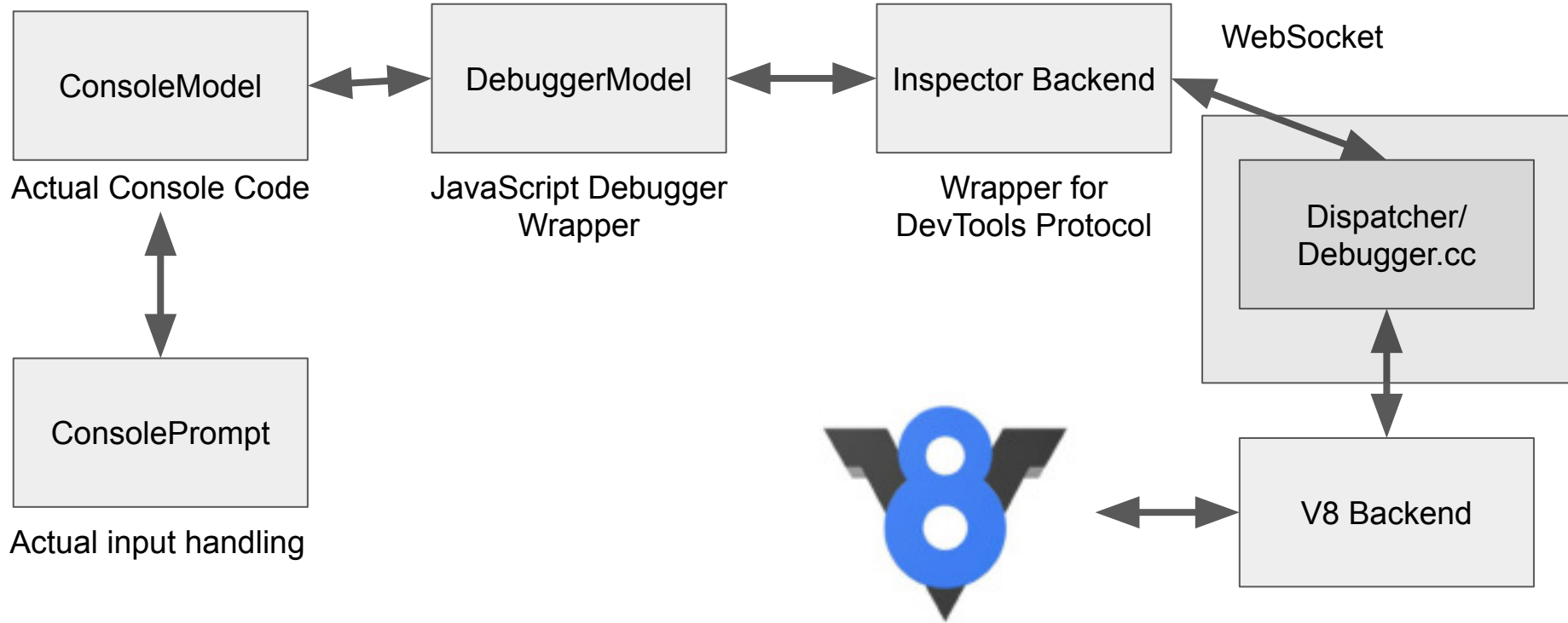
<code>callFrameId</code>	<code>CallFrameId</code> Call frame identifier to evaluate on.
<code>expression</code>	<code>string</code> Expression to evaluate.
<code>objectGroup</code>	<code>string</code> String object group name to put result into (allows rapid releasing resulting object handles using <code>releaseObjectGroup</code>).
<code>includeCommandLineAPI</code>	<code>boolean</code> Specifies whether command line API should be available to the evaluated expression, defaults to false.
<code>silent</code>	<code>boolean</code> In silent mode exceptions thrown during evaluation are not reported and do not pause execution. Overrides <code>setPauseOnException</code> state.
<code>returnByValue</code>	<code>boolean</code> Whether the result is expected to be a JSON object that should be sent by value.
<code>generatePreview</code>	<code>boolean</code> Whether preview should be generated for the result.
<code>throwOnSideEffect</code>	<code>boolean</code> Whether to throw an exception if side effect cannot be ruled out during evaluation.
<code>timeout</code>	<code>Runtime.TimeDelta</code> Terminate execution after timing out (number of milliseconds).

RETURN OBJECT

<code>result</code>	<code>Runtime.RemoteObject</code> Object wrapper for the evaluation result.
<code>exceptionDetails</code>	<code>Runtime.ExceptionDetails</code> Exception details.

<https://chromedevtools.github.io/devtools-protocol/tot/Debugger/#method-evaluateOnCallFrame>

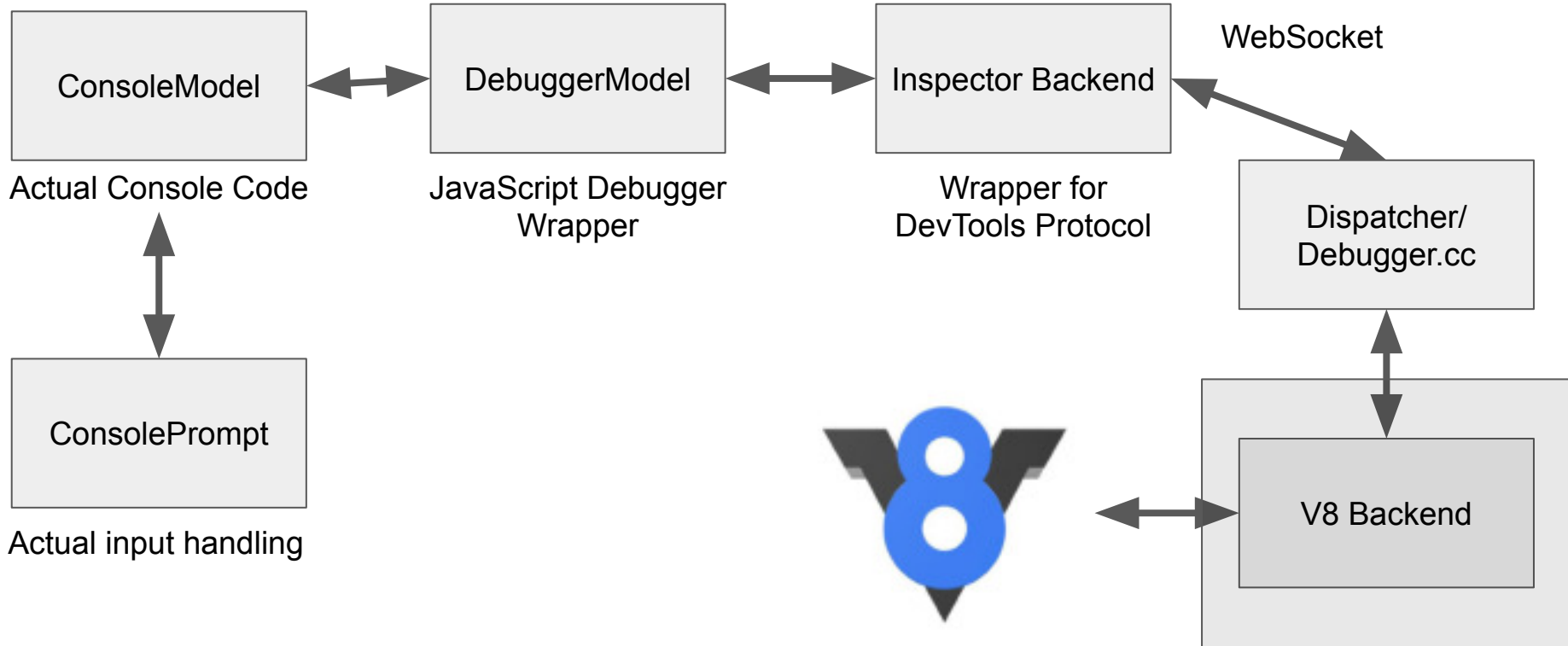
How does it work?



On the other end of the socket

```
    DispatcherImpl(FrontendChannel* frontendChannel, Backend*
backend): DispatcherBase(frontendChannel) {
    m_dispatchMap["Debugger.continueToLocation"] =
&DispatcherImpl::continueToLocation;
...
    m_dispatchMap["Debugger.evaluateOnCallFrame"] =
&DispatcherImpl::evaluateOnCallFrame;
...
}
```

How does it work?



On the other end of the socket

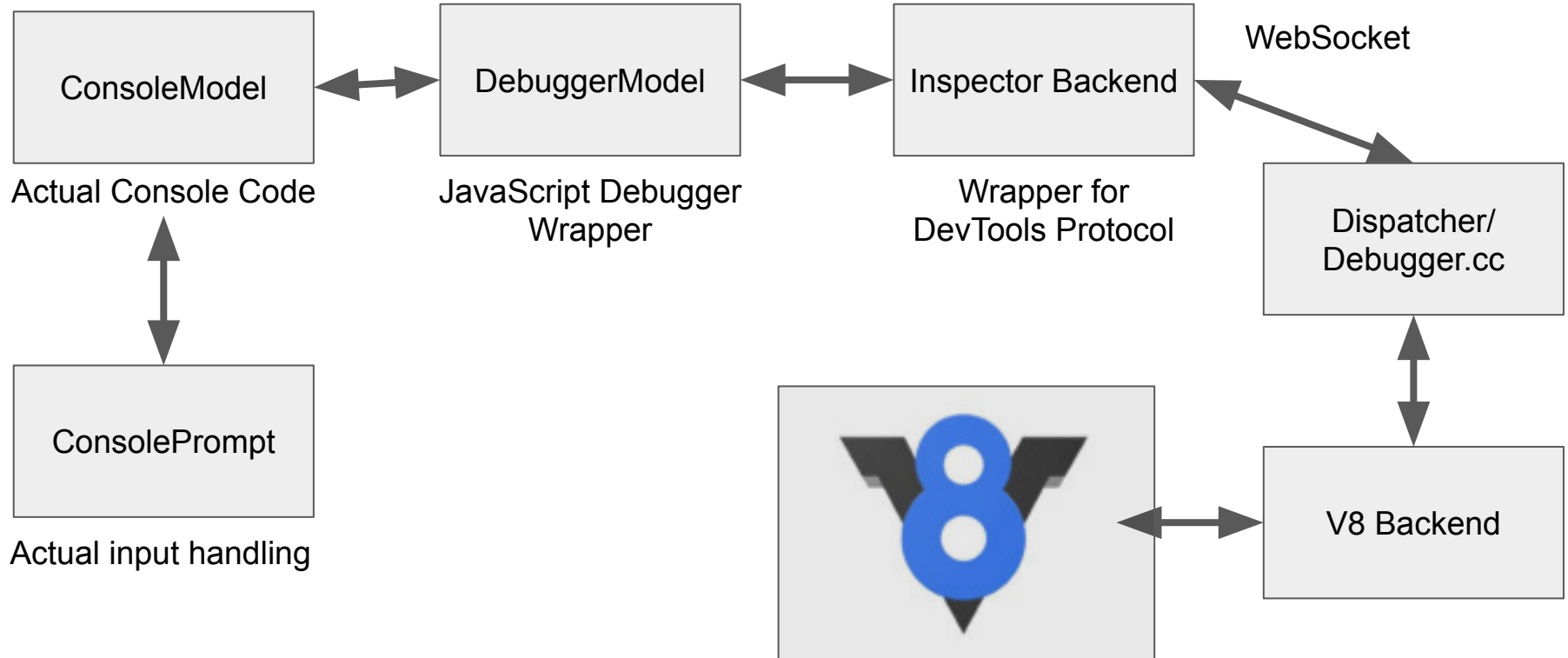
```
void DispatcherImpl::evaluateOnCallFrame(const ProtocolMessage&
message, std::unique_ptr<DictionaryValue> requestMessageObject)
{
    // ...

    DispatchResponse response =
m_backend->evaluateOnCallFrame(in_expression, &out_result);

    // ...

}
```

How does it work?



On the other end of the socket



```
Response V8DebuggerAgentImpl::evaluateOnCallFrame(  
    const String16& expression,  
    std::unique_ptr<RemoteObject>* result) {  
    InjectedScript::CallFrameScope scope(m_session, callFrameId);  
    Response response = scope.initialize();  
    v8::MaybeLocal<v8::Value> maybeResultValue;  
    {  
        V8InspectorImpl::EvaluateScope evaluateScope(scope);  
        maybeResultValue = it->Evaluate(toV8String(m_isolate, expression),  
                                       throwOnSideEffect.fromMaybe(false));  
    }  
    return scope.injectedScript()->wrapEvaluateResult(  
        maybeResultValue, scope.tryCatch(), objectGroup.fromMaybe(""), mode,  
        result, exceptionDetails);  
}
```

You enter a command and press “enter”

```
async evaluateCommandInConsole(executionContext, expression) {  
    const result = await executionContext.evaluate({  
        expression: expression  
    }, true, awaitPromise);  
    await Common.console.showPromise();  
    this.dispatchEventToListeners(CommandEvaluated,  
        {result: result, exceptionDetails: result.exceptionDetails});  
}
```

Evaluation result is printed

```
_commandEvaluated(event) {  
    const data = (event.data);  
    this.history().pushHistoryItem(messageText);  
    this._consoleHistorySetting.set(...);  
    this._printResult(data.result, data.commandMessage,  
data.exceptionDetails);  
}
```


And that's how the devtools work

Now, why do we care?

Other than that it's cool



testim



Selenium



- Most popular test automation tool.
- Basically just a specification - code belongs to browsers.
- HTTP server with a JSON REST API.
- In order to click an element you just POST a click and Chrome:

```
CommandMapping(kPost, "session/:sessionId/element/:id/click",  
                WrapToCommand("ClickElement")  
base::BindRepeating(&ExecuteClickElement)) ,
```

- Can anyone guess how it works?

Selenium

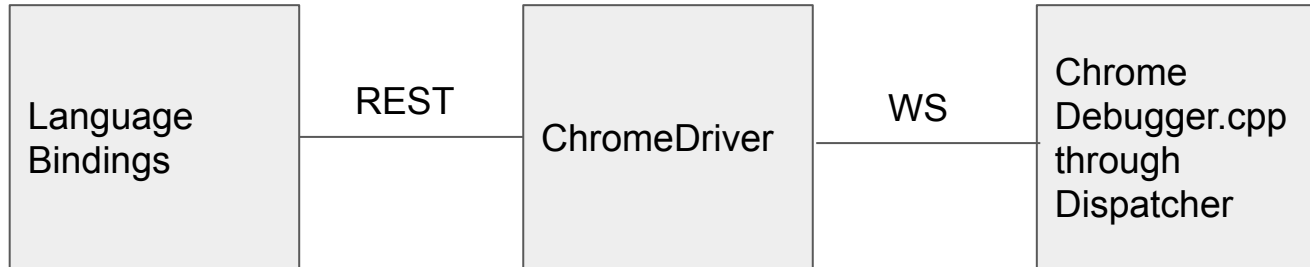


```
Status ExecuteClickElement(...) {
    std::string tag_name;
    Status status = GetElementTagName(..., &tag_name);
    events.push_back(MouseEvent(kMovedMouse, kNoneMouseButton));
    events.push_back(MouseEvent(kPressedMouse, kLeftMouseButton);
    events.push_back(MouseEvent(kReleasedMouse, kLeftMouseButton);
    status = web_view->DispatchMouseEvents(events)
session->GetCurrentFrameId());
    return status;
}
```

It's just the debugger protocol

```
Status WebViewImpl::DispatchMouseEvents(events) {
    for (auto it = events.begin(); it != events.end(); ++it) {
        params.SetString("type", GetAsString(it->type)); // ...
        status = client->SendCommand("Input.dispatchMouseEvent", params);
    }
    return Status(kOk);
}
```

Selenium

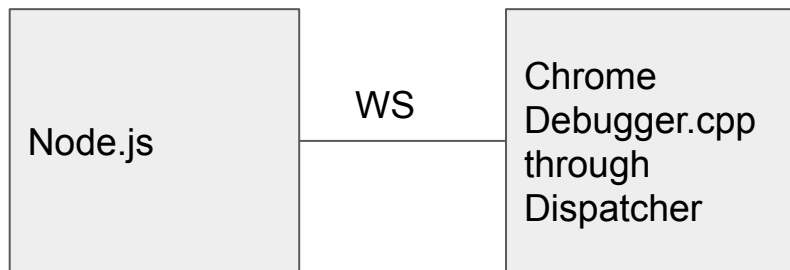


What about puppeteer?

- Popular “chrome only” test automation tool.
- Basically an even more direct client over the Devtools API
- Very tightly coupled to the protocol by design and to provides stability when running.
- Clicking does something similar



Puppeteer



Clicking with Puppeteer

```
async click(x, y, options = {}) {  
  const {delay = null} = options;  
  this.move(x, y);  
  this.down(options);  
  if (delay !== null)  
    await new Promise(f => setTimeout(f, delay));  
  await this.up(options);  
}
```

Clicking with Puppeteer

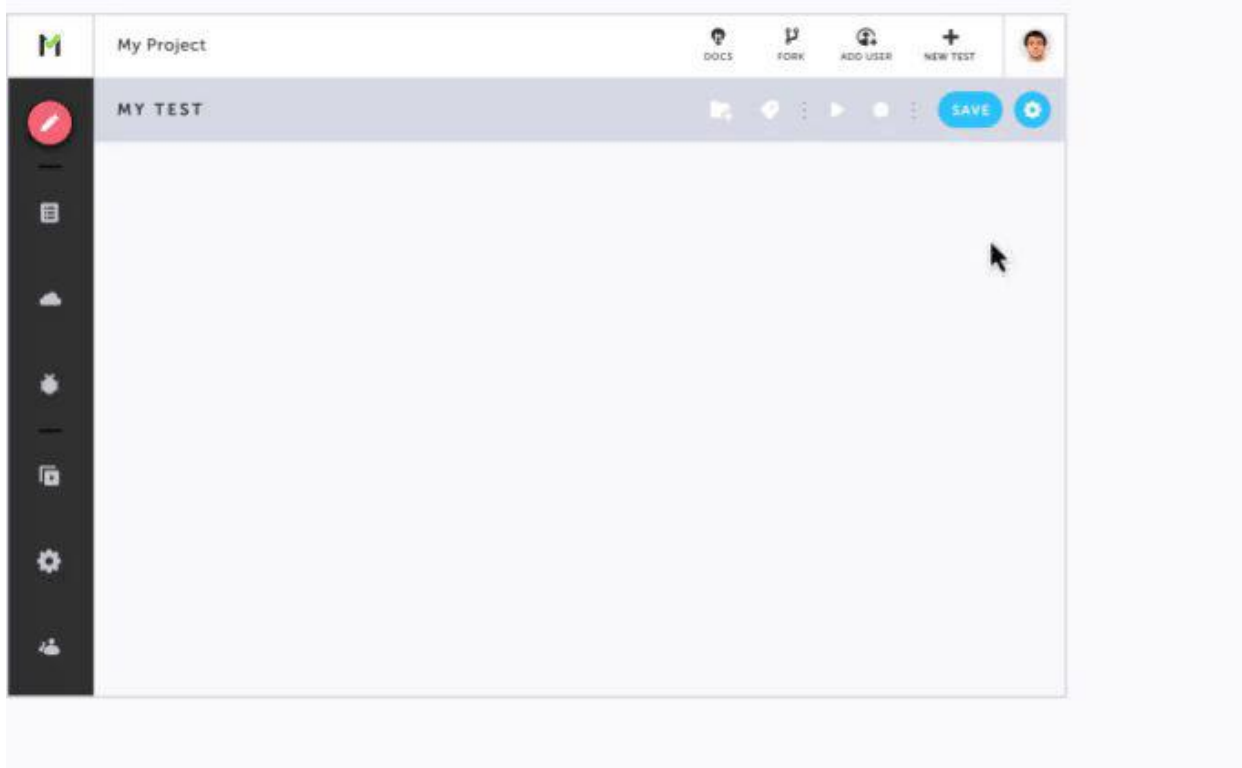
```
async down(options = {}) {  
  const {button = 'left', clickCount = 1} = options;  
  this._button = button;  
  await this._client.send('Input.dispatchMouseEvent', {  
    type: 'mousePressed',  
    button,  
    // ...  
  });  
}
```

What about Testim?



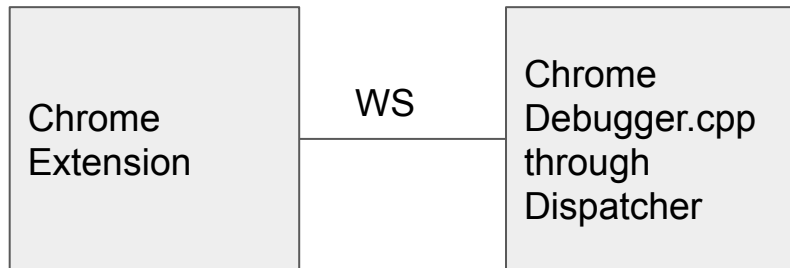
- Obviously the best test automation tool, and I'm not incredibly biased here 😊,
- Emphasis on stability.
- Connects to (**you guessed it**) the DevTools protocol protocol when playing in Chrome in a pretty tightly coupled way.
- Mostly `Input.dispatchEvent`
- Lots of code to deal with edge cases of doing that to stabilize automation.

What about Testim?

The logo for Testim, featuring the word "testim" in a dark grey, lowercase sans-serif font. The letter "i" is replaced by a blue checkmark icon.

What about Testim?

Similar to lighthouse or the DevTools



What about You?

- Let's talk about how you can use the devtools protocol to your advantage.
- There are two primary ways to use the devtools protocol:
 - From extensions `chrome.debugger.attach`
 - From consoles `cdpSession.send(...)`

Some ideas for the protocol

- Simulate location `Emulation.setGeolocationOverride`
- Override timers with `Emulation.setVirtualTimePolicy`
 - Stability with `pauseSelfNetworkFetchesPending`
- Warnings with `Log.startViolationsReport`
- Network cache clear with `Network.clearBrowserCache`
- Test accessibility with `Accessibility.getFullAXTree`
- Get code coverage with `Profiler.takePreciseCoverage`
- Force hover state with `CSS.forcePseudoState`

How to: boilerplate - CDP

```
const CDP = require('chrome-remote-interface');
```

```
const { launch } = require('chrome-launcher');
```

```
(async () => {
```

```
  const chrome = await launch({
```

```
    chromeFlags: ['--headless', '--disable-gpu']
```

```
  });
```

```
  const session = await CDP({ port: chrome.port });
```

```
  console.log(chrome, session);
```

```
})();
```


How to: boilerplate - chrome-launcher

```
const CDP = require('chrome-remote-interface');
```

```
const { launch } = require('chrome-launcher');
```

```
(async () => {
```

```
  const chrome = await launch({
```

```
    chromeFlags: ['--headless', '--disable-gpu']
```

```
  });
```

```
  const session = await CDP({ port: chrome.port });
```

```
  console.log(chrome, session);
```

```
})();
```

How to: boilerplate

```
const CDP = require('chrome-remote-interface');
```

```
const { launch } = require('chrome-launcher');
```

```
(async () => {
```

```
  const chrome = await launch({
```

```
    chromeFlags: ['--headless', '--disable-gpu']
```

```
  });
```

```
  const session = await CDP({ port: chrome.port });
```

```
  console.log(chrome, session);
```

```
})();
```

How to: boilerplate

```
const CDP = require('chrome-remote-interface');
const { launch } = require('chrome-launcher');

(async () => {
  const chrome = await launch({
    chromeFlags: ['--headless', '--disable-gpu']
  });
  const session = await CDP({ port: chrome.port });
  console.log(chrome, session);
})();
```

How to: take screenshot directly

```
const { Page, Network } = session;  
await Page.navigate({ url: "http://www.devdays.lt "});  
const { data } = await Page.captureScreenshot();  
await fs.writeFile("screenshot.png", Buffer.from(data, "base64"));
```



How to: take screenshot directly

```
await Page.navigate({ url: "http://www.devdays.lt "});
await new Promise(resolve => {
  timer = setTimeout(resolve, 2000);
  Network.requestWillBeSent(params => {
    clearTimeout(timer);
    timer = setTimeout(resolve, 2000);
  });
});
const { data } =
  await Page.captureScreenshot();
```



A quick utility library

- Starts express server, chrome and devtools connection
- Calls passed code into it using a disposer pattern
- Lets me run arbitrary chrome commands
- If anyone wants, I will put this on NPM

Some examples - using the library

```
use(`  
  function fib(n) { return n < 2 ? 1 : fib(n-1) + fib(n-2); }  
  function bar() {}`, async ({Page, Runtime, Profiler}) => {  
    const { result } = await Runtime.evaluate({  
      expression: 'fib(10)'  
    });  
    console.log(result.value); // 89  
  });
```

Some examples - coverage without nyc

```
await Profiler.enable();  
await Profiler.startPreciseCoverage();  
await Runtime.evaluate({ expression: 'fib(10)' });  
const { result } = await Profiler.takePreciseCoverage();  
const { ranges } = result[0].functions  
    .find(x => x.functionName === 'fib');  
console.log(ranges);
```




If anyone wants to get into Node.js - please shoot me an email at benjamingr@gmail.com

Thank You!