jsprobes

cross-platform browser instrumentation using JavaScript

Brian Burg (@brriian / burg@cs.uw.edu)
Background:

browser research template

1. Become puzzled by a question
2. Get real data and analyze
3. Write paper with answers

images from www.phdcomics.com
Right now, this is the hard part

1. Become puzzled by a question
2. Get real data and analyze
3. Write paper with answers

images from www.phdcomics.com
Browser instrumentation wishlist

• Cross-platform/architecture
• Low/no performance overhead
• Shareable/distributable
• Runtime flexibility
• Familiar programming model
• Cross-language/cross-component
Existing approaches: browser addons/extensions

Advantages
• Somewhat familiar programming model
• Easy to distribute
• Cross-platform
• Flexibility at runtime

Disadvantages
• Can’t access all parts of the stack
• Bad for critical path instrumentation

Image: Mozilla Test Pilot (www.mozillalabs.com/testpilot/)
Existing approaches: platform instrumentation

**Advantages**

- Extremely low/no performance overhead
- Can instrument any browser component
- Runtime flexibility

**Disadvantages**

- Needs escalated (root) privileges/kernel mod
- Platform-specific
- Limited programming language/model
- Cannot distribute

Images: dtrace.org, en.wikipedia.org/wiki/SystemTap
Existing approaches: modifying browser source

Advantages

• Can do anything!
• (possible) low performance impact

Disadvantages

• Difficult to understand and modify
• Easy to cause crashes
• Hard to distribute/share/reuse
• Very fragile vs. upstream changes
jsprobes: an experiment

• A browser instrumentation framework...
  – With instrumentation written in JavaScript
  – Accessible via XPCOM to addons
  – Available on all platforms/architectures
  – Which encourages experimentation
  – That can gather many kinds of data
  – Fast enough to gather low-level data
jsprobes terminology

probe point  probe handler  probe values

Source location  What to do at  Data to record
    to instrument  probe point  at probe point

images from www.phdcomics.com
jsprobes use case

1. Find an interesting probe point
   ```javascript
   probes.GC_DID_START
   ```
2. Decide which probe values to use
   ```javascript
   env.currentTimeMS,
   runtime.heapSize
   ```
3. Write a probe handler for that point
   ```javascript
   pendingGC = [env.currentTimeMS, 0,
               runtime.heapSize, 0];
   ```
jsprobes use case (2)

4. Write a matching handler to `GC_WILL_END`:
   ```javascript
   pendingGC[1] = env.currentTimeMS;
   pendingGC[3] = runtime.heapSize;
   data.push(pendingGC);
   ```

5. Register handlers with the probes service.

6. Periodically fetch data and do something, such as aggregate, graph, or report it.
Demo!
(source available at https://bitbucket.org/burg/aboutgc)
jsprobes architecture

Main browser thread + heap

Handler registry

Probe handler thread + heap
jsprobes architecture

Main browser thread + heap

Handler registry
- probe handler code
- probe handler code
- probe handler code

Register probe handler with service

Probe handler thread + heap
jsprobes architecture

Probe point reached.

Main browser thread + heap

if (!hasAvailableAvail)
removeFromAvail
JSRuntime *rt = inf
Probes::resizeHeap()
JS_ATOMIC_ADD(&rt->

Handler registry

probe handler code
probe handler code
probe handler code

Register probe handler with service

Probe handler thread + heap
jsprobes architecture

Probe point reached.

Data gathered, serialized, and enqueued

Main browser thread + heap

Probe handler thread + heap

Handler registry

- probe handler code
- probe handler code
- probe handler code

Register probe handler with service

```c
390  if (!hasAvailableAr
391    removeFromAvail
392    JSRuntime *rt = inf
394  Probes::resizeHeap()
395    JS_ATOMIC_ADD(&rt->
```
jsprobes architecture

Data gathered, serialized, and enqueued

Main browser thread + heap

Handler registry

Register probe handler with service

Probe handler thread + heap

asynchronously deserialize data and run handlers

```c
if (!hasAvailableAr) removeFromAvail
JSRuntime *rt = inf
Probes::resizeHeap(
JS_ATOMIC_ADD(&rt->
```

```javascript
current = [env.currentTimeMS, 0,
runtime.heapSize, 0]
...
current[1] = env.currentTimeMS;
current[3] = compartment.runtime
pendingData.push(current);
```
**jsprobes architecture**

Data gathered, serialized, and enqueued

Main browser thread + heap

if (!hasAvailableArea) removeFromAvail

JSRuntime *rt = inf

Probes::resizeHeap()

JS_ATOMIC_ADD(&rt->

Register probe handler with service

Handler registry

probe handler code

probe handler code

probe handler code

asynchronously deserialize data and run handlers

current = [env.currentTimeMillis, 0, runtime.heapSize, 0]

... current[1] = env.currentTimeMillis;

current[3] = compartment.runtime

pendingData.push(current);

... while (pendingData.length) postMessage(pendingData.pop());

Probe handler thread + heap

Post async messages back to main thread
Main browser thread + heap

Data gathered, serialized, and enqueued

Handler registry

Register probe handler with service

Post async messages back to main thread

Probe handler thread + heap

asynchronously deserialize data and run handlers

Dispatch message to callbacks

jsprobes architecture
Architecture implications

• Probe points can fire at times unsafe for JS
• Probe handlers have read-only access*
• Handler must specify what data to collect
• Probe data must be representable in JS
• Probe data must be serializable**

* Side-effects would complicate reasoning when multiple handlers are registered for the same probe point

** Probe data is marshalled using the HTML 5 structured cloning algorithm. This can be extended to support new data types.
jsprobes: current status

• Cross-platform/architecture
• Low/no performance overhead (TODO)
• Shareable/distributable (TODO)
• Runtime flexibility
• Familiar programming model
• Cross-language/cross-component

Current implementation available at https://bitbucket.org/burg/jsprobes-patches
Let’s fill in the research template...

1. “Do websites have a typical heap size?”
2. Use jsprobes to make an addon that measures per-site heap size
3. Implement better heap size heuristics based on real data
Future work

• More sophisticated implementation
  – No “probe effect” when probes inactive
  – Low performance impact when active
• Add probe points to more components
• Expose more types of data to probe handlers

Brian Burg – University of Washington
burg@cs.uw.edu  www.brrian.net
www.twitter.com/brrian
https://bitbucket.org/burg/
http://brrian.tumblr.com