JS Profiling @ TPAC 2021

acomminos@fb.com
cpescheloche@fb.com

Recap

API overview

- A web-exposed sampling profiler for client JS execution
- Provide insight into client performance characteristics on real users' devices
- Shipped in Chrome 94

- github.com/WICG/js-self-profiling

```
const profiler = new Profiler({ sampleInterval: 10, maxBufferSize: 10000 });
```

Starting a profiler

```
const profiler = new Profiler({ sampleInterval: 10, maxBufferSize: 10000 });

window.addEventListener('load', async () => {
  const trace = await profiler.stop();
  const traceJson = JSON.stringify(trace);
  sendTrace(traceJson);
});
```

Sending trace on load

```
const profiler = new Profiler({ sampleInterval: 10, maxBufferSize: 10000 });

window.addEventListener('load', async () => {
  const trace = await profiler.stop();
  const traceJson = JSON.stringify({
    entries: performance.getEntries(),
    trace,
  });
  sendTrace(traceJson);
});
```

Including data from the performance timeline

```
{
  "frames": [...],
  "resources": [...],
  "samples": [...],
  "stacks": [...],
}
```

```
[
  {
    "column": 80,
    "line": 311,
    "name": "caller",
    "resourceId": 0
  },
  {
    "column": 368,
    "line": 311,
    "name": "callee",
    "resourceId": 0
  },
]
```

Trace format: frames

```
{
  "frames": [...],
  "resources": [...],
  "samples": [...],
  "stacks": [...],
}
```

```
["https://www.fbcdn.net/script.js"]
```

Trace format: resources

```
{
  "frames": [...],
  "resources": [...],
  "samples": [...],
  "stacks": [...],
}
```

```
[
  {
    "stackId": 1,
    "timestamp": 15199
  },
  {
    "timestamp": 15209
  },
],
```

Trace format: samples

```
{
  "frames": [...],
  "resources": [...],
  "samples": [...],
  "stacks": [...],
}
```

```
[
  { "frameId": 0 },
  { "frameId": 1, "parentId": 0 }
]
```

Trace format: stacks

What’s working well?

- Initial data suggests **enabling profiling slows load time by <1% (p=0.05)** at FB
    - Strong evidence that sampling profiling can be implemented with minimal overhead
- API has provided a drop-in solution for FB app client perf analysis
- Strong adoption from other industry partners

What could be better?

- Non-JS execution is hard to identify in traces
  - Currently, top-level UA work is indistinguishable from idle execution
  - GC activity adds to the noise of long traces
  - Client code that causes asynchronous rendering work isn’t measurable
- Interactions with Long Tasks API can be cumbersome

Representing non-JS execution

Introducing state markers

- Tags a sample with a string representing top level UA work category
- Similar representation to traces visualized through devtool profilers

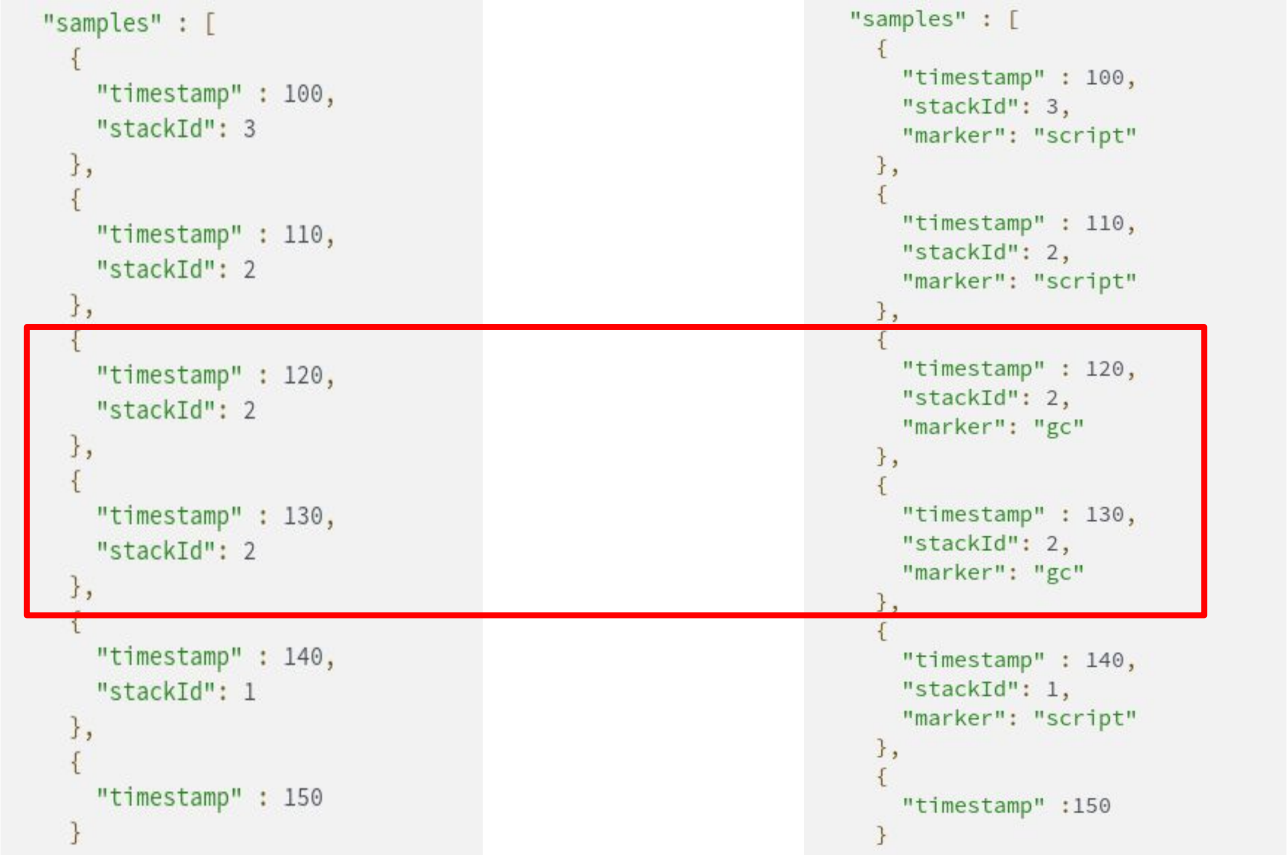
Marker candidates

- Need to be generic and interoperable
- Script related:
  - script: js execution, optional ?
  - parse: HTML? JS?
  - gc
- Rendering related:
  - paint: update the rendering part of the event loop or limited to actual paint
  - style
  - layout

API Modification

```
enum ProfilerMarker { "script", "gc", "parse", "paint", "other" };

dictionary ProfilerSample {
  required DOMHighResTimeStamp timestamp;
  unsigned long stackId;
  ProfilerMarker? marker;
};
```



```
"samples" : [
  {
    "timestamp" : 100,
    "stackId": 3
  },
  {
    "timestamp" : 110,
    "stackId": 2
  },
  {
    "timestamp" : 120,
    "stackId": 2
  },
  {
    "timestamp" : 130,
    "stackId": 2
  },
  {
    "timestamp" : 140,
    "stackId": 1
  },
  {
    "timestamp" : 150
  }
```

```
"samples" : [
  {
    "timestamp" : 100,
    "stackId": 3,
    "marker": "script"
  },
  {
    "timestamp" : 110,
    "stackId": 2,
    "marker": "script"
  },
  {
    "timestamp" : 120,
    "stackId": 2,
    "marker": "gc"
  },
  {
    "timestamp" : 130,
    "stackId": 2,
    "marker": "gc"
  },
  {
    "timestamp" : 140,
    "stackId": 1,
    "marker": "script"
  },
  {
    "timestamp" :150
  }
```

Example trace GC

Security and privacy concerns

- Profiles **must not** expose work done on a cross-origin document
  - Top level UA work may only appear in a trace if the **responsible document** for the work is same-origin with the associated Profiler
- New information exposed, need to limit granularity of marker types
  - Need to avoid introducing new side channels
  - Require cross-origin isolation for markers?

Prototype

- TAG review: https://github.com/w3ctag/design-reviews/issues/682
- Intent to prototype:
  - Implementation for review in V8/Blink
  - Target Origin Trial in Chrome 98

Open questions

- Interest in breaking down paint marker into:
  - Style
  - Layout
  - Paint
- Events like GC may be hard to isolate timing by origin
- Is JS self profiling the best place for this information?
  - Performance-timeline could be a candidate

Profiling and long tasks

Issues with the Long Tasks API

- Currently hard to debug root cause for long tasks
  - Partially solved by Long Tasks V2 API sketch, though not much movement
- Can profiling help here?

Supplementing with JS Self-Profiling

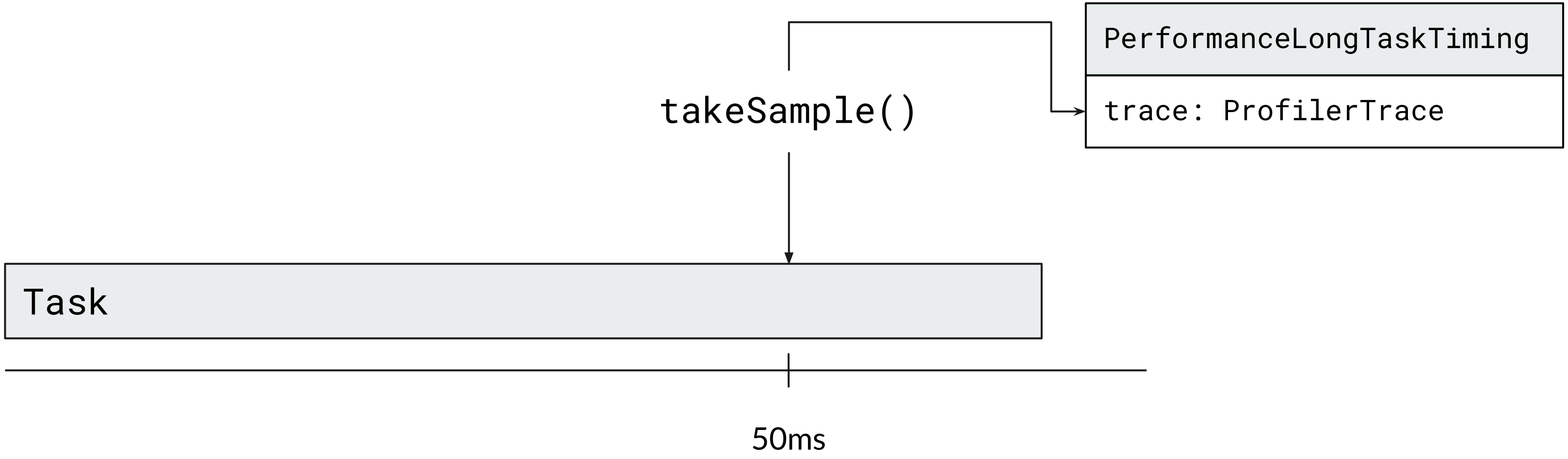
- If running a profiler, you can **cross-correlate with recorded samples to root cause**
    - Find expensive sampled functions
    - Identify UA-level work (e.g. GC/paint/layout) with the marker extension

Drawbacks of JSSP correlation

- Requires recording **all samples**, when we may only care about samples responsible for a long task
    - Increased memory and CPU pressure from sampling and trace processing

Can we do better?

- What if we added an option to **sample only during long tasks**?
    - Existing Document-Policy header js-profiling can hint to starting maintaining code map metadata
    - Active profiler not necessary
- Requires proactively detecting long tasks
    - Schedule 50ms background task to capture a sample (RAIL long task definition)



Sampling long tasks

API modification

```
[Exposed=Window]
interface PerformanceLongTaskTiming : PerformanceEntry {
  readonly attribute FrozenArray<TaskAttributionTiming> attribution;
  readonly attribute ProfilerTrace? trace;
  [Default] object toJSON();
};
```

Security and privacy

- Need to ensure we do not expose attributions across origins
- Actual sampling is a subset of existing profiling functionality
  - Existing cross-origin checks will continue to make this safe

Open questions

- How should we activate this?
  - Add samples when js-profiler document policy is present?
- Is a single sample enough to get signal?
- Will proactive long task sampling be *slower* than a long-running profiler?
  - Additional main-thread work is likely required before each task runs

Discussion

Appendix

Links

- Explainer: https://github.com/WICG/js-self-profiling/pull/55
- Markers TAG review: https://github.com/w3ctag/design-reviews/issues/682

```
[Exposed=Window]
interface Profiler : EventTarget {
  readonly attribute DOMHighResTimeStamp sampleInterval;
  readonly attribute boolean stopped;

  constructor(ProfilerInitOptions options);
  Promise<ProfilerTrace> stop();
};
```

IDL: Profiler