Performance Issues and Optimizations in JavaScript: An Empirical Study

Marija Selakovic
Michael Pradel

May 18, 2016
JavaScript is slow!
JavaScript is not slow! It's fast.
Why Do Developers Optimize JavaScript?

- Still possible to write slow code
- Compiler optimizations are limited
- Deopts and bailouts
This Talk: Empirical study of performance issues and optimizations in JavaScript
Contributions

Better understanding of performance issues in JavaScript

Set of reproducible performance problems [1]

Who Benefits From This Study?

- Application developers
- Developers of program analyses
- Developers of JS engines
Motivating Example

Iterates over all properties of \textit{arg}:

\begin{verbatim}
for (var prop in arg) {
    if (arg.hasOwnProperty(prop)) {
        ....
    }
}
\end{verbatim}

Provides enumerable properties of \textit{arg}:

\begin{verbatim}
var updates = Object.keys(arg);
for (var i=0, l=updates.length; i<l; i++) {
    var prop = updates[i];
    ....
}
\end{verbatim}

Ember.js pull request 11338
Methodology

Subject programs

- 16 popular JavaScript projects
- High number of pull requests

Selection of performance issues

- ~100 performance issues
- Reproducible
- Confirmed and accepted optimizations
What are the main root causes of performance issues in JavaScript?
52% of all issues are caused by *inefficient API usage*
Inefficient API Usage

Multiple functionally equivalent ways to do the same

```
str.split('\\').join('\\\\')
```

```
str.replace(/\//g, '\\\\')
```

Relatively small number of root causes
How complex the optimizations are?
Performance vs. Maintainability
Complexity of Optimizations

Slow code \(\rightarrow\) Fast code

- Median: 10 lines
- 37.5% do not modify number of statements
- 47.2% do not modify cyclomatic complexity
- 14.43% decrease cyclomatic complexity

Relatively simple changes can speedup JavaScript code
What is the performance impact of optimizations?
Performance Impact

- Only positive: 42.68%
- Positive or no impact: 15.89%
- Positive and negative impact: 41.43%

Developers apply optimizations that degrade performance
Are there recurring optimization patterns?
Recurring Optimizations

- 29 studied instances are recurring
- AST-based static analysis
- 139 new instances
- Reported 10 optimizations, 5 accepted

Many optimizations are instances of recurring patterns

For the full list of reported optimizations, see https://github.com/marijaselakovic/JavaScriptIssuesStudy
Can recurring optimizations be applied automatically?
"Apparently, V8’s JIT engineers require that we, as JavaScript developers perform this very simple transformation, since they do not seem capable of performing it themselves"

(Developer of Ember.js)
Preconditions for Automatic Transformations

Challenging to statically analyze these preconditions in JavaScript

Type check:

- `arg` must be an object

```javascript
for (var prop in arg) {
    if (arg.hasOwnProperty(prop)) {
        ....
    }
}
```

Native `hasOwnProperty` function must not be overridden
Conclusions

Systematic study of JavaScript performance issues

- Small number of root causes
- Inefficient API usage
- Relatively simple changes
- Many instances of recurring patterns

Thank you! Questions?
Instances of Recurring Patterns

- Use jQuery `empty` function instead of `html(' ')`
  
  ```javascript
  body.html('') → body.empty()
  ```

- Instead of checking object type with `toString` use `instanceof` operator
  
  ```javascript
  Object.prototype.toString.call(err) === '[object Error]' → err instanceof Error
  ```

- Prefer for loop over functional processing of array
  
  ```javascript
  styles.reduce(  
  function (str, name) {
    return ...;
  }, str);  
  ```

  ```javascript
  for (var i=0; i< styles.length; i++) {
    var name=styles[i];
    str = ...;
  }
  return str;
  ```