Опыт разработки

2006: PHP, MySQL, шаблонизаторы

2007-2009: Flex SDK 3

2009-2011: Гибридные приложения (AS3 + Cpp контейнер)

с 2012: SPA приложения с использованием Closure Compiler (в конце 2016 года в линейке проектов еще появились гибридные приложения JS+Cpp)

С 2018: фронтенд редактора статей

Closure Compiler

1. Релизы примерно раз в месяц
2. Стабильность
3. Не требует лишних зависимостей, работает с обычным JS
4. Оперативное реагирование на issue в гитхабе
5. Вектор на большую прозрачность процесса разработки?

План доклада

1. Минимизация кода
2. Принцип работы
3. Оптимизации компилятора
4. Пошаговый разбор оптимизации кода
5. Советы по использованию
6. Интеграция в живой проект
7. Выводы

Минимизация кода

UglifyJS

```
const cube = x => x * x * x;
const square = x => x * x;

console.log(cube(3))
```

```
const o = o => o * o * o,
  c = o => o * o;

console.log(o(3));
```

Closure Compiler

```
const cube = x => x * x * x;
const square = x => x * x;

console.log(cube(3))
```

```
console.log(27);
```

UglifyJS

```
class Rudder {
  turn() {
     console.log('turn');
  }
}

class Wheel {
  pump() {
     console.log('pump')
  }
}

const car = {
  wheel: new Wheel(),
  rudder: new Rudder(),
};

car.rudder.turn();
```

```
class e {
  turn() {
     console.log("turn");
  }
}

class n {
  pump() {
     console.log("pump");
  }
}

const l = {
  wheel: new n(),
  rudder: new e()
};

l.rudder.turn();
```

Closure Compiler

```
class Rudder {
  turn() {
     console.log('turn');
  }
}

class Wheel {
  pump() {
     console.log('pump')
  }
}

const car = {
  wheel: new Wheel(),
  rudder: new Rudder(),
};

car.rudder.turn();
```

```
console.log("turn");
```

UglifyJS

```
(function() {
  const SERVER_CONFIG = {
     url: 'http://example.com',
     port: 8080,
  };
  function getServerConfig() {
     return SERVER_CONFIG;
  }
  class UrlGenerator {
     getFullUrl(serverConfig) {
        const {url, port} = serverConfig;
        return `${url}:${port}`;
     }
  }

  const u = new UrlGenerator();
  const fullUrl = u.getFullUrl(getServerConfig());
  console.log(fullUrl);
})()
```

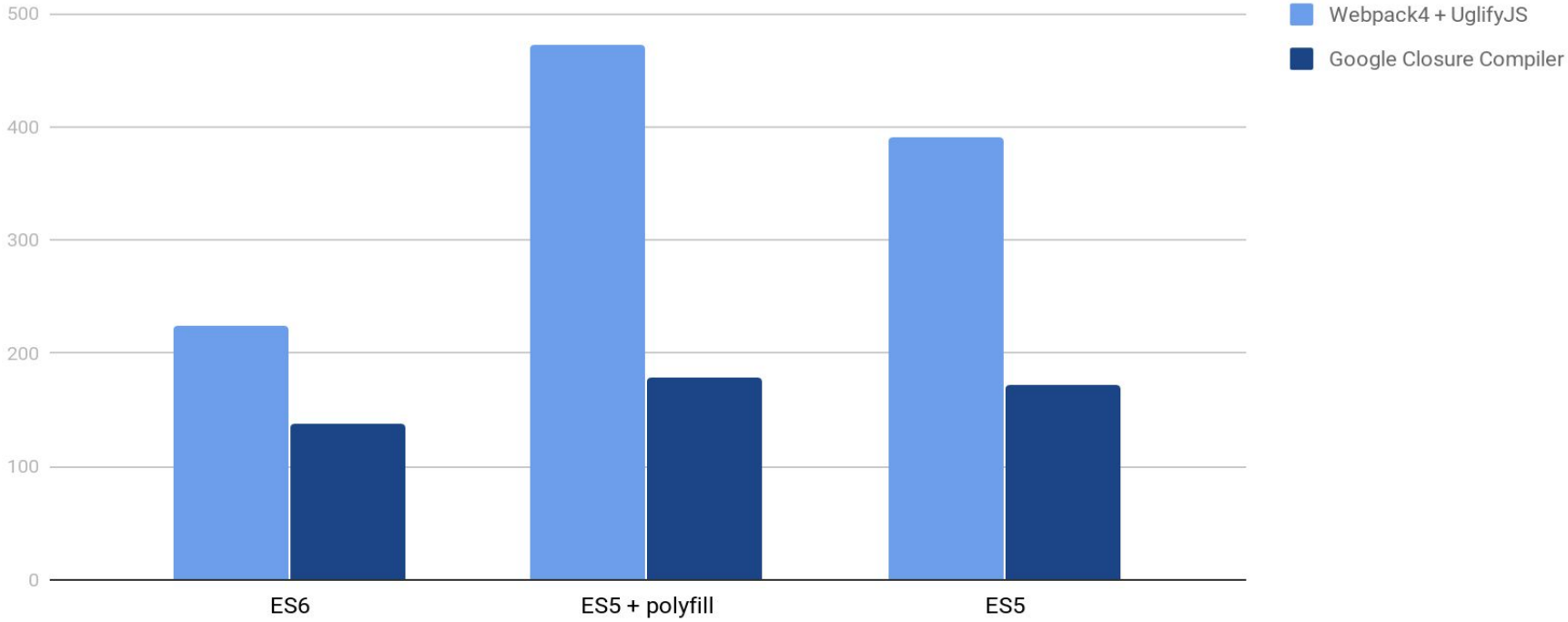
```
!function() {
  const l = {
     url: "http://example.com",
     port: 8080
  };
  const t = new class {
     getFullUrl(l) {
        const {url: t, port: o} = l;
        return `${t}:${o}`;
     }
  }().getFullUrl(l);
  console.log(t);
}();
```

Closure Compiler

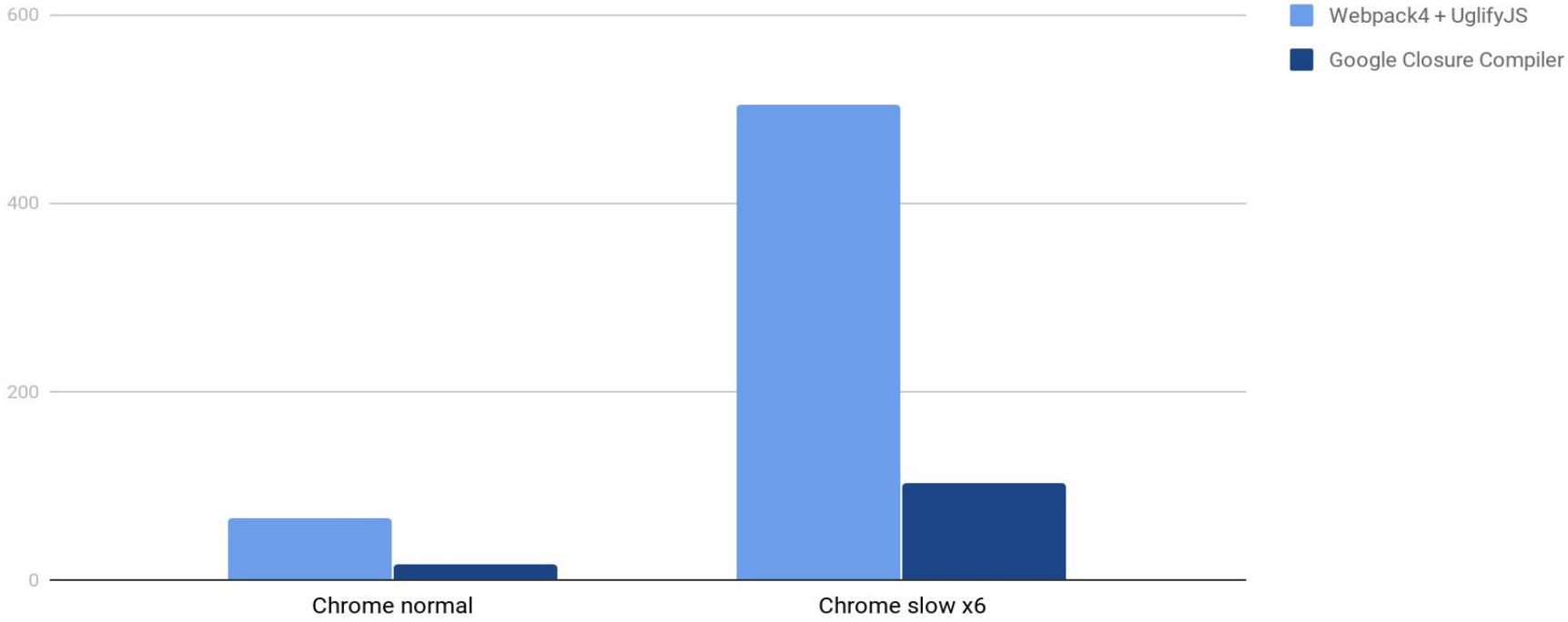
```
(function() {
  const SERVER_CONFIG = {
     url: 'http://example.com',
     port: 8080,
  };
  function getServerConfig() {
     return SERVER_CONFIG;
  }
  class UrlGenerator {
     getFullUrl(serverConfig) {
        const {url, port} = serverConfig;
        return `${url}:${port}`;
     }
  }

  const u = new UrlGenerator();
  const fullUrl = u.getFullUrl(getServerConfig());
  console.log(fullUrl);
})()
```

```
console.log("http://example.com:8080");
```

Размеры бандлов



Время выполнения скрипта



Принцип работы

Процесс компиляции

1. Парсит исходники, строится AST
2. Проверяет валидность кода, попутно происходит транспиляция
3. Оптимизирует AST
4. Генерирует JS-код

Externs

Используется для объявления API внешних библиотек.

```
// externs

/** @type {MyLib} */
var myLib;

class MyLib {
  doSomething(){}
}

// source code
const value = myLib.doSomething();
console.log(value);
```

Exports

Механизм для объявления публичного API приложения.

```
class Foo {
  bar() {}
}
const foo = new Foo;
console.log(foo.bar())

window['Foo'] = Foo;
Foo.prototype['bar'] = Foo.prototype.bar;
```

Аксиомы компилятора

1. Для всех внешних библиотек подключены externs-ы
2. Все методы, публичного апи экспортированы
3. Обращения к свойствам объектов через точку
4. Использование перебора методов класса не защищает методы от удаления, если они не используются явно
5. При объявлении/использовании глобальных свойств надо явно использовать window.prop
6. Разрешается использование this только в конструкторах и методах прототипа
7. Статические методы класса разрешается использовать только по прямой ссылке на класс
8. Запрещается использование super в статических методах, при наследовании динамически сгенерированного класса (например при использовании миксинов)
9. Геттеры/сеттеры не содержат side effect-ов
10. В реализациях toString и valueOf отсутствуют side effect-ы
11. Не должно бросаться исключений при обращении к переменным
12. Оригинальное значение Function.length может измениться

Оптимизации компилятора

Оптимизация AST

1. В зависимости от конфига составляется список оптимизаций
2. Все оптимизации выполняются последовательно
3. Существуют одиночные оптимизации и группы оптимизаций
4. В группах оптимизации повторяются до тех пор, пока есть изменения

Упразднены

- J2CL - специфика
- Closure специфика
- Сборщики информации
- Функционал разбиения на чанки

Порядок оптимизаций

1. normalize
2. optimizeArgumentsArray
3. aggressiveInlineAliases
4. collapseProperties
5. earlyInlineVariables
6. earlyPeepholeOptimizations
7. removeUnusedCode
8. disambiguateProperties
9. codeRemovingLoop
10. devirtualizePrototypeMethods
11. flowSensitiveInlineVariables
12. mainOptimizationLoop
13. flowSensitiveInlineVariables
14. removeUnusedCode
15. collapseAnonymousFunctions
16. extractPrototypeMemberDeclarations
17. ambiguateProperties
18. renameProperties
19. convertToDottedProperties
20. coalesceVariableNames
21. peepholeOptimizations
22. exploitAssign
23. collapseVariableDeclarations
24. denormalize
25. renameVars
26. renameLabels
27. latePeepholeOptimizations

normalize

Нормализует кодовую базу для упрощения дальнейших оптимизаций

- Разделяет объявление переменных
- Выносит инициализацию переменных за пределы циклов
- Конвертирует while в for
- Делает все названия переменных уникальными
- Удаляет дублирование объявления переменных
- Разворачивает стрелочные функции

normalize

Нормализует кодовую базу для упрощения дальнейших оптимизаций

- Разделяет объявление переменных
- Выносит инициализацию переменных за пределы циклов
- Конвертирует while в for
- Делает все названия переменных уникальными
- Удаляет дублирование объявления переменных
- Разворачивает стрелочные функции

```
// до
var a = 0,
  b = foo();

// после
var a = 0;
var b = foo();
```

normalize

Нормализует кодовую базу для упрощения дальнейших оптимизаций

- Разделяет объявление переменных
- Выносит инициализацию переменных за пределы циклов
- Конвертирует while в for
- Делает все названия переменных уникальными
- Удаляет дублирование объявления переменных
- Разворачивает стрелочные функции

```
// до
for(var i = 0; i < 10; ++i)
{
  // ...
}

// после
var i = 0;
for(; i < 10; ++i)
{
  // ...
}
```

normalize

Нормализует кодовую базу для упрощения дальнейших оптимизаций

- Разделяет объявление переменных
- Выносит инициализацию переменных за пределы циклов
- Конвертирует while в for
- Делает все названия переменных уникальными
- Удаляет дублирование объявления переменных
- Разворачивает стрелочные функции

```
// до
while(c < b)
{
  // ...
}

// после
for(; c < b;)
{
  // ...
}
```

normalize

Нормализует кодовую базу для упрощения дальнейших оптимизаций

- Разделяет объявление переменных
- Выносит инициализацию переменных за пределы циклов
- Конвертирует while в for
- Делает все названия переменных уникальными
- Удаляет дублирование объявления переменных
- Разворачивает стрелочные функции

```
// до
let a = 'outer';
{
  let a = 'inner';
}

// после
let a = 'outer';
{
  let a$jscomp$1 = 'inner';
}
```

normalize

Нормализует кодовую базу для упрощения дальнейших оптимизаций

- Разделяет объявление переменных
- Выносит инициализацию переменных за пределы циклов
- Конвертирует while в for
- Делает все названия переменных уникальными
- Удаляет дублирование объявления переменных
- Разворачивает стрелочные функции

```
// до
var a = 1;
var a = 2;

// после
var a = 1;
a = 2;
```

normalize

Нормализует кодовую базу для упрощения дальнейших оптимизаций

- Разделяет объявление переменных
- Выносит инициализацию переменных за пределы циклов
- Конвертирует while в for
- Делает все названия переменных уникальными
- Удаляет дублирование объявления переменных
- Разворачивает стрелочные функции

```
// до
() => 1;

// после
() => {
  return 1;
}
```

optimizeArgumentsArray

Убирает использование arguments в функциях

```
// до
function() {
  alert(arguments[0] + arguments[1])
}

// после
function(a, b) {
  alert(a + b)
}
```

aggressiveInlineAliases

- Инлайнит алиасы, когда они по факту являются константами
- В случае изменения объекта, после объявления алиаса, изменений не будет

```
// до
var obj = {
  foo: {
     bar: 3
  }
};
var bar = obj.foo.bar;
alert(bar);

// после
var obj = {
  foo: {
     bar: 3
  }
};
var bar = null;
alert(obj.foo.bar);
```

aggressiveInlineAliases

- Инлайнит алиасы, когда они по факту являются константами
- В случае изменения объекта, после объявления алиаса, изменений не будет

```
var obj = {
  foo: {
     bar: 3
  }
};
var bar = obj.foo.bar;
obj.foo = {};
alert(bar);
```

collapseProperties

Схлапывает глобальные объекты/неймспейсы путем замены ‘.’ на ‘$’ в их имени. Это уменьшает время доступа к свойству в браузере и позволяет переименовывать эффективнее.

```
// до
var a = {};
a.b = {};
a.b.c = {};
var d = a.b.c;

// после
var a$b$c = {};
var d = a$b$c;
```

inlineVariables

Находит переменные, которые используются один раз и инлайнит их.

```
// до
var a = foo();
a && alert(3);

// после
foo() && alert(3);
```

earlyPeepholeOptimizations

Оптимизация для удаления неиспользуемого кода

- Удаляет блоки без сайд эффектов
- Удаляет условные блоки/циклы, которые никогда не будут исполнены

earlyPeepholeOptimizations

Оптимизация для удаления неиспользуемого кода

- Удаляет блоки без сайд эффектов
- Удаляет условные блоки/циклы, которые никогда не будут исполнены

```
{
  let x;
  'hi'
}
```

earlyPeepholeOptimizations

Оптимизация для удаления неиспользуемого кода

- Удаляет блоки без сайд эффектов
- Удаляет условные блоки/циклы, которые никогда не будут исполнены

```
// до
if (1) {
  x = 1;
} else {
  x = 2;
}

// после
x = 1;
```

removeUnusedCode

Сборщик мусора для объявлений функций и переменных

- Удаляет неиспользуемые классы, свойства классов и переменные
- При наличии возможных сайд эффектов, оставляет правую часть

removeUnusedCode

Сборщик мусора для объявлений функций и переменных

- Удаляет неиспользуемые классы, свойства классов и переменные
- При наличии возможных сайд эффектов, оставляет правую часть

```
// до
var foo = 3,
var bar = 4;
window.foo = foo;

// после
var foo = 3;
window.foo = foo;
```

removeUnusedCode

Сборщик мусора для объявлений функций и переменных

- Удаляет неиспользуемые классы, свойства классов и переменные
- При наличии возможных сайд эффектов, оставляет правую часть

```
// до
var a = doSomething()

// после
doSomething()
```

disambiguateProperties

Переименовывает свойства объектов, добавляя информацию о владельце

```
// до
Foo.a

// после
Foo.Foo$a
```

codeRemovingLoop

1. inlineValiables
2. collapseObjectLiterals
3. removeUnusedCode
4. peepholeOptimizations
5. removeUnreachableCode

inlineVariables (1 из 5)

Описание

collapseObjectLiterals (2 из 5)

Безопасная версия collapseProperties

```
// до
function f() {
  var a = {
     x:x(),
     y:y()
  };
  f(a.x, a.y);
}
// после
function f() {
  var JSCompiler_object_inline_x_0 = x();
  var JSCompiler_object_inline_y_1 = y();
  f(JSCompiler_object_inline_x_0,
JSCompiler_object_inline_y_1);
}
```

removeUnusedCode (3 из 5)

Описание

peepholeOptimizations (4 из 5)

Включает в себя earlyPeepholeOptimizations

- Оптимизирует точки выхода return/break/continue
- Минимизирует условия
- Заменяет if-ы тернарным оператором
- Заменяет конструкторы Array/Object литералами
- Оптимизирует инициализацию массивов/объектов
- Схлапывает использование базовых методов
- Схлапывает константные выражения

peepholeOptimizations (4 из 5)

Включает в себя earlyPeepholeOptimizations

- Оптимизирует точки выхода return/break/continue
- Минимизирует условия
- Заменяет if-ы тернарным оператором
- Заменяет конструкторы Array/Object литералами
- Оптимизирует инициализацию массивов/объектов
- Схлапывает использование базовых методов
- Схлапывает константные выражения

```
// до
function f() {
  if (success()) {
     return 1;
  } else {
     return 2
  }
}

// после
function f() {
  return success()
     ? 1
     : 2;
}
```

peepholeOptimizations (4 из 5)

Включает в себя earlyPeepholeOptimizations

- Оптимизирует точки выхода return/break/continue
- Минимизирует условия
- Заменяет if-ы тернарным оператором
- Заменяет конструкторы Array/Object литералами
- Оптимизирует инициализацию массивов/объектов
- Схлапывает использование базовых методов
- Схлапывает константные выражения

```
// до
if (x) {
  foo();
}

// после
x && foo();
```

peepholeOptimizations (4 из 5)

Включает в себя earlyPeepholeOptimizations

- Оптимизирует точки выхода return/break/continue
- Минимизирует условия
- Заменяет if-ы тернарным оператором
- Заменяет конструкторы Array/Object литералами
- Оптимизирует инициализацию массивов/объектов
- Схлапывает использование базовых методов
- Схлапывает константные выражения

```
// до
if (x) {
  foo();
} else {
  bar();
}

// после
x
  ? foo()
  : bar();
```

peepholeOptimizations (4 из 5)

Включает в себя earlyPeepholeOptimizations

- Оптимизирует точки выхода return/break/continue
- Минимизирует условия
- Заменяет if-ы тернарным оператором
- Заменяет конструкторы Array/Object литералами
- Оптимизирует инициализацию массивов/объектов
- Схлапывает использование базовых методов
- Схлапывает константные выражения

```
// до
var x = new Array();
var y = new Object();

// после
var x = [];
var y = {};
```

peepholeOptimizations (4 из 5)

Включает в себя earlyPeepholeOptimizations

- Оптимизирует точки выхода return/break/continue
- Минимизирует условия
- Заменяет if-ы тернарным оператором
- Заменяет конструкторы Array/Object литералами
- Оптимизирует инициализацию массивов/объектов
- Схлапывает использование базовых методов
- Схлапывает константные выражения

```
// до
var a = [];
a[0] = 0;

// после
var a = [0];
```

peepholeOptimizations (4 из 5)

Включает в себя earlyPeepholeOptimizations

- Оптимизирует точки выхода return/break/continue
- Минимизирует условия
- Заменяет if-ы тернарным оператором
- Заменяет конструкторы Array/Object литералами
- Оптимизирует инициализацию массивов/объектов
- Схлапывает использование базовых методов
- Схлапывает константные выражения

```
// до
x = 'abcdef'.indexOf('b');

// после
x = 1;
```

peepholeOptimizations (4 из 5)

Включает в себя earlyPeepholeOptimizations

- Оптимизирует точки выхода return/break/continue
- Минимизирует условия
- Заменяет if-ы тернарным оператором
- Заменяет конструкторы Array/Object литералами
- Оптимизирует инициализацию массивов/объектов
- Схлапывает использование базовых методов
- Схлапывает константные выражения

```
// до
x = 1 + 7;

// после
x = 8;
```

removeUnreachableCode (5 из 5)

- Удаляет код после return/break/throw
- Удаляет обращение к свойствам, без сохранения результата
- Удаляет неиспользуемые литералы

```
// до
if (x) {
  return;
  alert('unreachable');
}

// после
if (x) {
  return;
}
```

removeUnreachableCode (5 из 5)

- Удаляет код после return/break/throw
- Удаляет обращение к свойствам, без сохранения результата
- Удаляет неиспользуемые литералы

```
MyClass.prototype.memberName;

element.offsetHeight;
```

removeUnreachableCode (5 из 5)

- Удаляет код после return/break/throw
- Удаляет обращение к свойствам, без сохранения результата
- Удаляет неиспользуемые литералы

```
true;

'hi';

if (x) {
  1;
}
```

devirtualizePrototypeMethods

Заменяет вызовы методов прототипа вызовом функции, первым параметром в которую this приходит первым аргументом

```
// до
A.prototype.accumulate = function(value) {
  this.total += value;
  return this.total;
}
var total = a.accumulate(2);

// после
var accumulate = function(self, value) {
  self.total += value;
  return self.total;
}
var total = accumulate(a, 2);
```

flowSensitiveInlineVariables

Находит переменные, которые используются один раз и инлайнит их.

mainOptimizationLoop

1. codeRemovingLoop
2. inlineSimpleMethods
3. inlineProperties
4. deadPropertyAssignmentElimination
5. optimizeCalls
6. inlineFunctions
7. deadAssignmentsElimination

codeRemovingLoop (1 из 7)

Описание

inlineSimpleMethods (2 из 7)

Инлайнит вызовы функций-геттеров. Сами функции не трогает.

```
// до
Foo.prototype.bar = function() {
  return this.baz;
}
var x = (new Foo).bar();

// после
Foo.prototype.bar = function() {
  return this.baz;
}
var x = (new Foo).baz;
```

inlineProperties (3 из 7)

Ищет константные свойства и инлайнит их использование

```
// до
/** @constructor */
function Foo() {
  /** @type {?number} */
  this.a = 1;
}
var f = new Foo;
var x = f.a;

// после
/** @constructor */
function Foo() {
  /** @type {?number} */
  this.a = 1;
}
var f = new Foo;
var x = 1;
```

deadPropertyAssignmentElimination (4 из 7)

Удаляет лишние присваивания свойству класса

```
// до
var foo = function() {
  this.a = 10;
  this.a = 20;
}

// после
var foo = function() {
  10;
  this.a = 20;
}
```

optimizeCalls (5 из 7)

Оптимизирует вызовы функций.

- Если во всех вызовах функции в параметр передается одно значение - добавляет переменную в тело функции, а параметр выпиливает.
- Если возвращаемое значение не используется, то функция перестает возвращать значение

```
// до
function foo(a,b,c) {
  console.log(a,b);
  return c;
}
foo(1,2,3);
foo(0,2,4);

// после
function foo(a,c) {
  var b = 2;
  console.log(a,b);
  c;
  return;
}
foo(1, 3);
foo(0, 4);
```

optimizeCalls (5 из 7)

Оптимизирует вызовы функций.

- Если во всех вызовах функции в параметр передается одно значение - добавляет переменную в тело функции, а параметр выпиливает.

- Если возвращаемое значение не используется, то функция перестает возвращать значение

```
// до
function foo(a,b,c) {
  console.log(a,b);
  return c;
}
foo(1,2,3);
foo(0,2,4);

// после
function foo(a,c) {
  var b = 2;
  console.log(a,b);
  c;
  return;
}
foo(1, 3);
foo(0, 4);
```

inlineFunctions (6 из 7)

Инлайнит вызовы функций

inlineFunctions (6 из 7)

- Любые функции, которые вызываются только 1 раз. Сворачивает IIFE
- либо тело функции меньше вызова функции
- При инлайне сложных функций использует метки

```
// до
function a() {
  return {
    foo: 'foo',
    bar: 'bar'
  };
}
var x = a();

// после
var x = {
  foo:"foo",
  bar:"bar"
};
```

inlineFunctions (6 из 7)

- Любые функции, которые вызываются только 1 раз. Сворачивает IIFE
- либо тело функции меньше вызова функции
- При инлайне сложных функций использует метки

```
// до
function a() {
  return 4;
}
a();
a();
a();

// после
4;
4;
4;
```

inlineFunctions (6 из 7)

- Любые функции, которые вызываются только 1 раз. Сворачивает IIFE
- либо тело функции меньше вызова функции
- При инлайне сложных функций использует метки

```
// до
function a() {
  if (window.b)
     return 1;
  return 2;
}
var x = a();

// после
var x;
{
  JSCompiler_inline_label_a_0: {
     if (window.b) {
        x = 1;
        break JSCompiler_inline_label_a_0;
     }
     x = 2;
  }
}
```

deadAssignmentsElimination (7 из 7)

Удаляет присваивания переменным, если в дальнейшем эти переменные не используются

```
// до
function foo() {
  var a;
  a=bar();
}

// после
function foo() {
  var a;
  bar();
}
```

flowSensitiveInlineVariables

Описание

removeUnusedCode

Описание

collapseAnonymousFunctions

Преобразует присваивание переменной анонимной функции в именованную функцию

```
// до
var f = function() {}

// после
function f() {}
```

extractPrototypeMemberDeclarations

Выносит алиас на прототип в отдельную переменную при наличии нескольких методов класса

```
// до
function B() {}
B.prototype.foo = function() {}
B.prototype.bar = function() {}

// после
function B() {}
x = B.prototype;
x.foo = function() {}
x.bar = function() {}
```

ambiguateProperties

Переименовывает свойства/методы различных классов в одинаковые имена. Использует информацию о типах, чтобы не допустить коллизий

```
// до
Foo.Foo$p1 = 1;
Foo.Foo$p2 = 2;
Bar.Bar$p1 = 3

// после
Foo.a = 1;
Foo.b = 2;
Bar.a = 3
```

renameProperties

Переименовывает свойства/методы, к которым обращаются через точку и упоминания отсутствуют в extern-ах. Не использует информацию о типах

```
// до
Bar.prototype.getA = function(){};
Bar.prototype.getB = function(){};
bar.getA();

// после
Bar.prototype.a = function(){};
Bar.prototype.b = function(){};
bar.a();
```

convertToDottedProperties

Конвертирует обращение к свойствам через кавычки, на обращение через точку

```
// до
a['b'];

// после
a.b;
```

coalesceVariableNames

Переиспользует имя переменной, если это возможно. Уменьшает количество уникальных переменных для лучшего переименования и в конечном итоге лучших результатов gzip-сжатия

```
// до
var x = 1;
print(x);
var y = 2;
print(y);

// после
var x = 1;
print(x);
x = 2;
print(x);
```

peepholeOptimizations

Описание

exploitAssign

Выстраивает присваивания в цепочку

```
// до
a = 1;
b = 1;

// после
a = b = 1;
```

collapseVariableDeclarations

Объединяет последовательные объявления переменных в одно

```
// до
var a;
var b = 1;
var c = 2;

// после
var a,
  b = 1,
  c = 2;
```

denormalize

Обратные normalize изменения

renameVars

Переименовывает переменные для уменьшения размера кода и обфускации

```
// до
var Foo;
var Bar, y;
function x() {
  Bar++;
}

// после
var a;
var b, c;
function d() {
  b++;
}
```

renameLabels

Оптимизация меток, которые остались в результате инлайна функций

- Переименовывает, если метка используется
- Иначе удаляет

renameLabels

Оптимизация меток, которые остались в результате инлайна функций

- Переименовывает, если метка используется
- Иначе удаляет

```
// до
Foo: {
  a();
  break Foo;
}

// после
a: {
  a();
  break a;
}
```

renameLabels

Оптимизация меток, которые остались в результате инлайна функций

- Переименовывает, если метка используется
- Иначе удаляет

```
// до
Foo: {
  a();
}

// после
a();
```

latePeepholeOptimizations

Работает аналогично peepholeOptimizations,
но приоритет на минимизацию кода.

Пошаговый разбор оптимизации кода

earlyInlineVariables

```
(function() {
  function getServerConfig() {
     return SERVER_CONFIG;
  }
  var SERVER_CONFIG = {url:"http://example.com",
port:8080};
  var UrlGenerator = function() {};
  UrlGenerator.prototype.getFullUrl =
function(serverConfig) {
     var $jscomp$destructuring$var0 = serverConfig;
     var url = $jscomp$destructuring$var0.url;
     var port = $jscomp$destructuring$var0.port;
     return url + ":" + port;
  };
  var u = new UrlGenerator;
  var fullUrl = u.getFullUrl(getServerConfig());
  console.log(fullUrl);
})();
```

```
(function() {
  var SERVER_CONFIG = {url:"http://example.com",
port:8080};
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl =
function(serverConfig) {
     var $jscomp$destructuring$var0 = serverConfig;
     return $jscomp$destructuring$var0.url + ":" +
$jscomp$destructuring$var0.port;
  };
  var fullUrl = (new
UrlGenerator).getFullUrl(function getServerConfig()
{
     return SERVER_CONFIG;
  }());
  console.log(fullUrl);
})();
```

removeUnusedCode

```
(function() {
  var SERVER_CONFIG = {url:"http://example.com",
port:8080};
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl =
function(serverConfig) {
    var $jscomp$destructuring$var0 = serverConfig;
    return $jscomp$destructuring$var0.url + ":" +
$jscomp$destructuring$var0.port;
  };
  var fullUrl = (new
UrlGenerator).getFullUrl(function getServerConfig()
{
    return SERVER_CONFIG;
  }());
  console.log(fullUrl);
})();
```

```
(function() {
  var SERVER_CONFIG = {url:"http://example.com",
port:8080};
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl =
function(serverConfig) {
    var $jscomp$destructuring$var0 = serverConfig;
    return $jscomp$destructuring$var0.url + ":" +
$jscomp$destructuring$var0.port;
  };
  var fullUrl = (new
UrlGenerator).getFullUrl(function() {
    return SERVER_CONFIG;
  }());
  console.log(fullUrl);
})();
```

inlineVariables

```
(function() {
  var SERVER_CONFIG = {url:"http://example.com",
port:8080};
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl =
function(serverConfig) {
    var $jscomp$destructuring$var0 = serverConfig;
    return $jscomp$destructuring$var0.url + ":" +
$jscomp$destructuring$var0.port;
  };
  var fullUrl = (new
UrlGenerator).getFullUrl(function() {
    return SERVER_CONFIG;
  }());
  console.log(fullUrl);
})();
```

```
(function() {
  var SERVER_CONFIG = {url:"http://example.com",
port:8080};
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl =
function(serverConfig) {
    return serverConfig.url + ":" +
serverConfig.port;
  };
  console.log((new
UrlGenerator).getFullUrl(function() {
    return SERVER_CONFIG;
  }()));
})();
```

inlineFunctions

```
(function() {
  var SERVER_CONFIG = {url:"http://example.com",
port:8080};
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl =
function(serverConfig) {
    return serverConfig.url + ":" +
serverConfig.port;
  };
  console.log((new
UrlGenerator).getFullUrl(function() {
    return SERVER_CONFIG;
  }()));
})();
```

```
(function() {
  var SERVER_CONFIG = {url:"http://example.com",
port:8080};
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl =
function(serverConfig) {
    return serverConfig.url + ":" +
serverConfig.port;
  };
  console.log((new
UrlGenerator).getFullUrl(SERVER_CONFIG));
})();
```

inlineVariables

```
(function() {
  var SERVER_CONFIG = {url:"http://example.com",
port:8080};
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl =
function(serverConfig) {
    return serverConfig.url + ":" +
serverConfig.port;
  };
  console.log((new
UrlGenerator).getFullUrl(SERVER_CONFIG));
})();
```

```
(function() {
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl =
function(serverConfig) {
    return serverConfig.url + ":" +
serverConfig.port;
  };
  console.log((new
UrlGenerator).getFullUrl({url:"http://example.com",
port:8080}));
})();
```

optimizeCalls

```
(function() {
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl =
function(serverConfig) {
     return serverConfig.url + ":" +
serverConfig.port;
  };
  console.log((new
UrlGenerator).getFullUrl({url:"http://example.com",
port:8080}));
})();
```

```
(function() {
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl = function() {
     var serverConfig = {url:"http://example.com",
port:8080};
     return serverConfig.url + ":" +
serverConfig.port;
  };
  console.log((new UrlGenerator).getFullUrl());
})();
```

collapseObjectLiterals

```
(function() {
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl = function() {
    var serverConfig = {url:"http://example.com",
port:8080};
    return serverConfig.url + ":" +
serverConfig.port;
  };
  console.log((new UrlGenerator).getFullUrl());
})();
```

```
(function() {
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl = function() {
    var JSCompiler_object_inline_url_0 =
"http://example.com";
    var JSCompiler_object_inline_port_1 = 8080;
    return JSCompiler_object_inline_url_0 + ":" +
JSCompiler_object_inline_port_1;
  };
  console.log((new UrlGenerator).getFullUrl());
})();
```

inlineVariables

```
(function() {
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl = function() {
    var JSCompiler_object_inline_url_0 =
"http://example.com";
    var JSCompiler_object_inline_port_1 = 8080;
    return JSCompiler_object_inline_url_0 + ":" +
JSCompiler_object_inline_port_1;
  };
  console.log((new UrlGenerator).getFullUrl());
})();
```

```
(function() {
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl = function() {
    return "http://example.com" + ":" + 8080;
  };
  console.log((new UrlGenerator).getFullUrl());
})();
```

peepholeOptimizations

```
(function() {
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl = function() {
     return "http://example.com" + ":" + 8080;
  };
  console.log((new UrlGenerator).getFullUrl());
})();
```

```
(function() {
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl = function() {
     return "http://example.com:8080";
  };
  console.log((new UrlGenerator).getFullUrl());
})();
```

inlineSimpleMethods

```
(function() {
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl = function() {
     return "http://example.com:8080";
  };
  console.log((new UrlGenerator).getFullUrl());
})();
```

```
(function() {
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl = function() {
     return "http://example.com:8080";
  };
  console.log("http://example.com:8080");
})();
```

removeUnusedCode

```
(function() {
  var UrlGenerator = function() {
  };
  UrlGenerator.prototype.getFullUrl = function() {
     return "http://example.com:8080";
  };
  console.log("http://example.com:8080");
})();
```

```
(function() {
  console.log("http://example.com:8080");
})();
```

inlineFunctions

```
(function() {
  console.log("http://example.com:8080");
})();
```

```
{
  console.log("http://example.com:8080");
}
```

peepholeOptimizations

```
{
  console.log("http://example.com:8080");
}
```

```
console.log("http://example.com:8080");
```

Советы по использованию

TypeScript как анализатор JS-кода

- Проверка кода в режиме реального времени
- Систему типов, доступную через JS-doc, можно расширить тайпингами

```
/**
* @param {{
*   a: number,
*   b: string,
* }} data
*/
function foo(data) {
  console.log(data.hello)
}

foo(123456789)
```

TypeScript как анализатор JS-кода

- Проверка кода в режиме реального времени
- Систему типов, доступную через JS-doc, можно расширить тайпингами

```
// typings
interface Foo {
  (id: string): void
  (a1: number, a1: number): void
}

// externs
/** @typedef {Function} */
let Foo

// source
/**
* @param {Foo} callback
*/
function foo(callback) {
  callback(1, 2)
  callback('1')
  callback(1)
  callback('1', 2)
}
```

JS Conformance Framework

Автоматическая проверка кода на соответствие определенным требованиям, таким как запрет доступа к определенному свойству или вызовы определенной функции.

Описание на Closure Compiler Wiki

Пример конфига

```
requirement: {
  type: BANNED_PROPERTY_WRITE
  value: 'Location.prototype.href'
  error_message: 'Assignment to Location.prototype.href '
                 'is not allowed. Use '
                 'goog.dom.safe.setLocationHref instead.'
  whitelist: 'javascript/closure/dom/safe.js'
}
```

JS Conformance Framework

1. BanNullDeref - проверка на null при вызове метода у объекта
2. Блокирование функционала, использование которого считается плохой практикой
3. Разделение evironment-ов

```
/**
* @param {Array<{name: string}>} users
* @param {number} index
*/
function getName(users, index) {
  return users[index].name
}


// output
input0:6: WARNING - Violation: BanNullDeref
  return users[index].name
         ^^^^^^^^^^^^
```

JS Conformance Framework

1. BanNullDeref - проверка на null при вызове метода у объекта
2. Блокирование функционала, использование которого считается плохой практикой
3. Разделение evironment-ов

```
if (window.orientation == 90) {
  console.log('isLandscape');
}


// output
input0:1: WARNING - Violation: use isLandscape
from environment module instead.
if (window.orientation == 90) {
    ^^^^^^^^^^^^^^^^^^
```

JS Conformance Framework

1. BanNullDeref - проверка на null при вызове метода у объекта
2. Блокирование функционала, использование которого считается плохой практикой
3. Разделение evironment-ов

```
/**
* @param {Element} element
*/
function markAsVisited(element) {
  element.classList.add('visited');
}

// output
input0:5: WARNING - Violation: classList is not
supported by IE. Use classlist module instead.
  element.classList.add('visited');
  ^^^^^^^^^^^^^^^^^
```

Объявление внешнего API

```
// externs
/** @interface */
class FooApi {
  bar() {}
}

// source code
/** @implements {FooApi} */
class Foo {
  bar() {}
}
const f = new Foo;
console.log(f.bar());
window['Foo'] = Foo;
```

```
var a = function() {};
a.prototype.bar = function() {};
var b = new a;
console.log(b.bar());
window.Foo = a;
```

Объявление внешнего API

```
// при использовании extern-ов
var a = function() {};
a.prototype.longMethodName = function() {};
var b = new a;
console.log(b.longMethodName());
console.log(b.longMethodName());
console.log(b.longMethodName());
window.Foo = a;
```

```
// при использовании export-ов
var a = function() {};
a.prototype.a = function() {};
var b = new a;
console.log(b.a());
console.log(b.a());
console.log(b.a());
window.Foo = a;
a.prototype.longMethodName = a.prototype.a;
```

Объявление параметров с сервера

```
/**
* @param {Object} data
*/
function process(data) {
  const id = data['id'];
  const name = data['name'];
  // ...
}
```

```
function b(a) {
  var c = a.id;
  var d = a.name;
}
```

Объявление параметров с сервера

```
// externs
/**
* @typedef {{
*   id: string,
*   name: string,
* }}
*/
let UserData;

// source code
/**
* @param {UserData} data
*/
function process(data) {
  const {id, name} = data;
  // ...
}
```

```
function b(a) {
  var c = a.id;
  var d = a.name;
}
```

Объявление параметров с сервера

```
// externs
/**
* @typedef {{
*   id: string,
*   name: string,
* }}
*/
let UserData;

// source code
class Foo {
  name() {}
}
```

```
class e {
  name() {}
}
```

Объявление параметров с сервера

```
// externs
/**
* @dict
*/
class UserData {}

// source code
/**
* @param {UserData} data
*/
function process(data) {
  const id = data['id'];
  const name = data.name; // warning
  // ...
}
```

```
function b(a) {
  var c = a.id;
  var d = a.name;
}
```

Интеграция в живой проект

Обобщенный механизм перевода на GCC

- Использовать обращение к свойствам объекта только через точку
- Не использовать динамическую генерацию имен свойств
- С помощью экспортов прописать API приложения
- Вынести сторонние библиотеки в отдельные файлы, прописать для них extern-ы

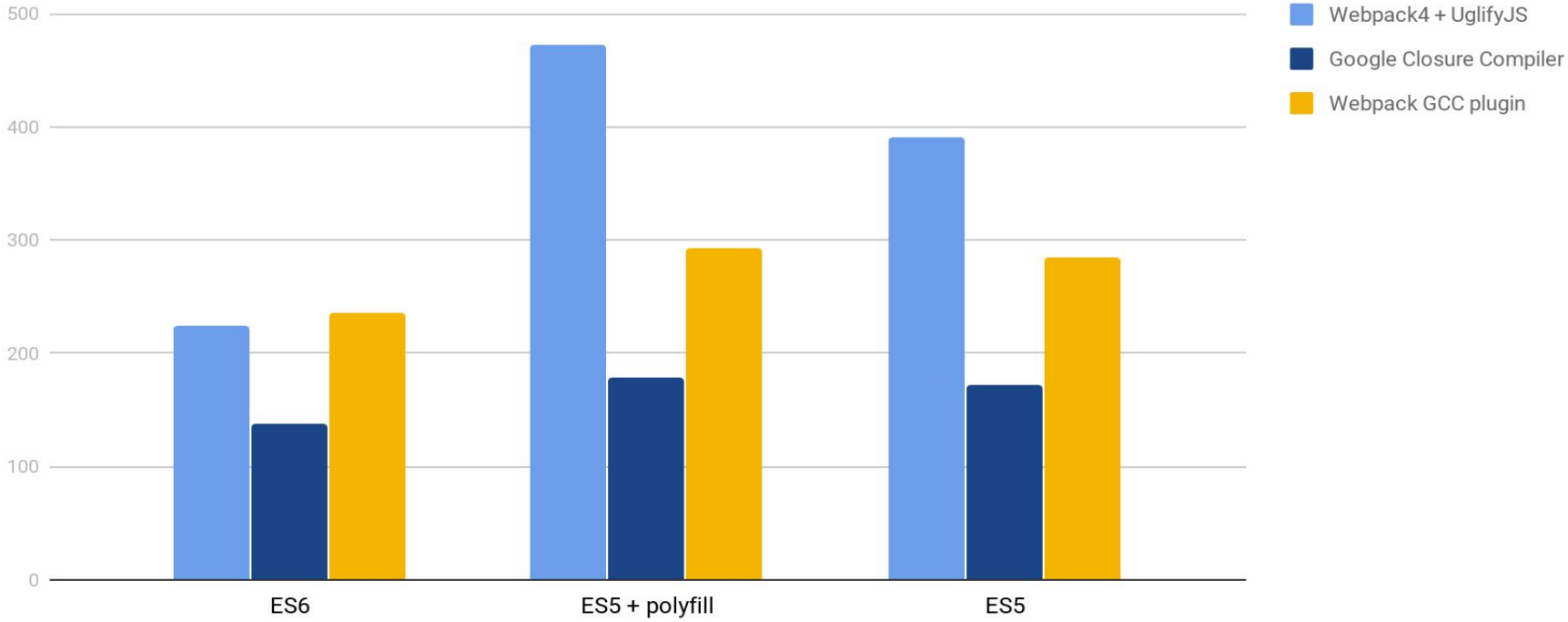
Поддержка node_modules

```
--module_resolution=NODE
--process_common_js_modules

--js node_modules/**/package.json
--js node_modules/**/**.js
```

GCC плагин для Webpack

- Минимизация: замена для UglifyJS
- Транспиляция: замена для babel

Размеры бандлов



Использование с Webpack

- Production без минимизации - переименовываются классы
- CssModules - сериализация объектов через JSON.stringify()

```
// CONCATENATED MODULE: ./src/foo.js
class foo_Foo {
  bar() {}
}
// CONCATENATED MODULE: ./src/index.js
/**
* @param {Foo} foo
*/
function hello(foo) {
  console.log(foo)
}
```

Использование с Webpack

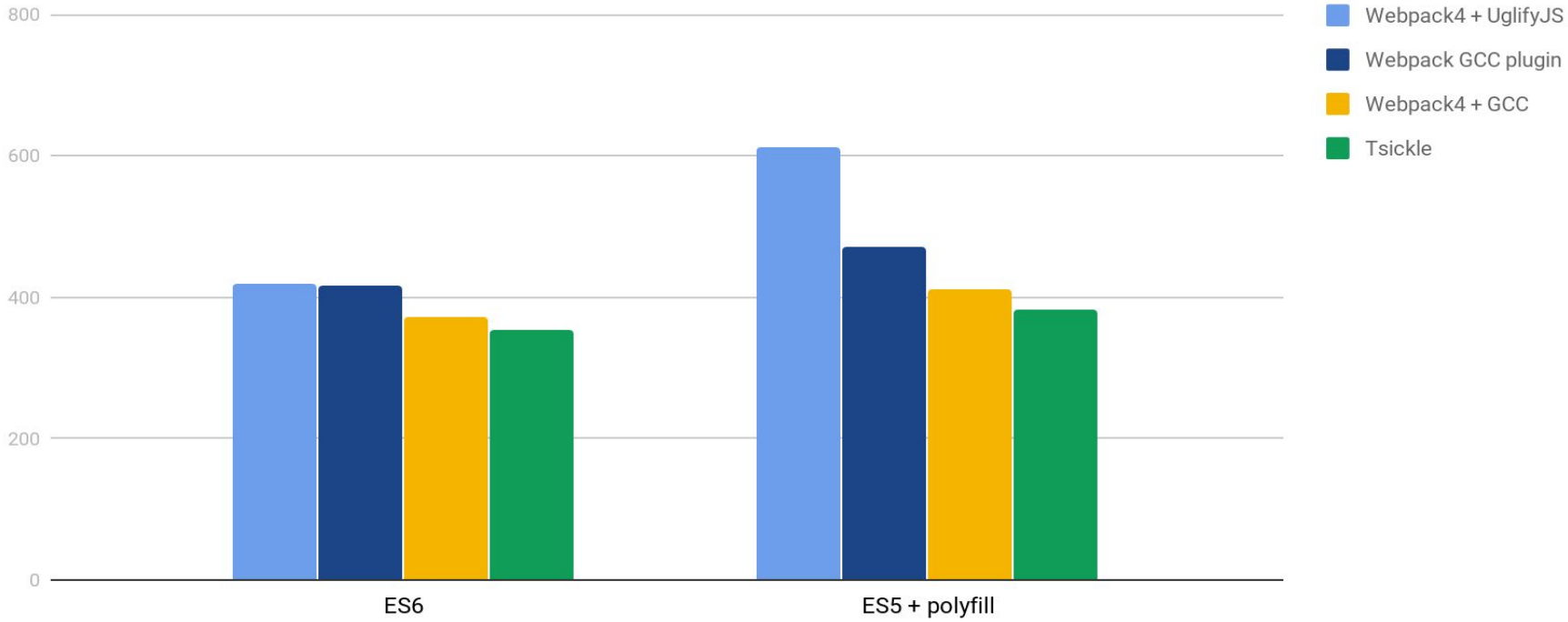
- Production без минимизации - переименовываются классы
- CssModules - сериализация объектов через JSON.stringify()

```
let html = `<h2
class="${_style_css__WEBPACK_IMPORTED_MODULE_0
___default.a.my_css_selector}">I should be
displayed in blue.</h2>`;

// removed by extract-text-webpack-plugin
module.exports =
{"my_css_selector":"app__my_css_selector__3ZGpn"};
```

Tsickle — TypeScript to Closure Translator

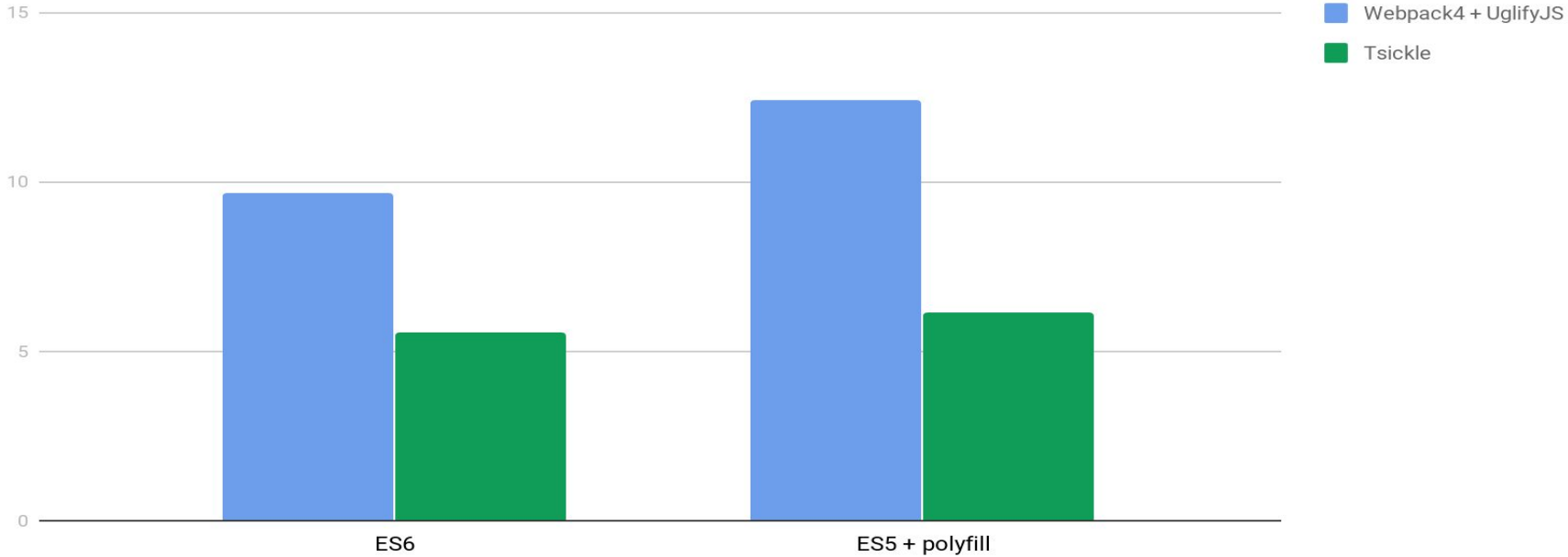
1. Создал shim файлы для библиотек из node_modules
2. Циклические ссылки: вручную заменить goog.require на goog.forwardDeclare
3. В кодовой базе все интерфейсы внешнего API объявить через declare
4. Прописать экспорты для точки входа

Tsickle — TypeScript to Closure Translator



Пример использования Tsickle

Репозиторий: https://github.com/ikabirov/snake



Выводы

1. Размер бандла имеет значение, присмотритесь к GCC
2. Если у вас Webpack проект с компиляцией в ES5, воспользуйтесь GCC плагином
3. При старте нового проекта, рассмотрите вариант связки TS и GCC

Вопросы?