es6 goodies in node 4

Patrick Mueller **`@pmuellr`**, **`muellerware.org`**
senior node engineer at NodeSource

http://pmuellr.github.io/slides/2015/09-es6-goodies
http://pmuellr.github.io/slides/ (all of Patrick's slides)

node 4.0 announced!

Have You Heard Of It?

> Node.js v4.0.0 has just been released. This is a huge milestone for Node under the new Node.js Foundation. All thanks to the development process inherited from the io.js fork.

All the io.js work has now been integrated back into the core node.js stream!

But wait, there's more ...

new in node 4.0

- **EcmaScript 6 features!**
  - because: V8 4.5
  - same level of V8 as shipped in Chrome 45
- moar stuff

wat EcmaScript 6 features?

my favorites:

- template strings - \`new kind of strings!\`
- classes - **`class X { foo() { log("in foo") } }`**
- arrow functions - **`cb => cb("shorter functions")`**

template strings

template strings - simple

```
x = "World"
y = "Hello"

console.log(`${y}, ${x}`)
// prints "Hello, World"

console.log(
`multi
line
strings`
)
// prints:
// multi
// line
// strings
```

template strings - tagged use case

```
"use strict"


// old school push-lines-to-array, join-when-done

const lines = []

lines.push("Hello")
lines.push("line number: " + (1+1))
lines.push("Later")

console.log(lines.join('\n'))

// prints:
//   Hello
//   line number: 2
//   Later
```

template strings - tagged use case

```
"use strict"
const LinePoster = require("./Line-Poster")

// new school push-lines-to-array, join-when-done

const p = LinePoster()

p`Hello`
p`line number: ${1+1}`
p`Later`

console.log(p())

// prints:
//   Hello
//   line number: 2
//   Later
```

template strings - tagged use case

```
"use strict"

const _ = require("underscore")

const interpolate = require("./interpolate")

module.exports = function LinePoster(lines) {
  lines = lines || []

  return function p(strings /*, value, value */) {
    if (!strings) return lines.join('\n')

    const values = _.toArray(arguments).slice(1)

    lines.push( interpolate(strings, values))
  }
}
```

template strings - tagged use case

```
p`Hello`
```

becomes

```
p( ["Hello"] , [ ] )
```

```
p`line number: ${1+1}`
```

becomes

```
p(["line number: ",""],[2])
```

template strings - tagged use case

```
const _ = require("underscore")

//-----------------------------------------------------------------
// f([a1,a2,..], [b1,b2,..]) -> "" + a1 + b1 + a2 + b2 ...
//-----------------------------------------------------------------

module.exports = function interpolate(strings, values) {

  // zip([a1,a2,..], [b1,b2,..])) -> [[a1,b1], [a2,b2], ...]
  strings = _.zip(strings, values)

  // flatten([[a1,b1], [a2,b2], ...]) -> [a1, b1, a2, b2, ...]
  strings = _.flatten(strings)

  return strings.join('')
}
```

template strings - moar info

- http://www.2ality.com/2015/01/es6-strings.html
- http://www.2ality.com/2015/01/template-strings-html.html

classes

classes - simple old school

```
"use strict"

function Animal(name) {
  this.name = name
}

Animal.prototype.speak = function speak() {
  console.log("hi, my name is " + this.name)
}

new Animal("Bob").speak()

// prints: hi, my name is Bob
```

classes - simple new school

```
"use strict"

class Animal {

  constructor(name) {
    this.name = name
  }

  speak() {
    console.log(`hi, my name is ${this.name}`)
  }
}

new Animal("Bob").speak()

// prints: hi, my name is Bob
```

classes - subclasses old school

classes - subclasses new school

```
"use strict"

class Animal {
  species() {                       // <----------------
    throw new Error("subclass responsibiity") // h/t Smalltalk
  }
}

class Frog extends Animal {         // <----------------
  species() {                       // <----------------
    return "frog"
  }
}

console.log(new Frog().species())   // prints: frog
console.log(new Animal().species()) // throws error
```

classes - super calls

```
"use strict"

class Animal {
  constructor(name) {
    this.name = name
  }
  speak() {
    console.log("hi, I'm " + this.name)
  }
}

class Frog extends Animal {
  constructor(name) {
    super(name)      // <--------------
  }
}

new Frog("Bob").speak()   // prints: hi, I'm Bob
```

class performance note

from Trevor Norris, one of the resident performance gurus at NodeSource:

> It's worth mentioning that **`super()`** isn't optimized yet. So should not be used in hot code.

classes - moar info

- http://www.2ality.com/2015/02/es6-classes-final.html

arrow functions

arrow function

```
const foo = ()     => console.log("in foo")
            // ~like function foo() { console.log(...) }
const bar = x      => console.log("in bar with", x)
            // ~like function bar(x) { console.log(...) }
const gru = (x,y) => console.log("in gru with", x, y)
            // ~like function gru(x,y) { console.log(...) }
const pup = () => {
  console.log("in pup with")
  console.log("...nothing")
}

foo(); bar(42); gru(1,99); pup()

// prints:
// in foo
// in bar with 42
// in gru with 1 99
// in pup with
// ...nothing
```

arrow function w/this

```
"use strict"

class FakeTransaction {
  expensiveThing(cb) {
    setTimeout( () => this.expensiveThingDone(cb), 500)
    //                ^^^^ look ma, no bind() or self/that
  }

  expensiveThingDone(cb) {
    cb()
  }
}

new FakeTransaction().expensiveThing(
  () => console.log("expensive thing done!")
)

// prints: "expensive thing done"
```

arrow functions - moar info

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

fin