Ext JS 7 - Modernizing the Ext JS Class System And Tooling

Marc Gusmano
Senior Sales Engineer
marc.gusmano@sencha.com

Agenda

- Ext JS 7 Class system content
  - Modules
  - Classes & Mixins
  - Life-cycle
  - Goals for language and tools
- Sencha’s New Tooling
- The Road Ahead

Ext JS Class System Contents

Namespaces vs Modules

ExtJS: Namespaces

- Namespaces are nested objects
- **Ext.define()** defines class & adds to the namespace
- The Loader maps names to URL's

```
Ext.Loader.setPath({
    Ext: '../ext/src',
    App: './app'
});
```

```
// Ext.grid.Panel
// => '../ext/src/grid/Panel.js'

// App.foo.Bar
// => './app/foo/Bar.js'
```

- Files are loaded at global scope

```
Ext = {};
Ext.global = window;
Ext.global.Ext = Ext;

Ext.define('App.some.Thing', {
    extend: 'App.foo.Bar',
    requires: [ 'Ext.grid.Panel' ],
```

```
Ext = {
    global: {
        Ext: Ext,
        App: {
            foo: {
                Bar: constructor
            },
            some: {
                Thing: constructor
            }
        }
    },
    grid: {
        Panel: constructor
    }
}
```

New: ES2015/ES6 Modules

- Modules are files
- Modules execute in a private scope (not global)
- Modules publish values using **export**
- Modules can have one, unnamed “default” export value
- Modules use **import** to acquire values exported by other modules
- Modules are imported by path or name

./path/file.js

```
var foo = 'bar';  // private, not global

export var abc = 'abc';

export default class Bar {
    // ...
}
```

./other/thing.js

```
import { abc } from '../path/file';

import Bar from '../path/file.js';

import Ext from '@extjs/kernel';

import { define } from '@extjs/kernel';
```

New: Importing Components

- Classes are exported in several ways:
  - By xtype
  - By alias family
  - By class name
- These would be used where strings might be used today
- These exports can be generated by the new build tool

```
import { button } from '@extjs/modern';

import { hbox } from '@extjs/modern/layout';

import { Ext_Button } from '@extjs/modern';
```

Sencha Cmd Understands Namespaces

New Build Tools Understands Modules

Classes

ExtJS: Classes

Ext.define()

- Inheritance
- Requires (directives)
- Properties
- Constructors
- Methods
- Static methods
- callParent

```
Ext.define('App.some.Thing', {
    extend: 'App.foo.Bar',
    requires: [ 'Ext.grid.Panel' ],

    alias: 'thing',
    text: 'Hello',

    constructor: function (config) {
        this.callParent([config]);
    },

    method: function (x) {
        return this.callParent([x]) * 42;
    },

    statics: {
        create: function (cfg) {
            cfg.type = 'thing';
            return this.callParent([cfg]);
        }
    }
});
```

New: ES2015 Classes

import - export - class - @define

- Inheritance
- Directives

- Properties
- Methods
- Static methods
- Super calls (callParent)

```
import { define } from '@extjs/kernel';
import { grid } from '@extjs/modern';
import Bar from 'app/foo';

@define('App.some.Thing', {
    alias: 'thing',
    text: 'Hello'
})
class Thing extends Bar {
    method (x) {
        return super.method(x) * 42;
    }

    static create (cfg) {
        cfg.type = 'thing';
        return super.create(cfg);
    }
}
export default Thing;
```

Decorator

New: Decorators

- Decorators are functions that manipulate classes (or methods)
- Decorators can accept arguments…
  - But must return a function to call as if no arguments were provided
- The **`@decorator`** syntax is just a different way to call these functions
- Decorators are currently a Stage 2 proposal (not yet standardized)

https://github.com/tc39/proposals

```
function define (className, body) {
    return T => {
        T.define(className, body);
    };
}

@define('App.some.Thing', {
   alias: 'thing',
   text: 'Hello'
})
class Thing extends Bar {
}

define('App.some.Thing', {
   alias: 'thing',
   text: 'Hello'
})(Thing);
```

ES2015: Restrictions on constructor

- A **constructor** can only be called with the new operator.
- Calling a **constructor** with `call()` or `apply()` throws a **TypeError**.

```
class Foo {
    constructor (x) {
        this.x = 42;
    }
}

new Foo(42);

let obj = {};
Foo.call(obj, 42);
```

```
▶ Uncaught TypeError: Class constructor Foo cannot be invoked without 'new'(…)
```

Mixins

ExtJS: Mixins

- Mixins is multiple inheritance
- Mixins are classes
- Methods from the mixin prototype are copied to the target class prototype
- Unless there is already a method present

```
Ext.define('App.mixin.Foo', {
    method: function (x) {
        return x * 42;
    }
});

Ext.define('App.some.Thing', {
    extend: 'App.foo.Bar',
    mixins: [ 'App.mixin.Foo' ],
});

Ext.define('App.some.OtherThing', {
    extend: 'App.foo.Bar',
    mixins: [ 'App.mixin.Foo' ],

    method: function (x) {
        return 10 * this.mixins.foo.method.call(this, x);
    }
});
```

ES2015: Mixins

- Mixins are classes (that extend **Ext.Base**)
- Use `@define` to mix in the mixin(s)
- Mixins will be able to act more like a true base
  - The **extend** directive accepts mixins
  - Calls to overlapping methods are handled in the super call

```
import { Ext, define } from '@extjs/kernel';
import Bar from 'app/foo';

class Mixable extends Ext.Base {
    method (x) {
        return x * 42;
    }
}

@define({ mixins: [ Mixable ] })
class Thing extends Bar {
    method (x) {
        return 10 * this.mixins.mixable.method.call(this, x);
    }
}

@define({ extend: [ Mixable ] })
class OtherThing extends Bar {
    method (x) {
        return 10 * super.method(x);
    }
}
```

Common Object Life-cycle

The Life-cycle of Ext.Base

Creation

- Instances are created with operator **`new`**
  - or **`Ext.create()`** or other factory
- The native **constructor** is available to ES2015 classes
  - Not recommended in most cases
- The **`construct`** method is available to all classes
  - The new name for the old **`constructor`**

The Life-cycle of Ext.Base

Configuration

- Use the **`config`** directive to define configuration properties
- Config system calls your apply and update methods
- **Ext.Base `construct`** calls **`initConfig`**

The Life-cycle of Ext.Base

Destruction

- Cleanup is handled by **`destroy`**
- Sets flags like **`destroying`** and **`destroyed`**
- Ignores multiple calls
- Calls **`destruct`** (just once)
- New: Clears instance properties to avoid memory leaks

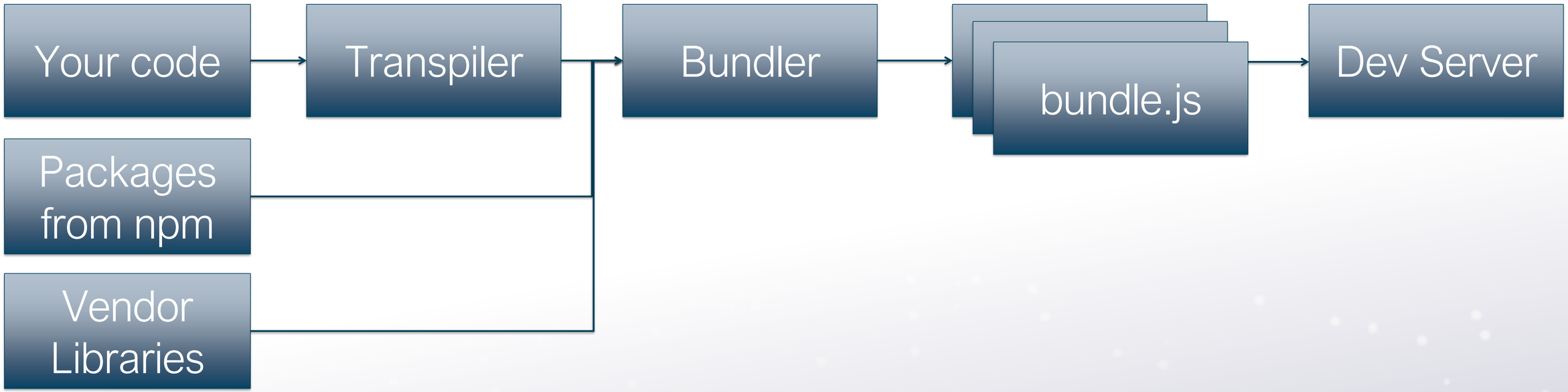
Sencha’s New Tooling

Sencha Cmd is Java based, and makes use of Apache Ant

It's a task runner, which allows you to:

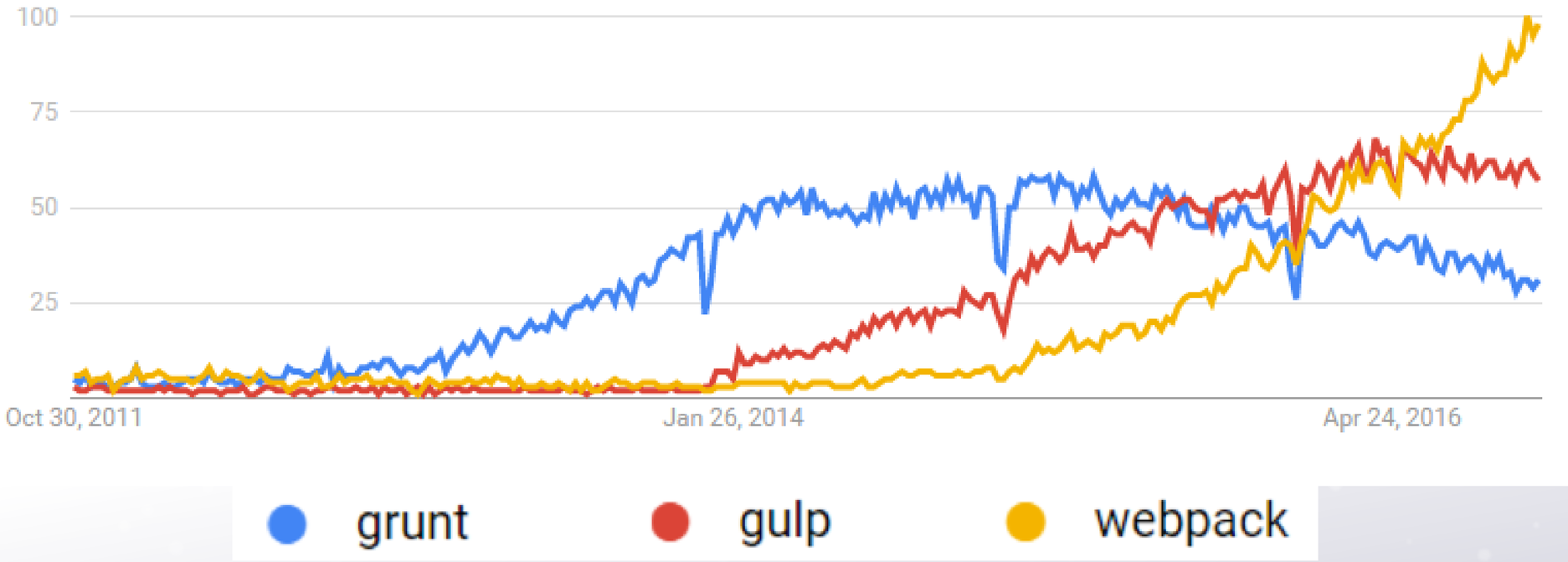
- Generation of code and applications
- Built-in server (Jetty server)
- File watcher
- Load assets (Microloader)
- Compilation of Sass Themes (Fashion)
- Bundling files
- Compress, Optimize and Transpile files (Google Closure Compiler)
- Support for Cordova, Electron, PWA's etc.…
- Upgrade between versions

Building Modern JavaScript Applications



Tools Enable Language Adoption

Build Tool Trends



https://www.google.com/trends/explore?cat=5&q=grunt,gulp,webpack

New build tool, based on NodeJS

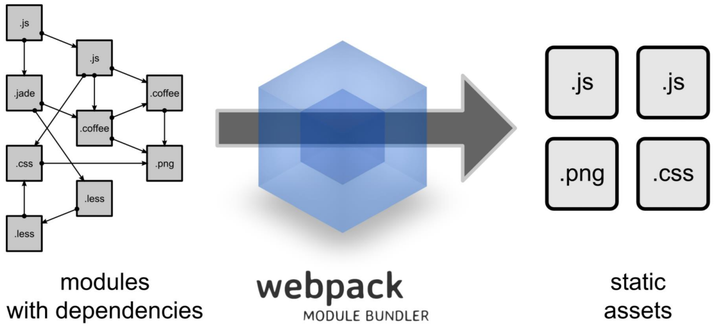
“NodeJS is a JavaScript runtime built on Chrome’s V8 JavaScript engine”

Open toolchain

maven

New Build Tool

Open tooling platform



New build tool – We are looking into these technologies:

- **Package managers for fetching all dependencies**
  - NPM & YARN
- **Bundling your modules to use in a browser**
  - Webpack 2
- **Loading assets**
  - Webpack 2
- **Built-in server and file watcher**
  - Webpack server
- **“Upgradinator”**
  - Custom Sencha tool to upgrade existing code to ES6 decorator syntax.
- **Transpiling ES2015 code to browser ready ES5 code**
  - Babel
- **Node modules organization**
  - Mondorepo
- **Compilation of Sass themes**
  - Fashion
- **Generation of code**
  - Custom template solution vs. Yeoman
- **Support for:**
  - Electron, Cordova, Service Workers

The Road Ahead

The Road Ahead

Sencha Cmd 6.5

- Available now!
- Uses Google Closure Compiler
- Supports much of the ES2015 syntax (http://kangax.github.io/compat-table/es6/)
- Provides polyfills as well
- Transpiling is optional
- New compressor (replaces YUI) can process native ES2015 code

The Road Ahead

Ext JS.Next

- Framework will have changes in **Ext.Base**
- New Build Tool and new Sencha Cmd
- Upgrade tool to convert from projects from Cmd to Build

Questions?

SenchaCon Roadshow