YUI

YUI is a free, open source JavaScript and CSS framework for building richly interactive web applications.

- Initial seed file of only 7kb
- Lazy-loaded modules on demand
- Integrated, inheritance-based application development support including attribute and class management
- Mature, consistent evolution between releases

By learning another library, not only your core JavaScript skills improve, you'll also develop a deeper understanding of how libraries work and benefits they bring.

- 常用方法
- 对象生成
- Apps
- Mobile
- Event

常用方法

namespace

- YUI.namespace -- static
- Y.namespace -- instance
- Y.namespace.apply

```
var YUI = function() {
    // implementations
};
```

Both YUI and Y have:

applyConfig, applyTo, add, use, namespace, log, message, dump, error, guid, stamp, destroy, cached

don’t destroy YUI

- Y.log
- Y.message -- 压缩时会保留
- Y.error -- erroFn
- Y.stamp -- 使用Y.guid, 可能返回null
- Y.cached -- memoization
- Y.later

- Y.Env, Y.config

- Y.Array
- Y.Queue
- Y.Lang
- Y.Object
- Y.UA
- Y.Get
- Y.Features

- color
- escape
- collections
- merge
- number_format
- substitute
- types
- valuechange

对象生成

```
Y.SubClass = Y.extend(
  function() { /* constructor */ },
  /* extend */ SuperClass,
  { /* Instance members */ },
  { /* Static members */ }
);
```

- Good for basic class inheritance
- If you can extend Y.Base, there are more options

```
Y.extend = function(SubClass, SuperClass, ...) {
  var superProto = SuperClass.prototype,
      subProto = Y.Object(superProto);
  SubClass.prototype = subProto;
  // ...
}
// Reusable constructor function for the
Object.create() shim.
```

Factory constructor

```
function Set() {
  var that = (this instanceof Set) ?
                 this :
                 Y.Object(Set.prototype);
  // use that instead of this
  [].push.apply(that._items, arguments);
  return that;
}
```

Use Y.Object

- Avoids copying a lot of properties
- Can be used to make factory constructors
- Can be used to store original values
- Any object can be the prototype
- Avoids class explosion

Augmentation

```
Y.augment = function(to, from, force, whitelist, config);
Y.ModelList = Y.extend(
  function() {
    /* constructor */
    ModelList.superclass.constructor.apply(this, arguments;)
  },
  Y.Base,
  { /* prototype */ },
  { /* static */ }
);
Y.augment(Y.ModelList, Y.ArrayList);
```

- Defers constructor overhead
- Can augment with multiple classes
- Supports class or instance augmentation
- instanceof is false for augmenting classes
- Use it to simulate lazy multiple inheritance
- Y.Base-based classes should use class extensions

Plugins

- host, namespace
- this.beforeHostMethod
- this.afterHostMethod
- this.onHostEvent
- this.afterHostEvent

Class extensions

- Host Method Overlap - AOP
- Y.Do.after(this._doSomethingAfterMainClassMethod, this, ‘doSomething’);
- Y.Do.before(this._doSomethingBeforeMainClassMethod, this, ‘doSomething’);
- Replace exists method

If the extension implements a method which exists on the main class, it will be replaced with the extensions version.

```
Y.Overlay = Y.Base.create(‘overlay’, Y.Widget, [
  Y.WidgetStdMod,
  Y.WidgetPosition,
  Y.WidgetStack,
  Y.WidgetPosition,
  Y.WidgetPositionConstrain
]);
```

Extensions vs Plugins

- Extensions can be used to contribute core behavior
- Extensions modify the class prototype, plugins are always namespaced
- Feature extension constructors are always executed, plugin constructor on plug()
- Feature APIs/attributes on the prototype vs class plugins in namespace is a stylistic choice

App

Y.Base

- Base itself augments Attribute, which in turn augments EventTarget; Base is therefore both an *Attribute provider* and an *Event Target*.
- NAME and ATTRS
- Initialization and Destruction
- ATTRS: initialized, destroyed (readonly)
- events: init, destroy
- Base.create, Base.mix

Y.Widget

- extends Y.Base
- ATTRS: id, boundingBox, contentBox, srcNode, tabIndex, width, height, visible, focused and disabled, rendered, strings
- events: render
- renderUI, bindUI, syncUI, render, renderer
- Progressive Enhancement -- HTML_PARSER
- BOUNDING_TEMPLATE, CONTENT_TEMPLATE (prototype properties)
- getClassName(instance method / static method), CSS_PREFIX
- String Localization

- renderUI is for making changes to the DOM
- bindUI is for hooking up event listeners to the DOM
- syncUI is for changing Widget state based on items in the DOM

Developing Your Own Widgets

- ATTS, initializer, renderUI, bindUI, sycUI, destructor
- Extensions - A Class Level Concept
- widget-[position|position-align|stack|stdmod|parent|child|buttons|autohide|modality]
- Plugins - An Instance Level Concept

- Do a little bit of analysis and design for your modules
- Decide ahead what type of implementation to use
- Consider plugins for advanced functionalities
- Organize your web app as a module repository

- Inheritance-based architecture and class management through the Attribute interface, and Base and Widget classes producing performant, reusable and organized code
- Separation of presentation from model and data using the Widget class to render alternate views (inline or overlay) based on the application’s location within the site

Build Webapps with Widget

```
var Manager = Y.Base.create('manager', Y.Widget, [], {
  renderUI: function() {},
  bindUI: function() {},
  syncUI: function() {}
}, {
  ATTRS: {},
  HTML_PARSER: {}
});
```

Teach it about the DOM and what we want it to do as we go.

App Framework

Mobile

- CSS components -- RWD
- Mojito -- With the YUI App Framework
  Run on either the server or client

Event

- EventTarget
- Base
- Widget
- Y

EventFacade

- methods: halt, preventDefault, stopImmediatePropagation, stopPropagation
- properties: currentTarget, relatedTarget, type, details, target

- hover
- outside
- focus
- key
- mouseenter/mouseleave
- mousewheel
- windowresize
- valuechange
- simulate
- touch
- synthetic