Using Emacs for Javascript development

When I started into web development at work, I took the chance to do the work with Emacs, because different from Java, there are no ultimate tools that offer reliable capabilities that are required for work but are not provided sufficiently well in Emacs.

I build on js2-mode, js2-refactor-mode, company-mode, and dumb-jump.

For a basic setup, see Setting up Emacs for JavaScript (part 1 and part 2).

However we're using a nice advanced Javascript environment that provides fledging features that are not yet in js2-mode, so I need some adaptions.

Inhaltsverzeichnis

private class fields

Private class fields are currently in stage 3. I added an issue and a pull-request to support them in js2-mode, but since they are not yet in the final specification, these won't be pulled.

Therefore I patch them in using `advice-add`.

```
(advice-add #'js2-identifier-start-p
          :after-until
          (lambda (c) (eq c ?#)))
```

There are warnings about undeclared private class fields (since it does not catch their definition), but no errors.

customElements

I also add `customElements` as defined externs:

```
(js2-global-externs '("customElements"))
```

Final setup with use-package

This is the Javascript-related part of my work-setup for Emacs. I extracted it from my init.el, but it still might be missing parts.

```
(use-package js2-mode :ensure t :defer 20
  :mode
  (("\\.js\\'" . js2-mode))
  :custom
  (js2-include-node-externs t)
  (js2-global-externs '("customElements"))
  (js2-highlight-level 3)
  (js2r-prefer-let-over-var t)
  (js2r-prefered-quote-type 2)
  (js-indent-align-list-continuation t)
  (global-auto-highlight-symbol-mode t)
  :config
  (setq js-indent-level 2)
  ;; patch in basic private field support
  (advice-add #'js2-identifier-start-p
            :after-until
            (lambda (c) (eq c ?#))))

(use-package projectile :ensure t :defer 1
  :config
  (projectile-mode)
  :config
  (define-key projectile-mode-map (kbd "C-c p") 'projectile-command-map)
  (bind-key "C-c p s" 'projectile-ripgrep)
  (setq projectile-sort-order 'modification-time))

(use-package which-key :ensure t
  :config
  (which-key-mode))

(use-package company :ensure t :defer 20
  ;; This is not perfect yet. It completes too quickly outside
  ;; programming modes, but while programming it is just right.
  :custom
  (company-idle-delay 0.1)
  (global-company-mode t)
  ;; otherwise this throws lots of errors on completion errors
  (debug-on-error nil)
  :config
  (define-key company-active-map (kbd "TAB")
  ↪  'company-complete-selection)
  (define-key company-active-map (kbd "<tab>")
  ↪  'company-complete-selection)
  (define-key company-active-map [return] 'company-complete-selection)
  (define-key company-active-map (kbd "RET")
  ↪  'company-complete-selection)
  ;; auto-complete compatibility
  (defun my-company-visible-and-explicit-action-p ()
    (and (company-tooltip-visible-p)
         (company-explicit-action-p)))
  (defun company-ac-setup ()
    "Sets up `company-mode' to behave similarly to
    ↪  `auto-complete-mode'."
    (setq company-require-match nil)
    (setq company-auto-complete
    ↪  #'my-company-visible-and-explicit-action-p)
    (setq company-frontends
          '(company-echo-metadata-frontend
            company-pseudo-tooltip-unless-just-one-frontend-with-delay
            company-preview-frontend))
    (define-key company-active-map [tab]
                'company-select-next-if-tooltip-visible-or-complete-sel⌋
                ↪  ection)
    (define-key company-active-map (kbd "TAB")
                'company-select-next-if-tooltip-visible-or-complete-sel⌋
                ↪  ection))

  (company-ac-setup)
  (add-hook 'js2-mode-hook (lambda () (company-mode))))

(use-package company-quickhelp :ensure t :defer 30
  :config
  (company-quickhelp-mode t))

(use-package dumb-jump :ensure t :defer 10
  :custom
  (dumb-jump-rg-search-args '())
  :config
  (defun jump-to-mouse-position (event &optional promote-to-region)
    (interactive "e\np")
    (mouse-set-point event promote-to-region)
    (dumb-jump-go))
  (global-unset-key [C-down-mouse-1])
  (define-key global-map [C-mouse-1] 'jump-to-mouse-position))

;; Highlight TODO, FIXME, ... in any programming mode
(require 'fic-mode)
(add-hook 'prog-mode-hook 'fic-mode)

(use-package flymake-eslint :ensure t :defer 10
  :custom
  ;; add glasses-mode to bolden capitals in CamelCase here. Could also
  ;; be done elsewhere.
  (glasses-face (quote bold))
  (glasses-original-separator "")
  (glasses-separate-capital-groups t)
  (glasses-separate-parentheses-p nil)
  (glasses-separator "")
  :config
  (add-hook 'js-mode-hook
            (lambda ()
              (flymake-eslint-enable)
              (flymake-mode -1)
              (flycheck-mode 1)
              (glasses-mode 1)))
  (add-hook 'js2-mode-hook
            (lambda ()
              (flymake-eslint-enable)
              (flymake-mode -1)
              (flycheck-mode 1)
              (glasses-mode 1)))
  (custom-set-variables
   '(help-at-pt-timer-delay 0.3)
   '(help-at-pt-display-when-idle '(flymake-overlay))))
(use-package flymake-diagnostic-at-point :ensure t :defer 20
  :config
  (flymake-diagnostic-at-point-mode t))

(use-package tern :ensure t :defer 30
  :if (locate-file "tern" exec-path)
  :hook (js2-mode . tern-mode))
(use-package json-mode :ensure t :defer 20
  :custom
  (json-reformat:indent-width 2)
  :mode (("\\.bowerrc$"     . json-mode)
         ("\\.jshintrc$"    . json-mode)
         ("\\.json_schema$" . json-mode)
         ("\\.json\\'" . json-mode))
  :bind (:package json-mode-map
         :map json-mode-map
         ("C-c <tab>" . json-mode-beautify)))

(use-package company-tern :ensure t :defer 30
  :config
  (add-to-list 'company-backends 'company-tern)
  (define-key tern-mode-keymap (kbd "M-.") nil)
  (define-key tern-mode-keymap (kbd "M-,") nil))

(use-package js2-refactor :ensure t :defer 30
  :config
  (add-hook 'js2-mode-hook #'js2-refactor-mode)
  (js2r-add-keybindings-with-prefix "C-c C-m"))
;; context menu for keybindings
(use-package discover :ensure t :defer 30
  :config
  (global-discover-mode 1))

;; ido-preview
(use-package ido-preview ;; no need to ensure: it is part of emacs
  :config
  (add-hook 'ido-setup-hook
        (lambda()
          (define-key ido-completion-map (kbd "C-M-p")
                      (lookup-key ido-completion-map (kbd "C-p")))
          (define-key ido-completion-map (kbd "C-M-n")
                      (lookup-key ido-completion-map (kbd "C-n")))
          (define-key ido-completion-map (kbd "C-p")
                      'ido-preview-backward)
          (define-key ido-completion-map (kbd "C-n")
                      'ido-preview-forward)))
  (defun rebuild-resources (folder)
    "Rebuild resources and generate code in the to-be-selected module."
    (interactive
     (progn
       (list (ido-read-directory-name
              "Select module: " "~/Cadenza/cadenza-master/cadenza"))
       ))
    (async-shell-command (concat "bash -i -c 'cd " folder
                                 "; source ~/.bashrc; cd "
                                 folder
                                 "; Xvfb :3 -screen 0 1024x768x16 & "
```

```
                                              "time DISPLAY=:3 "
                                              "LD_LIBRARY_PATH=$HOME/.guix-profile/l⌋
                                               ↪  ib
                                               ↪  "
                                              "mvn11 generate-sources
                                               ↪  process-resources "
                                              "install validate -DskipTests=true'")))
  :custom
  (ido-buffer-disable-smart-matches nil)
  (ido-cr+-auto-update-blacklist t)
  (ido-cr+-function-whitelist nil)
  (ido-cr+-max-items 30000)
  (ido-cr+-replace-completely nil)
  (ido-enable-dot-prefix t)
  (ido-enable-flex-matching t)
  (ido-everywhere t)
  (ido-max-work-file-list 10)
  (ido-mode (quote both) nil (ido))
  (ido-ubiquitous-mode t)
  (ido-use-filename-at-point (quote guess))
  (ido-use-url-at-point t))


(use-package tabbar :ensure t)

(use-package rainbow-identifiers :ensure t
  :config
  (defun rainbow-identifiers--bolden-faces ()
    (dotimes (i 15) ;; TODO: use number of faces as customized
      (face-remap-add-relative
       (intern
        (format
         "rainbow-identifiers-identifier-%d" (1+ i)))
       :weight 'bold)))
  (add-hook 'rainbow-identifiers-mode-hook
            'rainbow-identifiers--bolden-faces)
  (rainbow-identifiers--bolden-faces))
```

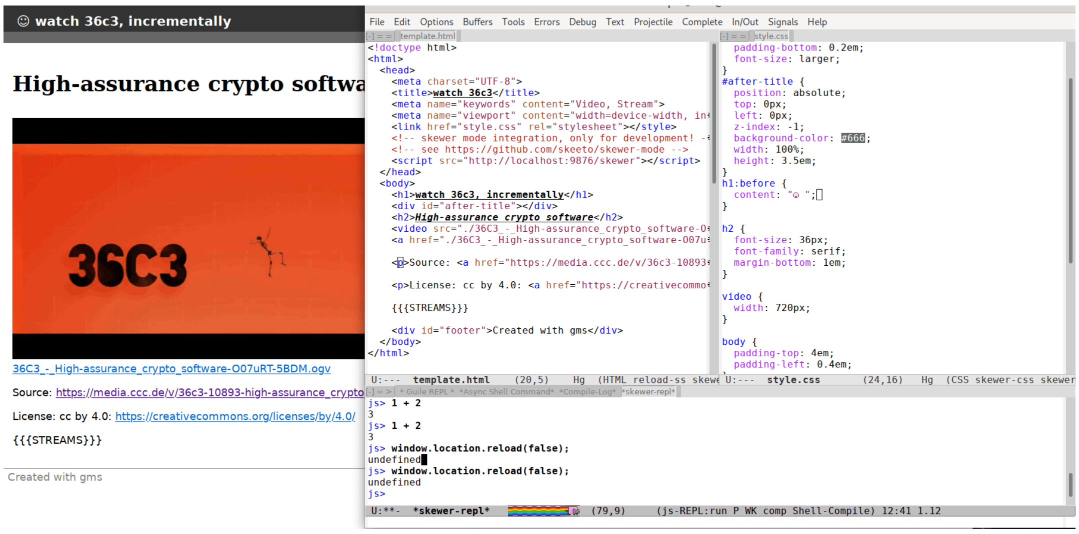
skewer-mode

For quick prototypes skewer-mode is pretty nice.

You can reload specific CSS rules and HTML tags with `C-M-x` and you have a live REPL, so you can tell the browser to reload the whole page from Emacs when needed.

Setup is simple: Just run `M-x run-skewer` to launch the server and then `M-x skewer-repl` to get the REPL for live javascript invocation. The REPL is still a bit brittle — I need to reload the website when it breaks — but updating changed parts of the site in the live browser-tab is beautiful.

And it already works really well for CSS development: Just hit `C-M-x` inside the script tag you just changed to update the style in the browser.


☺ watch 36c3, incrementally
High-assurance crypto softwa
36C3
36C3_-_High-assurance_crypto_software-O07uRT-5BDM.ogv
Source: https://media.ccc.de/v/36c3-10893-high-assurance_crypto
License: cc by 4.0: https://creativecommons.org/licenses/by/4.0/
{{{STREAMS}}}
Created with gms
File Edit Options Buffers Tools Errors Debug Text Projectile Complete In/Out Signals Help
template.html
style.css
* Guile REPL * *Async Shell Command* *Compile-Log* *skewer-repl*

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>watch 36c3</title>
    <meta name="keywords" content="Video, Stream">
    <meta name="viewport" content="width=device-width, in
    <link href="style.css" rel="stylesheet"></style>
    <!-- skewer mode integration, only for development! -
    <!-- see https://github.com/skeeto/skewer-mode -->
    <script src="http://localhost:9876/skewer"></script>
  </head>
  <body>
    <h1>watch 36c3, incrementally</h1>
    <div id="after-title"></div>
    <h2>High-assurance crypto software</h2>
    <video src="./36C3_-_High-assurance_crypto_software-0
    <a href="./36C3_-_High-assurance_crypto_software-007u

    <p>Source: <a href="https://media.ccc.de/v/36c3-10893

    <p>License: cc by 4.0: <a href="https://creativecommo

    {{{STREAMS}}}

    <div id="footer">Created with gms</div>
  </body>
</html>
U:---  template.html     (20,5)     Hg   (HTML reload-ss skewe
```

```
  padding-bottom: 0.2em;
  font-size: larger;
}
#after-title {
  position: absolute;
  top: 0px;
  left: 0px;
  z-index: -1;
  background-color: #666;
  width: 100%;
  height: 3.5em;
}
h1:before {
  content: "☺ ";
}

h2 {
  font-size: 36px;
  font-family: serif;
  margin-bottom: 1em;
}

video {
  width: 720px;
}

body {
  padding-top: 4em;
  padding-left: 0.4em;
U:---  style.css         (24,16)    Hg   (CSS skewer-css skewer
```

```
js> 1 + 2
3
js> 1 + 2
3
js> window.location.reload(false);
undefined
js> window.location.reload(false);
undefined
js>
U:**-  *skewer-repl*          (79,9)      (js-REPL:run P WK comp Shell-Compile) 12:41 1.12
```

(click for full size)

As you see in the screenshot, I used `M-x customize-variable http-port` to change the skewer port to 9876, because port 8080 interfered with too many other services.