
CPI
CSSNITE
after Dark

Sass 徹底解説

SassがもたらすCSSのパラダイムシフト

nodot 小久保浩大郎

自己紹介

- 小久保浩大郎（@kotarok)
- bA → iA → nodot
- 2000年頃からフルCSS実装
- CSSレイアウトの黎明期からやってる古参
- 当時2chカスイケスレに常駐
- 某CSSコミュニティ中〜後期の人(多分)
- **Sassエバンジェリスト(勝手に言ってるだけ)**

TOC

- Sass とは何か?
  - 概要と歴史
  - CSS の問題点
- Sass を使う
  - ツールの使い方
  - 文法と機能
  - 実際にどう書いて使うか
- Sass 周辺の今後の展望

Sassとは何か?

Sassとは何か?

概要と歴史

Sassを一言で表すと?

"Sass makes CSS fun again."

もうちょっと詳しく

- “Sass is a meta-language on top of CSS that’s used to describe the style of a document cleanly and structurally, with more power than flat CSS allows. Sass both provides a simpler, more elegant syntax for CSS and implements various features that are useful for creating manageable stylesheet.
- Sassは素のCSSをパワフルにし、ドキュメントのスタイルをクリーンで構造的に記述するためのメタ言語です。Sassはシンプルでエレガントな記法と、管理しやすいスタイルシートを作るための様々な機能を提供します。

Sassの歴史

- 元々は Hampton Catlin (Wikimedia Foundation) が Haml というテンプレート言語と処理系を開発。Haml は HTML のタグのネスト構造をインデントによって記述するテンプレート言語。
- CSS も 同じように書きたい、ということで Haml と同じくインデントでCSSの構造を記述する言語としてSassを開発。
- Nathan Weizenbaum 通称 @nex3 (当時はワシントン大学の学生) が開発者として合流。

Sassの歴史

- Hampton は Ver 2.0 まで開発に携わる。現在は Nathan がメインの開発者。ちなみに Nathan は大学卒業後 Google に join し、Gmail のオフライン機能開発に携わってる。
- Ver 2.2 から Chris Epstein が参加。彼は Sass ベースのCSS フレームワーク、Compass の作者でもある。

実はこの手のものは幾つかある

- CSS-SSC (PHP)
  - サーバーサイドでの動的コンパイル
- Clever CSS (Python, Django)
  - Pythonチックなインデント記法
- Sass (Ruby, RoR, Haml)
  - Haml のサブセットとして誕生、インデント記法
  - Python のパーサー実装もある
- LESS (Ruby → JS (client side & on node.js))
  - CSS互換の記法、最初はRubyベースだったが今はJSに

色々あるけど…

- 実際あんまり使われてない。
- 何であまり使われてないのか?
  - コマンドライン操作
  - いちいちコンパイル
  - サーバーで動的に動かす必要
  - 独自の構文
- いわゆるウェブデザイナーやウェブ制作者には敷居が高い
- 独自の構文とかマジ勘弁(笑)

Sassの歴史

- Hampton は Ver 2.0 まで開発に携わる。現在は Nathan がメインの開発者。ちなみに Nathan は大学卒業後 Google に join し、Gmail のオフライン機能開発に携わってる。
- Ver 2.2 から Chris Epstein が参加。彼は Sass ベースのCSS フレームワーク、Compass の作者でもある。
- **2010年5月、Sass 3 リリース。CSS3 完全互換文法のSCSS 記法を新たに採用。新機能 @extend によりスタイルの継承が可能に。また --watch オプションの追加によりファイルの変更監視が可能に。**

そもそも何でそんなものが必要なのか?

CSSは言語としてショボすぎる

Sassとは何か?

CSSの問題点

CSSの(言語としての)ショボさ

- そもそも言語と言えるのか?
  JSONみたいなデータフォーマットに近いものではないか?
- 昔からこれについては思うところがあり、いい機会があったのである人物を激しく問い詰めてみた。

2011/January /22(Saturday)

Håkon Wium Lie

W3C で CSS の原案作った人

現在は Opera の CTO

- Q: どうしてCSSには**変数やマクロなどのコードを再利用する仕組みがない**んですか?
- A: それは長い間議論されてきたしたくさんの人によって提案されてきた。しかし結局導入されていないのは**互換性の問題と、「シンプルさを保つ」という命題のため**だ。

- Q: たしかに「シンプルさ」というのはCSSやHTMLを**普及させるために重要なキー**だったと思います。それらが**誰でも書けるくらいシンプルだったからこそWebはここまで広がった**んですものね。
- A: **そのとおりだね。**

- Q: しかし、当初はそれが優先事項でありまた十分なスペックだったのかもしれませんが、**今日のように大規模で複雑なウェブサイトを作るには力不足だとは思いませんか?**
- A: それも**確かにそのとおりだと思う。実はマクロ的なしくみはCSS4に向けてAppleから提案されている。**また現状のCSSの仕様では難しいくらい複雑なことをしたかったら**JavaScriptなんかの補助技術を使う**ってのはどうだい?

- Q: JavaScriptですか?**サーバーサイドの技術ではなく**?
- A: もちろんそれも答えの一つだ。今でもいくつかのそういった**拡張CSSパーサー**は公開されてるよね。そういった技術を使って**拡張仕様を実現すれば将来仕様が決まってブラウザ側で対応したときにも切り分けて出力できる**しね。

- Q: なるほど。ありがとうございます。関連しているかもしれませんが別の質問です。以前slashdotで誌上インタビューがあった際、**「なぜマクロ定義のような再利用のための仕組みがないのか」**という質問に**「HTMLで複数クラスをつければいいよ」**答えてましたよね。

- A: そうだね。ただそれが**CSSに再利用の仕組みがない事を100%カバーするとは思っていない。やはり本質的にそれらは別の問題**だとは思う。

- Q: なるほど。ありがとうございます。最後になりますがもう一つシンプルな質問をさせてください。CSSはバージョンじゃなくてレベルという言葉を使っていますが、これは**文法的な拡張はあまりなく、語彙やブラウザ側の表現の拡張が主だから**なのでしょうか?

- A: **そのとおり**。確かにそれもひとつの理由だ。最初に言ったように後方互換性を保っている事を表したかったんだ。

和解

CSSの問題点まとめ

- CSSは意図的にシンプルであるため、変数やマクロ定義のようなコード再利用の仕組みを持たない。
- できた当初は十分だったシンプルなスペックは、今日のように複雑化したニーズに対しては不十分である。
- スタイル定義を組み合わせるにはHTML側に複数クラス指定というアドホックな解決を取らざるをえない。
- CSSは最初のバージョンから現在のCSS3まで発展してきたが、表現や語彙の拡張が主で、言語としての進化はあまりない。
- **クロスブラウザ用のコードでぐちゃぐちゃ**

一言でいうと

抽象化機構が貧弱すぎて

DRYにできない

DRY = Don’t Repeat Yourself (同じことを繰り返すな)

いわゆるCSSフレームワーク

- Blueprint とか 960.gs とか他にも色々ありますが…
- 使ってますか? 多分使ってないと思いますが。
- 理由:
  - カスタマイズしにくいから
  - 他人のルールに縛られるから
  - HTML側に手を入れる必要があるから
  - そもそもあの手のやつは「CSSを作るフレームワーク」じゃなくて「スタイルの当たったHTMLを作るフレームワーク」なんじゃない?

Sassによる解決

- プログラマブルCSSであるSassに出来ること
  - @mixin によるコードの再利用
  - 変数の導入による値の名前付きハンドリングや演算
  - @extend によるスタイル定義の継承
  - それらを利用したクロスブラウザハックの隠蔽
- 抽象性と再利用性を導入できるため
  - 複雑な設計をエレガントに記述が可能
  - コード資産の蓄積と再利用が可能

Sassを使う

Sassを使う

ツールの使い方

の前に一応インストールの仕方を…

- Sass は Ruby で書かれているので動かすには Ruby が必要
  - Mac や Linux の場合は最初から入ってる
  - Windows の場合は Ruby と Gem をまず入れる
    - やり方はぐぐってね

- 後はおもむろにターミナルを立ち上げ
  - `$ gem install haml`
  - 以上。

コンパイルの仕方と種類

- 基本的な使い方
  - `$ sass {元のファイル} {出力先ファイル}`
  - `$ sass coolstyle.scss:coolstyle.css`
- ディレクトリまるごと変換も可能
  - `$ sass ./scss:./css`
- さらにリアルタイムで変更を監視
  - `$ sass ./scss:./css --watch`

Sassを使う

文法と機能

Sassの言語仕様

- 基本的な記法が2種類ある。
  - SASS: 従来のインデントにより構造を記述する記法
  - SCSS: Ver3から導入されたCSS上位互換の記法

従来の SASS 記法

```
.header
  width: 700px
  background: white
  h1
    font-size: 150%
    a
      text-decoration: none
  p
    margin: 0
```

Ver3から導入されたSCSS記法

```
.header {
  width: 700px;
  background: white;

  h1 {
    font-size: 150%;
    a {
      text-decoration: none;
    }
  }

  p {
    margin: 0;
  }
}
```

コンパイルして出力されたCSS

```
.header {
  width: 700px;
  background: white; }
  .header h1 {
    font-size: 150%; }
    .header h1 a {
      text-decoration: none; }
  .header p {
    margin: 0; }
```

出力オプション -t expanded

```
.header {
  width: 700px
  background: white;
}

.header h1 {
  font-size: 150%;
}

.header h1 a {
  text-decoration: none;
}

.header p {
  margin: 0;
}
```

出力オプション -t expand

```
.header {
  width: 700px
  background: white; }

.header h1 {
  font-size: 150%; }

.header h1 a {
  text-decoration: none; }

.header p {
  margin: 0; }
```

出力オプション -t compact

```
.header { width: 700px; background: white; }
.header h1 { font-size: 150%; }
.header h1 a { text-decoration: none; }
.header p { margin: 0; }
```

出力オプション -t compressed

```
.header{width:700px;background:white}.header h1
{font-size:150%}.header h1 a{text-
decoration:none}.header p{margin:0}
```

コメントの書式

```
// このスタイルのコメントはコンパイル時に出力されない
// 本音はここに書く

/* 従来スタイルのこのこメントはコンパイル時にも残る
   建前はここに書く */
```

どっちを使うべき?

- SCSS 一択。Ver.3が革命的な理由の半分。
- というかこの独自の SASS 記法こそが無駄な学習コストを強いる障壁であり、今まで広く使われてこなかった最大の理由。すでに CSS に慣れてる人がSASS記法を使う理由はない。
- また、SASS 記法は独自パーサーを通るため、CSS パーサーの特性を利用した vendor specific property や IE の filter やハックなどが使えない。
- SASS 記法のことは忘れましょう。以後 SCSS 記法の話しかしません。

SassがCSSに持ち込む機能

- 変数と演算 (variables and arithmetic)
- ルールのネスト (nested Rules)
- ミックスイン (mixins)
- セレクターの継承 (selector inheritance)
- @for, @if などの制御構文

Sassを使う ― 文法と機能

変数と演算

変数の定義、代入

```
//変数名は $ で始める
$varName: ‘value’;

//変数はスコープを持つ
.alert {
  $textColor: #f00;
  color: $textColor;
}

.attention {
  $textColor: #ff0;
  color: $textColor;
}

.note {
  color: $textColor; //error
}
```

変数を使える場所

- 基本的に value の中でしか使えない

```
.someClass {
  color: $textColor1;
  background: $bgColor1;
}
```

それ以外の場所で使いたい場合

- 差し込み記法 (interpolation)
- #{} の中に入れると評価される

```
$className: ‘coolClassName’;
#{$className} {
  // whatever
}
```

- #{} の中は評価されるので演算も出来る

```
$className: ‘coolClassName’;
.#{$className + ‘_’ + 1} { //-> .coolClassName_1
  // whatever
}
```

変数の種類(type)

- Sass の変数には次の4つの型(type)がある
  - Number (数値)
  - String (文字列)
  - Color (色)
  - Boolean (真偽値)

- Array や Hash みたいな複雑な物は無い

数値型 (number)

- Sass の 数値型は一般的なプログラミング言語と少し違う
- 実際には 整数 + 実数 + CSS の Length
  - -1
  - 0.2
  - 3px
  - 4em
  - 5cm
  - 60%

数値演算

- 数値に対する一般的な四則演算が可能 (+, -, *, /, %)
- 単位は自動的に考慮してくれる

```
$unit: 50px;
margin: $unit / 2;  //->25px
padding: 2cm - 5mm; //->1.5cm
```

- 異なる単位系を混ぜるのはNG

```
90% - 10px; //-> error
1em + 1ex;  //-> error
```

数値関数

- 数値を操作するいくつかの関数がある

```
//単位なし数値を%に変換
percentage(.5) //-> 50%

// その他
round()
ceil()
floor()
abs()
```

文字列型 (string)

- 文字列は一般的なプログラミング言語と同様にクォーテーションで囲む

```
“some string”
‘some string’
```

- CSSのルールに則ってワンワードであればそのまま裸で書いても文字列として扱われる

```
helloworld
hello-world
```

文字列演算

- 連結のみ可能 (+)

```
“hello” + ‘world’ //-> helloworld
```

- 違うtypeと連結しようとすると全部 string にキャストされる

```
‘the length is ’ + 10px; //-> the length is 10px
```

- 文字列関数

```
quote()
unquote()
```

色型 (color)

- 色リテラル
  - CSS3 の color value が全部そのまま使える
  - キーワード、3桁カラーコード、6桁カラーコード、rgb、hsl、rgba、hsla

```
red
#f00
#ff0000
rgb(255,0,0)
rgba(255,0,0,1)
hsl(0deg,100%,50%)
hsl(0deg,100%,50%,1)
```

色演算

- 色どうしの加算
- 色に対する四則演算
- その他たくさんの関数
  - lighten(), darken(), saturate(), desaturate() and etc...
  - http://sass-lang.com/docs/yardoc/Sass/Script/Functions.html

真偽値 (boolean)

- リテラル
  - true
  - false
- 演算
  - and
  - or

Sassを使う ― 文法と機能

ルールのネスト

みんな大好きルールのネスト

- 誰でも思いつくし欲しいやつ
- 最初の方にあげた同種のライブラリも全部実装してる

単純なルールのネスト

```
.header {
  width: 700px;
  background: white;

  h1 {
    font-size: 150%;
    a {
      text-decoration: none;
    }
  }
}
```

コンパイル結果

```
header {
  width: 700px;
  background: white;
}
.header h1 {
  font-size: 150%;
}
.header h1 a {
  text-decoration: none;
}
```

&でネストの内側から自分自身のセレクタを参照できます

```
.header {
  width: 700px;
  background: white;

  h1 {
    font-size: 150%;
    a {
      text-decoration: none;
      &:hover {
        text-decoration: underline;
      }
    }
  }
}
```

コンパイル結果

```
.header {
  width: 700px;
  background: white;
}
.header h1 {
  font-size: 150%;
}
.header h1 a {
  text-decoration: none;
}
.header h1 a:hover { //ネストのルートまで遡って展開
  text-decoration: underline;
}
```

Sassを使う ― 文法と機能

ミックスイン

コードの断片を名前をつけて定義して再利用

- 簡単な例を見てみます

```
@mixin reset {
  margin:0;
  padding:0;
  border:0;
  font-size:100%;
  text-decoration:none;
  background:transparent;
}
```

- 定義した mixin を使うときは @include します

```
.someClass {
  @include reset;
}
```

コンパイルするとこうなる

- こうなります

```
.someClass {
  margin:0;
  padding:0;
  border:0;
  font-size:100%;
  text-decoration:none;
  background:transparent;
}
```

- ね、簡単でしょう?

実は mixin は引数が取れる

- 引数付きの mixin を定義します。

```
@mixin v-gradient($c1,$c2){
  background-image: -webkit-gradient(linear, left top, left
bottom, color-stop(0, $c1), color-stop(1, $c2));
  background-image: -webkit-linear-gradient(top, $c1, $c2);
  background-image: -moz-linear-gradient(top, $c1, $c2);
  background-image: linear-gradient(top, $c1, $c2);
  filter: progid:DXImageTransform.Microsoft.Gradient
(GradientType="0",StartColorStr="#{$c1}", EndColorStr="#{$c2}");
  -ms-filter: 'progid:DXImageTransform.Microsoft.Gradient
(GradientType="0",StartColorStr="#{$c1}", EndColorStr="#{$c2}")';
}
```

実は mixin は引数が取れる

- 引数付きで @include します

```
.someClass {
  @include v-gradient(#f00,#00f);
}
```

コンパイルするとこうなる

- こうなります

```
.someClass
```

- ね、簡単でしょう?
- という具合にクロスブラウザ処理を隠蔽できたりします。

Sassを使う ― 文法と機能

セレクターの継承

セレクターの継承とはどういう事か?

- 機能的にはちょっと mixin と似ている
- 公式にはセレクターの継承と呼んでいるが、複雑なセレクタをそのまま継承できるわけではない
- クラスベースの OOP 的なクラスというより実体を継承する JS 的なプロトタイプに似てる
- なので個人的には「プロトタイプ継承」とでも呼びたい
- あとクラスって呼ぶとCSSのクラスと区別がつかなくてややこしいので
- 説明は難しいのでコードを見てみます

まず何か普通に class を作ります

当然これはこれでそのまま普通にHTMLの中で使えます

```
.boldred {
  font-weight: bold;
  color: red;
}
```

このクラス定義を別のところでも使い回す

これをコンパイルすると…

```
.boldred {
  font-weight: bold;
  color: red;
}

.error {
  @extend .boldred;
}
```

シンプルに考えるとこうなるような気がする

```
.boldred {
  font-weight: bold;
  color: red;
}

.error {
  font-weight: bold;
  color: red;
}
```

しかし、実はこうなる

```
.boldred, .error {
  font-weight: bold;
  color: red;
}
```

何がうれしいのか?

- 何がうれしいの?
- mixin と変わらないんじゃないの?
- mixin でいいんじゃないの?
- では典型的な嬉しい例を見てみます。

clearfix

```
.clearfix {
  *zoom: 1;
}

.clearfix:after {
  content: '';
  display: block;
  clear : both;
  height: 0;
}

@media print {
  .clearfix:after {
    height: 1px;
    margin-bottom: -1px;
    visibility: hidden;
  }
}
```

clearfix

- 複数のルールで構成される再利用単位
- @media や :after などややこしい記述
- これを使い回す方法
  - 1. HTML 側で使うたびに class="clearfix" と書く
  - 2.CSS 側に該当セレクタをどんどん追記していく
- どちらもエレガントとは言い難い
- が、@extend を使うと…

```
.someClass {
  @extend .clearfix;
}
```

こうなる

```
.clearfix, .someClass {
  *zoom: 1;
}

.clearfix:after, .someClass:after {
  content: '';
  display: block;
  clear : both;
  height: 0;
}

@media print {
  .clearfix:after, .someClass:after {
    height: 1px;
    margin-bottom: -1px;
    visibility: hidden;
  }
}
```

Sassを使う ― 文法と機能

制御構文

以下のようなものが使えます

- @if {}
- @else {}
- @for {}
- @while {}

- この辺は特に変わったことはないので
  ある意味普通で簡単です。

さっきのグラデーションのやつは実はもっと複雑

```
@mixin v-gradient($c1,$c2,$applyForIe:true){
  background-image: -webkit-gradient(snip);
  background-image: -webkit-linear-gradient(top, $c1, $c2);
  background-image: -moz-linear-gradient(top, $c1, $c2);
  background-image: linear-gradient(top, $c1, $c2);
  @if $applyForIe {
     @if $use_PIE {
        -pie-background: linear-gradient(top, $c1, $c2);
        @include PIE;
     }@else{
        filter: progid:DXImageTransform.Microsoft.Gradient
(GradientType="0",StartColorStr="#{$c1}", EndColorStr="#
{$c2}");
        -ms-filter:
'progid:DXImageTransform.Microsoft.Gradient
(GradientType="0",StartColorStr="#{$c1}", EndColorStr="#
{$c2}")';
     }
  }
}
```

簡単に CSS sprite するやつとか

```
@mixin sprite($img,$w:30,$h:30,$x:0,$y:0,$xo:$x,$yo:$y,$xa:
$xo,$ya:$yo) {
  @include inline-block;
  @include wh($w,$h);
  direction: ltr;
  text-align: left;
  text-indent: -9999px;
  display: block;
  overflow: hidden;
  @if $img != '' {
    background-image: url($img);
  }
  background-repeat: no-repeat;
  @include spp($x,$y,$xo,$yo,$xa,$ya);
}

@mixin no-outline {
  outline: none;
  hidefocus: expression(hideFocus='true');
}
```

簡単に CSS sprite するやつの続き

```
//sprite position
@mixin spp($x:0,$y:0,$xo:$x,$yo:$y,$xa:$xo,$ya:$yo){
  background-position: -#{$x}px -#{$y}px;
  @if $xo != $x or $yo != $y  {
     &:hover {
        background-position: -#{$xo}px -#{$yo}px;
     }
  }
  @if $xa != $xo or $ya != $yo  {
     &:active,
     &:focus {
        background-position: -#{$xa}px -#{$ya}px;
        @include no_outline;
     }
  }
}
```

レイアウト用のグリッドに応じたクラスを自動生成とか

```
@mixin use_layout_span{
   .#{$layout-span-class-prefix}0\.5{
      @include span(0.5,-$gatter-width/2);
   }

   .#{$layout-span-class-prefix}0 {
      width: 0;
   }

   @for $n from 1 to $column-num+1 {
      .#{$layout-span-class-prefix}#{$n} {
         @include span($n);
      }
      .#{$layout-push-class-prefix}#{$n} {
         margin-left:($body-width+$gatter-width)/$column-num*$n;
      }
      .#{$layout-pull-class-prefix}#{$n} {
         margin-right:($body-width+$gatter-width)/$column-num*$n;
      }
   }
}
```

作るものは複雑になるけど

- 一回作ってしまえば使い回すのはとても簡単で安全
- 修正するのもとても簡単で安全 (一箇所直すだけでいい)
- 誰かが作ったものを使い回すのもとても簡単で安全
- 抽象化して再利用出来るというのはそういう事

Sassを使う ― 文法と機能

Tips & FAQs

Tips

- ついつい不必要にネストしてしまうのでしすぎないように注意
  - そもそもセレクタはHTMLの構造から切り離したほうがいい
  - セレクタは冗長すぎてもシンプルすぎてもいけない。最適でユニークに。
  - クエリーやif分の条件のようなもの。その時動くからと言って無駄に詳しかったり簡潔すぎたりは書かないよね?

Tips

- IE6は詳細度の算出やセレクタ周りの挙動がおかしい。Sassを使うとネストしたセレクタになりやすいのでこれにはまりやすい。またセレクタをネストさせるとコード量がかなり増える。Sassを使っても不必要にネストした記述にならないようにしよう。extendしたときに高度なセレクタが混ざるとIE6がそのルール全体を無視するので注意。
- ほかのSassを読み込むのに絶対パスが使えない。スタイルシートをいろんな場所に分散させる場合は注意。
- 出現位置に制限のある記述に注意 @import とか @charset とか。同様に宣言ブロックの中で@mediaも使えない
- mixinは使う前に定義されてないとダメ。extendはパース対象にそのクラスが入ってれば大丈夫

FAQ

- 詳細度とか出てくる順番によっておかしくなったりしないの?
  - @extend するとプロパティの適用順が定義元クラスの出現順によって決まるので注意が必要。
- 出力結果を考えながら書くなんてめんどくさいだけじゃない?
  - 使ってみればわかりますが、デバッグとかややこしいmixin を書く時以外に出力コードを気にすることは全くなくなります。

Sass 周辺の今後の展望

Tab Atkins のスライド

- Tab Atkins
  - Google の Chrome チームの人 / W3C CSS Working Group のメンバー
- 1月12日に CSS に関するプレゼンをしてスライドを発表
  - 今考えられている CSSOM をこき下ろす
  - 新しい Variables, Nesting, Mixins, Modules のアイデアを提案
  - 「今年中に何らかの形で Chrome に実装したい」
  - 追加エントリー来ました。
    - http://www.xanthir.com/blog/b49w0

Nathan のリアクション

「Sass に基づいた Chrome における CSS の改善について @tabatkins と話してるなう。」

Nathan のリアクション

「Sass に基づいた Chrome における CSS の改善について @tabatkins と話してるなう。」

ktkr!!!!!

ということで

- Sass のアイデアを取り入れた形での CSS の仕様策定があり得るかも?
- そして年内に Chrome に入るかも?
- なので今から Sass に慣れておくといいことあるかも。
- しかしどっちみち他のブラウザのサポートが追いつかなかったり古いバージョンとの互換性を取る必要はある。
- しかしそれでもローカルで Sass を使っておけば出しわけたりすることも可能。
- Sass 使いましょう!!!11

One More Thing...

オレオレSassフレームワーク作ってます

FLECSS

- Latest front-end technic included.
- Robust and easy layout framework.
- Many useful mixins and helper JS library.
- Cross browser caring. Even CSS3 PIE integrated.
- Open Source, definitely.
- Will be available soon at http://flecss.com/