2017
春号

百萬石
Hyakumangoku Vol.48

Published by MINA

部長挨拶

hogas

hogas@mma.club.uec.ac.jp

「百萬石」をお手に取っていただき、ありがとうございます。弊誌 (以下、部誌) は、電気通信大学の公認サークル MMA が新歓期 (春号) と弊学が開催する調布祭時期 (秋号) の年 2 回発行・頒布しているものです。MMA とは、”Microcomputer Making Association”の頭文字に由来しており、1975 年に創立されたサークルです。現在では計算機に関連することを中心に、ネットワークやセキュリティなどの分野でも活動しています。部誌には、MMA の部員が各々に行なっている活動を記事にしたものが掲載されています。内容は多岐に渡り、高度なプログラミングの話から旅行記など、MMA の活動の自由さが感じられるものとなっております。

新入生の皆さん、ご入学おめでとうございます。この大学に入学した、というご縁を大事にしていただきたいと思います。加えて、新入生でも新入生でなくても、この部誌を手に取っている、という MMA とのご縁も感じてみてください。MMA では沢山の活動をしています[*1]から、中にはひとつくらい好きなことが見つかるかもしれません。全然わからないという人も、すごく詳しいよという人も、大歓迎です。

最近では、毎日の活動の他に、CTF[*2]や競技プログラミングといったイベントへ参加していたり、年に数回ほどの頻度で Dentoo.LT というライトニングトーク[*3]大会を開催していたりします。この 3 月に参加した ICT トラブルシューティングコンテストでは、最優秀賞をいただくことができました。

「毎日の活動」が何かって……？詳しいことは、お話を聞きにいらしてください。お待ちしております。

2017 年 4 月吉日

[*1] たとえ入部をしなくても、ちょこっとは触れてみてくださいね

[*2] Capture The Flag; コンピュータセキュリティのコンテスト

[*3] 短いプレゼンテーション。Dentoo.LT では 10 分

目次

第 1 章

不可能から生まれる図形

hogas

1.1　3 大作図問題

古代ギリシャの数学に，**3 大作図問題**というものがあります:

角の 3 等分問題　与えられた任意の**角**の，**3 等分**を作図する
倍積問題　与えられた任意の大きさの**立方体**の，**2 倍の体積**の立方体を作図する
円積問題　与えられた任意の大きさの**円**と，**同じ面積**の正方形を作図する

というものです．ここで「作図する」というのは，**直定規** (目盛りのない; 長さを測ることができず，直線を引くためだけの定規) と，**コンパス** (方位磁針ではない方) のみを使って作図をすることを言います．

この 3 問題は長らく数学者を考え込ませ続けましたが，19 世紀になって，ようやっと「不可能である」と証明され (てしまい) ました．代数的に数式で表すと，直定規は 1 次元 (直線)，コンパスは 2 次元 (円) となり，いずれの問題もそれらのみによっては解くことができなかった，ということです．[*1]

ここでは不可能であることについて何か言うわけではなく，長らく考え込み続けた数学者の方に少し焦点を当てるお話をします．

1.2　ニコメデスのコンコイド

その中に，**ニコメデス**という数学者が居たとされています[*2]．ニコメデスは，**コンコイド (conchoid)** という曲線を発見しました．コンコイドは以下のように作られます[*3]:

1. 点 P と，それから適当に離れた直線 L をおく
2. P から，L に交わる半直線を放射状に引く
3. 各半直線上で，L との交点から，P とは反対側の一定距離 a の点が描く軌跡がコンコイドとなる

[*1] できそうですか？是非考え込んでみてください
[*2] 大分マイナーらしく，インターネット上にはあまり文献がありませんが……
[*3] ちなみに、負側のコンコイドも伴ってできます; 表す数式は Wikipedia などにありますから、グラフ作成ソフトなどで試してみてくださいね

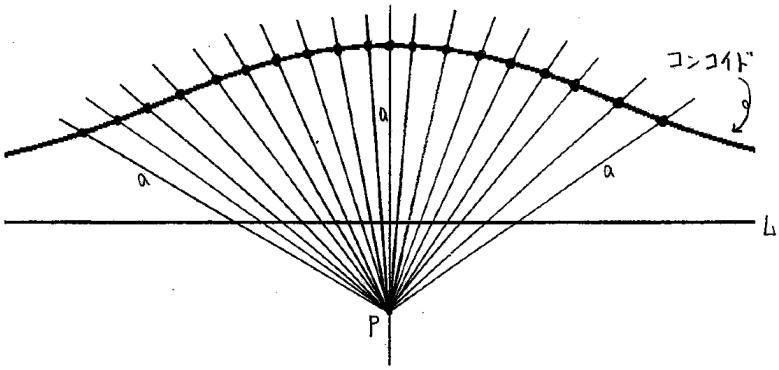

図 1.1　ニコメデスのコンコイドの作成

この曲線が，先に述べた 3 問題のうち「角の 3 等分問題」と「倍積問題」を解決する手段のひとつとなりました．ここでは，角の 3 等分問題について解いてみます．

1.3　角の 3 等分問題を解く

1.3.1　準備

はじめに，準備をします．

1. 点 P から直線 L に下りる垂線の，L との交点を Q とする
2. 与えられた角が $\angle QPR$ となるように，L 上に点 R をおく
3. このときの線分 PR の長さを h とおき，一定距離 $a = 2h$ としてコンコイドを作る
4. R からコンコイドへ，L に垂直な半直線を引き，交点を S とおく
5. 線分 SP を引き，それと L との交点を T とおく
6. 線分 ST の中点を M とおき，線分 MR を引く

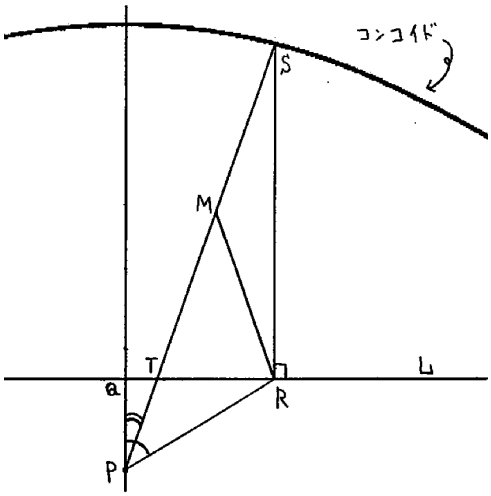

図 1.2　角の 3 等分問題を解く準備

ここまでできたら，実は既に $\angle QPT$ が，与えられた**角 $\angle QPR$ の 3 等分**になっています (！)．これを証明してみましょう (基本的な幾何学の知識だけで証明できます ♡)．

1.3.2　証明

以下のように証明することができます．

(証明)　まず，点 M は直角三角形 SRT の斜辺の中点であるので，直角三角形の 3 頂点すべてから等距離[*4]であり，ゆえに $SM = TM = RM = h$　($\because$ コンコイドの定義から $ST = 2h$)．

$SM = RM$ より三角形 SMR は二等辺三角形で，2 底角は等しく，$\angle MSR = \angle MRS = k^\circ$ とおくと，その外角は $\angle RMT = 2k^\circ$．

また $RM = RP$ より三角形 RMP は二等辺三角形で，2 底角は等しく，($\angle RMP$ は $\angle RMT$ のことなので，) $\angle RMP = \angle RMT = \angle RPM = 2k^\circ$．

ここで，線分 QP と線分 SR はいずれも直線 L に直交しているので $QP//SR$ であり，錯角の関係から $\angle QPT = \angle TSR = k^\circ$．

以上より，$\angle QPT = k^\circ$，$\angle RPT = 2k^\circ$ なので，与えられた角は $\angle QPR = 3k^\circ$．

よって $\angle QPT = \frac{1}{3}(\angle QPR)$ となるので，$\angle QPT$ は $\angle QPR$ の 3 等分である．(証明終わり)

如何でしょうか？見た目にも分かり易く，難しい性質なども無く，かの 3 大作図問題の 1 つを解くことができました．

[*4] 直角三角形をもうひとつ足して長方形にすれば、対角線の性質からわかりますね

1.3.3　結び

解くのが簡単だったとはいえ，実際にやってみた読者の方は，コンコイドを描くのがとっても大変だということにお気付きかもしれません．作図によって厳密なコンコイドを作ることができない，という点で，コンコイドを使えば解ける角の 3 等分問題も，やはり作図では不可能ということになります．

直定規とコンパスだけでは不可能と示されてこそいるものも，別の方法なら解くことができたりします．解くことができないと証明されてしまう前に数学者が生み出したものは，そう証明された後でもその意味を無くしてしまうことはなく，(程度はともかく少なくとも) 有用なものとしてちゃんと残っています．先人の知恵をありがたく使っていきましょう (できることなら，先人となっていきたいですね).

参考文献

[1] 須賀（惣田）正明,「数学史」,(2017 年 4 月 2 日現在)，西須賀（惣田）正明 http://www2m.biglobe.ne.jp/~m-souda/mysouda/math/smith/chapter4/math4.html#04

[2] オニ・パパス (訳: 安原和見),「数学の楽しみ 身のまわりの数学を見つけよう」，2007 年，筑摩書房，pp128-130, pp168-171.

第 2 章

Vue.js でタイマーを作る

mizdra

概要
フロントエンド向け View ライブラリである Vue.js を用いてでカウントダウンタイマーを作成する.

2.1 はじめに

フロントエンドの View を担うライブラリ/フレームワークは Backbone.js, React, Angular など様々なものがありますが, ちょっとしたものを書くためだけにこれらを使うと成果物の大きさの割にコード量 (ライブラリを含む) が膨大になったり, 冗長なコードになってしまうことがあります. Vue.js はこれらの問題を解決し, 少ないコストで楽しく View の開発をすることができます. Vue.js には次のような特徴があります.

- コンポーネント指向
- シンプルな設計
- VirtualDOM のサポート
- リアクティブ

ここでは Vue.js の基本や特徴に触れ, 簡単なカウントダウンタイマーの作成を目指します. 最終的なコードは `https://gitlab.mma.club.uec.ac.jp/mizdra/vue-timer` で公開しているので参考にしてください.

2.2 セットアップ

2.2.1 Vue.js の導入

Vue.js の導入方法は以下の 3 つの方法がありますが, 今回は開発用サーバ, Hot Module Replacement(HMR) などの機能がすぐに利用できる CLI で行います.

- `<script>` 埋め込み

- npm
- CLI

Vue.js コミュニティではオフィシャルな CLI を提供しており, いくつかの質問に対話的に答えるだけでモダンなフロントエンド開発環境を構築することができます.

```
# オフィシャルCLIツールをインストール
$ npm install -g vue-cli

# ビルドツールにWebpackを用いたテンプレートからプロジェクトを作成
$ vue init webpack vue-timer
? Project name vue-timer
? Project description A timer with Vue.js
? Author mizdra <pp.mizdra@gmail.com>
? Vue build standalone
? Install vue-router? No
? Use ESLint to lint your code? Yes
? Pick an ESLint preset AirBNB
? Setup unit tests with Karma + Mocha? No
? Setup e2e tests with Nightwatch? No

$ cd vue-timer
$ npm install
$ npm run dev
```

ブラウザが自動で立ち上がり, Web ページが表示されたはずです. これで開発サーバが立ち上がりました. 今回はビルドツールを Webpack としましたが, 他にも Browserify を使うことができます. 利用可能なテンプレートの一覧は `vue list` で確認できます.

2.2.2　プロジェクトの構成

ソースコード 2.1　プロジェクトの構成

```
.babelrc
.editorconfig
.gitignore
.postcssrc.js
README.md
index.html
package.json
build/ ... 開発サーバやWebpackの設定
config/ ... その他設定(環境変数等)
node_modules/
src/
  App.vue
  main.js ... エントリポイント
  assets/ ... アセット(画像等)
  components/ ... コンポーネント
    Hello.vue
static/
```

プロジェクトルートにはツールの設定ファイルや README などが配置されています. デフォルトで扱いやすい設定になっているので必要になったら弄る程度で良いでしょう. 重要なのは`/src` ディレクトリでここにコンポーネントやリソースを配置していきます.

2.3 Hello World!

チュートリアルがてらに「Hello World!」と表示するコンポーネントを書いてみましょう．まず `Hello.vue`, `App.vue` を次のように編集します．

ソースコード 2.2 src/components/Hello.vue

```
<template>
  <div>
    <div>{{ msg }} World!</div>
    <button v-on:click="msg = msg.split('').reverse().join('')">Reverse</
        button>
  </div>
</template>

<script>
export default {
  name: 'hello',
  data() {
    return {
      msg: 'Hello',
    };
  },
};
</script>
```

ソースコード 2.3 src/App.vue

```
<template>
  <div class="app">
    <hello></hello>
  </div>
</template>

<script>
import Hello from './components/Hello';

export default {
  name: 'app',
  components: {
    Hello,
  },
};
</script>

<style scoped>
.app { text-align: center; }
</style>
```

変更を保存すると自動でブラウザがリロードされます．この時 HMR 機能により変更部分のみがリロードされるため, (今回はありませんが) `<input>`要素に入力したデータなどを残したまま変更を適用することができます．

リロードされたページを見ると「Hello World!」という文字と, 押すと文字が「olleH World!」に変わるボタンが表示されているはずです．まずここで押さえるべき点は「単一ファイルコンポーネント」,「テ

ンプレート」,「コンポーネントがどのように挿入されるか」の 3 つです. 1 つずつ見ていきましょう.

2.3.1　単一ファイルコンポーネント

コンポーネントとは GUI を構成する部品であり, 再利用可能なように設計されます. Vue.js のコンポーネントはテンプレート, ロジック, スタイルの 3 つから構成されます.

ソースコード 2.4　src/App.vue

```
<!-- テンプレート -->
<template>
  <div class="app">
    <hello></hello>
  </div>
</template>

<!-- ロジック -->
<script>
import Hello from './components/Hello';

export default {
  name: 'app',
  components: {
    Hello,
  },
};
</script>

<!-- スタイル -->
<style scoped>
.app { text-align: center; }
</style>
```

あなたが React や Backbone.js などの他の View ライブラリを使ったことがあれば同一ファイル内にスタイルを含んでいることを不思議に思うでしょう. なぜならテンプレート, ロジック, スタイルの 3 つの要素は言語とそれが表現するモノは全く異なるためです. 大抵は `.html`, `.js`, `.css` のようにファイルタイプごとにファイル分割をします. しかし Vue.js では「3 つの要素は互いに関係しており, 1 つのファイルで管理したほうが保守性に優れている」という考えを元に 1 つのファイル (.vue) に全てを記述できるようになっています (もちろん, あなたがファイルタイプごとにファイル分割したければそうすることも可能です. ですがここではその方法は省きます.). これを**単一ファイルコンポーネント**と呼びます.

> 注意すべき重要な点の 1 つは、関心事項の分離がファイルタイプの分離と等しくないことです。現代の UI 開発では、コードベースを互いに織り交ぜる 3 つの巨大なレイヤーに分割するのではなく、それらを疎結合なコンポーネントに分割して構成する方がはるかに理にかなっています。コンポーネントの内部では、そのテンプレート、ロジック、スタイルが本質的に結合されており、実際にそれらを配置することで、コンポーネントがより一貫性と保守性に優れています。
>
> `https://jp.vuejs.org/v2/guide/single-file-components.html#関心の分離について`

単一のファイルでコンポーネントに関する多くの情報を記述することで統合的にコードを捉えられる, 開発時の頭の切り替え (コンテキストスイッチ) が減る等のメリットがあります.

単一ファイルコンポーネントは Webpack Loader の vue-loader で実現されています. vue-loader は他にもテンプレートやスタイルを Pug(旧 Jade) や Sass で書けるように単一コンポーネントの各要素に対する Webpack Loader との連携, scoped CSS のシミュレート機能 (擬似的な scoped CSS), デフォルトでの ES2015 対応などモダンな要求に答えられうる機能を有しています. [*1]

例えば, テンプレートを Pug で書くには Pug をインストールしてルートの template 要素の属性に `lang="pug"` を追加するだけで実現できます.

ソースコード 2.5 Pug をインストールする

```
$ npm install --save-dev pug
```

ソースコード 2.6 Hello.vue

```
<template lang="pug">
  div
    div {{ msg }} World!
    button(v-on:click="msg = msg.split('').reverse().join('')") Reverse
</template>
```

2.3.2 テンプレート

Vue.js のテンプレート構文は HTML として正しく, 仕様に準拠したブラウザやパーサーがパース可能な有効な HTML です. ですから, HTML に馴染みのある皆さんにとってこのテンプレート構文は容易に理解できるでしょう.

ソースコード 2.7 App.vue のテンプレート

```
<template>
  <div>{{ msg }} World!</div>
  <button v-on:click="msg = msg.split('').reverse().join('')">Reverse</
      button>
</template>
```

しかしながらいくつかの制約があります. テンプレートは template 要素で囲む必要があり, その template 要素はただ 1 つの子要素を持つようにしなければなりません. 例えば, 次のようなテンプレートは無効です.

ソースコード 2.8 無効なテンプレートの例

```
<!-- ルートの template要素が子要素を1つも持たない -->
<template>
  {{ msg }} World!
```

[*1] Vue.js はモダンなツールや考えを元に他の View ライブラリよりもシンプルになるように設計されています. これは Vue.js がレガシーなブラウザを除いた IE9 以上のみをサポートしていることからもわかります. Vue.js がどの点において優れているのか, 劣っているのかについては公式ガイド (`https://jp.vuejs.org/v2/guide/comparison.html`) に非常に丁寧にまとめられています. 興味があれば読んでみると良いでしょう.

```
</template>

<!-- ルートの template要素が子要素を2つ以上持っている -->
<template>
  <div>item 1</div>
  <div>item 2</div>
</template>
```

{{ ... }}は **Mustache** 構文 (二重中括弧) といい, **データバインディング**を表します. Mustache は対応するオブジェクトの msg プロパティの値 (サンプルコードでは'Hello') に置き換えられます. データバインディングということからもわかるように, プロパティの値が更新されれば, Mustache もその値を元に自動的に再計算されます. つまり, プロパティの値を更新する際に対応する Mustache を更新するための特別なコードは必要ありません. 「メッセージの送受信を隠蔽し値同士の関係 (data-flow) を宣言的 (関数型的) に記述する」[2], これが**リアクティブ**です.

また, Mustache は JavaScript 式を完全にサポートしています. ただし文や制御構文は単一の式ではないため, これらを含めることはできません. この式を**テンプレート式**と呼びます.

ソースコード 2.9　テンプレート式の例

```
<!-- テンプレート式では JavaScript式が使える -->
{{ msg.toUpperCase() }}

<!-- if式の代わりに三項演算子を使う -->
{{ items.length === 0 ? ('アイテムが登録されていません') : (items.length + '
    件が登録済み') }}
```

Mustache は属性値に利用することはできません. 属性値でデータバインディングするには v-bind ディレクティブを使います. **ディレクティブ**とは, v-から始まる特別な属性のことです. **ディレクティブ属性値**は Mustache と同様に JavaScript 式が利用できます. 属性値に真偽値を使用すると, 真偽値が偽である時にその属性は削除されます.

ソースコード 2.10　v-bind ディレクティブの例

```
<button v-bind:id="buttonId" v-bind:disable="!valid">{{ text }}</button>
```

Mustache は XSS 対策のため自動でサニタイズを行います. つまりデータは HTML ではなくプレーンテキストとして扱われます. データを HTML として扱いたい場合は v-html ディレクティブを使用します.

ソースコード 2.11　v-html ディレクティブの例

```
<template>
  <div v-html="rawHtml"></div>
</template>

<script>
export default {
  name: 'hello',
  data() {
    return {
      rawHtml: '<h1>Hello</h1>',
    };
```

```
  },
};
</script>
```

サンプルコードでは v-on ディレクティブを使って button 要素の click イベントを購読し, 発火したら msg プロパティの文字列を逆順にするようになっています. データバインディングにより msg プロパティの更新は対応する Mustache に伝播され, View が瞬時に更新されます.

v-bind ディレクティブは:, v-on ディレクティブは@を使った省略記法があります. これらの記号を用いた属性名は特殊に見えるかもしれませんが, Vue.js のサポートしている全てのブラウザで正しくパースすることができます.

ソースコード 2.12 v-bind v-on ディレクティブの省略記法

```
<!-- 以下のコードは同義 -->
<button v-bind:id="buttonId" v-on:click="alert('click!')">ボタン</button>
<button :id="buttonId" @click="alert('click!')">ボタン</button>
```

2.3.3 コンポーネントがどのように挿入されるか

作成したコンポーネントをネイティブの DOM に挿入するには, 挿入したいコンポーネントのオプションオブジェクトに el オプションを追加し, インスタンスにどの要素にマウントさせるかを教えてあげる必要があります. オプションの値は id 指定なら'#rootId', class 指定なら'.classId', といった形です.

ソースコード 2.13 main.js

```
import Vue from 'vue';
import App from './App';

Vue.config.productionTip = false;

new Vue({
  el: '#app',
  template: '<App/>', // オプションでtemplateを渡す
  components: { App },
});
```

ソースコード 2.14 index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>vue-timer</title>
  </head>
  <body>
    <!-- エントリポイント(main.js)は
         Webpackによってバンドルされ自動でここに挿入されます -->
    <div id="app"></div>
  </body>
</html>
```

import 文で読み込んだ**オプションオブジェクト**を components オプションに渡すと読み込んだ側の (親) コンポーネントのテンプレート内で読み込まれた (子) コンポーネントが利用できるようになります. これをコンポーネントの**ローカル登録**といいます. 読みこまれた (子コンポーネント) は (親) コンポーネントのテンプレート内で name プロパティの値, もしくは components プロパティで登録した key を使ってアクセスできます.

ソースコード 2.15　様々なアクセス方法

```
// テンプレート内で ‘<AppComp>‘ で
    Appコンポーネントにアクセスできるようにする
new Vue({
  template: ’<AppComp/>’,
  components: { AppComp: App },
});

// 同名でもOK
new Vue({
  template: ’<App/>’,
  components: { App: App },
});

// ES2015のShorthand property namesを利用
new Vue({
  template: ’<App/>’,
  components: { App },
});

// nameプロパティの値を使う
new Vue({
  template: ’<app/>’,
  components: { App }, // 必須
});
```

ローカル登録はその親コンポーネント内に対して子コンポーネントを利用することを許可しますが, これとは反対に components オプションにオプションオブジェクトを渡さずとも全てのコンポーネントに対して利用を許可する**グローバル登録**もあります.

ソースコード 2.16　コンポーネントのグローバル登録

```
// Appコンポーネントをグローバル登録する
Vue.component(’app’, {
  // オプション
});
```

2.4　タイマーを作る

さて, ここまで Vue.js の基本について学んできました. 次は今まで学んだことを生かして簡単なカウントダウンタイマーを作ってみます.

2.4.1　DurationView コンポーネントの作成

はじめに, 残り時間を表示する DurationView コンポーネントを作ります. コンポーネントの要件は次の通りです.

1. プロパティで残り時間 value (単位は ms) を受け取る
2. 「mm:ss.cs」(cs は centi second) 形式で残り時間を表示
3. mm, ss, cs それぞれは整数で左詰め 2 桁

ソースコード 2.17　src/components/DurationView.vue

```
<template>
  <div>
    <span>{{ Math.trunc(value / 1000 / 60) }}分</span>
    <span>{{ Math.trunc(value / 1000) % 60 }}秒</span>
    <span>{{ Math.trunc(value / 10) % 100 }}</span>
  </div>
</template>

<script>
export default {
  name: 'duration-view',
  props: ['value'], // 単位はms
};
</script>
```

とりあえず要件の 1 と 2 を満たしたものを書いてみました. 早速作成したコンポーネントを読み込みましょう. DurationView コンポーネントを読み込む Timer コンポーネントを作成し, さらに Timer コンポーネントを App コンポーネントで読み込みます.

ソースコード 2.18　src/components/Timer.vue

```
<template>
  <div>
    <duration-view :value="duration" />
  </div>
</template>

<script>
import DurationView from './DurationView';

export default {
  name: 'timer',
  components: { DurationView },
  data() {
    return {
      duration: 80456, // 1分20秒45
    };
  },
};
</script>
```

ソースコード 2.19　src/App.vue

```
<template>
  <div class="app">
    <timer />
  </div>
</template>

<script>
import Timer from './components/Timer';

export default {
  name: 'app',
  components: {
    Timer,
  },
};
</script>

<style scoped>
.app { text-align: center; }
</style>
```

ブラウザを確認してみると「1 分 20 秒 45」と表示されているはずです．期待通り動いてますね．DurationView コンポーネントのコードを見ると，テンプレートに単位の変換の処理 (ロジック) が書かれています．このままテンプレートへのロジックの挿入をし続けるとメンテナンスコストの増大に繋がる恐れがあります．算出プロパティを使うとこうしたテンプレートに埋め込まれていたロジックをオプションオブジェクトの中に書くことができます．

ソースコード 2.20　src/components/DurationView.vue

```
<template>
  <div>
    <span>{{ minutes }}分</span>
    <span>{{ seconds }}秒</span>
    <span>{{ cs }}</span>
  </div>
</template>

<script>
export default {
  ...
  // valueプロパティを元に計算される3つのプロパティ(算出プロパティ)を作成する
  computed: { // `this.value` でvalueプロパティにアクセスできる
    minutes() { return Math.trunc(this.value / 1000 / 60); },
    seconds() { return Math.trunc(this.value / 1000) % 60; },
    cs() { return Math.trunc(this.value / 10) % 100; },
  },
};
</script>
```

value プロパティが更新されると自動で minutes, seconds, cs の 3 つのプロパティが更新されます．そう，この 3 つのプロパティは value プロパティに**バインディング**しているのです．

次に要件 3 を満たすために**フィルタ**を使って 0 詰めするフィルタを作成します．フィルタは与えられた JavaScript 式に変換を施し，その結果を mustache 展開と v-bind 式に渡すことができます．

ソースコード 2.21　src/components/DurationView.vue

```
<template>
  <div><!-- フィルタは'|'で区切り, JavaScript式の末尾に追加する -->
    <span>{{ minutes | pad }}分</span>
    <span>{{ seconds | pad }}秒</span>
    <span>{{ cs | pad }}</span>
  </div>
</template>

<script>
export default {
  ...
  filters: { // フィルタはfiltersオプションに書く
    pad(val) { // 左0詰めフィルタ
      return val.toString().padStart(2, '0');
    },
  },
};
</script>
```

これで「01 分 20 秒 45」と正しい表示になりました.

2.4.2　Timer.vue の作成

いよいよタイマー部分を作っていきます. まず Timer コンポーネントにタイマーの開始, 停止ボタンを設置し, それぞれからメソッドを呼び出します.

ソースコード 2.22　src/components/Timer.vue

```
<template>
  <div>
    <duration-view :value="duration" />
    <button @click="start">開始</button>
    <button @click="stop">停止</button>
  </div>
</template>

<script>
...
export default {
  ...
  methods: {
    start() { console.log('start'); },
    stop() { console.log('ended'); },
  },
};
</script>
```

メソッドは算出プロパティと同じようにロジックをテンプレートから排除するために用いられます. ただし, こちらは算出プロパティと違ってリアクティブではないのでその点は注意しましょう.

残りのコードを追加すると次のようになります.

ソースコード 2.23　src/components/Timer.vue

```
<template>
```

```
  <div>
    <duration-view :value="duration" />
    <button @click="start">開始</button>
    <button @click="stop">停止</button>
  </div>
</template>

<script>
import DurationView from './DurationView';

const raf = window.requestAnimationFrame;

export default {
  name: 'timer',
  components: { DurationView },
  data() {
    return {
      initialDuration: 80456, // 開始時間
      duration: 0,            // 残り時間
    };
  },
  computed: {
    // 残り時間が0なら終了フラグを立てる
    ended() { return this.duration === 0; },
  },
  methods: {
    updateDuration(newDuration) {
      this.duration = Math.max(newDuration, 0); // durationは0以上
      if (this.ended) {
        console.log('ended');
      }
    },
    startRAFLoop() {
      const cb = (currentTime, prevTime) => {
        // 残り時間が0になったらrafループを抜ける
        if (this.ended) return;

        // duration = duration - 経過秒数
        this.updateDuration(this.duration - (currentTime - prevTime));

        // 再帰呼び出しでループする
        raf(time => cb(time, currentTime));
      };
      const initialTime = window.performance.now(); // 現在時刻をmsで取得
      raf(time => cb(time, initialTime)); // rafでコールバック関数を呼び出す
    },
    start() {
      if (!this.ended)
        throw new Error('Cannot call start() when timer is started.');
      this.updateDuration(this.initialDuration); // 残り時間を初期化する
      this.startRAFLoop();
      console.log('start');
    },
    stop() {
      if (this.ended)
        throw new Error('Cannot call stop() when timer is ended.');
      this.updateDuration(0); // 残り時間を0にして終了する
    },
  },
```

```
};
</script>
```

ついでに initialDuration を input 要素から取得するように変更してみましょう.

ソースコード 2.24　src/components/Timer.vue

```
<template>
  <div>
    <duration-view :value="duration" />
    <input v-model.number="initialDuration" />
    <button @click="start">開始</button>
    <button @click="stop">停止</button>
  </div>
</template>
...
```

なんとたった 1 行で対応できました! シンプルで良いですね!

v-model ディレクティブは双方向バインディングのためのディレクティブです. これは input イベントを利用した双方向バインディングのシュガーシンタックスです.

ソースコード 2.25　src/components/Timer.vue

```
<!-- 以下のコードは同義($eventはイベントオブジェクト) -->
<input :value="val" @input="val = $event.target.value"  />
<input v-model="val" />
```

v-model ディレクティブの後ろに付いている number は修飾子です. 通常, $event.target.value で取得できるフォームの値は文字列ですが, number 修飾子を使うとフォームの値を数値として扱うように強制できます.

2.5　おわりに

ここでは Vue.js の基本を学び, 実際に簡単なアプリケーションを書くところまで行いました. コードを書いていく中で, シンプルな記法でありつつもモダンで柔軟な表現ができるという Vue.js の特徴を体験できたかと思います. ここでは紹介しませんでしたが, Vue.js は公式の日本語のドキュメントがあったり, Routing や Flux 開発のための**公式プラグイン** (こちらにも公式日本語ドキュメントがあります!) が提供されていたりと公式のサポートが手厚いのも特徴の 1 つです. 特に, そうした豊富な日本語の情報は新しいライブラリを使い始めるあなたをきっと強く後押ししてくれるでしょう. Vue.js に興味を持っていただけたなら是非公式ドキュメントを読み, さらなる機能をご自身で試してみて下さい.

参考文献

[1] はじめに - Vue.js, https://jp.vuejs.org/v2/guide

[2] 2015 年に備えて知っておきたいリアクティブアーキテクチャの潮流 - Qiita, http://qiita.com/hirokidaichi/items/9c1d862099c2e12f5b0f

第 3 章

自宅サーバラックの勧め 電源編

h-otter

概要

この部誌を読んでいるあなたなら一度はほしいと思ったことがあるはずであるサーバラック。以前はラックを買う話 [1] をしたので、今回はラックの電源周りについて話そうと思います。自宅サーバーラックを運用するにあたって電源において重要なことは 2 つです。1 つは安定的な電力を確保することです。これは家の系統がどのように分かれているか把握し、ブレーカーが落ちないような系統を選択することが重要です。また、UPS を導入すると雷などの災害の際にも対応できます。2 つ目は自分の身の潔白を証明することです。ラックのようなものを持つと同居人の目は厳しいものになります。特に電気代の変化はまずすぐに自分に嫌疑がかかります。そのようなときのためにも節電になるような機材の選定と、今自分がどの程度電力を使っているのかを把握する必要があります。そのためには電気代を計測する機材があるほか、PDU というラック用の機材があったりします。自宅の具体的な例とともに話していきたいと思います。

3.1　設計

3.1.1　安定的な電力

安定的な電力を得るためには前提としてブレーカーが落ちないような系統にラックを置く必要があります。しかし、普通の使い方をしていればブレーカーが飛ぶようなことはありません。そのような電気使用量になった場合、先に財布の中身が飛ぶでしょう。電子レンジや、ドライヤーなどと同じ系統でなければ問題はないでしょう。

次にコンセントが抜けにくいことが挙げられます。DC などではコンセントが抜けることなどはないでしょうが、自宅ラックの場合は生活圏です。何が起こるかわかりません。3P の抜け止めコンセントなどに交換すると不慮の事故を未然に防ぐことができるでしょう。その場合には第 2 種電気工事士の資格が必要になります。

また、サーバなどは 3P であることも多いので、3P コンセントを用意できるといいでしょう。部屋にクーラー用のコンセントがあり、使ってない場合にはそれを代用するといいかもしれません。もし、電気工事士の資格を持っている場合には部屋にアースが入ってきているかを確認し、ある場合にはコンセントを付け替えるなどの作業をするのもロマンがあります。

UPS

安定的な電力を供給するには UPS が不可欠でしょう。一般家庭では雷の際などに停電することもあります。急な停電や電圧のノイズは機械の寿命などを縮めることにつながります。UPS はそれらを吸収してくれるのでラックを買ったときには入れたほうがいいでしょう。(2U くらい使ってくれるので、いい感じに埋まります)

また、UPS は利用者が多いのでヤフオクにも量が出回っています。いい状態のものを安く手に入れることができます。自分のものはバッテリの状態も良い状態で 5000 円程度で購入することができました。近年のものはネットワークインターフェースで監視も行うことができるので、コスパはいいと思います。

図 3.1　UPS

3.1.2　電気代

ラックを持っていると確実に電気代がかさみます。非常に高いので電気代は可能な限り抑える必要があるでしょう。となると一番効果があることは消費電力を可視化することです。

PDU

PDU は高機能な電源タップです。近年のものは消費電力、温度、またネットワーク機器等のために 1 コンセントごとに電源を切ることができます。

自分が買ったものは少し古いので消費電力を SNMP などで監視できるものです。それで 7000 円くらいでした。普通の消費電力チェッカーが 3000 円くらいであり、電源タップもかねていることを考えるとお得感があります。ラックにマウントされているので、抜き差しも容易なのが便利です。

zabbix で消費電力を監視しているのですが、それはまた別の記事で。

3.2　最近の話

とある月、休み期間中であったため新しく買った ”Dell R610” を常用を始めました。

図 3.2　Dell R610

しかし…

```
母「なんか今月、いつもより4000円くらい電気代が高いんだけど…あんたなんか知らない？」
@h-otter「シラナイナー」
```

ラックを持っているからって電気代が増えたことについて自分が最初に疑われるのは、まったくもって心外です。実際に R610 によって増えた電気代を計算してみましょう。

身の潔白を証明

電気代の計算は 3 段階の従量課金方式が一般的です。うちの場合にはすでに一番高い段階にあるので、30 円/kWh[2] として計算していきます。

また、PDU とサーバの消費電力モニタを使用して、平均してだいたい 150 Wh であることを確認しました。つまり、24 時間 30 日間動かしていたので以下の式の通り計算すると 3240 円 になります。

$$\text{電気代 [円]} = \text{消費電力 } [kW] * 24[h] * 30[days] * 30[\text{円}/kWh]$$

以上より、その月の電気代の増分と R610 の消費電力が大体一致しました。なぜか自分の身の潔白を証明するはずが、真っ黒であると証明されてしまったようです。ナンデカナー

3.3　まとめ

電源周りは昨今のクラウドや DC 事業によって触る機会がどんどん減ってきているように感じます。これらのことをやるには自宅ラックが一番の近道でしょう。

しかし、前述もした通り電気代などお金や家族の目が冷たくなることも多くなります。損得勘定でいうと少しマイナスくらいですが、自宅サーバラックは無限に浪漫を感じることができるのでやはりいいものです。この記事を読んで、用法用量を守って楽しくサーバラックライフを送ってくれる人が増えてくれることを期待しています。

参考文献

[1] h_otter/AdventCalendar2016 (自宅サーバーラックの勧め ラック購入編) https://wiki.mma.club.uec.ac.jp/h_otter/AdventCalendar2016

[2] http://www.tepco.co.jp/ep/private/plan/chargelist01.html

第 4 章

Don't stop PC now

su_zu

概要

皆さん想像してみてください, 少しでも停止すると多くの人が困ったりするサービスってありますよね？例えば医療系のシステムなどは命と直結していますので絶対に止めることは許されないですしエンジニアであれば GitHub などの git ホスティングサービスが停止すると仕事ができない人がいることは容易に想像できると思います (git は分散管理なんだから一つサービスが落ちただけで何もできなくなるのは git を使っている意味がない云々のつまらない話は置いときましょう). 今回はそんな停止させてはいけないものの中で宇宙空間で稼働するシステムを中心に話したいと思います.

4.1　主な hot deploy 方法

- 新しいプロセスを立ち上げてそれに切り替える方法
  主に LL 言語で用いられることが多いと思います.ruby などの Unicorn などが該当します.
- jvm 上でクラスが動的にロードされる方法
  主に開発環境で用いられることが多いと思います.
- repl にログインする方法（今回話すのはこれです）
  repl サーバを立てておいてそこにログインして動的に書き換えます.repl とは Read-eval-print loop の略で文字通り読んで評価して表示してを繰り返す対話型評価環境を指します.

4.2　Deep Space 1

NASA の New Millennium Program[*1]の一つに Deep Space 1 という宇宙探査機というものがありました. そこでは Common Lisp で書かれた Remote Agent(RA) という自動航法可能な AI システムが搭載されておりました.

[*1] 宇宙探査・人工衛星における新技術の革新を目指す NASA のプロジェクト

4.2.1 Debugging code from 60 million miles away

地上でテストした際には見られなかったバグが宇宙空間で発見されたときエンジニアは repl にログインしてシステムを一切止めずに debug しました.1 億 6000 万キロ離れた場所から debug するのはとても面白い話だと思います.

4.3 Common Lisp swank

Common Lisp で RA のようなことをやってみたいと思います.

まず環境としては Common Lisp の処理系に SBCL を使い package 管理に Quicklisp を使います. 今回は分かりやすい例として clacl という web framework を使って web アプリを動的に書き換えたいと思います. swank サーバには emacs の SLIME を使用してログインします.SLIME は vim や atom にもあるので頑張ってください.

ちなみに Quickdocs.org[*2]などでもこのような開発環境で開発しているようです. `http://blog.8arrow.org/entry/20130320/1363787619`

ソースコード 4.1 app.lisp

```
(ql:quickload :clack)
(ql:quickload :swank)

; 書き換える関数
(defun handler (env)
  '(200 nil ("HOGE")))

; webサーバ
(defparameter *clack-server*
  (clack:clackup (lambda (env) (funcall 'handler env))))

; swankサーバ
(swank:create-server :port 4005)
```

```
$ sbcl --load app.lisp

$ curl localhost:5000
HOGE

$ emacs
slime-connect
localhost
```

[*2] Lisp ライブラリのドキュメント集約サイト

```
4005
CL-USER> (defun handler (env)
           '(200 nil ("Hello world")))

$ curl localhost:5000
Hello world
```

今回は localhost の swank サーバにログインしましたがリモートに接続する場合などは ssh トンネルなどを使うといいと思います.

4.4 Clojure nREPL

Clojure でも Common Lisp と同様なことを行いたいと思います.

プロジェクトツールには Leiningen を使い web framework には compojure を使い repl サーバには nREPL を使います.

```
$ lein new compojure hello-world
```

ソースコード 4.2 hello-world/src/hello-world/handler.clj

```clj
(ns hello-world.handler
  (:require [compojure.core :refer :all]
            [compojure.route :as route]
            [ring.middleware.defaults :refer [wrap-defaults site-defaults
                ]]))

; 書き換える関数
(defn index []
  "HOGE")

(defroutes app-routes
  (GET "/" [] (index))
  (route/not-found "Not Found"))

(def app
  (wrap-defaults app-routes site-defaults))
```

ソースコード 4.3 hello-world/project.clj

```clj
(defproject hello-world "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :min-lein-version "2.0.0"
  :dependencies [[org.clojure/clojure "1.8.0"]
                 [compojure "1.5.1"]
                 [ring/ring-defaults "0.2.1"]]
```

```
  :plugins [[lein-ring "0.9.7"]]
  :ring {:handler hello-world.handler/app
         :nrepl {:start? true ; ここでnREPLの設定を行います
                 :port 5000}}
  :profiles
  {:dev {:dependencies [[javax.servlet/servlet-api "2.5"]
                        [ring/ring-mock "0.3.0"]]}})
```

```
$ lein ring server-headless
Started nREPL server on port 5000
2017-04-03 14:44:05.135:INFO:oejs.Server:jetty-7.6.13.v20130916
2017-04-03 14:44:05.244:INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.0.0
Started server on port 3000

$ curl localhost:3000
HOGE

$ emacs
cider-connect
localhost
5000
user> (in-ns 'hello-world.handler) ; 名前空間の変更
hello-world.handler> (defn hoge []
                        "hello world")

$ curl localhost:3000
hello world
```

4.5 spacemacs が良い (余談)

最近 MMA の先輩がおすすめしてくれたり使ってみて最高だったので vim から spacemacs に乗り換えました. 入力インターフェイスは vim が良くて実行環境としては emacs が良いいなと思っていた私にピッタリのものが見つかりました. まさにエディタ戦争を終結させる最強のエディタだと個人的には思っています.

emacs のキーバインドは腱鞘炎になるなどは言いすぎだと思いますが良いものとは個人的には言えないと思います. 設定は vim script で書くよりも Emacs Lisp で書く方が個人的には好きですね. まさに両者の良いところを取った最強のエディタですね！！.

4.5.1 Layer

spacemacs は Layer という独自のパッケージ管理システムを持っていて Layer を入れるだけでメーラー, git, 言語 support などがインストールされます. もちろんプライベートな Layer も作成することが簡単にできます.

言語 support Layer 一覧

- agda
- asciidoc
- asm
- autohotkey
- bibtex
- c-c++
- clojure
- common-lisp
- csharp
- csv
- d
- elixir
- elm
- emacs-lisp
- erlang
- ess
- extra-langs
- faust
- fsharp
- go
- graphviz
- haskell
- html
- idris
- ipython-notebook
- java
- javascript
- latex
- lua
- markdown
- nim
- ocaml
- octave
- php
- plantuml
- purescript

- python
- racket
- ruby
- rust
- scala
- scheme
- shaders
- shell-scripts
- sml
- sql
- swift
- typescript
- vimscript
- windows-scripts
- yaml

4.6　まとめ

終始 Lisp 良いよねという記事になっと思いますが Lisp は本当に良いと思うので皆さん書いてみては如何でしょうか.

参考文献

[1] JPL（ジェット推進研究所）における Lisp の顛末 `http://postd.cc/lisping-at-jpl/`

[2] Practical Common Lisp 2. Lather, Rinse, Repeat: A Tour of the REPL `http://www.gigamonkeys.com/book/lather-rinse-repeat-a-tour-of-the-repl.html`

[3] The Remote Agent Experiment: Debugging Code from 60 Million Miles Away `https://youtu.be/_gZK0tW8EhQ`

[4] The Remote Agent Experiment: Debugging Code from 60 Million Miles Away (slide) `http://www.flownet.com/ron/RAX2.pdf`

[5] spacemacs `https://github.com/syl20bnr/spacemacs`

百萬石 2017 -春- ©電気通信大学 MMA

2017 年 4 月 5 日　初版第一刷発行　【本書の無断転載を禁ず】

著　者　hogas, mizdra, h-otter, su_zu
表　紙　g_nootoko
編集者　kyontan
発行者　電気通信大学 MMA
発行所　電気通信大学 MMA 部室
〒182−8585 東京都調布市調布ヶ丘 1−5−1 サークル会館 208 号室
http://www.mma.club.uec.ac.jp/
印刷所　電気通信大学 MMA 部室
製本所　電気通信大学 MMA 部室

電 気 通 信 大 学