著者 KoRoN/香り屋

2011/12/31

目次

第 1 部 スパルタン Vim

本稿はスパルタン Vim のα版です。著者が普段考えていることを、まずはとりあえず文章にして出力しただけのものです。そのため将来にわたって追記・修正されることを保証しますが、それが読者のもとにどのような形で届くかはわかりません。あらかじめご了承ください。

はじめに

スパルタン Vim(以下、本書)は最近あまり見かけなくなった質実剛健(スパルタン)な Vim ユーザを目指す読者を対象に書かれています。本書は Vim どころかコンピュータを利用する際に、一切の甘え、妥協を許しません。Vim を含めツールの機能へ過度に頼ったり、CPU の速度や潤沢なメモリに甘えたりすることは禁忌としています。そう、コンピュータ利用の際に隠蔽されているあらゆる詳細を意識することをユーザへ強要します。

本書が目指すのは Vim を使うことを通じてユーザの革新、ユーザ自身が持つハードウェア(脳を含めた肉体)の最適化です。いわばニュータイプです。「ツールが人間に合わせる」など愚の骨頂、「人間がツールに極限まで適応する」のです。なぜならば人間が合わせたほうが速いから。ツールが人間に合わせるにはプログラムの修正など少なからず時間がかかります。しかし人間の脳の可塑性、柔軟さ、適応力は非常に高く、ツールが人に合わせるよりも遙かに速く、現場でツールに合わせて順応できます。また Vim を正しく理解し本書が目指すレベルで使いこなすならば、作業時間そのものは「ツールが人間に合わせる」場合に比べても、遜色ないかより短くなります。

というわけで本書はただでさえ急な Vim の学習曲線を絶望的なほどに高くします。あまりに過激でスパルタンな内容のために、読者によっては動悸や息切れ、場合によっては命に危険が及ぶ可能性が懸念されます。読み進めるにあたっては健康状態その他に気をつけ、転んでも泣かないの精神を決して忘れないでください。

Vim 以前

歴史を学べ

あるツールを真に使いこなすには、そのツールの誕生に至る歴史、思想を知る必要があります。ツールがどのような考え方、目的で作られたのかを知ることで、どういう使い方をすべきか、またどのような時には使うべきではないかが見えてきます。原則としてなんにでも使える万能ツールという幻想は存在しないのです。この原則は Vim にも例外なくあてはまります。

Vim という名前は初期においては「Vi IMitation」、現在は「Vi IMproved」の略称です。この名前から分かる通り vi を前身としています。vi は BSD/Unix において生まれました。BSD/Unix は Unix の派生 OS、正しくは拡張パックと考えるのが良い、です。便利なツールを多数開発しパッケージしたものが BSD/Unix、そう覚えておけば良いでしょう。Unix は Multics へのアンチテーゼとして誕生しました。おおまかではありますがこの歴史が、Vim のツールとしての性格にどのような影響を与えているかを考えてみましょう。

Multics は当時としては巨大な OS でした。開発目標も非常に野心的かつ先進的で、高い拡張性、高い性能を目指していました。簡単にいえば、全てのコンピューティングを一つの OS、枠組みで行おうとしたのです。が、開発当時の技術水準がその目標に届いていませんでした。結果、パフォーマンスが大変悪く、現在 OS としての Multics のことを知っている人はあまり多くないでしょう。

Unix はそんな Muiltics での失敗の経験から、シンプルで独立したモジュール群で構成することを目標としました。この Unix では特徴であるモジュールで構成さ

れる点を、さらに推し進める重要な機能が開発されます。あるモジュール(プログラム)の出力を別のプログラムの入力とするための機能、そうパイプです。またあらゆるデータをテキストファイルとした点も、本書の主題である Vim を語る上では見逃せません。つまりプログラムの入出力をテキストデータとして規定し、それをパイプで繋いで複数のプログラムを連携させて目的を達成するスタイルは Unix により確立し、広く世の中に受け入れられることになりました。

vi はそんな Unix のプログラムの 1 つ(ビジュアル)テキストエディタとして誕生します。ですからテキストデータを取り扱い、vi 以外のプログラムとパイプを通じて連携することで真の価値を、性能を発揮するのです。

そんな vi をベースとした Vim は、vi に比べてより多くの機能を持ちますが、本質は vi のそれと全く変わりません。より多くのプログラムと連携してこそ、真の力を発揮します。そのため Vim を使いこなせるかどうかは、連携できるプログラムをどれだけ知っているかにかかっている、そう言っても過言ではないのです。

Vim 以外のツール

Vim と併用すべき代表的なプログラム(ツール)を挙げましょう。

- 各種シェル(bash/tcsh/zsh 等)
- grep
- wc
- find
- xargs
- cat
- 各種ページャ(less/lv)
- head

- tail
- cut
- sort
- uniq
- sed
- make
- スクリプト言語処理系(perl/python/ruby など)

これらのツールを知っている程度では全然足りません。その特性を見極め、用途に応じて使い分け、使いこなさなければいけません。大事な事なので 2 回言いますが、これらのツールは必修です。以下、各ツールを簡単に紹介します。

各種シェル

Unix 系 OS にログインした際に最初に触れるインターフェースです。このシェルについても sh 系と csh 系、さらには別派閥を登場させ宗教戦争を煽ることができるのですが、本書では取り上げません。最近は bash がデフォルトであることが多いようです。また BSD 由来という共通項から csh 系の tcsh を使うのも良いでしょう。ただ Vim 使いの理想を言えば sh 系(bash)と csh 系(tcsh)の両方に習熟し、シェルスクリプトが読める程度に習熟しているのが好ましいです。

特にシェルでの抑えておきたい機能を以下に挙げます。知らないものがあったら必死に勉強しましょう。

- コマンドラインの編集/履歴呼び出し
- パイプ、リダイレクト
- ジョブ操作
- pushd/popd

- エイリアス

本書ではシェルスクリプトは敢えて必ずしも書けなくても良いとします。シェルスクリプトでできる大半のことは、後で説明するスクリプト処理系言語を使えば実現可能だからです。しかし、もちろん書けたほうが良いことは当たり前です。

grep

ファイル内の特定の文字列を検索し表示するツール。

wc

文字数、単語数、行数を数えるコマンドです。特に行数のカウントは手っ取り早く作業の検証をするのに便利です。例えば一連のファイル内から特定のキーワードを別のものに置き換える作業では、次のコマンドを実行して 0 が表示されるかどうかで検証が可能です。

```
$ grep -nr {置き換え前のキーワード} . | wc -l
```

find

条件を絞ってファイルを探すツール。

xargs

入力を別のツールの引数にするツール。

cat

入力されたファイルを出力するだけのツール。単純で忘れがちなので、案外使うので覚えておきたいものです。

head

ファイルの先頭を切り出すツール。

tail

ファイルの末尾を切り出すツール。

cut

入力の各行から決められたカラムを切り出すツール。

各種ページャ(less/lv)

入力をページ単位で画面に表示するツール。

sort

行を並び替えるツール。

uniq

同じ行をフィルタするツール。

HTTPD のアクセスログから IP アドレスを cut で抜き出し、sort して uniq -c してさらに sort -n すると、簡易なアクセス元 IP のランキングが得られるなど、応用範囲が広いです。ちなみに uniq は sort -u で代替できる場合がありますが、この例は代替できません。

sed

Stream EDitor の略。編集するファイルを一度すべてメモリへ格納する Vim とは

異なり、sed では逐次処理していきます。そのために人間にとっては編集が行いづらい傾向がありますが、大きなファイルを大量にかつ高速に(定形)編集するのには向いています。

make

プログラムをビルドする際に使うツールですが、各種手続きの自動化にと大変便利に使えます。代替のツールはいくつも提案されていますが、残念ながら汎用性の面でこれを超える定番とは成り得ていませんし、Vim との親和性も非常に高いので是が非でも使いこなすべきです。

入力である Makefile を書けることはもちろんですが、少なくとも純粋な make と GNU make のバリエーションの違いを意識し、それら用の Makefile を書き分けられることはほぼ必須です。欲を言えば Windows の nmake の方言も理解していると良いでしょう。

スクリプト言語処理系

Vim を使うという視点からでは、標準入力を読み標準出力へ書ける言語処理系であれば、どの言語を選んでも良いでしょう。

ただ多くの言語を知っておくことは良いことです。各言語の提供するプログラミングに対する概念や哲学は、たとえ他の言語やこれまで解説してきたツールを使いこなすにあたっても、多くの示唆を与えてくれます。また Vim のスクリプト言語を使う際の助けにもなるでしょう。

その他のツール

これらの他にも「テキストデータを扱い入出力にパイプを利用可能なツール」で

あれば、使えるものが多ければ多いだけ Vim を活用できることになります。また特にスクリプト言語処理系は Vim と連携可能なツールを新たに手軽に作れる非常に有用なツールです。1 つ以上の言語に存分に習熟し参考書などを見ることなく自在に書けるようになっているのが好ましいです。

Vim の少し手前

Vim について書く前にもう少し知っておくべきことがあります。

キーボードを選べ

あなたはどのようなキーボードを使っているでしょうか? メーカー製 PC に付属のキーボード? 論外です。Vim を使う上で正しいキーボードの選択は切っても切り離せません。

英語配列キーボード

まず大前提として英語キーボードでなければいけません。英語キーボードと日本語キーボードでは特に記号の配置が異なります。Vim の前身である vi は英語キーボードを前提に作られました。そして vi にも Vim にも記号キーに多くの機能が割り当てられています。つまり英語キーボードでの記号の配置を前提に、各機能が割り当てられています。ですから配置が異なる日本語キーボードを使うことは、それだけで不自然なキーマッピングを甘受することになります。

通常、キー配列は OS の設定で変更できます。つまり日本語キーボードでありながら英語配列として利用できます。ただしそれはおすすめしません。なぜならキーボードに印刷された記号と実際の記号が著しく異なってしまうため、英語キーボードと日本語キーボードの特性をよく理解した一定以上の上級者、でなければその混乱から這い出ることはほぼ不可能です。

真の上級者は逆に日本語キーボードでも操作できるようになっている必要があります。実生活の中で自身が使い慣れないキーボードをあてがわれることは珍しくあ

りません。また他人のキーボードを借りて、ちょっとした作業をするということもあるでしょう。そのような時にあきらかに入力がもたつくようでは真の Vim 使いには程遠いでしょう。どのようなキーボードであっても、その場で適応していく柔軟さが欲しいものです。

その他のキー配置に関して

キー配置という意味でコントールキー(Ctrl)の位置は非常に重要です。Vim ユーザであれば A の左隣が Ctrl であるべきです。なぜなら Emacs 程ではないにせよ Ctrl は Vim でも多様します。また Vim と連携すべきツールでも Ctrl を用いた操作は避けて通ることができません。

特に ESC キーを Ctrl+[で代用することは重要です。ESC キーはキーボードの種類によってその場所が大きく異なります。それに対して Ctrl+[であればほとんど位置が変わりません。位置が変わらないのであればキーボードの種類が変わったところで困ることは何もありません。

また ESC に限らず、特殊キーのいくつかは Ctrl との組み合わせで表現できることは、是非とも頭に叩きこんでおくとよいです。以下のようなキーが利用可能ですが、これらは入力コードの 31 以下の値が等しいことを利用している、という事実は覚えておいて損はないでしょう。

- TAB：Ctrl+I
- Enter：Ctrl+M
- BACKSPACE：Ctrl+H (ターミナル経由ではいくつか問題があるが)

なおファンクションキーを活用するのは諦めましょう。環境によっては正しく入力できないことがあります。これはファンクションキーの端末上での入力コードが

単純ではないことによるのですが、結果的にまともには使えないと考えたほうが良いです。

推奨キーボード

以上を踏まえて Vim ユーザが使うべきキーボードを挙げるとすれば、これはもう Happy Hacking Keyboard Pro 以外は在り得ません。値段が 2 万 5 千円ほどしますが、真の Vim ユーザであればコンピュータに向かい合う時間のうち 80%以上はキーボードというインターフェースに触れていることになるのですから、そこにお金をかけるのは投資としての効果が十分あると言えます。

また折角買うのであれば打鍵音が静かな type S をオススメします。ほぼ 3 万円になってしまいますが、2 万 5 千円が 3 万円に増額したところで、わずか 20%です。たいした出費ではないと言えるでしょう。

ちなみに筆者は Windows であっても 90%以上をキーボードで操作します。また購入した HHK Pro は 4 台以上になっています。

タイピング

ここまでハードウェアな話が続きましたが、一度ソフトウェア的な話に切り替えましょう。ただしソフトウェアと言っても、あなた脳内ソフトウェアの話です。

あなたはタッチタイピングもしくはブラインドタッチができますか。ちゃんと 10 本の指を使って入力できているでしょうか。できていない人はまずできるようになってください。英字だけなら打てるという人は、数字までタイプできるようになってください。英数字だけなら打てるという人は、記号まで合わせて確実にタイプできるようになってください。Vim は数字キーも記号キーも多用します。それらを除外するわけにはいきません。

タッチタイプができるようになったら次は速度と正確さです。目安として最低でも 1 分あたり 100 打鍵は欲しいところです。私の知り合いには 400 に届こうかという方もいますが、正確でさえあれば速ければ速いほど良いものです。

自分のやり方を捨てろ

Vim はあらゆる意味で特殊なエディタです。その特殊な思想を理解し、受け入れて活用してこそ真価を発揮します。

行指向で考える

Vim は行指向のエディタです。だから行単位の編集操作は非常に得意です。それに対して文字単位の編集は比較的苦手です。

ところが日本語の文章というのは、一般的には文字単位の編集を必要とします。そのため日本語を良く書く人の中には一段落を一行にまとめて書き、Vim では編集しづらいと指摘する人がいます。しかしそれはその人のやり方が Vim に適合してないのです。

Vim で日本語を編集するときのベストプラクティスはこうです。

- 一文以下の短い単位で改行しつつ入力し
- 行単位で切り貼りして編集
- 連結機能(join)や整形機能で体裁を整える

特に最後の整形は、本来 Vim の仕事ではありません。整形専用のツールを使うのが真の Vim 使いの姿でしょう。

再度タイプするほうが速い

このような元文がありカーソルが Phrase B の先頭にあるとします。このとき Phrase B を Phrase B'に変更するにはどうするべきでしょうか。ただし Phrase C は 10 文字を超えない程度だとします。

```
{Phrase A} {Phrase B} {Phrase C}
```

通常のエディタの感覚では Phrase B を選択し、その状態で Phrase B'の内容を入力することでしょう。Vim でも同様の操作は可能です。ビジュアル選択して c、ct に続けて Phrase C の先頭文字を指定して…ということができます。しかしここでは C で Phase B と C をまとめて消して、Phrase B'と Phrase C を再入力するということを選択肢に入れましょう。

```
{Phrase A} {Phrase B} {Phrase C}
{Phrase A}
{Phrase A} {Phrase B'}
{Phrase A} {Phrase B'} {Phrase C}
```

十分なタイピング速度があり、あまり Phrase C が長くない場合には、実は再度入力してしまった方が、いろいろ考えるよりも速かったりもするのです。

スパルタン Vim の世界

いよいよスパルタン Vim の世界に案内しましょう。とはいえ「スパルタン」なので Vim について書くべきことは、本当はとても少ないのです。

マニュアルを読め

そうマニュアルを読んで自分で学んでください。

Vim を使う上で何よりも大事なのはマニュアルを読むことです。Vim はオープンソフトウェアとしては珍しいほどにドキュメントが充実しています。スパルタン Vim としてはもちろん英語の原文を読むことを推奨します。現在は日本語訳も揃って実用上は問題の無いほどに完成しています。しかし翻訳は間違っていたり古くなっていたりするかもしれませんし、それらがなくても原文の微妙なニュアンスが伝えきれるレベルとは言えません。ですから是非とも原文を読むべきです。

マニュアルを読んで Vim の全体を把握したい場合には、次のようなコマンドが役に立ちます。単に:help とした時はいわゆる目次が表示されます。また:help index とした時は、おおよそのキー操作の一覧が表示されます。まだマニュアルを読んだことがないなら、このあたりから読みはじめるのが良いでしょう。

慣れている人は:help に続けて調べたいキーワードを入力して、解説箇所に直接飛べるでしょう。Vim のヘルプにはちょっとしたハイパーリンク機能が備わっています。特定のキーワードの上で Ctrl+]を押すことで、そのキーワードについての解説文にジャンプできます。この Ctrl+]によるジャンプは履歴として記録されるので、ジャンプ前の場所には Ctrl+O で戻れます。これは Vim にもとからある機能と全く

同じです。

問題は初めは:help のキーワードがわからない点でしょう。それでもキーワードの入力途中に Ctrl+D を押すと、存在するキーワードの中から入力済みのキーワードを含むものを検索し一覧を表示してくれます。このキーワードは全て英語なので、Ctrl+D の検索のために入力するのも英語になります。つまり英語の語彙がある程度あれば、なんとなくでも機能が探せます。これがスパルタン Vim として英語マニュアルに馴染むことを推奨する理由の一つにもなっています。

またキーワードにはいくつか規則性があるものもあります。が、それらの規則性は自分でマニュアルを読んで身に付けてください。

利用スタイル

スパルタン Vim における利用スタイルはカスタマイズを極力しません。可能な限り設定ファイル vimrc は短く保ち、またプラグイン(plugin)も入れません。

必要な設定はその場その場で行うのが原則です。どうしても必要な設定は modeline に書くか filetype-plugin 書くことを優先します。それらに分類できないものについてはしかたがありません。vimrc に書きましょう。

プラグインには便利なものがあるのは確かです。ただプラグインを利用し始めると何をいれて何をいれないのかの線引きが難しくなり、結局たくさんのプラグインで溢れかえり Vim 自体の動作が多くなります。仮に Vim の起動が 1 秒以上かかるようであれば、もう何かを間違えています。一度全部のプラグインを捨ててしまいましょう。Vim は一瞬で起動してこそ、その他のツールとの連携がしやすいのです。

またプラグイン同士の相性という問題もあります。組み合わせ次第では正しく動作しないというようなことが簡単に起こります。これはプラグインを書く上で、競合を避けるようなルールがないことが原因です。

そのためにプラグインの作成だけでなく、ちょっとしたキーマップなどのカスタマイズをするのも、本当は考慮すべき要素が大変多いのです。

カスタマイズの鉄則

どうしようもなくカスタマイズをせざるを得ないということもあるでしょう。そのような場合の鉄の掟を示します。

- 既存のキーマップは潰すな
- (可能な限り)キーマップは増やすな
- コマンドを増やせ

既存のキーマップは潰すべきでありません。なぜなら Vim のデフォルトのキーマップは良く練られたものだからです。明らかに適当に付けたのだろうというキーマップもあるような気もしますが、使ってみると案外しっくりします。また既存のキーマップを変えなければ利用するプラグインが normal!ではなく normal コマンドを使ったために、自分が潰したキーマップとかち合ってヒドイ目を見るということを事前に防げます。

キーマップを増やすべきではない理由は、あらゆるキーマップは Vim が将来利用する可能性があるためです。唯一 mapleader/maplocalleader を除いては。つまり mapleader を用いないキーマップは一切するべきではありません。もっともプラグイン同士で mapleader がかち合って痛い目をみることはありますが、日常茶飯事すぎるので対策は各自で取ってください。

コマンドを増やすことは許容します。なぜなら他のプラグインとぶつかり嫌な目にあうことはまずありません。またコマンドだけ定義してキーマップは定義しないというスタイルが、そもそも推奨されないカスタマイズにおいては最も洗練された方法でしょう。

Vim 使いの持つべき素養

スパルタンな Vim 使いのもっとも重要な 3 つの素養があります。記憶力と思考力、そしてキーボードを叩く運指の瞬発力です。

理想的にはすべてモードにおけるキーマップを記憶し、またツール毎に異なる正規表現など言語仕様を完全に把握し、現在の操作対象に応じてそれらを瞬時に切り替えて使いこなすべきです。それらを特定のツールに合わせるためにカスタマイズを始めだすと、いくら時間があっても足りなくなります。思い出してください。あなたは仕事を片付けるために Vim を使っているのです。それぞれのツールの適切な使い方を知っていれば、そもそもカスタマイズにかける時間は必要ありません。

短期記憶も同様です。ジャンプリストやディレクトリスタック、これらを Vim に任せるのではなく人間の側で常に把握しておければ、効率的な操作を実現するでしょう。たとえ把握できないとしても、それらを表示する手段を覚えておくことは非常に意味の有ることです。

思考力とはすなわち応用力です。Vim では基本的な操作を組み合わせて、複雑な編集を実現することができます。それらの操作の組み立てをいかに速くできるかが、Vim を使う上での決定的な効率の違いとなってあらわれます。

最後の要素、運指の瞬発力は意図したキーを正確に速く入力することです。しかもタイプそのものは無意識のうちに。人間の脳は良くできたもので、特に身体(指)

を動かすことは、訓練を重ねさえすれば大脳を使わずに小脳のレベルで完遂できます。つまり複雑なキータイプを小脳だけで実行できるようになっていれば、どのようなキータイプをするのかを大脳で意識した(命令を発行した)あとは小脳が勝手にキータイプをしてくれるので、そのキータイプが終わるまでの数秒間に大脳は休んでいます。しかし大脳を休ませるなんてもったいないことはすべきではありません。次の編集内容と操作方法を考えることができるのです。

つまりスパルタン Vim が目指す境地とは、大脳の前頭葉が司る思考力・統率力のもと、視覚野からくるディスプレイの情報に応じて、海馬から適切な記憶を引き出しつつ、より良い判断をくだして小脳へキータイプの命令を行う、このサイクルを並列に絶え間なく行うことです。Vim は脳内でこのサイクルを実現するのに適しているといって良いでしょう。一度このサイクルが完成されればエンドルフィンなどの脳内物質が分泌されるのか、快感を覚えながら疲れを忘れて一心不乱に Vim を操作し続けられます。

そうして一節には一般人の 300 人分もの仕事をこなせる、スパルタン Vim が誕生するのです。

第 2 部 Vim 昔語

Vim 昔語は 2011 年の夏に私の Web サイト(http://www.kaoirya.net)で集中的に連載したブログ記事を修正して収録したものです。

遭遇編

mattn さんのブログ記事を読んでいたら懐かしくなったので思い出話でも

当時私は大学生で、自宅、研究室、バイト先の 3 箇所で開発をしていた。Visual C(Studio の前身)、ViVi、jvim なんかを使ってプログラムを書いていたと記憶している。 jvim のサイトには gvim(version 5)のバイナリがあったので試してみたが、ろくに設定もされていなかったものだから「ああ jvim で良いな」と思ったものだった。

ところがふとしたことから本家の Vim のマニュアルを読み、添付されているサンプルの設 定(vimrc_example, gvimrc_example)を利用したとき、私に衝撃が走る。それまで書いて いた C、Perl、TeX のコードがカラフルに色づけされていた。圧倒的に読みやすい。今で は珍しくないシンタックスハイライトも当時はキーワードハイライトが出始めた頃でまだ珍しかった。しかも vim では多くの(見たこともないのが大半だった)ファイル形式について既にシンタックスハイライトが提供されていた。そこには雲泥の差があった。ただ 問題が無かったわけではない。

TeX ファイルに日本語が混ざるとハイライトが崩れていたのだ。原因は Vim の正規表現エンジンが日本語(マルチバイト文字)に対応していなかったこと。卒論を TeX で書いていた私には致命的とも言える問題だったため、期限の迫りつつある卒論そっちのけで正規表現エンジンの改良にのめり込んだのだ。regexp.c を印刷&製本し、短くない通学時間の大半を解読と修正方法の検討に費やした。そうして正規表現エンジンのマルチバイト文字対応化は完成した。またソレ以外にも多くの修正を完成させ、普段の使い方では困らないようになっていた。

私はこの成果を誇らしく思い、共有&公開しようと考え Bram 氏(説明するまでもないと思うが Vim の原作者である)にメールで連絡をとった。思えば明確な意図をもって英語でメールを書いたのはこのときが最初だった。ほどなく Bram からの快諾を受けコンパイル済みバイナリを配布するようになった。当時の香り屋はまだ独自ドメイン kaoriya.net 取得前で研究室のサーバを使っての配布だった。配布にあたってのコンセプトは「ダウンロードしてすぐに無設定で便利に使える」。当時の私の技術・知識・思慮不足はあったものの、今もそこは揺らいでいない。

この時、一緒に考えていたことがある。日本独自のモノになってはいけない。成果は可能な限り Official に還元し、世界全体に貢献するべきである。厨二病のソレである。逆に言えば、変更を行う場合にはそれが(自分・日本人だけではなく)世界的に見てどういう意味を持ち、どうすればその有用性を理解してもらえるかを考えながら、具体的な作業に落としこむ必要がある。この作業には最終的にどういう文面で vim-dev に送ればすんなり受け入れられるか考える工程も含まれる。特にこれは英語力に(今も)不安のある私にとっては重要で、議論になればまず勝てない。だったら議論の余地が無いほど、ぐぅの音もでないほど説得力のある変更内容とパッチを作れば良いのだ。この時の訓練は、仕事をするようになった今でも、相手が日本人であっても役に立っている。

そうやって vim-dev で活動を重ねて拙いながらも英語でコミュニケーションすることに苦 を感じなくなった頃、日本人からと思われる投稿を見つけた。そう mattn さんである。私はまたしても衝撃を受けた。彼の「とりあえずパッチできたから必要かどうかわからないけど送るわ」という姿勢にである。当時から彼のパッチのクオリティは非常に高く、しかし先進的な機能ほど本家に取り込ませる上ではお世辞にも良い戦略とはいえず、また英語に堪能というわけでもなかった。今にして思えば大阪系の積極性がなせる技だったのかもしれないが、当時の私はやや反感を覚え

たほどである。でも Official に送ると いう結論と行動そして変更内容のセンスの良さには非常にシンパシーをも覚え、どちらからともなく連絡を取り、気が付けば毎日のようにチャットをする仲になっていた。

以下、続くかもしれない。

ちなみに mattn さんとは今日までオフラインでの面識がない(笑) Bram とすら会ったことがあるにも関わらず。

スクリプト編

Vim スクリプトにまつわる愉快な思い出話。

前のエントリがどうやら好評なので、調子にのって続編として Vim スクリプトにまつわる愉快な思い出話など。

チャットで mattn さんとやりとりをする中で、Vim スクリプトが話題になったキッカケはよく覚えてない。たぶん最初は calendar.vim あたりを見せられたのだとおもう。今に比べればまだまだ単純だった calendar.vim はそれでも良くできており「おおっ」と感心しながらも、ちょっとしたプログラマ特有の嫉妬をこめて「皇紀での表示はできないのか?」みたいなボケをかましたのだが、本場のボケ返しはそれを上回っておりすぐに皇紀 に対応した版ができあがって舌を巻いたりしていた。大体はこんな感じで mattn さんが小物スクリプトを作り、私が添削したり改良したり一般化して返す、という感じで幾つか のスクリプトが生まれていった。特に新聞サイトの見出し表示(当時 RSS は普及してなかったので HTML パースが一般的だった)や、今は無き Excite 翻訳あたりが記憶に残っている。

時間的には前後するが、私が役に立つ Vim スクリプトを書いたのは autodate.vim や format.vim あたりが最初だろう。特に format.vim は西岡さんが日本のメーリングリストに投稿したスクリプトがキッカケだったハズだ。そのスクリプトは非常に泥臭い方法を用いてはいたが、ある一定幅でテキストを折り返して整形するというものだった。今では当たり前の機能だが、当時はそんなこともできなかった。私はそのスクリプトが採用していた泥臭い手法にはガッカリしたものの、その目指す先・行き着く先に大きな感銘とインスピレーションを受け、西岡さんの版を基に Vim 固有の機能(正規表現で画面上のカラム位置を取り扱える)を使った版を書いた。そ

の後、そのスクリプトは西岡さんの手 によって、しっかりコメントリーダーやらなにやらに対応した素晴らしい版に成長した。今では中平さんの autofmt.vim にその役目を譲り渡したが、それを可能にした過程においても実は format.vim が大事な役目を果たしていたこと、さらには modern な Vim の機能を実現可能に至らしめる上での最初の一里塚だったこと、ひいては「行き着く先に大きな感銘とインスピレーションを受け」の内容と合わせてそのうち書くかもしれない。

という感じで、私が Vim スクリプトに取り組む姿勢は万事受け身だ。誰かが書きおこしたスクリプトを眺め、少し気にいる所があれば、気に入らないところを直して返す。というのも私は本質的に Vim スクリプト、プラグインは少ないほど良いという超保守派で、今もそうである。そんな私が自ら Vim スクリプトを大量に書き起こすことなどない…ハズ だった。あの日までは。

ある日 mattn さんがいつものようにある一本のスクリプトを送ってきた。巨大掲示板 2 ちゃんねる(通称 2ch)を読む 2ch.vim である。またしても私にあらゆる意味での衝撃が走った。知っている人もいるだろうが、私は筋金入りの 2 ちゃんねらーだった。今では書くことも少なくなったが当時はバリバリ書きこんでは煽られてファビョっていたような気がする。そしてこの 2ch.vim、確かに動くが例によってツッコミどころ満載である。簡単に言うと新聞サイトリーダーの延長で HTML をパースしていた。mattn さんは私につっこませるためにわざとやってるんじゃないかとも思ったが、その時点で私の怒りが有頂天で寿命がストレスでマッハだった。いや当時まだブロント語はなかったと思う。これなら w3m(テキストブラウザ)で見たほうがはるかに良い、少なくとも dat(2ch の生データ)で読むべきだろう!

私はその衝撃と怒りに身を任せ Vim スクリプトをゼロから書き始めていた。そうして 2ch.vim を受け取った昼過ぎからコードを書き続け、翌朝には 2ch を読めるブラウザーが誕生した。Chalice for Vim だ。さらにその晩か、翌日だったかには 2ch

に投稿できるようになっていた。その後の Chalice は、まぁ順調に機能を追加していったものの、私が 2ch に飽きた最近ではもっぱら github の肥やしというところだろうか。恐らく Chalice は Vim と Vim スクリプトをプラットフォームとして作られた世界初の本格的なアプリケーションだったと言って良いだろう。もしかしたら今もあとに続くアプリはないかもしれないが。ともあれ Chalice の誕生で私が 2ch を利用する時間は拡大し、煽られてファビョる頻度も劇的に増大した。

Chalice を開発する中で Vim スクリプトに限らず、Vim のあらゆる性格を体験したと言って良いと思う。見えてきたのは Vim に足りないものと弱点とバランス、Vim では何をすべきで何をすべきではないか。

以下、続くかもしれない。

ちなみにあとでわかったことだが 2ch.vim を書いた当時の mattn さんは dat の存在を知らなかった。

激闘編

思えばあの頃は常に何かと戦っていた、そんな思い出話。

> *Vim スクリプトの暗黒時代。この頃はあの手この手を使って機能を実現してた。この頃に得たテクニックは今でも使ってるしいろんなスクリプトの断片に散らばってるはず。 --- mattn*

Vim スクリプトで大きなアプリケーションを書くことで、スクリプトのみならず Vim 全体の弱点が見えてきていた。なんでもスクリプトで書いてやろうとしたがなんでもは書けるわけもなく、Vim 本体に手を入れてでもスクリプトでいろいろできるようにしたりもした。mattn さんのボツになったパッチのうちソケットやプロセス操作などのほとんどがこの文脈にあったと記憶している。そんな mattn さんの玉砕を時に加担したり、やっぱりねと横目に見たりしつつ、私はといえばある青写真を描いてその実現に舵を切っていた。

私たちが何故そうまでしたのか。その問に答えるのは簡単だ。戦っていたのだ。相手は Emacs であり、Visual Studio や Eclipse といった IDE である。Emacs は説明するまでもない万能エディタで、始まりからしてアプリという枠を超え統合された環境だった。アドバンテージといえば Emacs は Lisp というアベレージユーザには決して使いこなせない難解な言語で構成されていた点と、あまりに統合されすぎていたためにいかんともし難くなっていた足回りの遅さ。片や IDE はプログラミング・開発という決して広くない分野へ特化したために、多くの使いやすい機能を手に入れた攻撃力のとにかく高いアプリケーション。特に当時 Visual Studio の C++において Intellisense と言われたコンテキトに応じた補完機能は他の追随を許さないほど完成された、それでいて誰にでもわかりやすい恩恵のある機能だった。

特化しているゆえのアドバンテージ、それだけに汎用性に乏しいことが弱点もなり得た。

Vim にはもともと高度な補完機能が備わっている。複数の補完方法をユーザが随時決定できるのだ。特に insert モードにおける CTRL-N/P はキーワード補完と言い、入力済みのキーワードで始まる単語をすべてのバッファから検索し順次補完候補として仮入力してくれる優れものである。ただしこれは使いこなすのが難しい。慣れてしまえば簡単なことなのだが、現在 Vim が持つバッファの内容を、どこにどんな単語が入っているかをおおよそ把握しておけば良いだけだ。実際の開発業務において、言語が決まってライブラリフレームワークが定まってしまえば使うべき語彙はさほど多くない。ある程度の熟練者であれば「これこれこういう処理を書きたい」「そのために必要な関数はあそこで使っていたな」「たしかこんなキーワードで始まるぞ」というような思考のもとで、キーワード補完は Intellisense が不要なほどの絶大な効果を発揮する。これは嘘でも大げさでもない、JARO も不要である。

だがしかし、2ch で IDE の Intellisense を引き合いにだされ「だから Vim はダメなんだ」と主張されれば、最終的に勝ち目がないことを悟るにはかなりの時間とファビョることを必要とした。幾度となく遠回りをした結果 Intellisense を Vim で実現することに決めた。 結局のところ補完だ。Vim にできないことではない。実際言語やライブラリを固定できるのであれば補完機能を作るのは難しくない。特に Java などは性質が良すぎてアルゴリズムが簡単に思い浮かんだ。だが必要なのはソレじゃない。どんな言語フォーマットでも補完でき、場合によっては IME すら記述しうるフレームワークだ。しかも世界中の Vim 開発関係者、誰もが納得する形にしなければならない。ややこしい議論に巻き込まれてしまえばフォローしきれない。

アイデアはこうだ。まず Vim に新たな補完方法を増やす。その補完方法ではスクリプトで補完候補を決定する。そして補完候補一覧を表示するメニュー機能を実装

する。こうすればあとからいくらでも補完機能を増やすことができる。Intellisense も IME も実装し放題だ。私はメニュー機能は棚に上げ(私には必要なかった)、早々にスクリプト補完までを実装し機会を待った。

その機会は思いのほか早くやってきた。世の中のトレンドがそういう時期にあったのだろう。海外のあるユーザが Windows 専用の Java(だったよね?)の補完機能を作成し vim-dev に提案したのだ。私は却下されるだろうと思っていたし実際そうなった。「それ Windows 以外のプラットフォームどうすんだよ」理由まで予想したとおりで、待ってましたとすかさず議論に参加する。その時のメールの文面はだいたいこんな感じだった。

Vim が UI を提供したらどうかな? Vim には補完機能があるけど今はファイル(バッファ)からしか持ってこれないよね。もしもスクリプトで補完候補を作ることができ、メニューみたいな UI があれば Intellisense みたいな機能をいろんな言語について実装できるじゃん。そうそうココに僕の作った補完候補のパッチがあるんだよ。こんな風に使うのだけどどうかな? これに補完候補を表示するフローティングウィンドウがあれば、Vim で Intellisense を理論上は全てのプラットフォームと言語に展開できるだろ。

日本人が大好きな「こんなこともあろうかと」をしれっと実行したのである。しかもメニューUI の実装は丸投げで。

さてこのパッチ、当時は議論されることはなかった。すぐに取り込まれることもなかったと思う。確か 7.0 へのメジャーアップグレードでマージされ、メニューUI もその時におそらく Bram 氏により実装された。つまり私の提案に賛同してくれていたと解釈して良いだろう。最初のメニューUI は特にマルチバイト絡みでかなりバグが多く、mattn さんが相当 に頑張ってデバッグしてくれていたと記憶している。

私が提案した機能は名前やキー マップこそ変わったものの、ほぼそのまま取り込まれていた。現在の補完関数の奇っ怪な引数の仕様がその証拠である。興味がある人は:help complete-functions を見て欲しい。

こうして Vim には Intellisense も実装しうる汎用の補完機能が実装された。この機能は後に neocomplcache.vim などに代表される補完系のスクリプトに繋がることになる。私が当初意図した Intellisense にとどまらず、その先の機能へ昇華したことは心の底から嬉しい。「あれはワシが育てた」ならぬ「あれはワシが生んだ」そんな心境だ。

以下、続くかもしれない。

ちなみに私はそれらのスクリプトを一切使用していない。いったいどういった了見なのだろう w

昇華編

スクリプトとネイティブコードによる拡張が本体に取り込まれていく、そんな昇華の思い出話。

回を重ねる毎にちょっとくどくなりすぎているので今回はアッサリめに。

前話から少し時期が遡るかくらいのころ、バージョン 5.x まで Vim では文字コードを扱えなかった。私は qkc を使っていたので困っていなかったが、私に届く要望・質問としてはやはり最優先事項だったと思う。もともと ABrowser を作って公開していたくらいなので文字コード認識&変換ロジックを書いて組み込むのは容易かったが、jvim のように日本特有の変更になってしまうのは許しがたかった。そんな時、開発の進んでいた新バージョ ン 6.0 に光明を見た。

今もあるが Vim の開発版に'charconvert'(以下 ccv)というオプションが追加された。このオプションは Vim でファイルを開く際のフックで、Vim スクリプト関数で文字コードを変換するという使い方をする。またこのオプションとは関係無く、どうしてかは覚えてないが Vim 自身が libiconv を使うようになっていた。私はこれをチャンスと思い、Vim スク リプト(libcall)+ネイティブコード(dll/so)+libiconv を駆使して、文字コードを変換する vim_ccv 拡張を作成し vim-dev メーリングリストに投稿した。

私のメールは例によってシンプルで短かったが、Bram からの返事は長かった。返信されたメールを一瞥した瞬間「こんなに長くちゃもうダメだ〜」とも思ったが、内容はまったく逆で絶賛と言って良いだろう。小泉元首相風に言うのであれば「感動した!」というニュアンスの書き出しになっている。結果から先に言うと、この Vim スクリプト+ネイ ティブコードによる vim_ccv 拡張はかなり形を変えたが、そ

の意図したものは Vim 本体に取り込まれている。Bram はオランダ人ということもあってか、日本などのマルチバイト文字文化圏の事情(ひいては多様なユーザニーズがあるということ)に非凡な理解があり、一旦問題を認識してもらえればすんなりとそしてベストな形で解決してくれるので、とてもやりやすい。仮に彼がアメリカ人かイギリス人だったらこうはいかなかったかもしれない、なんて考えてしまうのは偏見だろうか。

またこの vim_ccv には Bram の心を動かす仕掛けがしてあったことも重要だったかもしれない。libiconv.dll の動的読み込みだ。動的読み込みとは Windows 向けの機能で、通常 Windows 版では DLL をリンクしてしまうとその DLL がない場合に起動すらしないのだが、その DLL に依存した機能だけを無効化した状態で起動できるようにするものだ。この動的読 み込みの良い点は、1 つのバイナリでその機能を必要とする人としない人両方をサポート できることだった。ある機能は必要な人にとってはとても重要だが、不要な人にとっては心底あってほしくないものなのだ。私はこの時点で既に Perl と Python の動的読み込みパッチを提供しており accept されていた。libiconv の動的読み込み化という仕事を他人に渡したくはなかった。

ともあれ幾らかの議論はあったが、こうして Vim スクリプト+ネイティブコードによる拡張は Vim 本家を動かすものとなった。いま改めてこの事例から学べることはなんだろう。目的のためプログラミング言語の境界を超えるのに躊躇しないこと、日本特有のニーズを世界的なニーズに昇華させること、読むだけでその意味・意図が伝わるコードを書くこと、自分のアイデアや技術が必要とされる時期を逃さないこと…

以下、続くかもしれない。

ちなみつい先日も libiconv を使うコードを書いたがエンバグしてた。いろいろと

ややこしいやつだ。

翻訳編

ソフトウェアの翻訳は単に英語に堪能なだけでは足りないかも、という思い出話。

Vim の魅力の一つに充実した有用なドキュメントがある。ここまでドキュメントが充実ししかも有用なオープンソースソフトウェアは実はかなり珍しい。しかしそれが英語であっては、日本人にとって無いも同然というのもまた事実だ。個人的には苦手ではあるものの読むだけならば辞書片手に頑張ればいいじゃんと思うのだが、なかなかそうもいかないらしい。

ある日、いつものように mattn さんが「訳したんだけど」と tutor の翻訳を送ってきた。 tutor というのは Vim のチュートリアルドキュメントで、書かれているとおりに 1 時間未満の訓練をすることで基本的な Vim 操作を覚えられる、というなかなか価値の高いドキュメ ントだ。それを mattn さんが訳したのでレビューして欲しいとのこと。さっそく幾つか気になるところを直しつつ、mattn さんが意図的に訳さなかったであろう部分にとまどった。おそらく詩だと思われる原文はこうだった。

> *---> 1) Roses are red,*
> *---> 2) Mud is fun,*
> *---> 3) Violets are blue,*
> *---> 4) I have a car,*
> *---> 5) Clocks tell time,*
> *---> 6) Sugar is sweet*
> *---> 7) And so are you.*

2.6 節「行の操作」の対象例文なので訳さなくてもチュートリアルに支障はなか

ったが、 意味がわからないよりはわかるほうが良いだろうと訳すことにした。が、どうしても 7 行目の雰囲気がでない。6 行目の"sweet"には「甘い」以外に「すてき」とかの意味があるので、7 行目にかかってロマンチックで洒落た雰囲気があるのだが、どうしてもその雰囲気を伝える訳がでない。そこで英語の得意な友人に相談したところ次のような大変満足の行く訳となった。

> *---> 1) バラは赤い、*
> *---> 2) つまらないものは楽しい、*
> *---> 3) スミレは青い、*
> *---> 4) 私は車をもっている、*
> *---> 5) 時計が時刻を告げる、*
> *---> 6) 砂糖は甘い*
> *---> 7) オマエモナー*

ロマンチックはぶち壊しだが洒落た雰囲気は伝わってくる。ただ本当にもったいないことだが、あまりちゃんと読まれていないのかもしれない。このチュートリアルのこの文言を指摘してきた人はあまり多くはない。

Vim の翻訳すべき文章には大きく分けて二種類ある。一つは tutor のように英語を恐れさえしなければたいがいは訳せるもの。これにはリファレンスマニュアルやユーザマニュアルが含まれ、メニューなども含めて良いだろう。マニュアルは文章ごとに原著者によっては訳しやすかったりそうでなかったりの違いはあり(Bram の英語は読みやすい)、 また量が膨大なのでたしかに大変でもあるが平均的には別段難しいということはない。 私も最初は少し関わったが、vimdoc-ja プロジェクトのメンバー(特に膨大な量のユーザマニュアルを手がけた清水さんの名前を紹介しておきたい)が頑張ってくれたおかげで、日本語に翻訳されたマニュアルが利用できる。

Vim の翻訳すべき文章のもう一つは Vim の画面に表示されるメッセージだ。英語自体はマニュアルよりもはるかに簡単で量も比較的少ないが、自惚れでもなんでもなく当時は私を除けば mattn さんにしか翻訳はできない、そういう難しさがあったと思う。その難しさとは英語の文面だけでは解釈できないことによるのだが、その一端をいくつかの例から解説してみよう。

ID	原文	実際の訳文
001	E817: Blowfish big/little endian use wrong	E817: Blowfish 暗号のビッグ/リトルエンディアンが間違っています
002	E716: Key not present in Dictionary: %s	E716: 辞書型にキーが存在しません: %s
003	number changes when saved	通番 変更数 変更時期 保存済

001 は序の口、Blowfish が暗号アルゴリズムの一つだと知っていればすぐにわかる。エンディアンもプログラマなら一般教養の範囲だ。002 は普通の英語では Dictionary と Key の 間の関連が説明できないためやや難しい。鍵のかかった辞書? eval の辞書型(Perl におけ る連想配列)だとわかればすんなり訳せる。003 は最も難度の高いものの一つ。ソースコード中の利用場所とその周辺を読みといて始めて適切な訳が決定される。ちなみにこれはアンドゥリストのヘッダーに相当する部分だった。やっかいなのは 003 タイプの原文が自分の使ったことも見たことのない機能のものであるケースで、その場合はかなりの 時間をソースコードとマニュアル(原文)とのにらめっこに費やすことになる。つまり翻訳に際してコンピュータそのもの、各種 OS、ツール、プログラミング、ソフトウェア (Vim)とソースコード、etc…そういう知識が必要になる。いやどんな翻訳でも訳者には幅広い興味と造詣があるに越したことはないと思う。

もちろんリテラルに翻訳してしまうことはできるし、そういうものも世の中には少なくない。だけど自分が翻訳するにあたって、たとえ英語が苦手だからといってそうしてしまうのか…私はできなかった。もちろんいまだに変な訳は残っているし、全部に満足の行く翻訳を付けられたわけではない。でもせっかく手間暇をかけるのだからできるだけユーザにわかりやすく、できればちょっとでも楽しんでもらいたいものだ。

実は、メッセージの原文は時間と共にわかりやすく改善されている。そのため最初にメッセージを訳した時よりも、難しい訳ははるかに減っている。見慣れたということも多少はあるかもしれないが、こんなところでも Vim は進歩し続けているのだ。

以下、続くかもしれない。

ちなみに「オマエモナー」と訳した友人は生粋の 2 ちゃんねらーである。

未来編

人は歩みを止めないし止めるべきではない、そんな明日への話。Vim 昔語、最終話。

昔語で未来編とはどういうわけだ、というツッコミは甘んじて受ける。人は歩みを止めないし止めるべきではない。常に何事にも一歩を踏み出す勇気が必要、そんな明日への姿勢を新たにした思い出話。

2008 年秋の早朝、私は赤坂プリンスのロビーにいた。緊張していた。話はその数ヶ月前にさかのぼる。Bram から個人的にメールが来た。

「秋に東京に行くんだけど会えないか?」

「いいね、ぜひ会おう」

Bram が毎年一ヶ月ほどを世界のあちこちへ旅行していることは知っていた。この年は日本だった。私は彼が自分を気にとめていてくれて声をかけてくれたことがとても嬉しくて、軽い気持ちで会う約束をした。そして Bram の宿泊先ホテルの上階のレストランで朝食をとりながら歓談しようということになったのだ。

ロビーの電話で Bram を呼び出し、彼を待ってる間も胸中は複雑だった。メールでしかやり取りしたことのない一生会うことのなかったかもしれない相手と直接会える喜びと、メールや翻訳程度は苦ではないが会話などまともにしたことのない英語力への不安とでゴチャゴチャだった。しかも助けは誰一人いない。会うと安請け合いした数ヶ月前の自分を呪ったりもした。なお高校時代の英語の試験は一回を除いて全て赤点&追試だったと付け加えておく。

エレベーターから降りてきた Bram はとにかくデカかった。私の身長は 170cm 弱と大きくないので、見上げる Bram は 2m 近いんじゃないか。歳を重ねているため、よく目にする若い頃の写真のイメージとはかなり違うが、漂う知的な雰囲気に同じところがあった。

挨拶と握手を交わす。手の大きさは今も覚えている。お土産にと用意した浅草の小桜の かりんとうを、異文化への理解が深い彼は喜んで受け取ってくれた。彼からはスイスのチョコレート(Bram はスイスの Google 勤務だ)をいただいた。海外のチョコレートの日本とは根本的に異なる風味が大好きな私は、必要以上に喜びを表現していた気がする。

告白しよう。その後、何を話したか正確には記憶していない。ちゃんと応対できていたかも怪しい。さらに惜しいことにもう今では食べられない赤坂プリンスの朝食ビュッフェの味すら記憶にない。

しかし楽しい笑いの絶えないひとときだった。前日訪れた谷中の風景をとても気に入ったこと。ビルに囲まれた景色は世界中どこでも一緒だということ。「赤坂」と「浅草」の発声、区別が難しいこと。今日は鎌倉をみて、そのあとは関西に向かうこと。シーズンなので台風が心配だが、それを含めての日本だということ。ほうっておけば PC ばかり触っていること。この旅行では電子機器を一切封印していること。最新の Vim スクリプト は Python からの影響が強いこと。Vim の将来のメンテナンスの話。そして Vim という素晴 らしい仕事への感謝と、それを通じて出会えた喜びを。

楽しい時間を過ごした後、朝食を提供するレストランの最後の客になろうかという頃、 私たちはわかれた。私はその足で出勤し、Bram はたぶん鎌倉へ向かったのだろう。日本を楽しんでくれただろうか、台風は大丈夫だっただろうか、かりんと

うは口にあっただろうか。もっと英語ができれば、もっといろいろ喋れたのかもしれない。

でもそれは重要なことではなかった。終わってみれば当初の不安は何だったのかと思う。大事だったのは相手に対し真摯に尊敬し興味を抱くこと。そして知り合うための一 歩を踏み出す、小さな勇気。

独自改変バイナリを配布するために、ほとんど初めての自分の意志で書いた英語のメー ル、その送信ボタンを押すときのあのためらい。あれを超えるのに必要だったのは本当に小さな勇気と決断だったけれど、それらの積み重ねが今に、さらに明日へ繋がってゆく。今後、何か行動するのをためらうようならば、この経験が一歩を踏み出す次の勇気になってくれるだろう。きっと踏み出せば違う景色が見えてくるから。

そして私の手元にあったのは過去の勇気へのご褒美とも言えるチョコレート。…チョコ レート、とても美味しかったです。

以上、Vim 昔語はおしまい。

ちなみに食事している時の目線は Bram とほとんど変わらなかった。うわっ… 私の座高、 高すぎ…?

あとがき

本来あるソフトウェアをどのように使うかはユーザの自由です。しかしながら本書では全く逆に、ソフトウェア自身がその効果的な使い方を規定しているのだからそれに従うべきだという、過激で真っ向からぶつかる主張を展開してみました。でもやっぱり Vim をはじめあらゆるツールは自由に使って良いのです。自由に使う上でも新たなインスピレーションを得られるものに本書がなれば良いな、と心から思う次第です。

Vim についての詳しい情報は http://vim-jp.org/ で得られます。筆者の Web サイトは http://www.kaoriya.net/ です。乱丁落丁については筆者の Web サイトから、筆者に連絡を取ってください。

最後に謝辞を。Vim 昔語でネタにした皆さま、表紙を書いてくれた@ebicue、コミケ参加に関して実務的なことを引き受けてくれている@rin2_、みんなのおかげで本書が形になりました。ありがとうございます。

2011/12/31 村岡 太郎（KoRoN）