

創刊30周年

WORD

WORDカップ出場決定!号

2009
June

10

バトルロード

IXの
プロジェクト
オブ「フロントスター」教室

NHK大学ロボコン
2009



コーヒー
飲み比べ

「F#入門」

連載

RoR 入門
Ruby on Rails

TAKE
¥0
FREE

From College of Information Science

Vol.

10

2009年06月号

CONTENTS WORD

P3 創刊 30 周年記念 特別企画

P5 バトルロード

P14 IXの
パーフェクトオーブントースター教室

P26 NHK 大学ロボコン 2009

P43 徹夜の友
コーヒー見み比べ

P54 F# 入門

P59 入門 Ruby on Rails

P73 編集後記



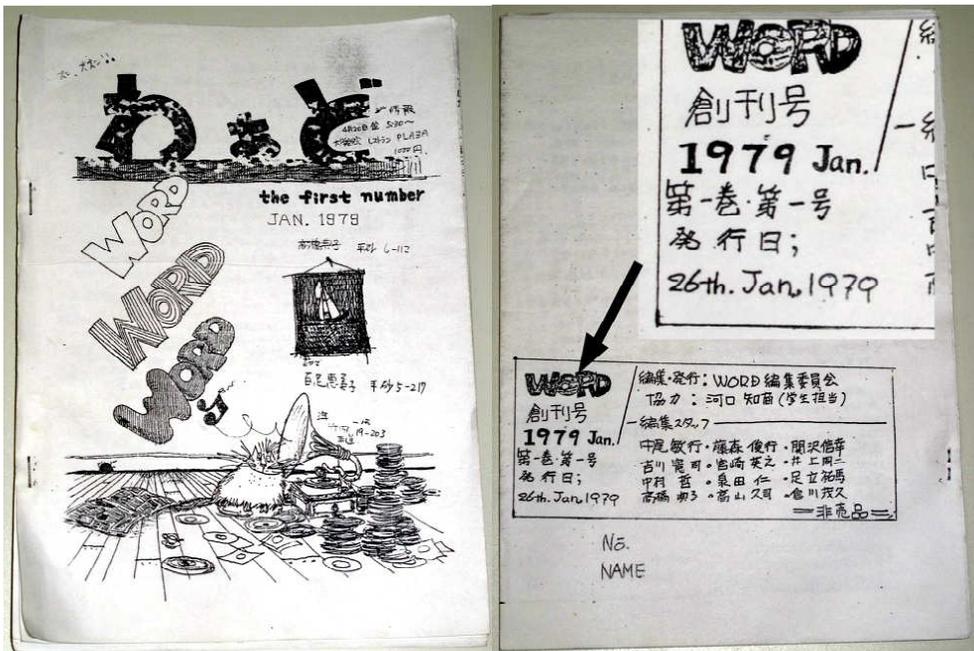
WORD

創刊30周年記念 特別企画

文 編集部 いのひろ

WORDがまさかの創刊30周年らしい

誰か 「WORD が今年で創刊から 30 年らしい。」
編集部 「ΩΩΩくな、なんだってー！！！」



と、誰かが引っ張り出してきた WORD 創刊号を確認すると、裏表紙に確かに「発行日; 26th, Jan 1979」という文字列が！ 驚くべきことに、なんと記事はすべて手書きです。ページをめくればそこには「現在現れつつある 8086^{*1}、Z-8000^{*2}、68000^{*3}などのマイクロコンピュータ〜」や「HITAC10^{*4}で〜」など、時代を感じさせる単語がたくさん含まれています。

*1 Intel 8086 プロセッサ。80x86 アーキテクチャにおける最初の CPU。800KIPS (IPS=命令/秒)。
 *2 日本人が設計した 16 ビットマイクロプロセッサ。1979 年にザイログ社が発売。1MIPS。
 *3 モトローラ社の CPU。Intel 8086 のライバルとなった。Apple 社の Macintosh シリーズが採用。
 *4 日立製作所が開発し、1969 年 2 月に発売した日本初のミニコンピュータ。約 500 万円。

WORD 創刊 30 周年記念 特別企画

1977 年（昭和 52 年）に情報科学類の前身である情報学類が設置され、そして 2 年後、学生間および学生と先生との意見交換に役立ちたいという願いで WORD が創刊されました。

創刊号には著名な先生方の文書が掲載されていますが、以下は、森亮一先生^{*1}の「無限に近い可能性について」からの抜粋です。

“最近、8086 等の性能で十分であり、その先に進んでも何に使うのかという意見を聞くことがある。これは全く誤っている。我々は、大きな山に登ったのではなく、山の麓にたどりついた処なのであって、マイクロコンピュータには無限に近い可能性が残っているのである。”〈略〉“一台の計算機はプログラムを実行するという点でやっと、1 個の遺伝子と比較できるかもしれないが、その上には細胞、組織、器管、個体、異なる種からなる社会、宇宙と続くのである。我々は、大きな山に登ったのではなく、麓にたどりついたに過ぎない。”

私たちが現在、情報科学・工学を学ぶことができるのは WORD 創刊当時のこのような熱い思いがあったからではないでしょうか。そして我々 WORD 編集部は「WORD が、この熱い思いを、情報科学・工学の無限の可能性を後生に伝えていかなくては！」と強く感じました。そこで WORD 創刊から現在まで、そして未来を描くための催しを企画しました。

企画概要

1, WORD 創刊 30 周年 記念イベント

日時：2009 年 9 月 26 日（土）16 時～21 時（JST）

場所：第一エリア 大食堂

内容：情報学類・情報科学類の著名な方による講演（16 時～18 時）

（内容は未定ですが、情報学類・情報科学類に関する講演を予定しています）

食事会（18 時～21 時）

（伝説のアレも 8 リットル版を予定しています）

参加費：2700 円（仮）

招待者：情報科学類誌 WORD の読者のみなさま、先生方

情報学類・WORD 編集部 OB/OG の方

情報学類・情報科学類の著名な方

2, 「情報科学類誌 WORD ～創刊 30 周年記念総集編～（仮称）」の作製・配布

過去の原稿をすべてひっくり返し、編集部選りすぐりの記事をお届けします！この冊子は 30 周年記念イベントでしか配布しません！

写真は過去（1986、99 年あたり）の原稿です。このような原稿が沢山入った大きな段ボール箱が全部で 4 つあります。ご期待ください！



*1 元情報学類長。筑波大学名誉教授。独自のデジタル・コンテンツの流通システム「超流通」を考案。1986 年、IFIP（国際情報処理連合）から SilverCore 賞を受賞。

たたかうかえる

闘蛙

書いた人： 編集部 ふあい

あいさつ

前回の記事から半年以上経ってしまいました。こんなに放置してしまった言い訳を書いていくとそれだけで **WORD の別冊が出せるくらい長くなる**ので省略させていただきます。

前回の記事の反省なども含めて、筆者の謝罪会見を行います。

Q.前回の記事で、マップがつぶれて見づらいのですが。

A.ごめんなさい。

Q.どうして前号にバトルロードの記事が無かったのですか？

A.ごめんなさい。

Q.お兄ちゃん、なんで単位落としたの？

A.ごめんなさい。

そんなわけで、**鬼畜さに定評のあるバトルロード・イン・バトルマニアックの攻略第2回目**です。今回は、2面の攻略を行います。

Stage2:円盤でGO!



1面とは違い、2面は縦スクロールに変わります。まく分からない**高性能な円盤**(左画像参照)にのって洞窟の下層を目指して進みます。基本的には下方向に進む強制スクロール面ですが、敵を倒すまでスクロールが止まる箇所がいくつか存在します。

敵キャラ紹介(名前はテキトーです)

◆蜂



針で刺して攻撃してきます。

1回殴るだけで倒せます。

倒した後の死体を殴ることもでき、複数回殴ると 1UP ボーナスがあります。 **ボーナスステージよりも稼げます。**

◆龍



やたらメカニックな首だけの龍です。

噛みついて攻撃してきますが、噛んでくる時しかダメージ判定がないので、**見かけによらず弱い**です。

殴っている間は攻撃しないため、ひたすら殴るだけで楽に倒せます。

関蛙



◆ビット

真横にいかづちを飛ばして攻撃してきます。

当たると3ダメージと、かなりの痛手を負うので注意しましょう。

こちらがダメージを与える度に、ランダムに移動するので、倒すのが面倒くさい敵です。



◆砲弾オーク

何故か砲台から発射され、捨て身で攻撃してくる悲しい敵です。

殴ることも出来ますが、殴ってもダメージを受けるので、なんとかして避けましょう。



◆なんか磁石もってる奴

ネーミングに困る敵。

手に持っている磁石で主人公達を引き寄せて、パンチで攻撃してきます。このパンチも当たると3ダメージ食らうので、何とかして避けたいところです。

1回殴るだけで倒せますが、うまく攻撃を避けないと返り討ちにあいます。

操作法

◆基本操作

Bボタン：パンチ

十字キー：移動

左右方向のみ、キーをすばやく2回押すことにより、ダッシュできます。

◆特殊な攻撃

・円盤アタック

画面の端にいる状態で、画面外に出ようとする方向のキーを押しながらBボタンを押すと、主人公が壁や画面端に掴まり、円盤を飛ばして攻撃します。飛ばした円盤は、画面の端まで行った後に、そのまま引き返して自分の所に戻ってきます。ヨーヨーみたいな動きです。

右図の場合、画面の一番右端にいる状態で→キーを押しながらBボタンを押せば、円盤を左に飛ばし、円盤が画面の左端までいったら、そのまま右に戻ってきます。

同じ敵に何回も円盤を当てたり、一度に複数の敵に円盤を当てたりできるので、非常に強力な攻撃です。

壁だけではなく、何故か画面の上端からも円盤アタックができます。



MAP と攻略

前回と同様、MAP に注釈をつけていく形で攻略していきます。

※1：左右に3体ずつ、計6体の蜂がいます。

この蜂は最初は動きませんが、ある程度画面がスクロールすると、突然襲いかかってきます。

刺されても0.5のダメージしか食らいませんが、襲いかかる前に倒しておいた方が安全です。

この蜂の死体は(タイミングがやや難しいですが)何回も叩くことができ、5回叩くと1UPするボーナスがあるので、頑張ってフルボッコにしましょう。

※2：右側の壁に龍が2体います。

こいつは噛みついてくる時しかダメージ判定がないので、どんどん接近してバシバシ殴れば簡単に倒せます。

しかし、倒すためには8回殴る必要があるので、少々面倒です。

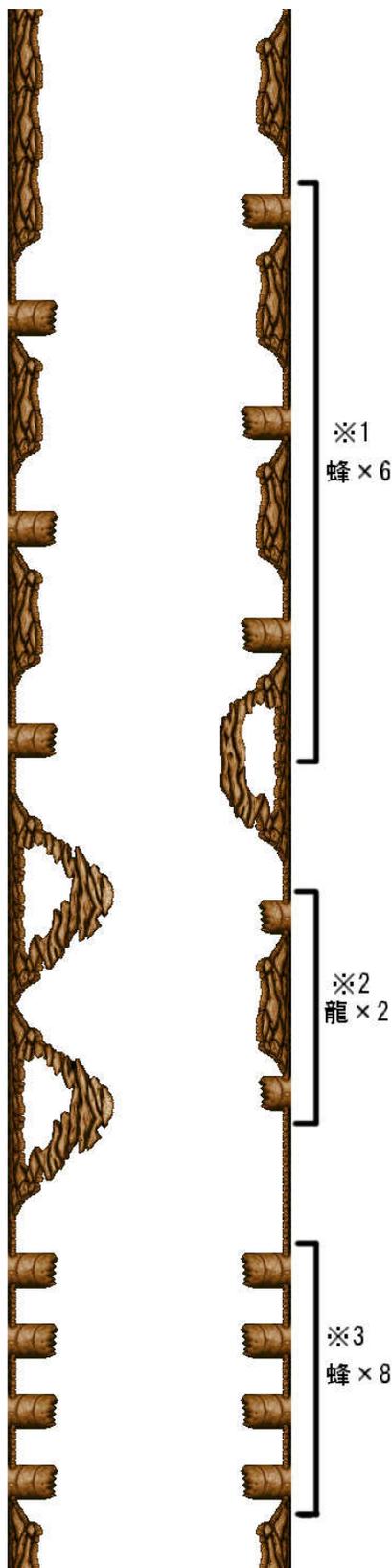
そこで、左側の壁に掴まって、円盤アタックをしましょう。当たればすぐに倒せます。

※3：左右に4体ずつ、計8体の蜂がいます。

全ての蜂が一斉に襲ってくる事はなく、最大2体までしか同時稼働しないので、あせることはありません。

画面の上端に掴まって円盤アタックをすると、一気に3体の蜂を倒せます。

また、※1でも言及したとおり、蜂の死体を5回殴ると1UPするので、頑張ってここで残機数を稼ぎましょう。



閻蛙

※ 4: 左右のトゲに当たると 3 のダメージを受けます。
最大 HP が 8 のこのゲームにおいてはかなりのダメージ
なので、避けたいところです。

スクロールがゆっくりなので、画面上部で落ち着いて避
けましょう。

また、トゲの当たり判定は**かなりアバウト**です。

手のひらや腕がトゲに触れたくらいではダメージを受け
ることはありません。

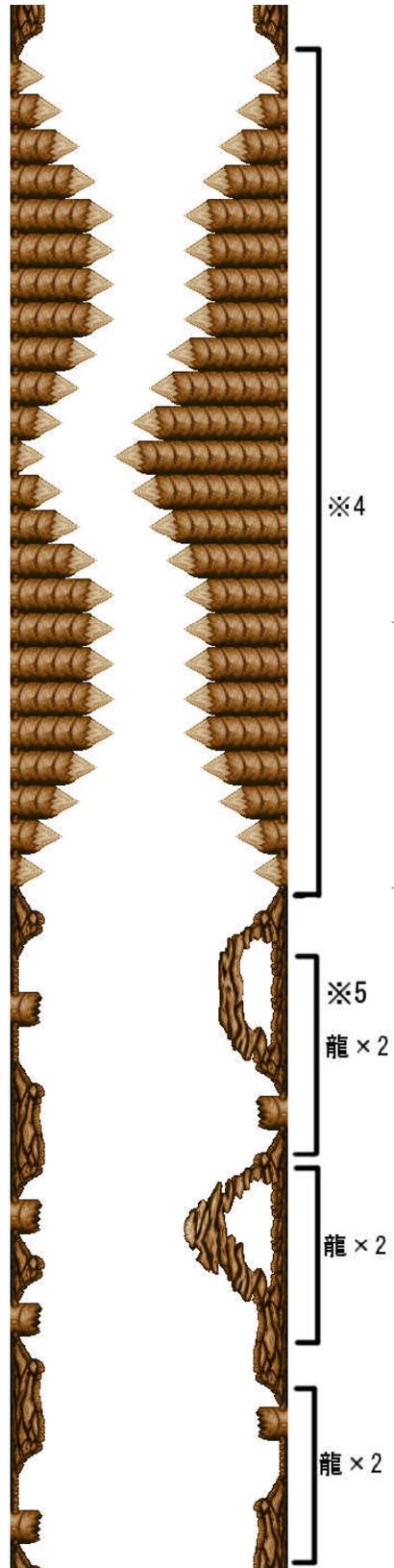
「だいたい体幹が触れるとダメージを食らう」と覚えて
おきましょう。

途中、くの字になっている箇所では、位置の調整に注意
しましょう。

位置を微調整しようとして十字キーを細かく連打すると、
ダッシュと見なされて**ものすごい勢いでトゲに突撃する**
という**大惨事**が発生します。

パンチを連打すると、若干前に進むという仕様があるの
で、それを利用すれば安全に位置を微調整できます。

※ 5: 龍× 2 体のセットが立て続けに 4 回あります。
円盤アタックでやっつけましょう。



※6：MAP 中の矢印の箇所から、砲弾オークが真横に飛んできます。こいつは殴ってもダメージを受けるので、頑張って避けるしかありません。

しかし下の方では、**あらかじめ出現位置を知っていない限り絶対に避けられないタイミングで砲弾オークが飛んでくるという、エスパー向け仕様**になってきます。

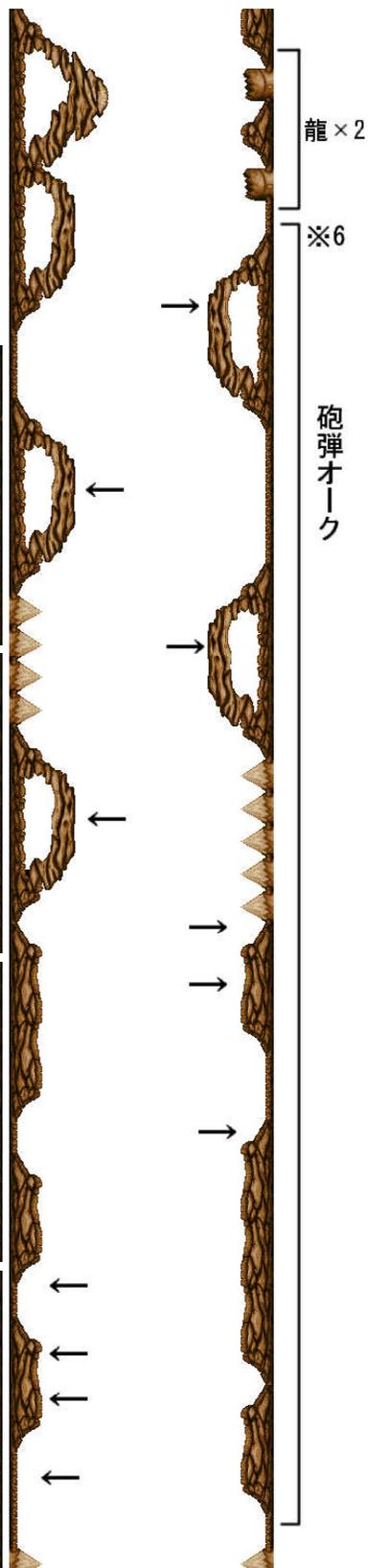
しかし頑張って出現位置を覚える必要はありません。

我々は驚くべき回避方法を発見しました。

弾丸オークの発射音とともに、円盤アタックをすると...

↑このフレーム間 無敵 ↓

これで簡単にノーダメージで切り抜けますね☆

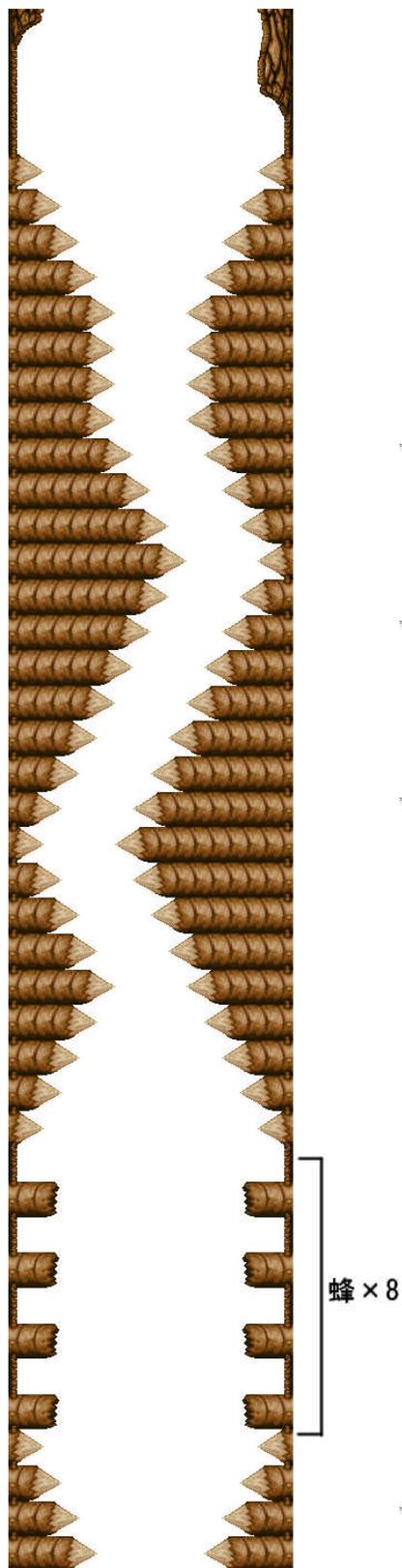


関蛙

この部分は、※ 4 と同じ要領で切り抜けるので、特に注釈を入れる箇所はありません。

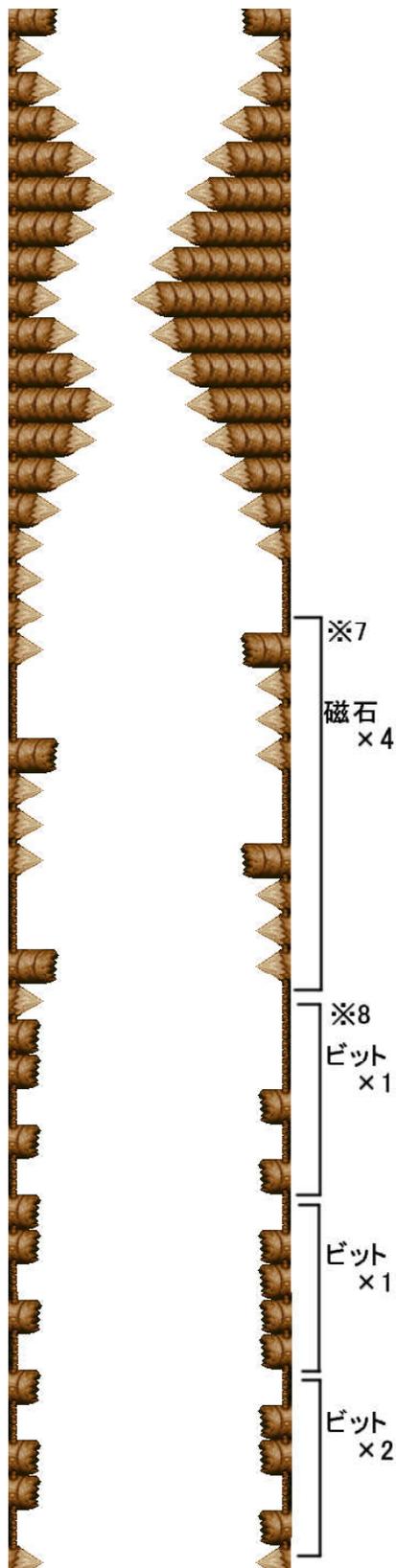
よって、このページに書くことはありません。

以下、**大量に余白が生まれるので、メモなどにご活用ください。**



※7：なんか磁石持ってる奴が4体出てきます。
 こいつは磁石で主人公を吸い寄せて、近づいたところで殴ってきます。
 殴られると3もダメージを食らうので、かなりの痛手を負います。
 こいつも殴る時以外はダメージ判定がなく、殴る前のモーションが若干あるので、その隙に近づいてすぐに倒しましょう。

※8：ビットが立て続けに4体(最後の2体は同時に)出てきます。
 こいつは真横にいかづちを飛ばして攻撃してきます。
 このいかづちに当たると、3のダメージを食らいます。
 また、この攻撃の特性上、主人公の横に並ぶという性質があります。
 これを利用してビットをおびき寄せ、壁に掴まって横に円盤アタックをすれば、一度に大量のダメージを与えることができます。
 砲弾オークの箇所でも述べたとおり、円盤アタック中は無敵状態なので、いかづちを避けることもできます。



閼蛙

※ 9：両側にトゲがあるため、横方向の円盤アタックができません。

画面上部に掴まって円盤アタックをするか、ひたすらパンチをするかしましょう。

※ 10：ここからしばらく、ファンが設置されているゾーンに入ります。

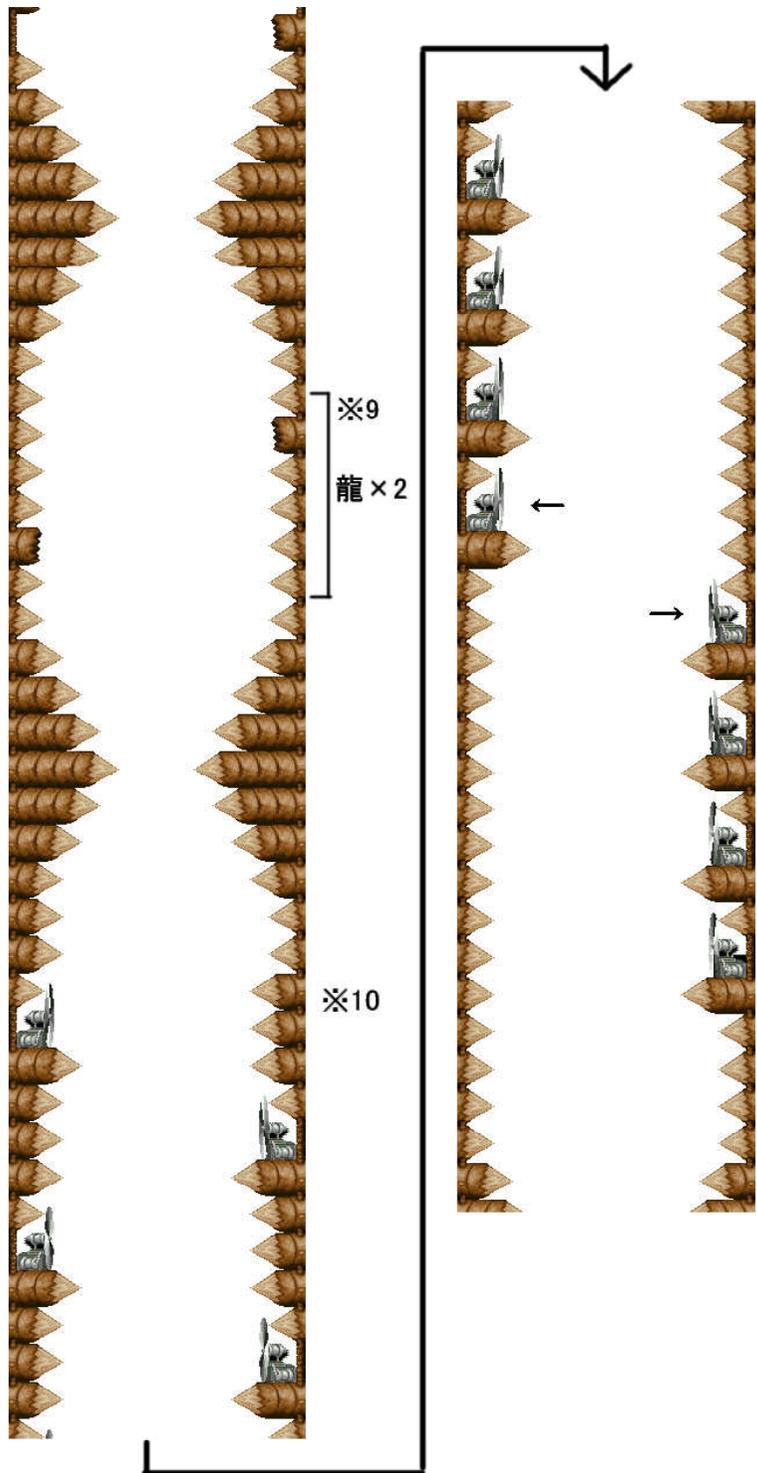
ファンの横にいくと、送風にあおられます。

何も抵抗しないと、そのままトゲに当たってダメージを食らうので、抵抗しましょう。

抵抗するときはダッシュする必要はありません。ダッシュすると、逆にファンの後ろにあるトゲに当たります。

基本的に風を押し出してくるだけですが、MAP中に矢印をつけてあるファンは**吸い込んでくる**仕様になっています。

見た目に差異が無いので、**あらかじめ知っていないと回避できません。**



※ 11：ここからスクロールが急に速くなります。3倍くらい速くなります。

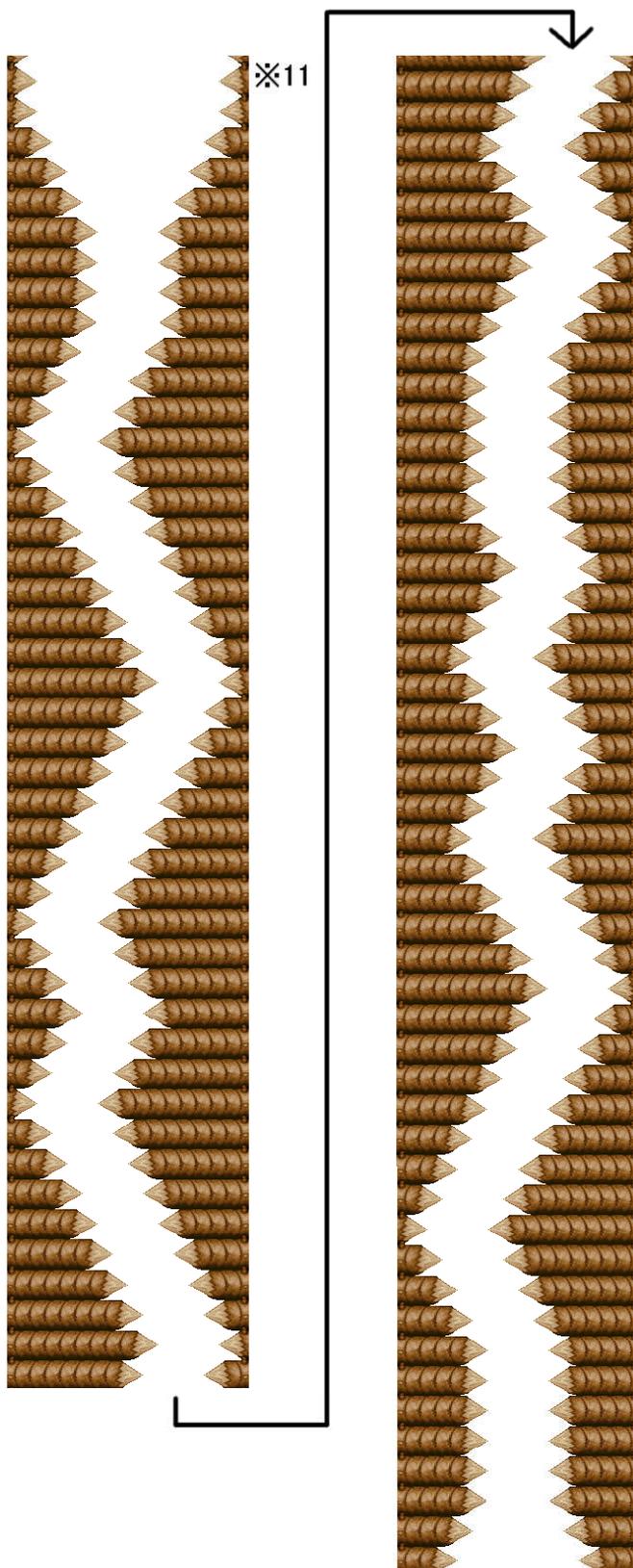
その点をふまえつつ、**あとはひたすら避けましょう**。そうとしか言えません。

一番下まで到達するとクリアです。

まとめとか

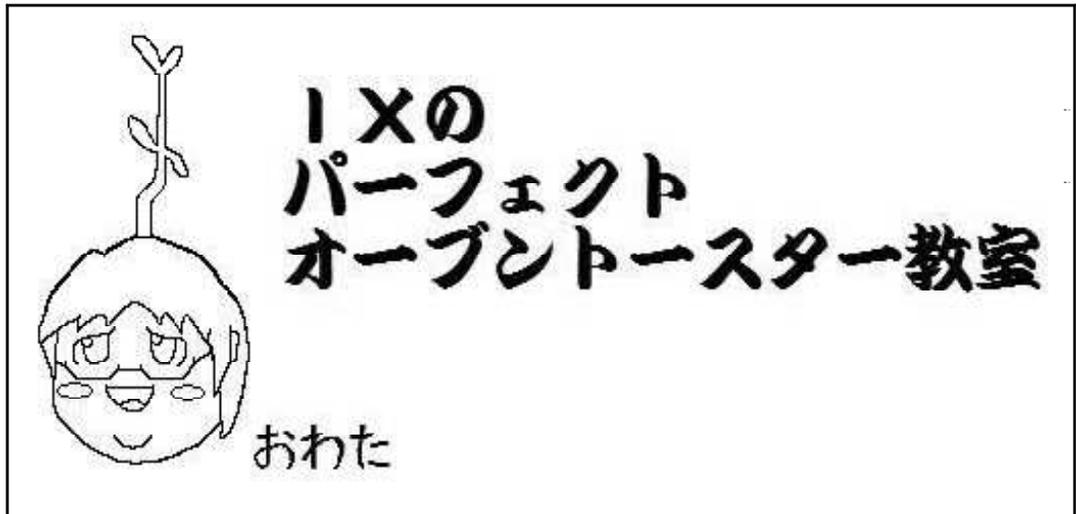
あらかじめ知っていないとダメージを食らうという**エスパー要素**が色々出てきましたが、ここまではまだまだ**序章に過ぎません**。

あと、次こそはちゃんと連載続けるよう頑張ります…



IXのパーフェクトオープントースター教室

文 編集部 IX



著者近影

みんなー！ IX がゆっくりしすぎたよ～！
こんなニートみたいにならないように、
適度にゆっくりして行ってね！！！！

■160円で得られる幸せとその可能性



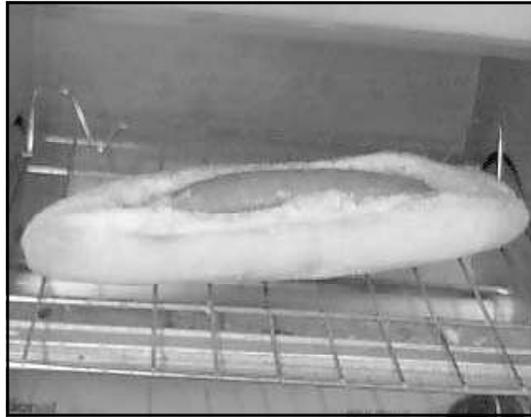
やあ、また会ったね

予告^{*1} で実施したチリドックですが、おさらいしてみます。さっそくオープンに入れましょう。とりあえず3分加熱します。

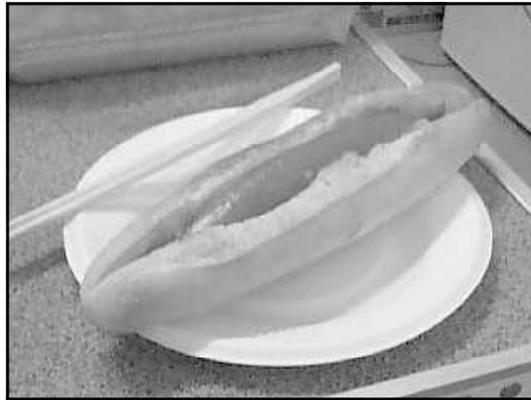


とらわれのチリドックさん

*1 予告：前々号参照。

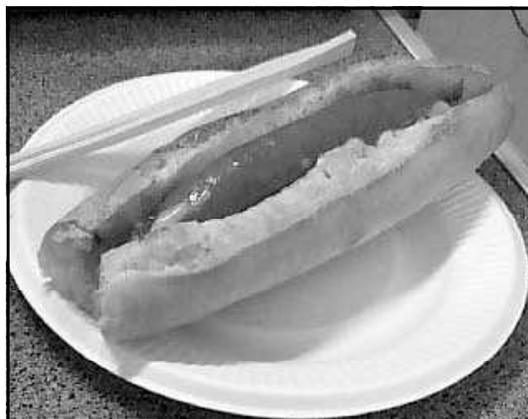


艶やかで香ばしいチリドックさん



まな板の上の鯉状態のチリドックさん

チリソースとパンの焼けた匂いが食欲をそそります。
まずは端を一口。少々硬いがそれがいい。



二人の初めての共同作業

パリッ……ザクッ……ジュワッ……ザクザク……モグモグ……モヒモヒ……

これを見てもろよっ！



これ肉汁ですよ、全部（不適切な画像^{*2}のため削除されました）

肉汁の復活したソーセージといい、パリッとしたパン、程よい辛さのチリソース……最高じゃ^{*3}。



ごちそうさまでおだぁ

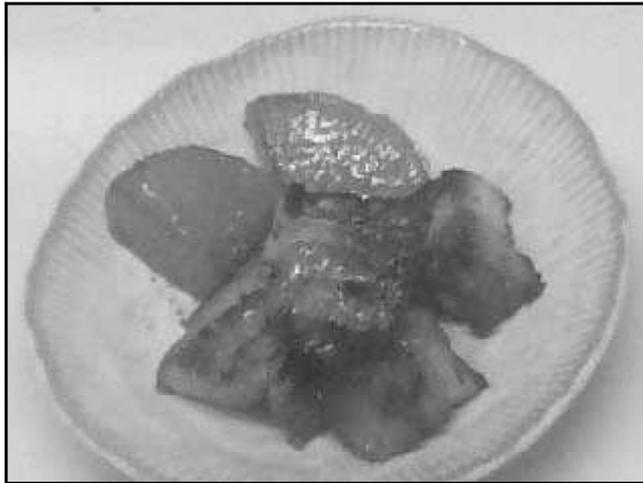
チーズを乗せたり、マヨネーズ、ケチャップ等の調味料を追加してみてもよいでしょう。

*2 不適切な画像：いかんせん画質が悪かったので。

*3 ~最高じゃ：某 GUN 道を広めた和尚さまリスペクト。

IX の POT 教室

■煮込み×炙り=驚異的な破壊力



ザ・カリカリ至上主義^{*4}

ある日の晩御飯。それは煮豚でした。茶碗一杯分のちらし寿司^{*5} とともにおいしくいただきました。

ところが、煮豚が残っています。おなかにも余裕があることですし、我が胃袋に収めてやろうと思ったのですが、満腹中枢からこんな通達がありました。

**「このバラ肉、脂がのりすぎてはいないか。
これ以上はやめておくことを推奨する。」**と。

*4 カリカリ至上主義：カリカリした食感が好きで好きで堪らないこと。

*5 ちらし寿司：す〇太郎。魚介類未使用のヘルシーメニュー。

だったら、炙ればいいじゃない。

脂も落ちるし、表面がカリカリになるし。

そこでオーブントースターの出番です。ガスバーナーや七輪などは持ってませんし。下準備^{*6}もお忘れ無く。

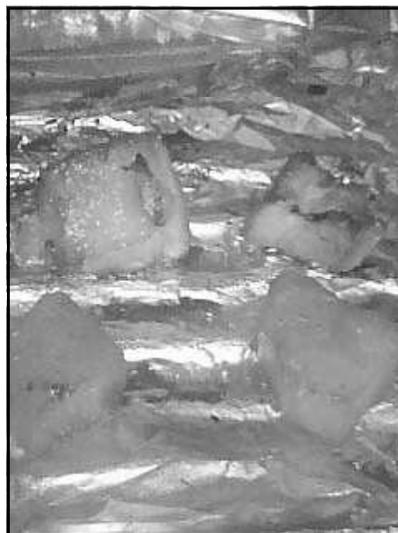


温度設定が可能なハイテク娘

しかし、ただ炙るわけではありません。まずは、低温で肉と大根を温め、次に高温で焦げ目を付けます。また、高温にする際に**煮汁を丁寧**にかけ、表面をカリカリにします。



バラ肉ゴロゴロ



手前が大根、奥がバラ肉

*6 下準備：トレーを入れる。アルミホイルを引く。それだけ。

IX の POT 教室

煮汁に含まれる醤油が焦げ、香ばしい香り^{*7}が漂ってきました。例えるならば、そう、縁日の屋台から漂う、焼きトウモロコシの香り。香りに誘われ思い出すは、すべてが新鮮だった幼き日々。あの頃は良かったなあ。もう二度とは戻らないあの夏。



CONGRATULATIONS !

しみじみしてしまいましたが、無事完成しました。

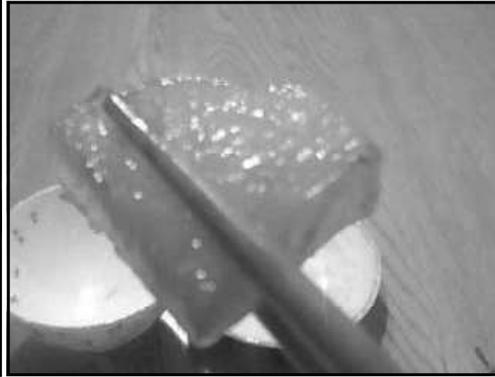


烏龍茶を添えて

*7 香ばしい香り：いわゆるひとつのバイオテロ。



ほどよく脂の落ちたバラ肉



煮汁が良く染みた大根

さあ、いただきます。

ハフハフ……ジュワァ……カリッ……モフッ……モグモグ……ハフハフ……

これですよ、これ。一手間掛けた甲斐がありました。焦げた脂身の部分が、実に良い。実験的に投入した大根も、ほのかな香ばしさがアクセントになり、なかなか良い。ただ、煮汁がよく染みてるからこそなのでしょう。中途半端な染み方では、水っぽさが際だって台無しになっていたにちがいありません。



すべて食べ終えた時、皿には光り輝く脂溜まりが

IX の POT 教室

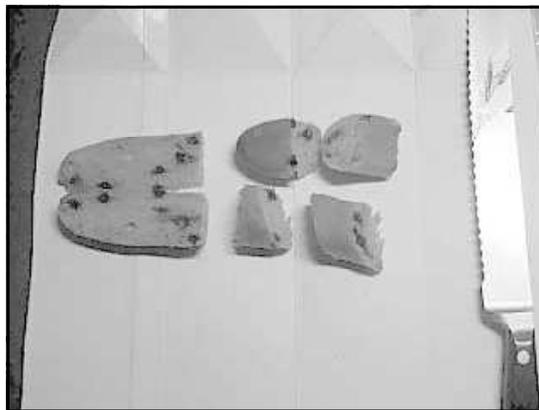
■ヤ○ザキ、梅雨のスイーツ祭

食後のスイーツ（笑）を作ります。賞味期限の切れたチョコチップスティックを加熱してラスクっぽくできたらいいなと思います。温度調整できないオーブントースターだと高確率で消し炭になるので要注意です。



えっ？チョコチップスティックなら俺の隣で寝てるよ

まずはスライスします。断面を加熱することがポイントです。定食の付け合わせのキュウリみたいな形を目指します。



ぶっつけ本番の結果がこれだよ！

早くも**挫折気味**だけど続けます。



肉屋のメンチカツを温めた後だったけど気にしないよ。

次に焼きます。今回は 120℃（最低）で 3 分間加熱します。以前、260℃（最高）で 5 分間加熱して、限りなく炭に近い何かを生み出してしまったことがあるので、慎重にいきます。実に苦かった。



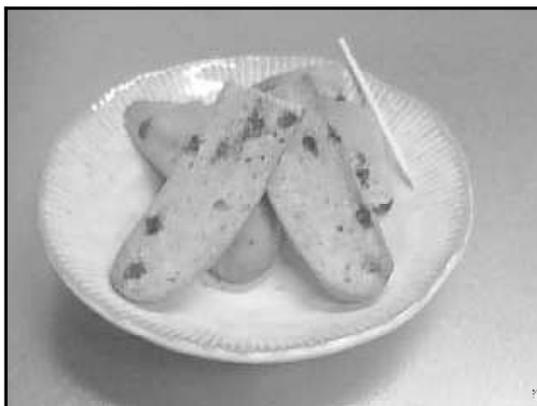
加熱開始



上手に焼けましたあ♪

IX の POT 教室

チョコチップが溶けていい感じになっています。開店前のパン屋の軒先のような香りが漂っています。



適当な盛り付け



ミルクを添えて

カロリーを気にする必要のない人は、グラニュー糖やはちみつ、チョコレートソースなどを合わせるといいかもしれません。

サクサク……ポチュン……サクッ……モキュッ……サクッ……モキュッ……サクサク……

さすがに一本分。あっという間に無くなってしまいました。



最後に牛乳をゴクリ、至福の時である

あとがき
懺悔

今回は、簡単な例ばかりになってしまいました。しかし、市販のソースを駆使したドリアや、焼きカレー、本家顔負けのチキンナゲットなど、オーブントースターの可能性は無限大です。

この機会に、オーブントースターのある生活をエンジョイしてみてもいいでしょうか？

NHK大学ロボコン2009レポート

文 編集部 yasuharu

今年は、筑波大学から工学系サークルの「つくろぼ」が NHK 大学ロボコン（以下、NHK ロボコン）に出場しました。筑波大学が NHK ロボコンに参加するのは3年ぶりのことです。

今回、私自身が NHK ロボコンのメンバーとして出場していたので、その時の様子を紹介していきます。NHK ロボコンとはどのようなものか、また、ロボットの制作過程はどのようなものかなど、裏側の部分を紹介します。

NHK 大学ロボコンとは？

NHK ロボコンとは、文字通り NHK エンタープライズが主催しているロボットコンテスト（以下、ロボコン）で、ロボコンの中でも高い知名度を誇ります。テレビでロボットが壊れる競い合っている様子を見たことがある方も多いのではないのでしょうか。このロボコンでは、与えられた課題をどれだけ早くこなせるか、ということを競います。

また、このロボコンは ABU ロボコンというアジアで行われるロボコンの代表選考会も兼ねており、優勝者にはその出場権が与えられます。今年はこのロボコンが日本でされるということもあり、上位2校にその権利が与えられます。

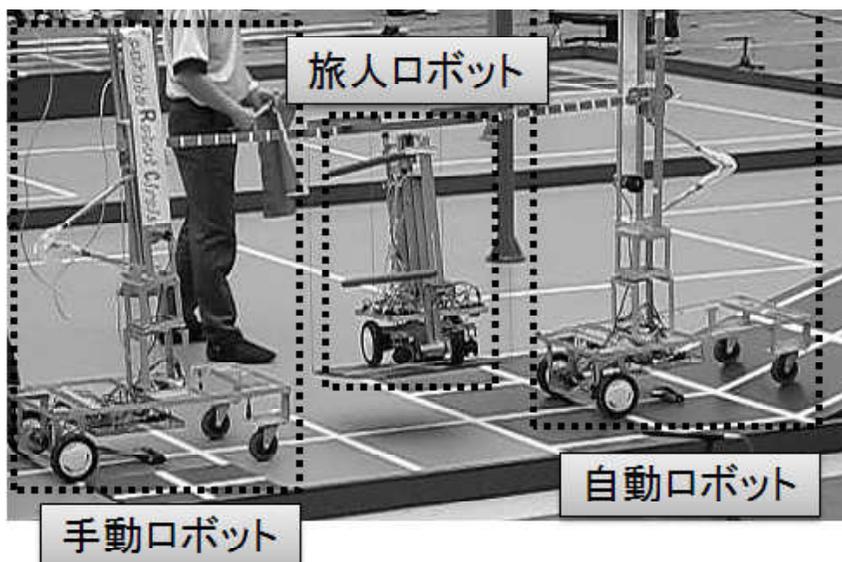
9回目となる今年は、50校以上もの応募の中から20校が選抜され、6月7日に国立オリンピック青少年センターにて行われました。

NHK 大学ロボコン 2009 の課題とルール

では、今年の競技の課題とルールについて説明します。

今年の課題は、日本の伝統的な乗り物である「かご」による旅をテーマに与えられました。タイトルは「旅は道連れ、勝利の太鼓を打て」です。

まず、この課題では旅人ロボットがかごに乗り込み、これを自動ロボットと手動ロボットの2台のロボットでかついで運んでいきます。そして、所定の位置まで運び、旅人ロボットがかごから降りて太鼓を叩きます。この太鼓がタイトルにある「勝利の太鼓」のことです。



人間が操縦できるロボットは手動ロボットのみで、旅人ロボットと自動ロボットは人間から操作されることなく、自律して動くことが必要です、

ちなみに、かごというのは名ばかりのもので、木の棒からワイヤで一枚の板が吊されているだけの代物となっています。**かごと言うよりもブランコ**と言った感じであり、非常に不安定で元々落とすことを意図しているかのような作りとなっています。

次に、ルールについて説明をします。

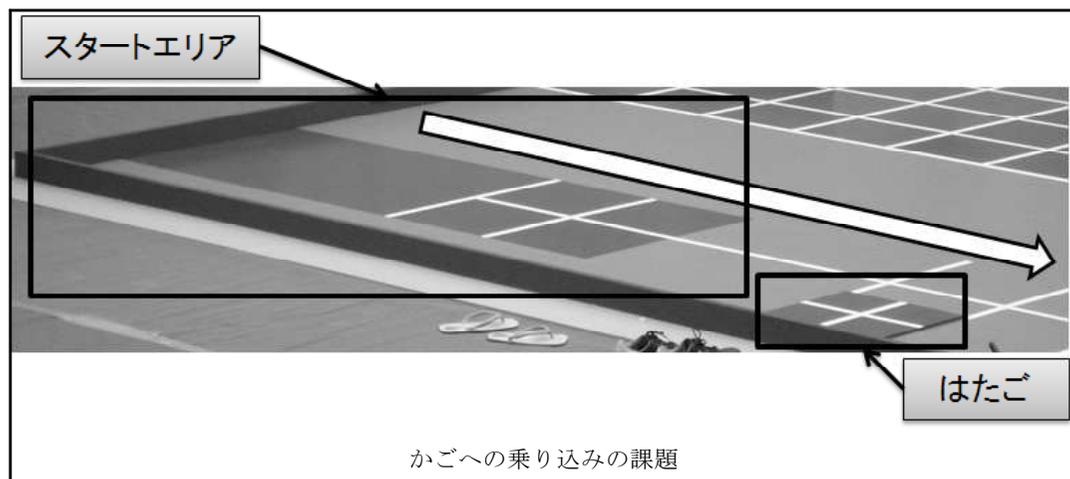
試合は、1 試合あたり 3 分間で行われ、先に太鼓を叩いたチームが勝ちとなります。試合の最中に違反があった場合には、リトライを行い、所定の位置まで戻ってやり直します。

また、3 分の試合でどちらのチームも太鼓を叩けなかった場合、太鼓までの距離が近いチームが勝ちとなります。

かごへの乗り込みの課題

自動ロボットと手動ロボットがかごを持って、スタートエリアから出発します。出発した 2 台のロボットは「はたご」*1 の位置まで移動してかごを降ろし、旅人ロボットが乗り込みます。

この課題では、かごを降ろす位置がずれてしまったりして、多くのチームが苦戦しました。

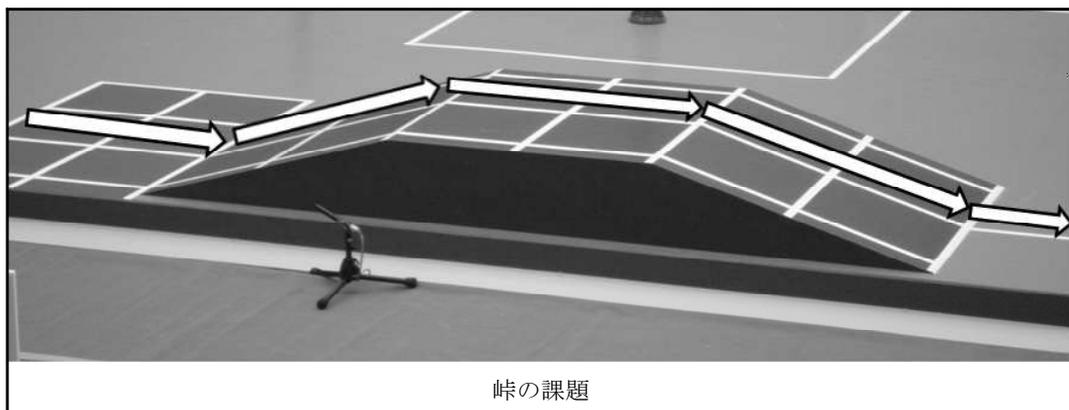


峠の課題

旅人ロボットが乗り込んだら、次に峠の課題です。ここでは、高さ 30cm の台形な台の上を通過します。勾配は 17 度あり、バランスを取ることが難しいです。実際に競技の様子を見ていると無惨にも床に叩きつけられる旅人ロボットが続出しました。

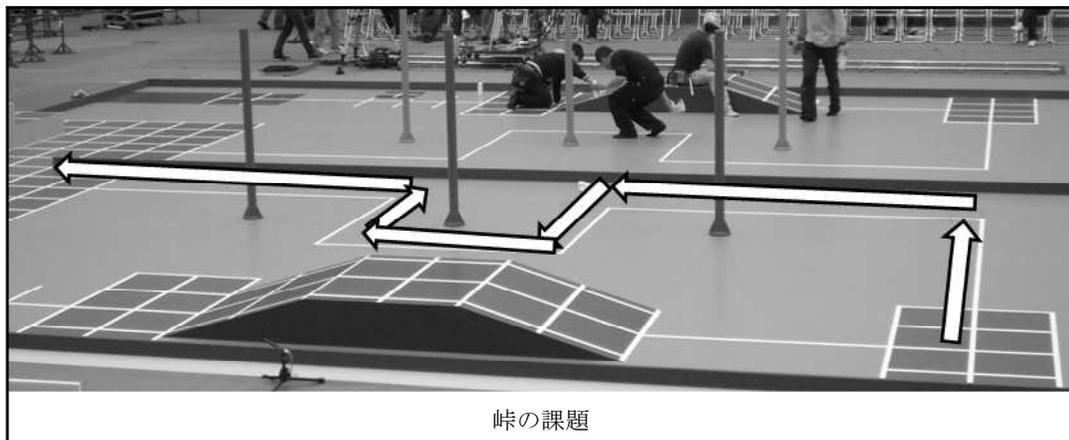
おそらく、この課題が今回の NHK ロボコンで一番難しいところでしょう。

*1 旅籠屋 (はたごや) : 「江戸時代、旅人を泊め、食事を供することを業とした家。公用以外の武士や一般庶民が利用した。はたご。」(三省堂「大辞林 第二版」より引用)



林の課題

峠を越えた後は、林に見立てたポールの間をぬって進みます。かごと旅人ロボットはポールに触れてはならず、ポールの間をうまく通っていかなくてはなりません。

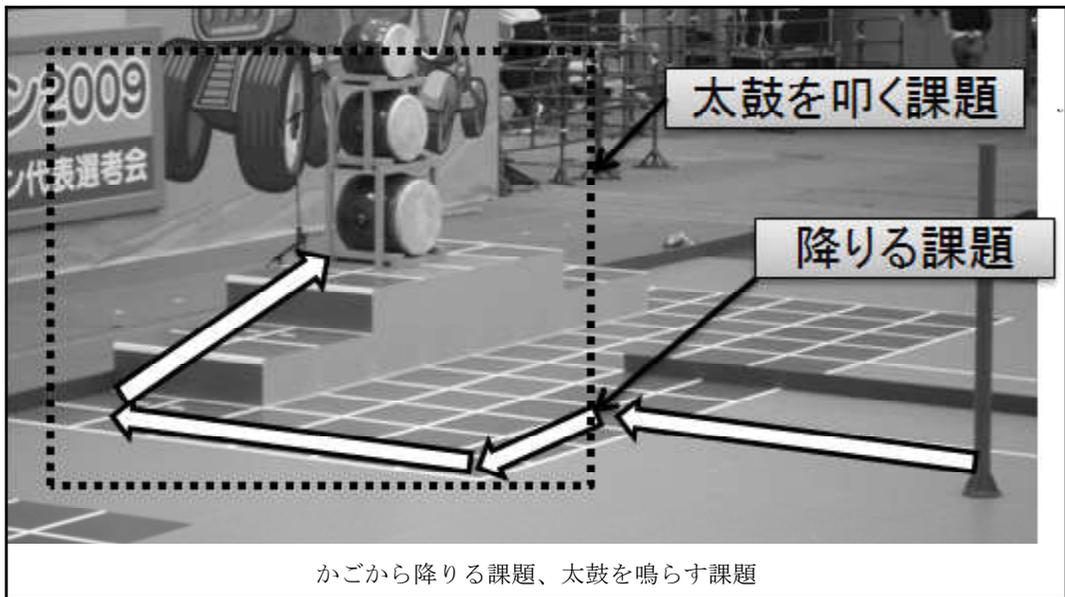


かごから降りる課題

林を抜けて太鼓の近くに来たら、手動ロボットと自動ロボットがかごを降ろします。旅人ロボットはかごが地面に降ろされたことを検知して、次の課題へ向かいます。

太鼓を鳴らす課題

かごから降りた旅人ロボットは最後の課題として「勝利の太鼓」を叩きます。太鼓は、1段が 25 cm ある階段の上に置かれており、太鼓の最長部は 2m 近くあります。旅人ロボットは、階段を上って叩いたり、周りからアームをのぼしたりして太鼓を叩きます。タイムを早くしたいチームでは、階段を上らずに周りから太鼓を叩きます。



作成したロボットの紹介

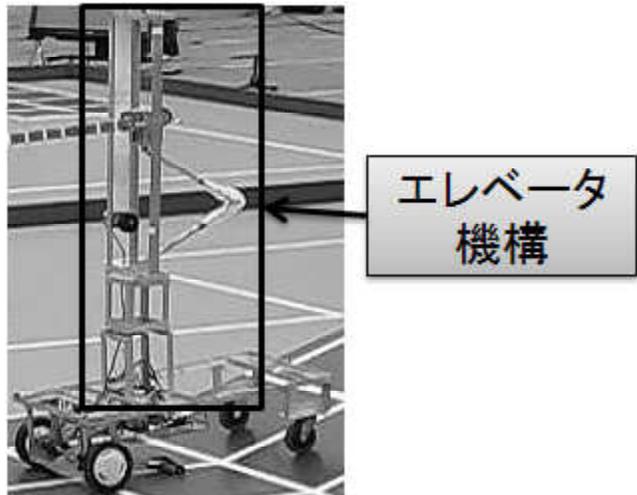
それでは、つくろぼが作成したロボットについて紹介します。

自動ロボット・手動ロボット

自動ロボットと手動ロボットの共通の特徴は、走行用モータに Maxon 社製のモータを使用していることです。1 個あたり数万円もするようなモータですが、その値段に見合った性能が得られます。

また、ロボットの頭脳と言えるマイコンには、ルネサステクノロジ社が開発している、SH7125 (SH/Tiny) を使用しました。これは、各ロボットに 2 個ずつ搭載されており、その間で通信をして協調して動作します。

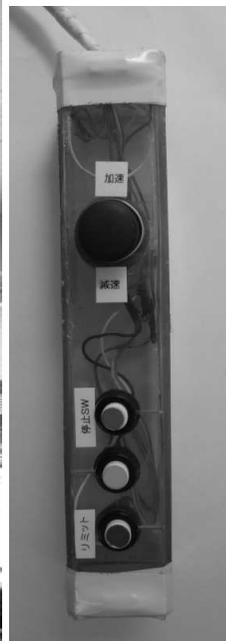
手動ロボットの特徴は、ロボットを操作するためのコントローラです。このコントローラは、かごの持ち手を上げ下げするためのエレベータ機構を制御する部分と、ロボットの移動方向を指定するための部分に分かれています。



NHK 大学ロボコン 2009 レポート

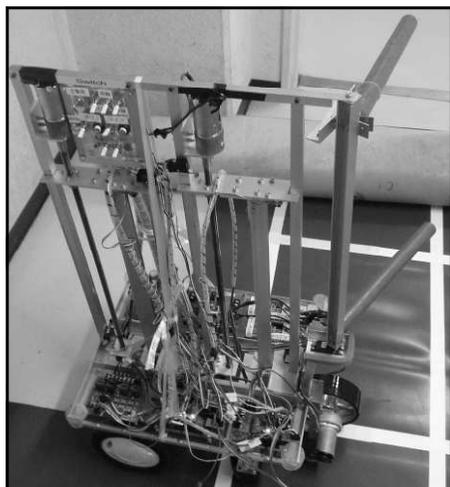
エレベータ機構の部分は、**某〇 YBERDYNE 社**のロボットスーツのように腕の部分に装着できるようになっています。コントローラを装着している腕を上下させるとその動きに合わせて、エレベータ機構が上下するようになっており、まさにかごをかついでいるような雰囲気を演出できます。

ロボットの移動方向を指定する部分は、**〇ii コントローラ**の形状をしており、スティックの部分を上下左右させることでロボットを移動させることが可能です。

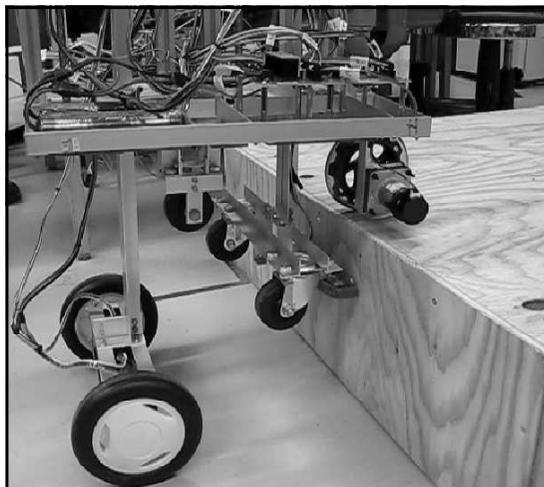


旅人ロボット

旅人ロボットの特徴は、階段を上するための機構を備えている点です。他のチームは、階段を上らない作戦を取ると予想される中、つくろぼでは、あえて階段を上らせることにしました。理由としては、あえて階段を上ることで注目を浴びよう、ということです。



旅人ロボット



階段を上る旅人ロボット

制作過程

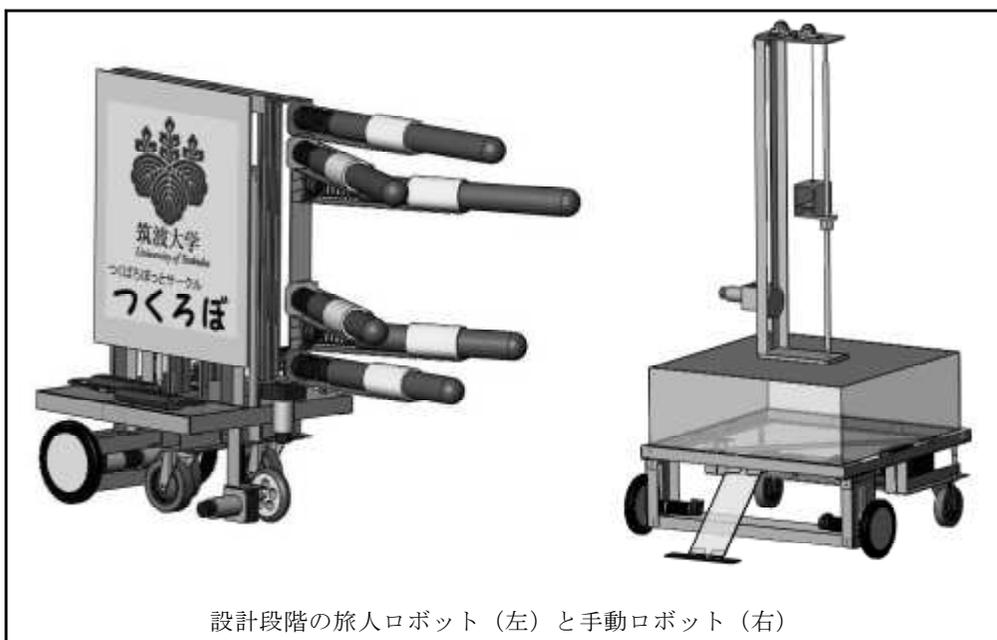
ロボットの制作は、昨年の12月から今年の6月まで、約半年にかけて行いました。1次審査のための書類を作成した期間も考えると約8ヶ月ほどになり、非常に長い時間をかけてNHKロボコンへの準備を進めてきました。

2008年10月～2008年11月：1次審査の書類作成

NHKロボコンの1次審査は、書類審査で行われます。ABUロボコンの選考会も兼ねているということもあり、9月頃にABUロボコンのページ（英語）でルールが公開された後に、NHK大学ロボコンのページ（日本語）でルールが公開されます。書類審査では、ありきたりのロボットを作るというのではなく、勝つための奇抜なアイデアや面白いロボットを求めているようです。

つくろぼは、ルールが発表されていることに気づくのが遅れて、書類を書き始めたのが10月の末でした。当初は時間がなくてまずいと思っていたのですが、応募締め切りが11月の頭から12月の頭に変更されたおかげで安堵していました。そうは言っても、筑波大学の3学期制だと、ちょうどテストの期間に重なってしまうということもあり、書類作成には、1ヶ月程度しか時間がかけられませんでした。

書類審査では、動作説明や各ロボットの仕様、アピールしたい点などの項目を記述します。その中で、自分たちの独自のアイデアを書いたり、それを実現できるかどうかなど主張していきます。



2008年12月：1次審査の結果発表

審査結果の通知は意外と早く、提出してから1週間ほどで届きました。

実はNHKロボコンの審査の中で、一番通りづらいのが1次審査だと言われています。実際、つくろぼでは毎年書類を提出しているにもかかわらず、1次審査を通過したのは3年ぶりのことでした。

1次審査の通過が決まったと同時に、ロボットの制作を始めていきました。

NHK 大学ロボコン 2009 レポート

2009年12月第2週 ～ 2009年4月第1週 : ロボット作成

ロボットの作成は、ハードウェア班、回路班、ソフトウェア班の3つの班に分かれて、作業を行いました。サークル自体の人数は15人程度であり、各班は4,5人程度でほぼ均等に分かれて作業を行っていきました。

ここでは、各班ごとの様子を見ていきましょう。

・ハードウェア班

ハードウェア班は、ロボットのフレームや各種機構の作成、組み立てを行いました。

フレームは、L字のアルミ材を加工して使用しました。アルミ材などを加工するためには、バンドソーで材料を切って、ボール盤で穴開けをして、ネジ止めてフレームを組み立てます。



バンドソー



ボール盤

ロボットのフレームの制作は、1月～3月にかけて行いました。右の画像が3月の時点での手動ロボットと自動ロボットです。6月の試合の時とは大きさや形など、全然違うものになっており、試合の時の写真と見比べてもらおうと面白いです。

3月の中頃から2次審査までは、実際のコースを作成してテストを行いました。また、このときに不都合が出てきた点について、作り直したりしました。

ハードウェア班が苦労した点は、材料の加工精度の問題です。



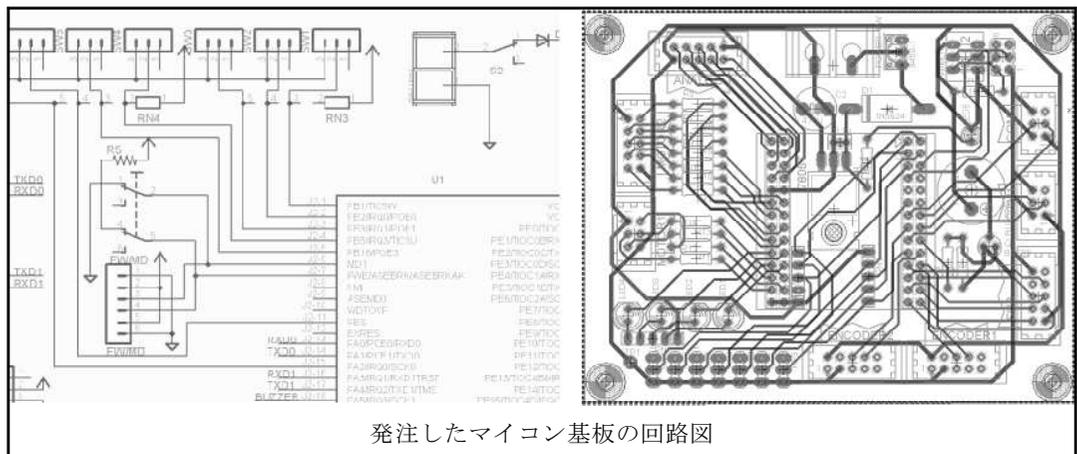
3月の時点での手動ロボットと自動ロボット

これは、加工を行う人によって精度がまちまちなので、うまくネジの穴位置や部品の寸法が合わないことがありました。うまく穴位置が合わなかったりすると、最終的にいろいろな部品を組み合わせてロボットを組み立てた時に、フレームがゆがんでしまいます。そうなると、ロボットを直進させた時に、直進しようとしているのに曲がってしまいます。^{*2}

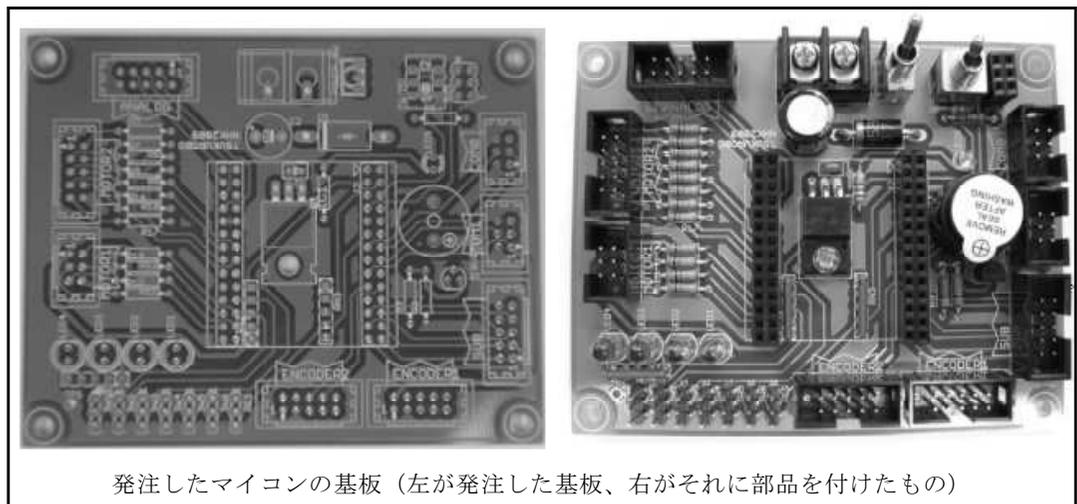
・回路班

回路班は、ロボットを動作させるために必要なマイコンやモータの回路を作成しました。また、センサの取り付けや電源周りの配線などを行いました。

今回作成したロボットでは、プリント基板としてマイコン用の基板とモータの制御回路の基板を作成してもらうために、外部の業者へ発注しました。



発注したマイコン基板の回路図



発注したマイコンの基板 (左が発注した基板、右がそれに部品を付けたもの)

また、地面にあるラインを読みとるためにフォトインタラプタという素子を使ったラインセン

*2 他大学に比べて使える加工機械が少ないため、精度を出すことが難しかったということもあります。慣れという話もあるのですが…。

抜けているためにスイッチが反応しなかったり、センサの素子が死んでしまっていたために値が正常に取れないということもありました。

全体として、3月までは2次審査に向けてのロボットを作っていました。3月の中頃にロボットの形ができあがって、ソフトウェアをロボットに入れて動かしました。4月の頭には走行する動作、エレベータ機構を上下させてかごを上げ下げするための動作、旅人ロボットが階段を上がる動作、など、個々の動作は完成していました。4月に入ってから、2次審査に向けてのプログラムを急いで仕上げ、2次審査を迎えました。

2009年4月第1週 : 2次審査

2次審査は、ビデオ審査の形式で行われ、各課題を行っているロボットを撮影して送付するという形式でした。

4月の頭の時点で個々の動作はできるようになっていたのですが、その動作をつなげる作業が遅れており、ビデオ撮影の前日から行っているような状況でした。急ピッチで作業を進めて何とか間に合わせることはできましたが、次のような問題が露呈することとなりました。

- ・乗り込みの課題でうまくロボットを乗せることができない
- ・ロボットが安定して峠を越えることができない
- ・2台でかごを持った状態で直角カーブを曲がることできない、等々

それでも何とか2次審査は通過して、NHK ロボコンに参加できることが決定しました。2次審査では、ロボットが完璧に完成しているかよりも、ロボットが動かされるかどうかを見ているようです。

2009年4月第2週 ~ 2009年5月第2週 : ロボットの作成

2次審査の時にできなかった部分を修正したり、未実装部分の実装を行ったりしました。

ちょうど、新歓やGWの時期ということもあり、このころはあまり作業がはかどっていませんでした。

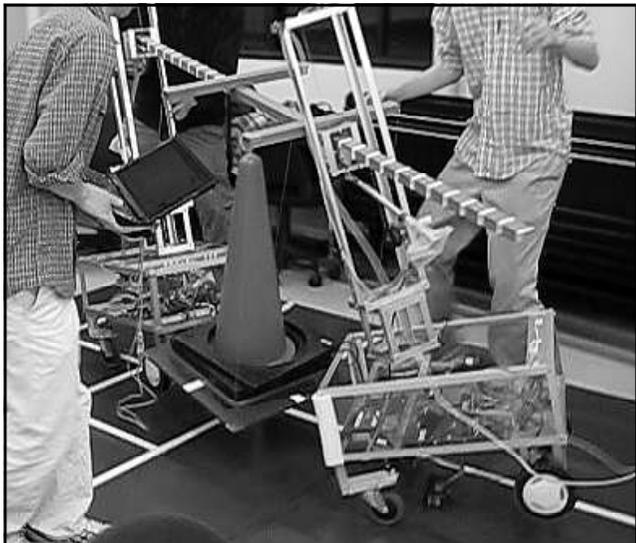
2009年5月第2週 ~ 2009年6月3日 : ロボットの調整

この段階になると、実際にコースを使ってロボットの動作を動かしたりすることが多くなり、それによって新しい問題が露呈し始めました。

一つには、旅人ロボットが重すぎるという重大な問題で、これが原因で峠の課題で制御ができなくなったりしました。その対策のために、ロボットの軽量化を行い、ロボットの様相が変わっていく日々でした。

また、自動ロボットや手動ロボットのバランスの問題により、峠の課題を練習している際にロボットが倒れてしまうことが頻発しました。^{*5}

ちょうどこの時期に新型インフルエンザが流行して、日本中がインフルエンザの話題で持ちきりの状態でした。日本中でいろいろなイベントが中止される中、インフルエンザでロボコンが無くならないかどうか不安になりながら、練習を行うのでした。



バランスを失って倒れる手動ロボット

2009年6月4日：ロボットの搬出

大会に参加するためにあたって、手でロボットを持って行くわけにはいかないので、ロボットを業者に頼んで輸送します。業者は NHK 側が手配をしてくれるので、参加者はロボットを自分たちで作った箱に箱詰めするだけです。ただし、ロボットは 3 台合わせると数十キロ近くある上に、ロボット自体の高さが 1m40cm 近くあり、非常に大きなものとなります。そのため、箱を用意するだけでも大変で、さらに、それを外に出すのでも一苦勞となります。

輸送日の前日に最後の練習を行い、午前 3 時頃、ごそごと L 棟 1 階のロビーのあたりで箱詰めを行いました。ロボットだけでなく、一緒に持って行くとかさばりそうな工具なども箱に入れました。

午前 10 時頃に業者の方がトラックで集荷に来たので、L 棟 1 階のロビーから下まで降ろして、トラックへと詰め込みました。

トラックで輸送するというのもあり、ベニヤ板に鉄のフレームを組み合わせた箱を用意しました。しかし、集荷を担当された方から「NHK ロボコンで、しっかりとした箱を作っているチームで良い成績を残せたところは、ほとんどない」とぼそりとつぶやかれ、反応に困るのでした。



L 棟の前でトラックに荷物を搬入

^{*5} 倒れてロボットが壊れることは少なかったのですが、それより、周りでロボットを支えている人への被害が大きく、手に生傷が絶えない状態でした。

こうして、ロボットはNHKロボコンの会場へと輸送されていったのでした。

2009年6月6日 12時 : 会場入り

TX に揺られて東京まで移動し、いろいろと乗り継いで会場の国立オリンピック青少年センター^{*6}へ。NHKロボコンでは、この中の体育館のうちの一つを借りて競技を行いました。

会場に入ると、既に設置されているコースがあり、NHKのスタッフの人が翌日の撮影に備えて準備を行っていました。会場には、各チームがロボットを整備するためのピットが用意されており、荷物が搬入されていました。



撮影用の大型機材

13時 : 試合順の組み合わせ

試合は、予選リーグと決勝リーグに分かれています。このうち、予選リーグは、7つのリーグに分かれた上で、3チーム（あるいは、2チーム）での総当たり戦で行われます。それぞれのリーグの中で勝った回数が一番多いチームが決勝リーグに進むことができます。また、全てのリーグの中の2位以下のチームの中で成績が一番良かったチームが1チームだけ、決勝リーグへ進むことができます。合計で、8チームが決勝リーグへと進むことができます。

組み合わせの抽選を終えたところ、次の結果となりました。

^{*6}1964年に開催された東京オリンピックの選手村の跡地にあり、体育館やプールなど各種の競技場を備えたスポーツ施設です。

NHK 大学ロボコン 2009 レポート

Aグループ	第1試合	第2試合	第3試合
九州大学			
工学院大学			
東京工科大学			

Eグループ	第1試合	第2試合	第3試合
信州大学			
長岡技術科学大学			
鹿児島大学			

Bグループ	第1試合	第2試合	第3試合
秋田県立大学			
ものづくり大学			
電気通信大学			

Fグループ	第1試合	第2試合	第3試合
静岡理工科大学			
東京大学			
長崎総合科学大学			

Cグループ	第1試合	第2試合	第3試合
筑波大学			
豊橋技術科学大学			
京都工芸繊維大学			

Gグループ	第1試合	第2試合
岡山大学		
三重大学		

Dグループ	第1試合	第2試合	第3試合
湘南工科大学			
金沢工業大学			
高知工科大学			

予選リーグ 組み合わせの結果

つくろぼはというと、なんと初戦から昨年の優勝校である豊橋技術科学大学（以下、豊橋科大）とあたってしまいました。⁷

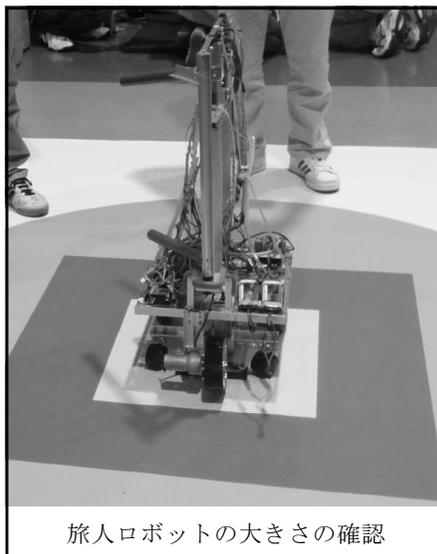
13時30分：ロボットの検査

ロボットの調整が終わると、NHK ロボコンのルールに沿った機体であるかどうかの検査が行われます。検査は、大きさや重さ、電源電圧などの項目を計測していました。

実際に検査を行ったところ、重さは、予備のバッテリーをあわせて 35.75kg でした。大体、旅人ロボットが 9kg 程度で、自動ロボットと手動ロボットは十数 kg 程度の重さでした。

周りのロボットを見ている感じだと、つくろぼのロボットは全体の中で見ても、重めなロボットでした。おそらく、他のところだと半分以下ぐらいの重量でした。

また、電源電圧については、どのようにチェックを行うのかと思ったら、実際にスタッフの方がテスターを持って電圧をチェックしていました。



旅人ロボットの大きさの確認

⁷ 個人的には、豊橋市出身の人間なので「何という巡り合わせ…。」といった感じでした。

14時～18時20分：試走

検査を終えたチームから、試走に入っていきます。本番まで、この時間のみがコースを使って確認を行える時間なので、最終チェックの時間となります。

他の大学のチームもどんどん試走を行っていき、大体、どのチームがどれくらいの速度を出すのか、この時点ではっきりとしました。

2009年6月7日：開会式

とうとう、NHK ロボコンの大会が始まりました。何台ものカメラで撮影されながら、入場をしていきます。



第1戦 筑波大学 vs 豊橋技術科学大学

初戦から昨年の優勝校との試合でした。

試合の前に1分間のセッティングを行い、チームメンバーの各人が所定の位置につきます。

審判のスタートの合図と同時に両方のロボットが動き始めました。試合は3分間で各課題を順番にこなして、先に太鼓を叩いたチームが勝ちとなります。

スタート早々、どちらのチームも乗り込みに失敗し、審判からリトライを宣言され、乗り込みの課題をやり直しました。

スタートエリアからやり直して、ほぼ、同じタイミングで両者のロボットの乗り込みが完了し、峠の課題へと向かっていきます。しかし、それ以降は、さすが昨年の優勝校ということもあり、次々と課題をクリアし、あっという間に太鼓を叩いてしまいました。

一方その頃つくろぼは、旅人ロボットの乗り込み位置が悪かったためか、非常に不安定な状態で峠に差し掛かっていきました。しかし、バランスの悪い状態では峠を越えることはできず、途中で旅人ロボットを落としてしまいました。

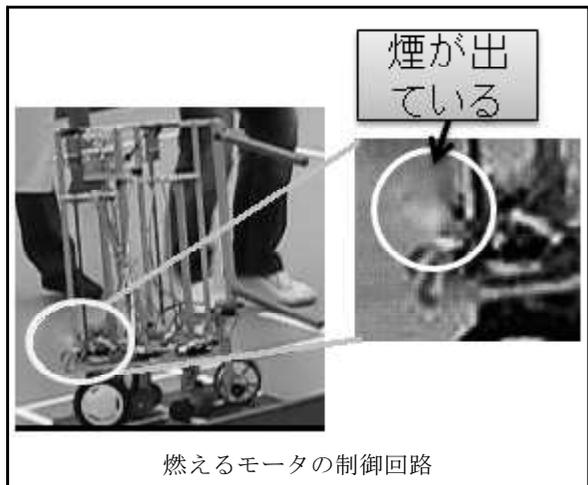
審判からリトライを宣言され、再度、峠の課題から行おうとしましたが、そのとき、審判が慌てながら「煙が出てないか!」と言いました。どうやら、落ちた拍子に触れてはいけないところが触れてしまったらしく、モータの制御を行っ



NHK 大学ロボコン 2009 レポート

ている回路の部分が燃えてしまいました。

あれこれ考えている時間はなかった*8 ので、モータを制御している回路の部分だけ電源を落として、せめて進めるところまで進むことにしました。しかし、途中で豊橋技科大に太鼓を叩かれ、あえなく競技は終わってしまいました。



第2戦 筑波大学 vs 京都工芸繊維大学

モータの制御回路が燃えてしまったので第2戦。ただし、**想定の範囲内**だったので、回路を交換するだけで何とかなりました。

第2戦の京都工芸繊維大学は、前日の試走を見ている感じだと、何とかがんばれば勝てないこともないという印象でした。せめて、**これだけは勝ちたい**という思いでの第2戦です。

スタート開始後、やはり、乗り込みの課題で失敗してしまいました。2度ほどリトライをして、何とか乗り込むことに成功。同じぐらいのタイミングで相手チームも乗り込みの課題が完了して、峠の課題へと差し掛かりました。

峠の課題は、操縦者の腕の動きに合わせてかごが上下する $H \ominus L$ のようなコントローラを駆使し、かごの動きを制御しながらゆっくりと進んでいきます。「これはいけるんじゃないか」と内心思ったのですが、かごがロボットフレームにあたってしまったらしく、審判にリトライを宣言されてしまいました。その間に、相手のチームに峠の課題をクリアされ、林の課題へと向かわれてしまいました。そこでタイムリミットになり、試合は終わってしまいました。

結局、つくろぼは1勝もできずに、6ヶ月間のロボット制作に幕を閉じました。

*8 もちろん、競技中に回路を変更することはできません。

全ての試合の結果

最終的な結果としては、以下の結果となりました。

Aグループ	第1試合	第2試合	第3試合
九州大学	×	○	/
工学院大学	○	/	×
東京工科大学	/	×	○

Eグループ	第1試合	第2試合	第3試合
信州大学	×	×	/
長岡技術科学大学	○	/	○
鹿児島大学	/	○	×

Bグループ	第1試合	第2試合	第3試合
秋田県立大学	×	×	/
ものつくり大学	○	/	×
電気通信大学	/	○	○

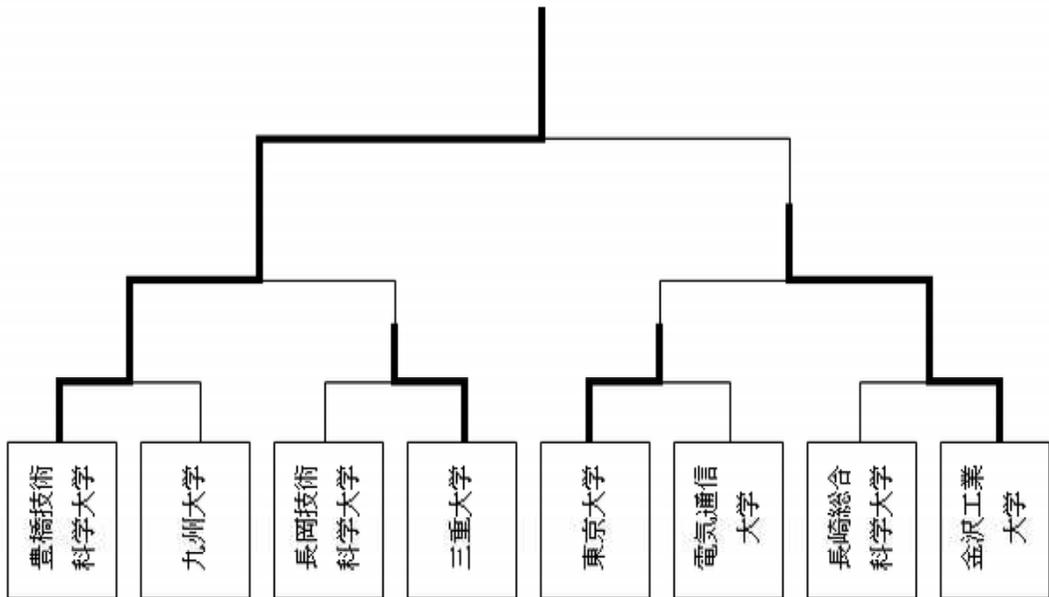
Fグループ	第1試合	第2試合	第3試合
静岡理工科大学	×	×	/
東京大学	○	/	○
長崎総合科学大学	/	○	×

Cグループ	第1試合	第2試合	第3試合
筑波大学	×	×	/
豊橋技術科学大学	○	/	○
京都工芸繊維大学	/	○	×

Gグループ	第1試合	第2試合
岡山大学	×	×
三重大学	○	○

Dグループ	第1試合	第2試合	第3試合
湘南工科大学	×	×	/
金沢工業大学	○	/	○
高知工科大学	/	○	×

予選リーグ 試合結果



決勝トーナメント 試合結果

昨年度優勝した豊橋技科大は、今年も優勝して ABU ロボコンへの出場権を獲得しました。また、2位には金沢工業大学が入り、こちらも ABU ロボコンへの出場権を獲得しました。

詳しくは述べませんが、他のチームでもいろいろなドラマがありました。特に、東京大学と金沢工業大学の試合では、ABU ロボコンへの出場権をかけた戦いとなり、いろいろな思いがぶつかり合ったと言えるでしょう。

NHK 大学ロボコン 2009 レポート

これらの様子については、NHK 総合テレビにて以下の日程で放映されます。また、全ての試合の様子を放映する番組は、別途 9 月に放映される予定だそうです。こちらについては、NHK 大学ロボコン 2009 のページ⁹⁾をご覧ください。

NHK 総合テレビ 2009 年 7 月 20 日 (月・祝) 13 時 5 分 ~ 13 時 59 分

授賞式

最後に、授賞式がありました。優勝したチームや技術賞のチームなど次々と表彰されていきました。

その中で、つくろぼは特別賞に選ばれました。受賞の理由は、操縦者の腕の動きに合わせてかごが上下するコントローラが審査員の方の目に留まり、その点を評価していただきました。

最後に

私にとって、最初で最後の NHK ロボコンが終わりました。元々、大学に入る前はロボットをやるなんて全く考えていなかったのですが、いつの間にか、このロボコンに出ることになっていました。ロボットを作成する作業を通して学んだことなど、良い経験になったと思います。

また、つくろぼとしては、今後とも NHK ロボコンに向けて努力していくことと思います。次回以降、再度、NHK ロボコンに出場することになった時には、是非、応援していただければと思います。



この記事に関して、疑問や質問などありましたら、以下のメールアドレスまでご連絡ください。

メールアドレス : word@yasuharu.net

⁹⁾ <http://www.official-robocon.com/jp/daigaku/daigaku2009/>

徹夜の友 ～コーヒー飲み比べ～

文 かづきお改め十七夜月

良いコーヒーとは
悪魔のように黒く 地獄のように熱く
天使のように純粋で 愛のように甘い

シャルル=モーリス・ド・タレーラン=ペリゴール伯爵

皆さん、こんにちは、お久しぶり、初めまして。

新年度が始まって二ヶ月が経ちましたが、皆さんはいかがお過ごしでしょうか。新入生にとって初めての期末テストまで、あとわずかとなりました。

大学生のテスト対策と言えば一夜漬け。そして一夜漬けと言えばコーヒーですよ。たっぷりのカフェインにより頭がすっきり。これで勝つ。

まあ私などは恒常的にコーヒーを飲んでいるわけなのですが、一人暮らしですとレギュラーコーヒーを豆からひいて……、という事にはなかなかならないと思います。しかし、世の中にはインスタントコーヒーという便利な代物が存在します。粉にお湯を注ぐだけ。三分間待つ必要もありません。

かく言う私も普段はインスタントコーヒーで済ましています。普段はというのは、実家に帰ればコーヒーメーカーが有りますし、大学では三食のパン屋さんのコーヒーを飲みます。缶コーヒーはよっぽどの事情がない限り飲みません。

インスタントコーヒーと一口に言っても、その数はインスタントラーメンほどではないにせよかなりの種類が有ります。初めてインスタントコーヒーを買う人はもちろんのこと、普段あまりコーヒーを飲まない人には選択肢が多すぎて何を選べばいいのか分からないのではないかと思います。

そこで今回私が「カワチで買うことができるインスタントコーヒー」をテーマに主観的な観点から飲み比べをしてみました。私自身もこうして複数の種類のコーヒーを買ってきて飲み比べる事自体初めてですので、非常に楽しみです。

味というものはかなりの主観が入るものなので私と異なった感想を持たれる方も多いと思いますが、少しでも皆さんのためになれば幸いです。

徹夜の友

選手紹介

四月某日、桜にあるカワチに行き、インスタントコーヒーの棚にあったモノを一種類ずつ全て買ってきました。



ただしこの中に、一人だけ異質な戦士がいます。皆さん、わかりますでしょうか。

後ろの列左から

- ・香味焙煎
- ・香味焙煎 柔らかモカブレンド
- ・香味焙煎 グアテマラ SHB
- ・MAXIM 華やかな香りブレンド
- ・KeyCoffee スペシャルブレンド

中段左から

- ・MAXIM 炭火焙煎
- ・AGF MAXIM
- ・AGF Blendy
- ・NESCAFE Excella
- ・MAXIM ちょっと贅沢な珈琲店

最前列左から

- ・香味焙煎 さわやかキリマンジャロブレンド
- ・NESCAFE PRESIDENT
- ・NESCAFE GOLDBLEND。

そして、最前列右端にいるのが NESCAFE FRAGILE。この選手だけは筆者推薦で参戦させてい

ます。素性を述べると、ネスカフェが毎年期間限定で秋から冬にかけて販売しているインスタントコーヒーです。最初のうちはネスカフェのメール会員限定販売だったのですが、最近は少数ですが店頭にも並びますし、最近は Amazon でも手に入ります。カワチで買ったものではありませんが、筆者の家で大量に備蓄されていたので参戦させました。

方法

評価方法は、香り、甘み、苦み、酸味、コク、の5つの観点から行いました。それぞれ5段階評価で、★の数が多いほど強いという事になります。また、その他に飲んだときの感想を載せてあります。飲み方は、コーヒースプーンに1杯の粉に、沸騰したてのお湯を注いで飲みました。1杯ごとに水を飲んで口の中をリセットさせています。



名前：MAXIM 炭火焙煎

グラム単価：7円

評価

香り：★★☆☆☆

甘み：★★★★☆

苦み：★★★★☆

酸味：★★★☆☆

コク：★★☆☆☆

感想

香りは焙煎の良さが目立つ。ほのかな甘い香りと少々の酸味も特徴。

あっさりとして飲みやすいがやや苦みが残る。



名前：AGF MAXIM

グラム単価：7円

評価

香り：★★★★☆

甘み：★★☆☆☆

苦み：★★★★☆

酸味：★★☆☆☆

コク：★★☆☆☆

感想

甘い香りが特徴的。逆に飲んでみるとあっさりしていて、飲みやすい。



名前：MAXIM ちょっと贅沢な珈琲店

グラム単価：8円

評価

香り：★★★★☆

甘み：★★☆☆☆

苦み：★★★★☆

酸味：★★★★☆

コク：★★★★☆

感想

焙煎の香りが強いが、炭火焙煎と違い上品な香り。

苦みが強く、口に残るテイスト。



名前：MAXIM 華やかな香りブレンド

グラム単価：3円

評価

香り：★★★★☆☆

甘み：★★☆☆☆☆

苦み：★★☆☆☆☆

酸味：★★☆☆☆☆

コク：★★☆☆☆☆

感想

香りは良いが、あっさりしすぎてコクがない。甘みも弱い。



名前：AGF Blendy

グラム単価：5円

評価

香り：★★★★★☆☆

甘み：★★☆☆☆☆

苦み：★★★★★☆☆

酸味：★★★★☆☆

コク：★★★★☆☆

感想

同じ AGF でも MAXIM とは一線を画すガッツのある香り。口当たりが良くマイルドで飲みやすいがコクもそれなりにある。



名前：KEYCOFFEE スペシャルブレンド

グラム単価：4円

評価

香り：★★★★☆

甘み：★★☆☆☆

苦み：★★★★☆

酸味：★★★★☆

コク：★★☆☆☆

感想

上品だが強い香り。まろやかな味の中で少しだけ利いている酸味が印象的。



名前：NESCAFÉ Excella

グラム単価：6円

評価

香り：★★☆☆☆

甘み：★★☆☆☆

苦み：★★★★☆

酸味：★★★★☆

コク：★★☆☆☆

感想

匂いは強いが、独特で好みの分かれるところ。

すっきり傾向だが苦みが若干残る。



名前：NESCAFÉ GOLDBLEND

グラム単価：9円

評価

香り：★★★★☆

甘み：★★★★☆

苦み：★★★☆☆

酸味：★★★★☆☆

コク：★★★★☆

感想

香りは強くないが甘く上品。甘みと酸味が利いている。



名前：NESCAFÉ PRESIDENT

グラム単価：14円

評価

香り：★★★★☆☆

甘み：★★★★☆☆

苦み：★★★★☆☆

酸味：★★☆☆☆☆

コク：★★★★☆☆

感想

強くないがしっかりした香り。すっきりと飲みやすいが苦みは強い。



名前：NESCAFÉ 香味焙煎 爽やかキリマンジャロブレンド

グラム単価：10円

評価

香り：★★★★☆☆

甘み：★★★★☆☆

苦み：★★★★☆☆

酸味：★★★★☆☆

コク：★★★★☆☆

感想

香りは控えめ。苦みと酸味が強いが、口に残るような感じはない。



名前：NESCAFÉ 香味焙煎

グラム単価：9円

評価

香り：★★★★☆☆

甘み：★★★★☆☆

苦み：★★★★☆☆

酸味：★★★★☆☆

コク：★★★★☆☆

感想

香りが強いというより「重い」という表現が当てはまる。コクがありバランスがよい。



名前：NESCAFÉ 香味焙煎 グアテマラ SHB

グラム単価：13 円

評価

香り：★★★★☆

甘み：★★☆☆☆

苦み：★★★★☆

酸味：★★★★☆

コク：★★★★☆

感想

香りが強いというより「重い」。
苦みが強いが口あたりはよい。



名前：NESCAFÉ 香味焙煎 柔らかモカブレンド

グラム単価：9 円

評価

香り：★★★★★

甘み：★★★★★

苦み：★★☆☆☆

酸味：★★★★☆

コク：★★★★☆

感想

とても甘い香りがする。飲んでみても甘みが強く苦みや酸味は僅か。

徹夜の友



名前：NESCAFÉ FRAGILE

グラム単価：21 円

評価

香り：★★★★★

甘み：★★★★☆

苦み：★★★☆☆

酸味：★★★★☆

コク：★★☆☆☆

感想

香りはエントリー中最も甘く強い。甘みが強いあっさり系。コクはそこそこ。

シチュエーションコーヒー

・徹夜の友：NESCAFÉ 香味焙煎 爽やかキリマンジャロブレンド

徹夜で試験勉強をがんばるあなたには爽やかキリマンジャロブレンドがお勧め。強い苦みが眠気を一扫。口に残るような感じが無いのも Good。

・読書の友：NESCAFÉ GOLDBLEND

読書や音楽鑑賞など趣味を楽しむあなたには GOLDBLEND がお勧め。上品な香りが至高の時間を演出してくれます。

・接客の友：NESCAFÉ FRAGILE

お客さんがきたときは迷わずこれ。独特の甘い香りでインスタントとは思われないかも。香りに反してきつい我が無く飲みやすいのでコーヒーが苦手な人にもお勧めです。

インスタントコーヒーの美味しい飲み方

インスタントコーヒーを淹れるときのポイントは、ずばり湯の温度です。コーヒーは基本的に淹れる温度が高ければ高いほど苦みが増す傾向にあります。これはレギュラーコーヒーの話なのですが、インスタントでも同じような傾向があります。さらに、コーヒーは時間がたつにつれ、ものすごい勢いで酸化していくためどんどん不味くなります。

お湯を沸かしたら、慌てて注がず、ある程度の間を置いてから淹れましょう。もちろんその間に粉をカップに注いでおくことはしないように。酸化してしまいます。

おわりに

今回初めてコーヒーを飲み比べてみて、思っていたよりそれぞれのコーヒーの個性に驚いています。たかがインスタント、されどインスタント。どうせ飲むのなら自分好みのおいしいコーヒーを飲みましょう。

それにしても、香りや味の表現は難しいです。読者の方にうまく伝わってくれるといいのですが。どうやら私は料理評論家にはなれなさそうです。

冒頭でも少し書きましたが、レギュラーコーヒーを飲みたくなったら三食のパン屋さんのコーヒーがおすすめです。三食と共通の食券で1杯100円で飲むことができます。これが個人的には大学内では1番美味しいと思います。三学書籍部前の自販機にはたまに行列を作っている人々を見かけますが、パン屋さんのコーヒーで列を作っている人は見かけません。穴場なのか私の嗜好が大多数の人と異なっているのかは定かではありませんが。

え？ MAX コーヒーはどうしたかって？

否！ 断じて否！

あんなものはコーヒーとは認めぬ！

あれは加糖練乳の親戚である。

コーヒーでは断じてない！

F#入門

文 編集部 (suma)

.NET で動作する F#という言語の概要や特徴、F#を使う利点、処理系の導入の仕方をぬるめに解説したいと思います。

はじめに

F#は Microsoft が関数型言語の OCaml を .NET 向けに拡張したプログラミング言語です。

今回は言語仕様よりも言語が持つ機能を中心に説明します。C/C++や Perl/Python/Ruby には慣れているけど、関数型言語の概念を知らないという方でも理解できるように F#について説明していきます。最後に、F#や OCaml を入門するのに適したウェブサイトを紹介합니다。

それでは、最初に OCaml とは何であるか説明し、続いて F#の特徴、コンパイラのインストールについて述べていきます。

OCaml と関数型言語

F#の説明に入る前に、まずは OCaml という言語について説明しましょう。OCaml は Objective Caml の略で、ML (Meta Language) という関数型言語の方言です。フランスの INRIA (フランス国立情報学自動制御研究所) が開発し、その処理系にはネイティブコンパイラとバイトコードコンパイラ、バイトコードインタプリタの 3 種類があります。

関数型言語

OCaml 以外の関数型言語として有名なものに Lisp や Scheme、分散処理に向いていると取り上げられる Erlang や Scala、純粋関数型言語の Haskell などがあります。どんな機能を持っていれば関数型言語なのか、という定義は持ち出しませんが、およそこれらの言語が関数型言語と呼ばれています。

OCaml は強い静的型付けのある言語で、型推論という機能を持っています。

強い型付けを持つ OCaml は、動的言語 (Perl や Ruby) などと違って変数や関数の返り値に型が存在します。しかも、C 言語などとは違って自動的に型が変化されること (暗黙の型変換) が許されていません^{*1}。そして、静的型付けというのはコンパイル時に型のチェックが実行されることを指します。

これによって、C/C++、C#、Java などの言語に比べてコンパイルが通りにくいものの、実行時の型エラーを完全になくすことができます。

型推論というのは、変数の型を自動的に推論してコードを解釈する機能です。例えば、if という構文に渡す値は bool 型と決まっているため、型が明示されてない変数が if に渡されても、自

*1 例えば C 言語では、int の数値から double の浮動小数点を足すような演算をすると、大きい方の型である double へ型が自動的に変更されてしまいます。強い型付けを持つ言語では、自動的に型が変換されることはないため、プログラマが手動で型を変換する関数を呼び出すことになります。

動的に `bool` という型を推論します。

この 2 つの特徴は OCaml に限ったことではなく、Haskell も同様の機能を持っています。他にも OCaml は以下の特徴を持っています。少し細かすぎたので OCaml (関数型言語) の概要はこれくらいにしましょう。

- 値の代入ができない
- 多相型
- パターンマッチ

参考になるかわかりませんが、OCaml でパターンマッチを使ってフィボナッチ数を再帰的に求める関数を作ると以下のように書くことができます。

```
let rec fib n =
  match n with
  | 0 -> 1
  | 1 -> 1
  | x -> fib (x-1) + fib (x-2)
```

Real World OCaml

OCaml 概要の最後として、関数型言語だと何が嬉しいのか簡単に説明してみましょう。

OCaml のような言語が得意なプログラムの 1 つとして、言語処理系 (コンパイラ・インタプリタ) があります。言語処理系は、プログラムのコード (文字列) を字句解析・構文解析してから、木構造 (普通のコンパイラは構文木と呼ばれる木構造を作る) でプログラム内で読み込んだコードを表現します。関数型言語の持つ、パターンマッチや多相型といった構文を使うことで、C/C++ や C#/Java などデータ構造を扱うよりも簡潔なコードを書くことができます。

その他、グラフや何かのモデルのような構造を表現して内部で実装するときは、関数型言語の方が操作を扱いやすいといったケースが存在すると思います。

最後に、OCaml が実際に利用されているプロジェクトを以下にあげてみます。

- OS 仮想化ソフトウェアである Xen の管理ツール
Xen And the Art of OCaml (Anil Madhavapeddy)
<http://cufp.galois.com/2008/slides/MadhavapeddyAnil.pdf>
- 産総研 RCIS Fail-Safe C
Fail-Safe C
<http://www.rcis.aist.go.jp/project/FailSafeC-ja.html>

Xen は 2 年ほど前から管理ツールの実装にも OCaml を利用し始めたそうです。関数型言語は言語処理系の実装に向いており、産総研の Fail-Safe C というプロジェクトに利用されています。どちらのプロジェクトもたくさんのソースコードが公開されているため、興味があればソースコー

F#入門

ドを参照してみてもよいでしょう。

F#概要

F#は OCaml を拡張した言語であると最初に述べました。ここからは、F#が具体的にどんな機能を持っているのか説明します。

lightweight syntax option

F#には OCaml の文法を拡張した、light シンタックスというオプションがあります。light シンタックスを使うと、いくつかの構文で必要なキーワードを省略できます。そのかわり、OCaml と文法上の互換性がなくなってしまいます。F#の仕様を記したドラフトや、出版されている書籍では light シンタックスでコードで記述しているらしいので、人が書いたコードを参考にするときは気を付けるとよいでしょう。

light シンタックスでコードを記述する時はコード内に「#light」を記述します。参考までに、通常の OCaml 互換のコードと、F#の light シンタックスで書いたコードを F#のドラフトから引用して載せておきます。

式ごとに区切り文字";;"の省略（左が light シンタックス、右が省略無し）

```
#light                                printf "Hello";;
printf "Hello"                        printf "World";;
printf "World"
```

let の後の in の省略（左が light シンタックス、右が省略無し）

```
#light                                let SimpleSample () =
let SimpleSample () =                  let x = 10 + 12 - 3 in
    let x = 10 + 12 - 3                 let y = x * 2 + 1 in
    let y = x * 2 + 1                   let r1,r2 = x/3, x%3 in
    let r1,r2 = x/3, x%3                 (x,y,r1,r2)
    (x,y,r1,r2)
```

「15.1.1 Basic #light rules by example.」より

http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/manual/spec2.aspx#_Toc207785554

.NET Framework への対応

F#は OCaml を .NET 向けに拡張しただけあって、.NET Framework を扱う機能は充実しています。「C#で .NET Framework を扱う機能すべてを F#が持っている」という記述は見られませんが、F#は .NET Framework を扱う多くの機能に対応していると考えてよいでしょう。

F#のドラフトから、いくつかの項目を抜粋すると以下のような機能を F#が対応していると確認できます。

- ・クラス、インタフェース、構造体
- ・クラスメンバの宣言
- ・オブジェクトのコンストラクタ
- ・.NET の CLR (Common Language Runtime) の型システムのためのシグネチャ
- ・デリゲート

軽くドラフトを軽く見たところ、.NET Framework を F#から利用することに特に不便はなさそうな印象を受けました。しかし、F#の言語仕様はドラフト（草案）の段階であり、今後仕様が変わることも考えられます。とりあえず、Visual Studio 2010 で F#が搭載されることは、Visual Studio 2010 Beta 1 に F#が搭載されていることからほぼ決定と考えてもよいでしょう。

F#を使う環境を整える

コードは一切なしで、処理系の導入方法だけ書いておきます。

処理系を導入する方法

F#処理系を利用するには、以下の3つの方法があります。

1. コマンドラインツールを使う（Mono を使えば Mac, Linux などでも実行可能です）
2. CTP (Community Technology Preview) 版を Visual Studio 2008 にインストールして使う
3. Visual Studio 2010 Beta 1 を使う

1. コマンドラインから使う

非 Windows 環境向けに、コマンドラインコンパイラ・ライブラリなどの実行に必要なファイルがパッケージ化されて配布されています。

F#の処理系は.NET Framework 互換の Mono を使うことで、Windows 以外の環境である Mac や Linux などから使うことができます。

2, 3. Visual Studio に統合したものを使う

CTP 版もしくは Visual Studio 2010 Beta 1 をインストールすると、Visual Studio に F#を統合した状態で使えるようになります。Visual Studio に統合することで、コンパイラ・デバッガだけでなくエディタのシンタックスハイライト（色分け）などもの機能が利用できるようになります。Visual Studio で C/C++, C#と同じようにインテリセンス（エディタの補完機能）の恩恵を受けることができます。

F#処理系や Visual Studio 2010 Beta 1 は以下のサイトからたどればパッケージをダウンロードできます（執筆現時点では Visual Studio 2010 Beta 1 は配布を続けています）。

F# - Microsoft Research

<http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/default.aspx>

Microsoft Visual Studio 2010

<http://www.microsoft.com/japan/msdn/vstudio/2010/>

F#入門

まとめ

以上、入門というほど踏み込んでいない「F#入門」でした。この記事を読んで関数型言語や F# について興味を持っていただけたら幸いです。次回、続くことがあれば F# でコードを書く段階まで踏み込んでみたいと思います。

参考科目

情報・情報科学類の 2 学期には「宣言型プログラム論」という科目があります。独学で学ぶよりも丁寧に ML 系の言語を学習したい方は履修を検討してはいかがでしょうか。標準履修年次が 3,4 年生となっていますが、プログラミングに自信のある方は 1 年生でも大丈夫だと思います（もちろん単位が取れることを保証するわけではありません）。

宣言型プログラム論 (Declarative Programming)

http://www.coins.tsukuba.ac.jp/syllabus/GB21204_L410304.html

参考文献

Objective Caml

<http://caml.inria.fr/ocaml/index.en.html>

F# Microsoft Research

<http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/>

The F# 1.9.6 Draft Language Specification

<http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/manual/spec2.aspx>

F#による関数型プログラミング入門

<http://lowlander256.web.fc2.com/fsharp/>

OCaml.jp

<http://ocaml.jp/>

Objective Caml 入門

<http://www.sato.kuis.kyoto-u.ac.jp/~igarashi/class/isle4/mltext/ocaml.html>

Functional Programming - お気楽 OCaml プログラミング入門/M.Hiroi's Home Page

http://www.geocities.jp/m_hiroi/func/index.html#ocaml

入門 Ruby on Rails Vol.2

文 編集部 いのひろ

こんにちは

さて、前回は準備編としたので、さっそく作っていきたいと思うところですが、その前に Rails を構成する 2 つの重要な要素を説明しておかないといけません。

- MVC アーキテクチャ
- REST

です。

MVC アーキテクチャ

ご存知の方も多いかと思いますが、MVC アーキテクチャとはソフトウェアアーキテクチャの一つで、ユーザインタフェースを伴うアプリケーションを効率的に開発する為のものです。MVC とはそれぞれ「Model」「View」「Controller」のことです。Rails はこの MVC アーキテクチャを採用しています。

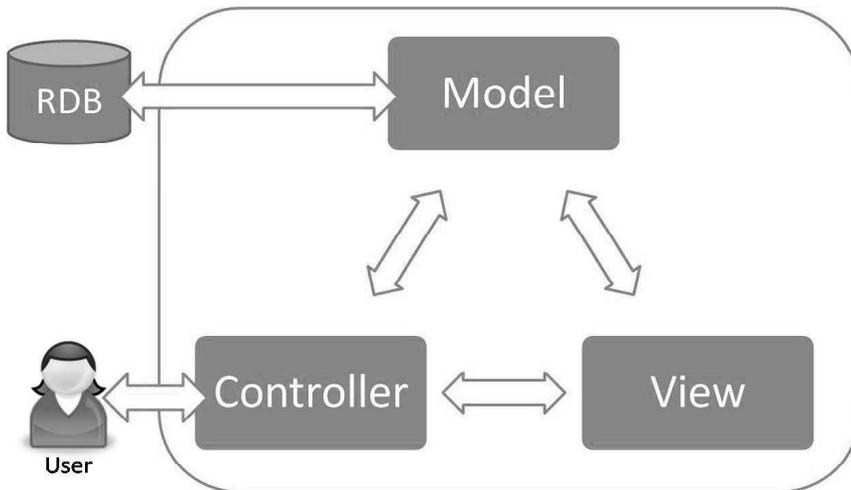
Model はアプリケーションが取り扱うデータオブジェクトとそれに対するメソッドを担当します。一つの Model クラスは ActiveRecord というコアライブラリ^{*1}を継承したもので、前号で書いた通りこの ActiveRecord はオブジェクト関係マッピング機能を備えています。従ってデータベース上のテーブルが各 Model と対応し、この Model に対する操作がデータベースに対する処理になります。

View はユーザインタフェースを担当し、ユーザーに向けた XML や HTML を Model から動的に生成します。PHP のように HTML/XML ファイルに Ruby のコードを埋め込みます。

Controller は View からユーザーのアクションを受け取り、Model のメソッドを呼んだり、View を表示させたり、別の View へリダイレクトさせたりします。但し Model の値を直接操作したりするわけではありません。

文字で書くと少し複雑な感じがしますが、処理の棲み分けを行っているわけです。

*1ActiveRecord とはそもそも、データベースからのデータの読み込みをプログラミング言語側から (SQL などの問い合わせ言語を用いずに) 行う為のデザインパターンであり、Rails における ActiveRecord はそのデザインパターンの Ruby 実装です。紛らわしいですね。(´・ω・｀)



上の図を見ると少しおかしいと感じるかもしれません。View は UI を担当すると書いたのに、実際に矢印が伸びているのは Controller だからです。実はすべてのリクエストは Web サーバーを通じてまず最初に Controller が処理します。Controller がリクエストによって View を返すか、何もしないかなど判断するわけです。

REST

Rails アプリケーションを作る上でもう一つ重要なものが「REST」です。REST とは「REpresentational State Transfer」の略で、Web におけるソフトウェアアーキテクチャのスタイル¹の事なのですが、これが概念的でなかなか難しい(´・ω・`)。なんとかわかりやすく書こうと思いますが、REST だけで記事を書けそうなくらいです。

リソースと URI

REST の考え方の中心となるのは「リソース (資源)」です。Web 上のすべてのオブジェクト (テキスト、HTML、XML、静止画、動画、音声など) はすべてリソースであり²、URI³によって識別できる／できるべきであるというものです。なんか難しく聞こえるかもしれませんが、「Web 上の“もの”は URI で指し示す事ができる (できるようにしよう、できるべきである)」という考えです。

*1 ソフトウェアアーキテクチャではなくそのスタイルです。さらに一つ上のレイヤです。

*2 逆に URI を定める事が出来るものがリソースであると考え事も出来ます。

*3 Uniform Resource Identifier/統一資源識別子。Identifier からわかるようにリソースの「識別子」になるものです。よく聞かれる URL (Uniform Resource Locator) は URI のサブセットであり、同じく URI のサブセットに URN (Uniform Resource Name) という物もあります。URI を拡張した物に IRI (Internationalized Resource Identifier; 国際化資源識別子) なんて言うのもあり、文字コードとして Unicode が指定され、日本語や中国語などを URI として利用できるようになります。

以下にいくつかのリソースとその URI を示します。

リソース：筑波大学の天気 (livedoor 天気情報)
URI : http://weather.livedoor.com/point/univ/8/107.html
リソース：CentOS 5.3 の DVD イメージファイル (ftp.riken.jp によるミラー)
URI : http://ftp.riken.jp/Linux/centos/5.3/isos/i386/CentOS-5.3-i386-bin-DVD.iso
リソース：XML 1.0 の仕様書 (W3C)
URI : http://www.w3.org/TR/xml/

リソースに名前 (URI) を付けられると何が嬉しいのでしょうか。URI はリソースへの参照です。例えばチャットなどでコミュニケーションしているとき、友人に動画を見せたいと思ってもすぐに見せることは困難ですね。しかし YouTube にある動画の URI 等を伝えることで簡単に見せてあげることができます。URI すばらしい(*´д`*)ハアアア。

さらに URI はアドレス可能性 (Addressability) という性質があります。データの集合から条件にあう部分集合を URI で定義が可能であるということです^{*1}。たとえば

Google で「筑波大学」と検索した上位 50 件

というリソースの URI は、

http://www.google.co.jp/search?as_q=筑波大学&num=50

と表現することができます。これをブラウザから開いてみれば、確かに Google で「筑波大学」と検索した上位 50 個の結果であることがわかります。また検索結果は時々刻々と変化するものですが、URI を用いればこのような抽象的なオブジェクト (データの集合) に対しても識別可能な ID (URI) を振ることが可能です。また REST な Web アプリケーションではこのようにデータの一部 (=リソース) を URI で表せるデータ構造を考えないといけません。

リソースと URI の関係は一对多になりえますが (リソースに対する URI が一意とは限らない)、多対一、つまり一つの URI が複数のリソースを表現することはできません。また URI が正しくリソースを表現しているというものの安全性は保たれていません。

極めつけに、URI を定める事で、リソースどうしの繋がりを表現する事ができます。繋がりはリソース同士の関係を表すことができます。URI だけでは関係を表現することはできないので、たいてい XML などのマークアップ言語を利用します。以下に具体例を示します。

*1 極端なことを言えば、「Web は超巨大なデータの集合」であり、Web の部分集合は URI で表現が可能 (アドレス可能) なのです。つまり WWW は一つに、REST の実装であるとも言えます。

```
<article id="http://localhost/articles/1">
  <title>記事タイトル</title>
  <previous href="http://localhost/articles/0" />
  <next href="http://localhost/articles/2" />
</article>
```

この XML は「適当に作った」ものであり、このようなスキーマがあったり、「このように表現しなくてはならない」という事ではありません。このようにすれば「articles/1」から「一つ前の記事 (articles/0)」と「一つ後ろの記事 (articles/2)」への参照 (リンク) を定義する事で、これらが相互に関係しているということを表している一例です。

繋がりを持てると何がうれしいのでしょうか。そもそも Web 上のリソースは、それに対する参照がなければ存在していないのと同じであると考え事が出来ます。

ただし Ajax をばりばり使った Web アプリケーションはアドレス可能とは言い切れません。GoogleMaps で「この場所を URI として伝えよう」とするときは、わざわざページ右上の「リンク」から座標情報が含まれた URI を取得しなければなりません。同じように Gmail でも単一のメールに対する URI (いわゆる ParmaLink) を取り出すことはできません。

REST でのアクション

リソースの URI を見ると、ブラウザのロケーションバーに貼付けて Return キーを押したくなりませんか? 「http://~」から始まる URI が指し示すリソースを取得する為に、ブラウザは HTTP の GET メソッドを用いて Web サーバーに取得しにいきます。REST ではリソースの操作に、HTTP の基本的なメソッドだけを利用する事にしています。リソースに対する操作を考えられるだけ挙げてみましょう。取得、作成、変更、削除……他にどのような操作が考えられるでしょうか?

データベースと連携した Web アプリケーションを考えてみましょう。Web アプリケーションが行っている処理は一般的に「CRUD」と呼ばれ、「Create, Read, Update, Delete」に分けられます。実はこれらの処理は HTTP の標準的なメソッドだけで十分実現可能であり (リソースはこれらのメソッドに対するインタフェースを持てば良く)、HTTP のメソッドを利用するという制限を持たせる事で一意のアクセス性をもたらします。

リソースの取得 : GET
リソースの作成 : POST
リソースの変更 : PUT
リソースの削除 : DELETE

Web アプリケーションへの操作に SOAP^{*1} などの RPC^{*2} を用いた場合、Web サービスへのアクセスは「`getArticle(ArticleId);`」といったような振る舞いを表すメソッド名を備えたメソッドを作り、それを利用することになるでしょう。しかし RPC が「**振る舞いベース**」であることに対して、REST では「**リソースベース**」であると言えます。リソースをやりとりするためには、HTTP の標準的なメソッドで十分であり、また RPC よりも単純でわかりやすいですね。

つまり私たちが Web アプリケーションを設計し、開発するときには大規模な便利 API を用意するのではなく、欲しいリソースを的確に URI で表現できる設計 (=アクセス可能な構造) を考えるべきなのです。そうすることで REST の恩恵を受けることができます。

また REST は HTTP を用いるためステートレス (セッションレス) です。サーバー側は通信状態を保持せず、クライアント側の情報はリクエストの度に HTTP のメソッドと共に送信します。保持しないことで Web アプリケーションのスケラビリティ^{*3} を確保しています。また 1 つの HTTP 通信がサーバーとクライアントの通信の単位となる為、サーバーの急なアクシデントに対しても損害を受けにくいと言えます。

これまで述べてきたように、REST は「リソースを中心」としたアーキテクチャスタイルで、REST な条件 (URI, HTTP, ステートレスなど) を満たす Web サービスを「RESTful である」と呼んだりします。但し繰り返すように REST はアーキテクチャスタイルであり、アーキテクチャではありません。実際に RESTful な Web アプリケーションを開発するときは「リソース指向アーキテクチャ (Resource Oriented Architecture)」と呼ばれる RESTful なアーキテクチャを利用します。

で、Rails は

長くなりましたがここからが Rails についてです (;ー A アセアセ。ずばり Rails は RESTful な Web アプリケーションを開発するアプリケーションフレームワークです。きれいな URI を設計する事ができ、セッション情報は Cookie に格納します^{*4}。

*1 Simple Object Access Protocol。ソフトウェアがオブジェクトの交換を行う為のプロトコルの一つ。

*2 Remote Procedure Call。ネットワークを跨いだプロセス間通信、と書くのがわかりやすいでしょうか。

*3 システムの拡張性

*4 Cookie に保持する事で CSRF (Cross-Site Request Forgeries; クロスサイトリクエストフォージェリ) 攻撃を仕掛けられるなどが考えられますが、CSRF 対策として Rails 2.x 系では正しいリクエストかどうか判定する仕組みが用意されています。

入門 Ruby on Rails Vol.2

まず Rails アプリケーションの基本的なディレクトリ構造について説明します。

sample/	プロジェクトルート
-- app	Model, View, Controller, Helper のコード
-- config	環境や DB の設定ファイル
-- db	DB の migration ファイル, SQLite の場合は実体ファイル
-- doc	生成されるドキュメント
-- lib	共有のライブラリコード
-- log	生成されるログファイル
-- public	Web から直接アクセス可能なディレクトリ
-- script	ユーティリティスクリプトファイル
-- test	各種テストファイル
-- tmp	一時的なランタイムファイル
-- vendor	サードパーティのツールなど

この連載ではおそらく「app/」「db/」「public/」くらいしか触らないと思います。可能ならば「test/」もやりたいと思っています。

さて、端末に移動してサンプルアプリケーションを作ってみましょう。今回は単純に Web サーバーを起動し、ブラウザから確認しただけでした。今回は Rails アプリケーションから簡単な操作をデータベースに対して行ってみます。

まずは足場を作る必要があります。前回 Web サーバーを起動したプロジェクトルートディレクトリ (sample) で

```
sample$ script/generate scaffold article title:string body:text
```

とタイプし、リターンしてください。ぱーっと十数行文字が流れると思います。2 行ほど warning が出ますが特に問題ありません。

scaffold とは足場という意味で、基本的なメソッドなどを Rails が組み立ててくれるものです。このコマンドで「String 型のカラム title」と「Text 型のカラム body」を持つテーブル「articles^{*}」を定義しました。この時点では RDB にはテーブルは作られていません。厳密には articles テーブルをデータベースに作るコマンドと、それらのレコードに対応する Model や取り扱う Controller が生成されました。

RDB にテーブルを作ったりするファイルを migrate ファイルと呼び、rake コマンドを用いて RDB にテーブルを構成する事を migration と呼んだりします。

この migration と呼ばれる操作は、テーブルの作成やカラムの定義を記述したスクリプトファイルを順番に実行していく事で、RDB に対してテーブルを作ったり構造を変えたりといった一連の

*1 コマンドでは「article」です。Rails が勝手に複数形に変換してくれます。

作業を簡単に行うものです。

また migration はいわゆる「リビジョン」の概念を持っています。具体的な例を挙げて説明すると、n 番目の migrate ファイルでカラム a を持つテーブル A を作成したと仮定します。しかし開発が進んでいくうちに「カラム a の名前を b に変更する」ことになりました。それならば n 番目の migrate ファイルで a としたところを b と書き換えればよいと考えます。しかしここでの正しい操作は、新しく m 番目の migrate ファイルを作成しカラムの名称を a から b に変更する処理を記述します。n 番目と m 番目の間に、もしも k 番目 ($n < k < m$) の migrate ファイルが存在し、カラム a のデータ型を変更しているかもしれません^{*1}。となると、n 番目の migrate ファイルで名前 a を b に書き直してしまうといろいろめんどくさい事になってしまいますよね。このようなありがたいな変更に対して柔軟に対応できるような仕組みになっています。

migration では過去の定義などを残しつつ、どんどんデータベースの構造を書き換えていく処理を行います^{*2}。

それでは先ほどのコマンドで生成された migrate ファイルを見てみましょう。db/migrate を ls すると「`~~_create_articles.rb`」というファイルがあるはずですが。中を見てみると「`self.up`」と「`self.down`」という関数があります。up メソッドにはその migrate ファイルで RDB に対して行う作業が、down メソッドにはそれを取り消すときにどのような操作を行えば良いかという事が書いてあります。`~~_create_articles.rb` の場合だと up メソッド (migrate ファイルのリビジョン番号を進める) には「articles テーブルの作成 (title カラム、body カラム、timestamps^{*3})」をするコード、逆に down メソッド (migrate のリビジョン番号を戻す) では「articles テーブルの削除」をするコードが書かれています。

実際に RDB に対して migration を行ってみます。

```
sample$ rake db:migrate
```

またばーっと文字が流れます。

```
(in /home/inohiro/Projects/sample)
== 20090612031154 CreateArticles: migrating =====
-- create_table(:articles)
   -> 0.0595s
== 20090612031154 CreateArticles: migrated (0.0597s) =====
```

*1 データ型の変更はよほどの事がない限りやるべきでないと思いますが。今回はカラム a を n 個目と m 個目の migrate ファイル以外も参照していたら、という例の為です。

*2 もちろん任意番目 (リビジョン) の migrate ファイルまで実行し、過去の構成を復元するという事が可能です。過去の定義を残しておくことで「前バージョンの構成は j 番目まで migrate すればよい」といったようなことができます。

*3 Rails が勝手に追加するレコードの作成/変更日時を記録するカラム

入門 Ruby on Rails Vol.2

これで DB に articles テーブルが作成されました。

Rails アプリケーションへのアクセス

準備は整ったのもう一度 Rails アプリケーションを動かしてみましょう。

```
sample$ script/server -p 8080
```

Web ブラウザから「<http://localhost:8080/articles>」にアクセスします。下のように表示されましたか？



「New Article」というハイパーリンクをクリックすると「<http://localhost:8080/articles/new>」に移動します。Title と Body を入力できるフォームが用意されているので適当に文字列を入力して、Create ボタンをクリックしてみましょう。適当にいくつか作ってみてください。また「/articles」には「Edit」と「Destroy」もあるので、これもいじってみてください。

これらのフォームなどは scaffold が勝手に作ってくれたもので、本格的なアプリケーション開発にはあまり使わないのではないかと思います。今回のように簡単な動くアプリケーションを作ってみるときは大変便利です。

RESTful な Rails

ちょっと遊んでみたところで、Rails が本当に RESTful な Web アプリケーションを開発できるフレームワークであるか確かめてみましょう。REST では同一のリソース URI に対して、HTTP の基本的なメソッドを使ってアクセスすると書きました。今回の場合は以下のようになります。

新しい article の作成 : POST /articles
article の一覧の取得 : GET /articles
既存の article の取得 : GET /articles/id
既存の article の更新 : PUT /articles/id
既存の article の削除 : DELETE /articles/id

今回は端末から「curl」というコマンドを用いて Web サーバーに対して POST, GET, PUT, DELETE を行ってみます*1。但し Rails は我々が意識しない所でセキュリティトークンを一緒に送信していたりするので、この検証ではその設定を解除しないとイケません。app/controller 以下にある application.rb を開いて、9 行目「protect_from_forgery」から始まる一行をコメントアウト（#をつける）します。

```
# 省略
class ApplicationController < ActionController::Base
  helper :all # include all helpers, all the time

  # See ActionController::RequestForgeryProtection for details
  # Uncomment the :secret if you're not using the cookie session store
  # protect_from_forgery # :secret => #アプリケーションごとのセキュリティトークン
  # 省略
```

新しい article の作成 (POST /articles)

新しい article の作成には「/articles」に POST する事で実現できます。article は「title」と「body」という 2つのフィールドを持つため、POST する場合は 2つのパラメータを指定します。

```
~$ curl -i -X POST -H 'Host: localhost'
  -d 'article[title]=hello&article[body]=from_curl_command'
  http://localhost:8080/articles
HTTP/1.1 302 Found
Location: http://localhost/articles/25
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
Date: Mon, 15 Jun 2009 09:21:33 GMT
Server: WEBrick/1.3.1 (Ruby/1.8.7/2008-08-11)
# 省略
```

*1 curl: 「sudo aptitude install curl」でインストールできます。wget コマンドでは GET, POST はできるものの PUT と DELETE ができません。



「/articles」にアクセスしたところ。新しいレコードが POST メソッドによって追加されているのが確認できます。

article 一覧の取得 (GET /articles)

article 一覧とは真上の画像の画面のことです。

```
~$ curl -i -X GET -H 'Host: localhost' http://localhost:8080/articles
HTTP/1.1 200 OK
Etag: "81e3c360593936f32d826fdb39e97e4"
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
Date: Mon, 15 Jun 2009 09:22:21 GMT
Server: WEBrick/1.3.1 (Ruby/1.8.7/2008-08-11)
X-Runtime: 0.06821
Content-Length: 1849
Cache-Control: private, max-age=0, must-revalidate
Set-Cookie:
_sample_session=BAh7BiIKZmxhc2hJQzonQWN0aW9uQ29udHJvbGxlcjo6Rmxhc2g6OkZsYXNo%0AS
GFzaHsABjoKQHvzZWR7AA%3D%3D--7e363cd71d20c427ff03f0c7608cb125bba0f2d2; path=/
# 省略
<tr>
  <td>Hello, Ruby on Rail Web Application!!</td>
  <td>This is first article for WORD.
  Ruby on Rails is happy.</td>
  <td><a href="/articles/1">Show</a></td>
  <td><a href="/articles/1/edit">Edit</a></td>
# 省略
```

既存の article の取得 (GET /articles/id)

記事の Id (Rails では migrate ファイルでレコード毎の Id を指定しなくても勝手につくってくれます) を指定するとそのファイルの詳細をとってこれます。



```

~$ curl -i -X GET -H 'Host: localhost' http://localhost:8080/articles/25
HTTP/1.1 200 OK
Etag: "ea1232254a64b4178be04706883797c8"
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
Date: Mon, 15 Jun 2009 09:23:14 GMT
Server: WEBrick/1.3.1 (Ruby/1.8.7/2008-08-11)
X-Runtime: 0.01024
Content-Length: 623
Cache-Control: private, max-age=0, must-revalidate
Set-Cookie:
_sample_session=BAh7BiIKZmxhc2hJQzonQWN0aW9uQ29udHJvbGxlcjo6Rmxhc2g6OkZsYXNo%0AS
GFzaHsABjoKQHVzZWR7AA%3D%3D--7e363cd71d20c427ff03f0c7608cb125bba0f2d2; path=/

# 省略
<p>
  <b>Title:</b>
  hello
</p>
<p>
  <b>Body:</b>
  from_curl_command
</p>
# 省略

```

入門 Ruby on Rails Vol.2

既存の article の更新 (PUT /articles/id)

既存の article に対して、POST と同じように 2 つのパラメータを設定し PUT メソッドを用いる事でファイルの置き換え（編集）が行えます。

```
~$ curl -i -X PUT -H 'Host: localhost'
  -d 'article[title]=edited&article[body]=edited_from_curl_command'
  http://localhost:8080/articles/25
HTTP/1.1 302 Found
Location: http://localhost/articles/25
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
Date: Mon, 15 Jun 2009 09:23:56 GMT
Server: WEBrick/1.3.1 (Ruby/1.8.7/2008-08-11)
# 省略
```



「articles/25」の Title と Body が更新されているのが確認できます。

既存の article の削除 (DELETE /articles/id)

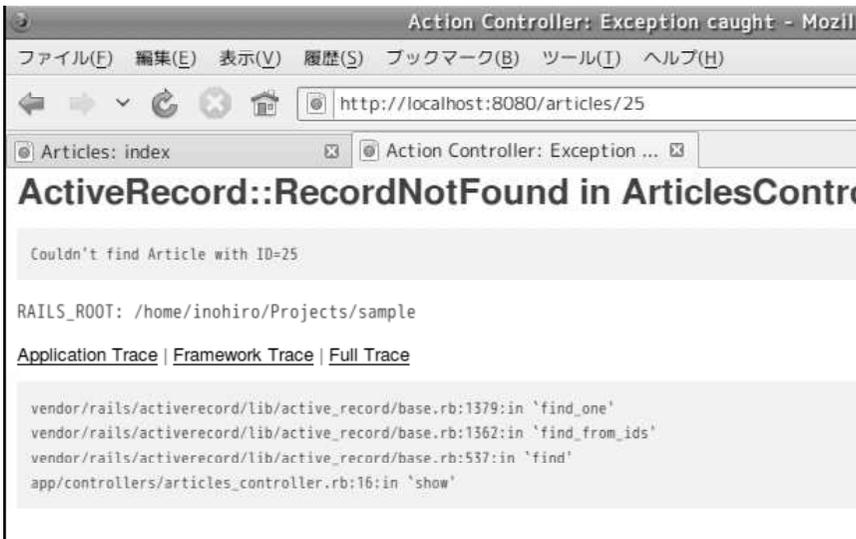
既存の article に対して DELETE メソッドを用いると削除する事が出来ます。

```
~$ curl -i -X DELETE -H 'Host: localhost' http://localhost:8080/articles/25
HTTP/1.1 302 Found
Location: http://localhost/articles
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
Date: Mon, 15 Jun 2009 09:24:41 GMT
Server: WEBrick/1.3.1 (Ruby/1.8.7/2008-08-11)
# 省略
```

削除したファイルを GET してみましょう。

```
~$ curl -i -X GET -H 'Host: localhost'
  http://localhost:8080/articles/25
HTTP/1.1 404 Not Found
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
Date: Mon, 15 Jun 2009 09:27:16 GMT
Server: WEBrick/1.3.1 (Ruby/1.8.7/2008-08-11)
# 省略
```

先ほど取得できた「/articles/25」に対して GET メソッドで取得しようとする Web サーバーが「404 Not Found」を返しているのが確認できます。



さらにブラウザからアクセスすると「RecordNotFound」とエラー出力されています。

まとめ

今回は Rails の重要な構成要素である MVC と REST について書きました。かなり長くなってしまいましたが、なるべくわかりやすく書いたつもりです。

特に Rails が RESTful である事を確認していただけたのではないかと思います。

逆に MVC については概念だけしか説明する事が出来ませんでした。Rails アプリケーションに対する URI は Controller が受け取り Model を操作したり View をブラウザに返す処理を行っている事を書きましたが、次回では具体的にどのように取り回しを行っているか書いていきたいと思っています。

「お前は何を言っているんだ」という状態でしたら以下の参考文献を参照してください。大変わかりやすい記事で、この記事を書くときの参考にしました。

入門 Ruby on Rails Vol.2

参考文献 (URI 素晴らしい(*´д`*)ハアア)

- REST 入門 (yohei-y:webblog)
-- http://yohei-y.blogspot.com/2005/04/rest_23.html

- REST と ROA (株式会社リコー ソフトウェア研究開発本部)
-- http://blogs.ricollab.jp/webtech/wp-content/uploads/2008/02/rest_and_roa.pdf

- RESTful Web Application の可能性
-- http://www.4bit.net/archives/2005/06/restful_web_app.html

- REST on Rails (www.xml.com)
-- <http://www.xml.com/pub/a/2005/11/02/rest-on-rails.html>

- ActiveResource の使い方 (前編) : Rails 同士で通信する (WebOS Goodies)
-- http://webos-goodies.jp/archives/how_to_use_activeresource_1.html

- RESTful best practices (www.slideshare.net by calamitas)
-- <http://www.slideshare.net/calamitas/restful-best-practices>

編集後記

情報科学類誌

WORD

WORDカップ^o出場決定!号

発行者 情報科学類長

編集長 柴田 泰晴

製作・編集 筑波大学情報学群
情報科学類WORD編集部
(第三エリアC棟212室)

印刷 情報科学類印刷室

2009年6月 初版第一刷発行