

# Babascript:

Babascript:

匿名で査読を行うため著者名なし\*

## Summary.

プログラムはコンピュータを制御できるが、人は制御できない。人をプログラムから制御できれば、コンピュータのみでは実現出来なかった処理も実現可能となる。そこで、本論文では、プログラムに人への処理命令を組み込むことのできるプログラミング環境 Babascript を提案する。Babascript では、人とコンピュータはプログラム上で対等な処理ノードとなり、相互補完的に動作することでまた、Babascript フレームワークによって実現する応用例を示し、起こりうる問題について考察する。

## 1 はじめに

プログラムはコンピュータを制御できるが、人を制御することはできない。既存のプログラムの仕様には、人に対する制御命令構文が存在するものではなく、コンピュータを制御するためのものである。

人はコンピュータに比べ、とても柔軟な能力を有する。単純な演算能力や記憶能力だけで言えば、コンピュータのほうが優れていることもあるが、場の空気を読んだり、人とのコミュニケーションは人のほうが得意であると言える。

人は、コンピュータにはない貴重な能力を有しており、プログラムにとっても有用である。しかし、人をプログラムから制御出来るような仕組みの研究はクラウドソーシングの分野に限られている。クラウドソーシングはインターネットを介した不特定多数の人をリソースとして扱うものであり、利用可能な能力は限られている。また、専用の記法を利用する必要があった。不特定な人ではなく、特定の人をプログラムから操作できるようになれば、プログラムが干渉可能な領域は更に広がると考えられる。

そこで、本論文では、Babascript プログラミング環境を提案する。Babascript プログラミング環境は、関数呼び出しによって人に処理命令の通知ができる DSL Babascript (図 1) と、Babascript からの命令を受け取り、処理結果を入力することのできるクライアントライブラリ (図 5) 及びスマートフォンアプリケーションの Babascript Client (図 2) から構成される。

Babascript を利用することで、通常のプログラムと人への処理命令をほぼ同じ記述で実現することができる。これによって、特別な記法の知識が必要なく、既存のプログラムの書き方人の力を借りたプログラミングが可能となる。

コンピュータと人が従順な処理ノードとして、プ

```
1 BabaScript = require("babascript");
2 baba = new BabaScript("takumibaba");
3 baba.書類整理をする(function(tesult){
4     // 返り値を得た後の処理を記述する
5 });
```

図 1. babascript プログラムの基本命令文

ログラムからの命令を積極的に処理していくことによって、相互補完しながら全体の処理を最適化できると考えられる。つまり、コンピュータはその演算能力を最大限に活かし人を助け、人はその柔軟な思考力や実世界への干渉力を活かしてコンピュータにはできない処理を実現する。こういった仕組みによって、人にとっての処理の最適化を狙う。また、その例として「仕事のプログラム化」と「人力実世界プログラミング」を示す。

## 2 BabaScript

Babascript は、関数呼び出しによって人に処理命令を送れるオブジェクト (以下、人オブジェクト) を宣言可能にする DSL である。人オブジェクトにおいて定義されていない全てのメソッドが人への命令 (以下、人への命令構文) として解釈され、人に通知される。また、人からの返り値を得ると、人への命令構文の引数で指定したコールバック関数を実行する。

### 2.1 人オブジェクトの宣言と処理命令構文

図 1 のようなプログラムによって、人に処理命令を送ることができる。人への命令構文は実行後、メソッド名と引数を元にした json オブジェクトを生成する。生成された json オブジェクトは、命令としてクライアントライブラリに配信される。この際、オブジェクトに定義されていない全てのメソッドが人への命令構文として解釈される。この機能は methodmissing



図 2. Web アプリのインターフェース

と呼ばれる、定義されていないメソッドが実行された際の処理を記述しておく、その処理を実行する機能によって実現している。

人オブジェクトは宣言時に ID を指定することによって、誰に対して命令を配信するかを決定する。例えば、id:baba に命令を送りたければ、人オブジェクト宣言時の第一引数には baba という文字列を指定する必要がある。また、クライアント側でも同様に、id:baba として宣言する必要がある。

この際、指定した ID を監視するクライアントが複数人だった場合は、後述する broadcast オプションをつかってない場合は、命令を実行していないクライアントへと順番に配信される。

人への処理命令構文の第一引数にはクライアント側に送信するオプション情報を指定し、第二引数には戻り値が得られた際に実行するコールバック関数を指定する。コールバック関数実行時には、その引数に戻り値と命令情報、実行者情報を格納したオブジェクトが与えられる。

## 2.2 オプション情報の付加

人への処理命令構文の第一引数に与えるオプション情報には、戻り値の型指定などの情報が考えられる。例えば Boolean 型を指定する場合には図 4 のようなプログラムが考えられる。他にも、指定したリストの中から値を選択してほしい、といった命令の場合は、リスト情報を格納した配列を指定する。このオプション情報の処理は基本機能には含まれていないため、ライブラリを利用して実装するアプリケーションごとに対応する必要がある。

特別なオプション情報として、broadcast オプションが存在する。broadcast 機能は、指定した ID を監

```
1 BabaScript = require("babascript");
2 baba = new BabaScript("takumibaba");
3 baba.書類整理をする(function(tesult){
4   // 戻り値を得た後の処理を記述する
5 });
```

図 3. オプション情報

視する全てのクライアントへとタスクを配信し、指定した数の戻り値を得られると処理を終了し、コールバック関数を実行するといったものだ。

## 2.3 サンプルコード

以下にサンプルコードを示す。

```
25 MEMBER = 3;
26
27 members.参加可能な日程を選んでください({
28   broadcast: MEMBER
29 }, function(results) {
30   var b, i, work, worker, workers, _k, _ref;
31   workers = _.pluck(results, "worker");
32   b = [];
33   work = [];
34   for (i = _k = 0, _ref = workers.length - 1; 0 <= _ref ? _k <= _ref : _k >= _ref; i = 0 <= _ref ? ++_k : --_k) {
35     worker = workers[i];
36     b.push(function(_this) {
37       return function(callback) {
38         var a, date, _l, _len1, _results;
39         a = [];
40         _results = [];
41         for (_l = 0, _len1 = list.length; _l < _len1; _l++) {
42           date = list[_l];
43           _results.push(worker.参加可能かどうか(
44             description: date
45           ), function(result) {
46             a.push(result.value);
47             if (result.task.description === "金曜(6限)") {
48               return callback(null, a);
49             }
50           });
51         }
52         return _results;
53       });
54     })(this);
55   }
56   return async.parallel(b, function(err, result) {
57     var a, c, funcs, k, len, m, _l, _m, _ref1;
58     len = result[0].length - 1;
59     b = [];
60     for (i = _l = 0; 0 <= len ? _l <= len : _l >= len; i = 0 <= len ? ++_l : --_l) {
61       for (k = _m = 0, _ref1 = result.length - 1; 0 <= _ref1 ? _m <= _ref1 : _m >= _ref1; k = 0 <= _ref1 ? ++_m : --_m) {
62         a.push(result[k][i]);
63       }
64       c = _.reject(a, function(bool) {
65         return bool === false;
66       });
67       b[list[i]] = c.length;
68     }
69     m = _.max(b, function(n) {
70       return n.val;
71     });
72     if (m[0].length >= MEMBER) {
73       return members.以下の日程から開催日時を選んでください({
74         description: m
75       }, function(result) {
76         return process.exit();
77       });
78     } else {
79       funcs = [];
80       funcs.push(function(_this) {
81         return function(args) {
82           var _n, _ref2;
83           if (args.length === 1) {
84             i = 1;
85           }
86           for (k = _n = 0, _ref2 = MEMBER - 1; 0 <= _ref2 ? _n <= _ref2 : _n >= _ref2; k = 0 <= _ref2 ? ++_n : --_n) {
87             if (i[m[k]]k) {
88               members.get(k, 以下の日程になりそうですが、調整できませんか({
89                 description: list[i]
90               });
91             }
92           }
93         }
94       });
95     }
96   });
97 }
```

図 4. サンプルコード

## 2.4 Babascript Client

Babascript Client は、Babascript からの命令受信と処理結果の送信機能を実現するクライアントライブラリだ。命令受信のイベントに対してコールバック関数を指定することで、処理命令を受け取ることができる。基本的な利用方法は、図 5 に示す。

クライアントオブジェクトの宣言時、ID を指定することによって、ID に対して処理命令が発行された際にタスクを受信することができる。また、ク

Babascript:

```
1 Client = require("babascript-client");
2 client = new Client("takumibaba");
3 client.on("get_task", function(task){
4     // プログラムからタスクを受け取った時の
5     // 挙動を記述する
6 });
```

図 5. クライアントライブラリの基本機能

クライアントオブジェクトが持つ `returnValue` メソッドを利用することで、戻り値を命令発行元に返すことができる。

クライアントライブラリは、UI と完全に分離した実装となっており、かつ利用方もシンプルだ。後述の Web アプリケーションだけでなく、様々なアプリケーションに組み込むことができる。

## 2.5 Web アプリケーション

BabascriptClient が得たタスクをユーザに提示するためのインタフェースとして、Web アプリケーションを実装した。タスク情報において型指定をすることによって、ユーザに提示する UI を変化させ、型に合った戻り値を選択できるようにしている。例えば、Boolean 型を指定していた場合、ユーザには `true` ボタンと `false` ボタンが提示され、どちらかを押すと、その結果が戻り値としてプログラムに返される。

スマートフォンに最適化した Web アプリケーションとして実装しており、様々な場面において利用可能であり、実世界におけるタスクを処理しながらでも十分に利用可能である。図 2 のような画面を持つ。

## 3 実装

上記システムは全て Javascript で実装した。Babascript は Node.js 上で動作し、Babascript Client は Node.js と Web ブラウザ上で動作する。全体図は図 6 のとおりだ。

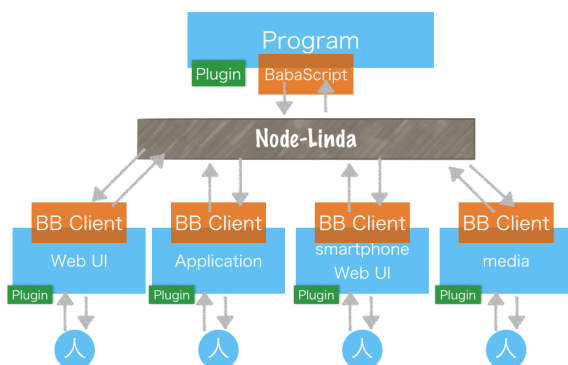


図 6. システム全体図

### 3.1 処理の流れ

Babascript 環境は、以下のようなフローで一回の人への命令構文が実行される。

1. 人への命令構文を実行する
2. 命令構文の内容に基づき、タスクが生成される
3. タスクが Node-Linda サーバを経由してクライアントへと配信される
4. タスクを受け取ったクライアントがユーザに処理を促す
5. タスク実行者が、処理結果を入力する
6. 処理結果を元に戻り値データが生成される
7. Node-Linda サーバを経由して実行元プログラムに戻り値が送信される
8. 指定されたコールバック関数が実行され、処理が続く

### 3.2 タスク情報の構成

人への処理命令構文の実行によってタスク情報が生成される。このタスク情報は、図 7 のような json オブジェクトとなる。

```
1 task = {
2     "name": "baba", // 命令配信先ID
3     "type": "eval", // 配信タイプ
4     "key": "部屋の温度は適切ですか", // 処理命令の内容
5     "cid": "1409227153_0.869041177676", //タスクID
6     "option": { // オプション情報
7         "format": "boolean",
8     }
9 };
```

図 7. タスクの JSON オブジェクト

### 3.3 実行結果の待ち方

人への命令構文に対する戻り値が、適切に戻り値として戻されるようにするために、命令ごとにユニークな ID を生成している。ユニーク ID は、人への命令構文実行時の UNIX time と、ランダムな数値を結合した文字列である。図 7 における json オブジェクトの `'cid'` の項目に示されているものだ。このユニーク ID は人への命令構文実行時に生成され、ユニーク ID をこのユニーク ID を伴った戻り値が戻ってくるか、指定したタイムアウト時間に到達するか、タスクキャンセルが発生するまで、

### 3.4 script-client 間の通信

script と client のタスク送受信のために、node-linda[2] を利用した。script, client 共に、この node-linda に接続するためのライブラリを組み込んでいる。

script は、プログラム実行時に node-linda サーバに接続し、人への命令構文を実行すると、タスクを node-linda サーバに書き込む。書き込み終了後、

そのタスクと同じ ID を持った返り値タスクが書き込まれるまで、node-linda サーバの監視を行う。

client は、プログラム起動中は常に node-linda サーバに接続し、指定した ID を監視し、タスク受信まで待機する。node-linda サーバに同一 ID にタスク書き込みが行われた際、そのタスクを取得する。このタスクは、broadcast オプションが付いている場合は、同一 ID を監視している全てのクライアントがタスクを受信する。通常の実行の場合は、キューの最初のクライアントのみがタスクを受信・実行し、他のクライアントは待機を継続する。client は、タスク実行中は通常のタスクを受信することではなく、タスク実行が終わって受信待ち状態になる際に新たにキューに追加される。このため、複数のクライアントが一つ ID を監視している状況であっても、一つのクライアントにタスクが集中するといったことはない。

## 4 応用例

以下のような応用が考えられる。

### 4.1 仕事のプログラム化

人への行動命令がプログラムとして記述可能になることで、人の仕事をプログラム化することができる。プログラム化し、実行することによって、状態をプログラムによって管理することが可能となる。つまり、条件判断や記憶が必要なことは出来るだけコンピュータ側に処理させる、といったことが可能になり、人は細かい条件などを覚えておいたり、自分で判断することなく、ただ指示されたことのみを実行することで、仕事を終わらせることができるようになる。ただ指示されたことのみを実行するだけで良いならば、仕事の引き継ぎなども必要なくなり、人の代替も容易となる。

また、人同士がコミュニケーションを取ることなく、複数人を協調させるといったことも可能となる。通常、複数人を協調させるためには、人同士が相談したり、上位の意思決定者が必要となる。しかし、コミュニケーションはコストのかかるものであり、適切に行われない場合、問題が生じることもある。意思決定をプログラムに委ねることは、複数人を効率よく協調させることに繋がるとも考えられる。

全ての仕事をプログラム経由にすることで、詳細な実行ログや実行状況を把握することもできる。仕事の実行量の定量化や、状況監視、状況の可視化などに活かすことができる。

### 4.2 人力実世界プログラミング

人をセンサーやアクチュエータとして利用することで、人によって実世界を操作する実世界プログラミングが可能となる。現在のセンサーやアクチュエータでは、実世界に干渉するのには限界がある。

例えば、その場の雰囲気や数値化したり文字列化するという、コンテキスト情報を伴ったセンシングは困難である。人というセンサーをプログラムから利用できるようにすることで、今までには実現しなかった処理が実現する。

## 5 考察

Babascript 環境について、以下のように考察する。

### 5.1 処理単位としての人

本研究では、人は処理を命令され実行するノードとなり、プログラム上においてコンピュータやセンサーと同等となる。こういったことに対して心理的な拒否感を覚えることも考えられる。しかし、一処理ノードとして扱うことによって、大きなメリットもある。人とコンピュータ、処理に応じて適切なように実行させることができれば、人は人にしかできないことや人がやるべき処理にのみ集中するといったことも実現できる。また、ただやるべきことのみを命令されて動くということは、自分で深く考えて行動する必要がないということだ。とても楽なことでもある。

### 5.2 タスク実行の遅延と実行保障性

Babascript によってタスク実行を依頼しても、人がすぐにタスクを実行し値を返すことを完全に保証することはできない。タスク受信端末を見ていない、受信しても実行できないといった状況の場合、すぐに値を返すことはできない。この際、Babascript による処理が原因で全体の処理が遅延する可能性がある。

また、労働関係にあるなど、タスク実行に強制力がある場合は、タスク実行が確実に行われると考えられるが、強制力がない場合はそもそもタスク受信を無視するといったことも考えられる。タスク実行に強制力がない場合は、金銭などのインセンティブを与えるといった手段によって、実行保障性を確保するといったことが考えられる。

### 5.3 例外処理

Babascript 環境において、命令文と現実との乖離によって適切な返り値を選択・記述ができなくなるといった可能性がある。これは、現実が刻々と変化していることなどから、完全に避ける事の出来ない問題であると考えられる。この際、無理やり値を返すといった処理をしてしまうと、本来の状態とは違った判断がなされてしまう危険性がある。命令文とは明らかに現実が異なっている場合などは、タスク実行者から例外としてプログラムに通知出来るような仕組みの実装によって、問題の解決へと繋がれると考える。



## 5.4 命令内容の粒度

Babascript では、タスクの命令文の記述には制限がないため、自由となっている。抽象度が高すぎる命令は、あいまいな表記となり、タスク実行者にとって理解しづらい文面となり得る。その結果、想定外の処理が実行され、意図しない結果を招く恐れがある。抽象度が低すぎる命令は、全体の処理内容にもよるが、プログラム自体が冗長となり得る。プログラムとタスク実行者の間のやりとりが増え、通信や待機時間などがボトルネックとなる可能性がある。また、タスク実行者にとっても、やりとりが増えることで負担増になると考えられる。処理ごとに異なると考えられるが、命令文は適切な抽象度に設計しなくてはならない。

## 5.5 複数命令の同時実行

複数のプログラムから同時に一人のタスク実行者へとタスクが配信される可能性がある。この際、異なるコンテキストにある命令が交互に配信され、タスク実行に大きな障害をもたらす可能性がある。例えば、料理プログラムと掃除プログラムが同時に実行された場合、鍋で煮ている途中で「洗剤を投入しろ」などといった命令が配信されることが考えられる。

この問題は、全ての Babascript プログラム中において、一人のタスク実行者は一つのプログラムからのみ、連続してタスクを受信できるような仕組みを用意することによって、解決可能であると考えられる。また、応用アプリケーションでの実装になるが、コンテキストを明示し、どの処理系におけるタスクなのかをタスク実行者に示すといった手段によっても解決可能である。

## 6 関連研究

計算機では処理が難しいようなタスクを解決するために、人を計算資源として利用する手法はヒューマンコンピューテーション [8] と呼ばれ、様々な研究が行われている。インターネットを介して不特定多数の群衆にタスクを実行させるクラウドソーシングと組み合わせた研究事例も多く存在する。クラウドソーシングのプラットフォームとしては、Amazon Mechanical Turk[1] が存在する。Barowy らは、CrowdProgramming という概念を提唱し、プログラミング言語内においてクラウドソーシングによる計算とコンピュータによる計算の統合を実現した [3]。Franklin らは、機械だけでは答えられないような DB へのクエリに対する応答を、クラウドソーシングを用いることで返答可能にする CrowdDB を提案している [6]。Morishima らは、人をデータソースとしてプログラムの中で利用する手法を提案している [7]。これらの研究では、クラウドソーシングを利用した問題解決手法の提案をしている。本研究

は、不特定多数の群衆をプログラムするためのものではなく、実世界の操作なども含むため、家族や職場の人のような特定可能な人物を対象としている。また、人を計算資源やデータソースとしてシステムに組み込むことを前提としているが、本研究は、計算資源やデータソースに限らず、実世界への干渉等も対象としている。本研究はより汎用的な枠組みであると言える。

ユビキタスコンピューティングの研究分野においては、Human as Sensor という概念も提唱されている。PRISM は、スマートフォンを利用したセンシングプラットフォームだ [5]。Liu らは、ソーシャルメディア上の人をセンサーとして扱った Q&A サービス MoboQ を提案し、その検証を行った。Human as Sensor に類する研究では、人をセンサーとして扱うことを対象としているが、本研究ではセンサーのみを対象としていない。

Cheng らは、モーションプラットフォームにおけるモーターやメカニカル機構の代替として人を利用した Haptic Turk を提案している [4]。Haptic Turk はゲームでの利用に特化したものだ。本研究は、用途を限らない汎用的な仕組みとなっている。

加藤らは、人とロボット間でのタスク共有システム Sharedo を提案した [9]。人とロボットのタスク実行における協調は、本研究の主眼である「何かの処理を実現するとき、人とコンピュータは相補的に動作すべき」という考えと大きく類似している。

## 7 おわりに

本論文では、プログラムに人への処理命令を組み込むことのできるプログラミング環境 Babascript を提案した。人をプログラムに組み込み、制御できるようにすることで、

何かの処理を実行していく上で、コンピュータにできることは極力コンピュータに処理させ、人は人にしかできないことや、人がやるべきことだけを処理するというのが本来あるべき処理の切り分け方であるが、現状ではその切り分け方はあいまいである。

Babascript によって、人はプログラム上においてコンピュータと同じ処理ノードとして存在し、関数実行によって処理内容を受け取り、実行して値を返す存在になれる。これによって、プログラマブルになりつつある世界において人自身もその一部になることが可能となる。

また、Babascript 環境によって実現する応用例を示すことによって、人をプログラマブルにした際のメリットを示した。

今後は、議論で述べた Babascript の問題点などを改善していく。

## 参考文献

- [1] Amazon Mechanical Turk.  
<https://www.mturk.com/mturk/>.
- [2] Linda implementation on Socket.IO.  
<https://github.com/node-linda/linda>.
- [3] D. W. Barowy, C. Curtsinger, E. D. Berger, and A. McGregor. AutoMan: A Platform for Integrating Human-based and Digital Computation. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA '12*, pp. 639–654, New York, NY, USA, 2012. ACM.
- [4] L.-P. Cheng, P. Lühne, P. Lopes, C. Sterz, and P. Baudisch. Haptic Turk: A Motion Platform Based on People. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '14*, pp. 3463–3472, New York, NY, USA, 2014. ACM.
- [5] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, and A. Sharma. PRISM: Platform for Remote Sensing Using Smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, MobiSys '10*, pp. 63–76, New York, NY, USA, 2010. ACM.
- [6] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: Answering Queries with Crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD '11*, pp. 61–72, New York, NY, USA, 2011. ACM.
- [7] A. Morishima, N. Shinagawa, T. Mitsuishi, H. Aoki, and S. Fukusumi. CyLog/Crowd4U: A Declarative Platform for Complex Data-centric Crowdsourcing. *Proc. VLDB Endow.*, 5(12):1918–1921, Aug. 2012.
- [8] L. Von Ahn. *Human Computation*. PhD thesis, Pittsburgh, PA, USA, 2005. AAI3205378.
- [9] 淳 加藤, 大介 坂本, 健夫 五十嵐. Sharedo: To-do リストによる人-ロボット間のタスク共有. WISS '11: 第19回インタラクティブシステムとソフトウェアに関するワークショップ, pp. 102–107, 2011.