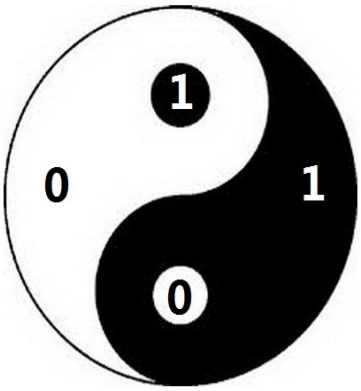



# 太极真经 ——

# 太极语言编程指南

曹星明 (太极真人)

# 太极语言的产生背景

太极语言是我长期自然语言编程偶然获得的一个意外收获。我从1995年开始研究自然语言编程，因此对于自然语言处理和计算机语言学的各种理论关注很多，特别是语法理论和解析技术。同时也对各种计算机语言，编译技术多有研究。尝试和研究过很多高级语言。特别是非常欣赏lisp的优雅和强大的功能。

从2007年开始，我尝试设计和实现解析器。试验了多种不同的解析技术实现方法。从2011年开始尝试实现解释器和编译器。期间完成了dao和daonode这两个项目，开源在Github。

# 太极语言的源流

太极语言的产生可以追溯到lisp之根。太极语言从内核上继承了lisp的精神，甚至比lisp更为lisp。lisp将程序或者说代码看作数据，也就是列表，一种递归数据结构的方法，深刻地影响了我的思维。lisp的宏就是以程序作为数据，并以程序作为结果的构造。

语法上拥抱了主流高级语言的语法。特别是缩进语法。我最早接触的缩进语言是python，非常喜欢这种简洁明快的语法。后来，我更多地使用javascript设计web系统，这期间接触到了coffee-script，大大提高了我的开发效率。coffee-script的语法特征也启发了我设计了Peasy这一解析器系统。从产生Peasy的创意到实现过程我体会到语言和语法对于思维之间的作用。因为正是coffee-script将赋值作为表达式，及其定义函数的语法使得用它手写语法规则特别直观可读，用它手写的解析器可以和专用于描述语法规则的dsl相媲美。从语法上，太极语言和coffee-script有很多相似之处。对此本书在附录中专门给出一节加以对比。

除了以上最重要的两点，太极语言也从很多高级语言吸取了营养。我曾经探索过rebol语言，感觉它是一种精致的语言。特别是它的方言功能，利用parse函数设计出的GUI方言的表现力令我叹服。受其影响，我设计和实现太极语言时，最开始我不自觉地选择了[]作为代码块的定界符。直到发布前夕，在实现某个功能时，我发现使用[]有某些不便。然后我突然意识到，可以使用更为通用的{}作为代码块定界符。

我也曾经了解过metalua的元语言功能，思维中植入了元编译这个概念。

虽然太极语言借鉴了这么多语言，但是太极语言并不是一个杂乱揉和各种语言特征的混合体。相反，太极语言在吸收的上述每个显著特征都明显地超过了最初的本体，上升到了一个新的层次，同时，也形成了自己的整体风格和设计原则，具有独立的灵魂。

# 太极语言的特点

javascript 穿上一层lisp的衬衣，然后外面再套一件类似coffee-script缩进语法的外套。

（coffeescript，haskell等同样语法特点的语言）

中间表示

灵活的语法 运算表达式 缩进表达式

解析 解析结果的格式 对解析结果的变换

编译

优化 优化指令 循环外提 函数外提 编译时间计算 无io作用的代码段 尾递归优化，普通递归优化

# 太极语言的相关资源

太极语言github项目：github.com/taijiweb/taijilang

发布在npm的包taiji：npmjs.org/package/taiji

google 邮件组：google groups: taijilang

google+帐号：taiji lang

QQ群: 太极语言 194928684

# 本书的组织

## 本书的目的

本书的目的是作为太极语言的一个太极语言的使用指南，特别是使用太极语言编程的指南。

## 本书的体系

本书主要分为两个部分：基础和高级。基础部分主要介绍太极语言作为编程语言的基本知识，特别是绝大多数高级语言所共通的知识。 高级部分介绍太极语言的高级功能。有很多功能是太极语言独有的创新。

## 各章内容介绍

基础部分

第一章到第八章是本书的基础部分。包含了太极语言的介绍性知识、基本功能以及和大多数高级语言所具有的共通功能。

- 第一章 介绍：本章就是第一章，介绍太极语言的产生背景，技术上的源流，特别是与现有高级语言的关系，太极语言自身的特点，编写本书的目的，以及全书的内容组织。
- 第二章 太极语言概览：介绍太极语言的功能特性，和具有相似特征的高级语言的对比，介绍太极语言的编译过程和中间表示的基本特点，太极图解以太极图的方式阐释高级语言和程序发展过程中观念和技术和技术演进，太极之禅是试图太极语言设计哲学的释宗禅语。
- 第三章 起步：介绍太极语言的基本工具和入门知识。包括npm安装命令，命令行工具的使用，repl，和一些简单的示例。
- 第四章 数据和变量：介绍太极语言的数字（整数和浮点数）、列表和字典等基本数据类型的文字表示，太极语言的变量定义以及与作用域相关的规定。
- 第五章 基本命令和运算：介绍太极语言的基本命令，如注释、打印、单引表达式、求值等，也将介绍太极语言的算符表达式，并给出太极语言预定义的各类运算符及其优先级规则。
- 第六章 控制结构：介绍常见的控制结构在太极语言中的使用方法，包括块结构、条件执行、重复执行、异常等各类语句。
- 第七章 介绍太极语言的函数，包括其定义和调用方法；类的定义和使用，太极语言的模块化功能，包括include!和import！语句的使用。

## 高级部分

从第八章到第十一章是本书的高级部分，介绍了太极语言的高级内容，体现了太极语言的独特功能。

第八章 编译过程和中间表示： 介绍太极语言的编译过程，涉及到编译过程各个阶段及其操作，包括代码解析，元编译和元变换、表达式转换和代码产生等。同时也介绍了贯穿所有这些阶段的中间表示。这一章是理解和掌握后续各章的基础性内容。

第九章 语法解析：将介绍太极语言关于语法及解析方法的设计原则，实现过程，介绍太极语言的词法、主要语法成分、算符表达式的解析特点。

第十章 元编译和宏：介绍太极语言的元编译特性以及它的最重要特例——宏功能。这包括太极语言发现元编译技术以及它的设计和实现，相关的元算符，并揭示了元编译的若干用途。说明了包含和导入模块与元编译的关系。另外也介绍了元编译的一个最重要特例：宏功能，解释了太极语言中进行宏定义、宏调用和宏扩展的方法。

第十一章 定制语言：介绍了太极语言各种定制和扩展语言的方法。包括定制运算符，利用解析器算符扩展语法规则和控制解析，定制语句等，并解释了如何配合函数和宏来进行定制。

# 体例

## 正名

必也正名乎

名可名，非常名。本书很多地方花费了不少文字介绍概念，谈到可能用到的术语和名词。由于历史的原因和交流的需要，同一个对象，可能会有不同的名词来称呼它。同一个名字，也有可能被用来指代不同的对象。请读者在阅读本书的时候，多注意这些正名的内容。既要注意名字本身，也要联系上下文以求得完整的理解。

功能和特性

和其它高级语言的比较

# 编译过程和中间表示

# 太极图解

## **lisp**风格和**c**风格

一种以追寻优美为前提，一种以简单实用为原则。

一种是纯粹的艺术家，一种是实际的工艺家。都有自己的原则，都有自己的审美观。如同理论物理学家与原子能核电站的工程师，各自专工不同，侧重不一。

一个领域的日常运用离不开前者，但是一个领域的整体的发展离不开后者。

一般来讲，lisp风格的研究将成为地底下的根和土，而c风格的研究会成为地面上的枝和叶。

理论和技术的关系，这两个领域的不同探索者之间的关系，可以用牛顿与瓦特。

还有另外两个层面的对比：技术和产品的关系。这方面我们可以用Dennis Riche和乔布斯作为最显著的例子。

一种讲究实用，一种讲究实利。一种面向工具，一种面向客户。一种通过工具的改善本身来赢得内心的快乐，一种能通过唤醒客户的需求来满足自己的欲望。

上述看似差异很大的两种风格，其实它们不是分裂的，对立的，而是互补，可以融合的。随着技术的发展，就会有机会实现两种风格的融合。随着计算机和互联网的高速发展，只是从理论转向技术、工业，转向生产和实用的周期越来越快。

太极语言也可以看作两种风格发展走向融合的一个结果。

# 太极之禅

有无相生

难易相成

多少善恶

是非因果

人机相应

习惯自然

道名非常

一以贯之

# 结语

# 安装太极语言

安装

```
npm install taiji -g
npm install taiji
npm install taiji --save
npm install taiji  --save-dev
```

# 读入求值打印循环

太极语言命令行工具可以执行读入求值打印循环，用习惯的术语叫repl(read eval print loop)。repl是lisp开创的传统。通过一个交互式的工具，让用户能执行简单的任务，熟悉语法，了解函数的功能、参数配置等。绝大多数动态语言，脚步语言，比如python，ruby，javascript都有这个功能。某些现代的高级语言也提供了repl，例如haskell。

在shell下输入不带参数的太极命令，即可进入repl，如下所示：

```
$> taiji
```

现在系统进入了太极语言的repl。系统提示为

```
taiji>
```

我们可以输入太极语言的语句或表达式：

```
taiji> 1
1
taiji> print "hello world!"
hello world!
```

如果希望退出repl，请按CTRL+C。

下一节我们将介绍一些简单太极语言的语句或表达式。你可以在repl模式下体验这些示例。

## 一些简单的例子

根据上节介绍，我们进入太极语言的repl，然后会看到如下提示符：

```
taji>
```

在提示符状态下，我们可以尝试输入这些例子：

a = 1

b = 2

a

b

1+2

a+b

print "Hello world!"

if a==1 then print "a is ", a

for x in [1 2 3] then print x

# 更多的例子

```
let a=1 then let a=2 then print a
```

```
let a=1 then let a=2 then print a
```

```
let a=1 then { let a=2 then print a }; print a
```

```
do print a, print b where a=1, b=2
```

```
i = 0; do print i++ until i==10
```

```
array? #= (obj) -> ` (Object::toString.call(^obj) == '[object Array]')
```

```
array? # 1
```

```
array? # []
```

# 一个完整的程序

```
taiji language 0.1
print 'hello taiji'
```

太极语言的模块文件由两部分：模块头部和模块体。模块头部是从第一行开始以及随后具有更多缩进的所有各行，一直到第一行没有缩进的行（也就是第一列不是空格或制表符的非空白行）。从该行开始直到文件尾是模块体部分。

模块头部第一行，也就是整个文件第一行，必须以如下文字开始：“taiji language x.y，其中x和y是两个数字，分别代表主版本号和副版本号。解析器会检验这两个版本号，如果不是解析器接受的版本号，将报告一个错误。随后各行可以是任意的格式。可以将作者，许可证，邮件，范例，文档等内容放置在模块头部。

模块体部分包含了有效的太极语言代码，解析器将依据解析模块体的规则产生太极语言的解析结果表示，作为后续编译阶段的输入数据结构。

# 本章结语

本章介绍了安装太极语言的方法，命令行工具taiji的功能，交互式执行太极语言代码的方法，也给出了一些太极语言的代码示例，并解释了太极语言代码文件的特征和要求。通过这些内容，我们对太极语言有了一个初步的印象。

接下来，从第四章到第七章，我们将全面地一步一步介绍太极语言各方面的基础知识。下一章（变量和数据）首先介绍太极语言处理数据和定义变量的方法。

# 字符串

太极语言用'some string'，'''some string', "some string", """some string"""四种形式表示字符串。通过使用不同的形式，可以控制字符串是否转义和是否插补。

转义字符串：

字符串转义指将字符串中出现的一个转义字符序列替换为另一个字符。

'...'或"..."中的字符串中如果出现了转义字符序列，将根据转义规则转换成目标字符。'''...'''或"""..."""不会转义其中的字符，即使出现了合乎规则的转义字符序列。 太极语言转义字符序列及其转换规则遵从javascript的习惯。以下是一些常见的转义序列：\n表示新行字符，\r表示回车字符，\t表示制表字符，\表示字符"\"，\'表示字符''', \"表示""。

插补字符串

字符串插补是指将字符串中满足插补语法的表达式求值后将求值结果转换成字符串插补到结果字符串中。

太极语言中"..."和"""..."""形式的字符串是可以插补的。 当这两种字符串中出现了形如(operatorExpression), [list], {block}，$compacetClauseExpression的字段,太极语言将先求值(operatorExpression), [list], {block}，compacetClauseExpression，然后将求值结果的字符串表示插补到整个结果字符串中。

根据上述规则，可以总结如下：

- '''...''': 既不转义也不插补
- '...': 转义但是不插补
- """...""": 不转义但是可插补
- "...": 既转义又插补

太极语言中上述四种字符串都可以是多行字符串。

# 列表

太极语言以[]表示列表，或者叫数组。比如[1 2 3], [a b c], [1, 2, 3]

[1 2 3]和[1,2,3]是等价的表示。

[1, 2, 3 4, 5, 6]

表示一个二维数组

[abs 1 sqr 2] 表示[abs(1), sqr(2)]

# 字典

{. 1:2 .}

{.1:2; 3:4.}

{. 1:2; 3:4; 5:6 .}

{. 1:2; 3: 5:6 .}

{. 1:2; 3: 5:6 7:8 .}

空字典 {}

# 变量

高级语言都有变量。用变量可以代表数据，从而是程序更为通用。

# 变量声明

var a

# 变量赋值

a = 1

b = 1

print a, b

常量 const MON = 'Monday' const PI = 3.14

常量和变量

根据语言的不同，程序可以定义常量和变量。早期的javascript实现在语言层面不区分常量和可变变量，但是新的javascript增加了const关键词，允许定义常量。Javascript有程序库可以产生不可变的数据类型，可以保证其内容一旦产生，即不能再被修改。然而，和变量本身不能被修改，也就是说变量不能够被多次赋值，是两个概念。因为这只能够保证数据本身不可变，无法保证变量不被多次赋值。

coffee-script没有提供const关键字，所以变量都是可变变量。

太极语言赋值的时候，如果遇到的是本地作用域没有定义的变量，会自动添加变量声明，而且通过这种方式定义的变量默认是个常量。

标识符

变量的命名是编程的一个重要问题。

太极语言的标识符规则有javascript基本一致，区别是除了首字符以外，后续字符也允许使用!和？。比如begin!, integer?等也是合法的太极语言标识符。太极语言标识符在需要转换到javascript标识符的时候，会将不合javascript规定的字符进行转换。现在的实现是都转成字符$。

# 变量的作用域

作用域控制变量定义的有效范围。高级语言最常用到词法作用域和动态作用域。现代语言更是以词法作用域作为首选。

在javascript中变量如果没有用var进行声明，而是有赋值产生，那么默认为全局变量。这是非常影响程序的模块性的一个语言特征。因为很可能在不知情或不小心的情况下修改了全局变量，从而改变其它函数、模块的功能。

局部作用域代码块

带有新一层本地作用域的代码块称之为局部作用域代码块，简称为局部代码块，或者局部块。太极语言有几种方法产生局部作用域代码块。使用let，letrec!, letloop!, block!语句, 函数定义语句。作为编译的目标语言，从javascript的角度来看，函数作用域和其它局部块有所不同，因为javascript自身并没有块级作用域，所有的var变量都位于函数级或者文件级别(浏览器下面还有文档级）

let a = 1 then let a = 2 then print a print a

a = 1 block! a = 2 print a print a

coffee-script的处理要好一些。赋值的变量要在模块中各层作用域中都没有定义过，那么总是默认产生一个本地变量声明。但是，如果在同一模块包含该变量的外层作用域预先定义了该变量，那么变量的赋值的是这个外层的变量。如此处理，模块化的程度要高一些，但是，还是有可能出现不注意而错误地修改外层变量的情况。实际上我在开发太极语言解析器中就曾经因此而导致了bug，花费了不少调试时间。

给外层变量赋值

因此，在太极语言中，默认情况下赋值总是针对本地的变量。如果赋值语句左边是一个变量，而该变量在该作用域下是第一次被赋值，那么太极语言将自动产生一个变量声明语句。

如果要对外层变量赋值，必须在变量名之前添加@@符号。

@@outerVariable = 1

# 本章结语

# 注释

//././//

# 基本命令

赋值运算 右结合 普通赋值 扩展赋值

打印 print 或console.log

let 和 where

单引式

求值(eval)

回引与消引 quasiquote

引用式

引用号~，波浪号，为什么不用单引号？尊重主流编程语言将单引号作为单引号字符串的习惯。 波浪号，其它编程语言一般用作位反。而取消求值某种程度与此有类似的涵义。而位反是个不常见的操作，移作它用不会有太多不便。 位反符号哪里去了？ ~是替代的位反符号

回引式 回引号` 键盘最左上角, 这也是lisp系语言的习惯。

消引号^ lisp系语言使用逗号，但是为了遵重包括主流高级语言以至于更一般的习惯，改用^。表示项的含义，将被回引的项重新项上来求值。 主流编程语言中，^一般用作位异或运算符。太极语言改用\^作为替代。

消列引号：lisp系语言使用,@，但是为了遵重包括主流高级语言以至于更一般的逗号使用习惯，改用^@。表示项的含义，将被回引的项重新项上来求值。

# 普通运算符

中缀算符，或称作二元算符。

前缀，后缀运算需要预先指定。

运算符，或者称作算符

系统预置的运算及其优先级

==、=== 和 !=、!===

一元算符 前缀运算符 前缀运算符是可以和二元运算符有重复的。比如-既用做加法的二元运算符，也表示求相反数的前缀运算符

后缀运算符不应该与二元运算发生冲突，或者说同一个符号不应该同时用做二元运算符和后缀运算符。

- 
  - 1 应该写(+ + 1), 否则是不合法的。因为这样将被解析为[+ + 1]

打印 print 或console.log

扩展运算符

# 算符优先级

系统预置的运算及其优先级

==、=== 和 !=、!===

空格、换行和缩进影响求值顺序

# 结语

# 块结构

语句块：一组缩进的代码行，关键字语句中的一个分句中一组句子或子句等等都构成一个语句块。不管是单个语句或者都个语句都可以用{}封包起来。{ print 1 2 3 }

[设计注记] 太极语言是一个缩进语法的语言。现有的其它缩进语法的语言没有显式的语句块定界符。比如python和coffee-script。那么为什么太极语言还要提供{}定界符来封包语句块呢？有心人很容易产生这个疑问。实际上，这个问题有一个自然而直接的答案，因为太极语言最强大的两项特性驱使它必须要有{}这种形式的代码块：元编译能力和动态语法能力以及它们用到的一组算符。

block!局部块语句

在上一节介绍过，利用block!可以定义一个带有新的内层局部作用域的块。

a = 1 block! a = 2 print a print a

# 条件执行

if语句 有时候又叫双分支语句，分支语句。这是所有高级语言最常见的语句。它的语法如下。 if condition then action [else action] if condition then action else action

if condition then action else acton

if condition then action else action

或者象这样类似python的写法： if condition: action [else action]

if sex=='female' then print "She is a woman." else print print "He is a man."

我们看到，和coffee-script相比，太极语言中的语句格式（特别是then分句的写法）一般要更加自由一点。

switch-case语句

对于多分支，太极语言提供了switch-case语句。在可能的情况下，switch-case语句会被转换为javascript的switch语句。否则，将转换为if-else if-else语句。

switch-case

switch day: case ["Mon", "Tue" , "Wen" , "Thu" , "Fri"] then print "I'm working." case "Sat" then print "I'm playing." else print "I'm having a rest."

# 重复执行

c-for语句 `for init; test; step then body`

`for i=0; i<10; i++ then print i`

for-in和for-of语句。

`for item [index] in range then body`

`for key [value] in hash then body`

```
guests = {. "张三": 39; "李四": 45; "王五": 22 .}

for guest in guests then
  print guest+"是我们的客人。"
```

```
for course score in sheet:
  print "$course得了$score分"
```

while语句

```
i = 0
while i<5 then
  print "闹钟响$i次了。"
if day=="Sat" or day=="Sun" then print "继续睡。"
else print "我必须起床了。"
```

do-while语句

```
do letter = tryGetALetterFromMailBox() while letter
```

do-until语句

```
do run() until outOfGas()
```

break语句和continue语句

```
tickingForTimeBomb = 30
while 1
  if --tickingForTimeBomb == 0 then break
print 'boom!'
```

```run() while 1 if not atStation() then print "没到站"; continue if reachFinalStop() then print "终点站到了"; break print "到站了" if noOneWantLeft() then continue stop() passengerLeft() run()

label!语句

label@ clause

i = 10 label1@ { while 1 while 1 if i = 0 then break i-- }
```

# 异常

javascript具有try-catch语句，通过它可以处理异常。太极语言提供了对于的构造

try语句

try fn() catch e then body else print 'other error' finally print 'always do this'

throw语句

抛出异常。

产生javascript的throw语句。

# 本章结语

下一章将介绍更多组织程序结构的方法：函数，类和模块。

# 函数

函数是一等公民 先求值参数，再求值函数整体。

(a) -> [print a] (a) => [print a]

传值方式 传引用方式带来的问题 javascript下的解决 (function(a){...})(a)
http://stackoverflow.com/questions/750486/javascript-closure-inside-loops-simple-practical-example
coffeescript的解决方案：do (x=x) -> ... http://rzrsharp.net/2011/06/27/what-does-coffeescripts-do-do.html

变量环境 一些简单的函数
所有变量都是局部变量 javascript: 上层声明过赋过值的自动在下层函数不再用var声明 taijilang将改为所有的都自动用var声明为局部变量，除非用scope关键字排除

所有变量自动是常量，除非用mutual声明？

递归函数

系统自动变换为循环的递归函数 尾递归变换 不是尾递归也可以进行变换的情况 loopable () -> ...

类

模块

# 结语

编译过程

中间表示

中间表示是太极语言的核心一环，理解太极语言的中间表示将对理解太极语言整体概念、语法及解析和具体使用，都会有很大的帮助。因此，在具体讨论其它内容之前，让我们先来看一看太极语言解析、编译、求解过程中都将涉及到的中间表示。我们也可以把这一中间表示看作一个语言，因此也可以将中间表示称之为中间语言。将这种形式的程序或代码称为中间代码，有时候也叫做中间程序。 太极语言采用json（http://www.json.org/）作为自己的中间语言。json除了原子以外，包括数组（也叫列表）和对象两种结构化数据。作为lisp风格的语言，理论上我们完全可以选择只使用数组这一种结构化容器。但是出于实用性的目的，并且为了和javascript，web开发，html等主流技术保持兼容性，太极中间语言也允许使用对象。

json

列表

对象（哈希表，映射）

字符串

符号

文字串

命令式

转义

# 代码解析

# 元编译和元变换

编译变换

表达式转换

# 编译优化

优化指令 循环外提 函数外提 编译时间计算 无io作用的代码段 尾递归优化，普通递归优化

## optimization 1:

## if 1 then a else b ---> a

## if 0 then a else b --> b

## 1+2 -> 3, etc.

## a and not a

## while 0, 1==0, 1==2, false, undefined, null, '', etc

## optimization 2:

## below need a has no side effects ( a is not function call(), etc.)

## if a then if a then b else c ---> if a then b

## if a then if not a then b else c ---> if a then c

## if not a then if a then b else c ---> if not a then c

## if not a then if not a then b else c ---> if not a then b

## if (if a then b else c) then d else e # no optimzation

## if (if a then b else c) then if a then d else e -->

## if a then if b then d

## else then if c then e

## while 0, while false

## optimization 3: assign optimization

## property optimization is not executed: be careful that getter, setter, watcher

## short distance

## conservative

http://www.refactoring.com/catalog/replaceIterationWithRecursion.html greatest_commonDivisor = (a, b) -> if (a > b) then return greatest_commonDivisor(a-b, b) else if (b > a) then return greatest_commonDivisor(a, b-a) else return a

greatestCommonDivisor = (a, b) -> while 1 if (a > b) then a -= b else if (b > a) b -= a else return a

代码产生

# 语法及解析方法的设计原则

太极语言设计语法的原则

对空白敏感的语法 排版的关键在于安排页面的空白，通过空白组织文字的结构。 增强可读性的关键是让程序员一瞥之间就能掌握程序的整体，包括整体结构和整体意义。

少即是多，多即是少。 更少的括号，更少的标点，更紧凑的语法，更少的限制，加上适当的空白，意味着能在更短的篇幅提供安排更多的内容，让程序员在更短的时间内输入更多，理解更多。 反过来，需要更多括号，更多标点，规则上增加更多的限制，不依赖人类可读的空白，而是面向计算机的括号匹配来组织结构，将意味着输入更慢，理解更慢，同样的内容需要更长的篇幅。

恨括号，shift键，标点 尽量减少使用括号，特别是远距离配对的括号。 利用行、缩进等方式，取消传统lisp风格语言中对括号的需要。 减少使用依赖shift键的符号。 使用[]而不是()作为s表达式的定界符，允许使用[]表示参数列表，在不影响理解和打破习惯的前提下尽量不使用需要shift键的符号作为常用的语法标记， 减少使用不必要的标点 参数的分隔，数组项目的分隔等不需要使用逗号。

缩进语法 表达式

太极语言在语法上带有两个小妖：一个叫精细鬼，一个叫伶俐虫:)，换句话说，太极语言拥有精细、灵活的语法控制能力。

注释 行内注释：/ /：视同空格 行注释 //：直到行尾 代码块注释 行首开始的单个/， 将注释随后的代码块，包括相同缩进的连词块。 块注释 行首开始的/.，后面的内容只要比该行有更多缩进，则为缩进注释

正规表达式： 为便于解析，太极语言对正规表达式有一个比javascript更为简单的规则：

在排除行注释//, 行内注释/*引导符的情况下，赋值号或小括号后面的单个“/”开启一个正规表达式。正规表达式必须在单行内，不能跨行和续行。 其它情况下/都不引导正规表达式，而是除号或别的符号。 行首的/ 将引导代码块注释

字符串： 转义和插补 转义：以\引导的一系列字符被替换为另一个字符。 插补：字符串中形如(...), [...], [/.../], {...}的部分先安装表达式进行求值，然后将该值的字符串表示插补到原字符串中。 '"..."'或"""..."""内的字符串将不被转义。 '...'或"..."内的字符串将被转义：将按照javascript的规则进行转义。例如相连的\和n会被替换为换行符号'\n' "..."或"""..."""内的字符串可以有插补表达式。 '...'或'"..."'内的字符串不会被插补。 原始字符串，原始插补字符串 转义字符串，转义插补字符串 转义字符串内部可以有续行符号

主要语法成分 表达式 模块，块，缩进块列表，连词子句，括号运算式

原子成分 字符串 数字 变量名：在算式中只允许javascript风格的标记符，另外允许其中出现#或？，但是不得出现其它符号 运算符：在算式中不允许出现#或？作为运算符。 \identifier: generate javascript word, identifier, etc.

基本项 括号表达式 封闭在括号中的表达式。默认使用[]。 为什么选用[]作为sexpression的默认括号？一、代码即数据，[]是各种现代语言中列表数据类型的通用括号、二、[]击键方便，不需要shift 调用式 对象元素访问式 js表达式：在小括号内，由若干操作数通过若干运算符根据一定的优先级组合而成的式子。 冒号式：以冒号引导，以连词、分号或换行结束的若干项目的列表。 连词子句：以连词引导，以分号或换行结束的若干项目的列表

行与块 模块 一组顶级的块

块 引导行 如果引导行由多个分句组成，则不允许跟随缩进子块和连词子块。 缩进子块 缩进子块由一组具有相同缩进的块组成。所有缩进块的起始列号大于引导行的起始列号。 连词子块 以连词开始的一个块

行 分句，连词子句，逗号小句 子句：以分号或换行结束的若干项目的列表 关键词语句：以关键词开始的子句，具有特殊的语法，类似于其它语言中的语句 连词子句：以连词开始，以分号、连词或换行结束的若干项目 逗号小句：以逗号、分号、连词或换行结束的若干项目的列表 连词：以冒号结束的词 if 1 then 2 else 3

```
    if 1
    then 2
    else 3

    使用标准的、习惯的连词，促进交流，增加可读性

  关键字：
    作为符号的字符串，出现在表头将影响程序的求解过程，出现在其它部位，taiji求解器将利用该符号从环境中查
    作为命令的字符
    if
    连词：then else
```

解析指令

3ab(1+2) sinAcosB a1+b2 A3+D5 AiHj

控制编译的与开发相关的指令 可以指定某一块在某种状态下不予编译。比如 (@quoteExpression =)@test quoteExpression = info { 表示只有在测试状态下才对该段代码予以编译，因为只有在测试状态下才需要探测某些私有成员

解析后的运算符变换

语法规则 列表综合 [expression for item in list if condition]

哈希综合 {expression for item in list if condition }

语法类型 gulp pipe stream: coffee copy clean mocha 否则 >> 应该表示向右移位运算 src >> coffee(...) >> dest

a = chan('a'), b = char('b') na na+ a *na* a! na, nb, na!, na!

w = literal('if') w!, spaces!, x=clauses!, colon!, then!

局部语法

单变量表达式（代数表达式）

数据块 [/ /]

怀念中学时代 当我们刚离开稚气的童年，迈入活力的少年，中学老师教了我们很多的知识，也培养了我们很多习惯。然而，不同的老师总是传授不一样的内容，即使是看起来应该一样的事务。比如，英语老师告诉我们，a是英文的第一个字母，abc是入门的第一课。，而数学老师说a可以表示随便什么东西。abc表示a×b×c，3a+4b表示3×a+4×n。当然，我们的英文作业本上从来没有出现过3a之类的东西。如果出现了，也许老师会疯掉。后来，我们成长为青年，进入了大学，有的人接触了一个叫计算机的东西，教这门课的老师颠覆了我们的很多光年。a×b×c不能写成abc，也不能写成a×b×c。从来没有学生这样尝试过。也许老师看到学生有试图这样尝试的意图，就已经把他赶出了课堂。 那么，那个老师是正确的？哪个时代有更值得怀念？为了工作，我们总是坚持大学养成的习惯，而内心，却总是希望回到更快乐的少年时光。太极语言给我们一个不需要纠结的机会。

# 关于解析器实现过程的注记

解析器是设计一门新语言的关键部分之一，也是非常困难的一项任务。我们可以用解析器产生器来完成这项任务，比如lex/yacc相关的系列产品，包括bison，jison，ply，antlr以及基于peg的解析器产生器等等。虽然这些解析器具有非常强的功能，产生的解析器也具有极其优化的执行速度，有很多的项目和语言都使用了这些工具，经过了长期的实际考验。然而，根据太极语言的设计目标，我无法选用这些产品。因为一直以来我都希望设计一门语法可以自由动态扩展的语言，基于解析器产生器的方案无法实现这一目标。为此，我做了长期的探索和研究。手写解析器也是很多语言选择的一种技术。但是以前的手写解析器其代码都相当复杂，不直观，可读性不好。特别是左递归语法的问题长期以来都没有简单的解决方案。

# 词法

太极语言没有单独的词法解析阶段。编写词法解析规则，匹配词法标记或单位的方法和语法解析的方法是一致的。

解析之前，在初始化阶段，解析器调用preparse函数，根据给定的文本参数执行一个预解析阶段。这个阶段将分析文本的行列信息，保存各行的起始位置和缩进位置。

太极语言解析器中以下匹配函数完成与词法相关的任务

taijiIdentifier

jsIdentifier

literal

symbol

escapeSymbol

escapeString

identifier

number

string

spaces

space：太极语言的行内注释和空白注释被识别为

bigSpace：跨行的空白，必须至少包含一个新行。

newline：新行，可以是\r, \n, \r\n, \n\r等四种组合。

# 主要语法成分

# 模块文件

模块头 模块体

表达式 模块，块，缩进块列表，连词子句，括号运算式

行与块 模块 一组顶级的块

块 引导行 如果引导行由多个分句组成，则不允许跟随缩进子块和连词子块。 缩进子块 缩进子块由一组具有相同缩进的块组成。所有缩进块的起始列号大于引导行的起始列号。 连词子块 以连词开始的一个块

行 分句，连词子句，逗号小句 子句：以分号或换行结束的若干项目的列表 关键词语句：以关键词开始的子句，具有特殊的语法，类似于其它语言中的语句 连词子句：以连词开始，以分号、连词或换行结束的若干项目 逗号小句：以逗号、分号、连词或换行结束的若干项目的列表 连词：以冒号结束的词 if 1 then 2 else 3

```
    if 1
    then 2
    else 3

    使用标准的、习惯的连词，促进交流，增加可读性

  关键字：
    作为符号的字符串，出现在表头将影响程序的求解过程，出现在其它部位，taiji求解器将利用该符号从环境中
    作为命令的字符
    if
    连词：then else
```

连词及其搭配

其它语法特征 续行符号 行尾紧跟\ 回退缩进提示符号 行首为\

原子成分 字符串 数字 变量名：在算式中只允许javascript风格的标记符，另外允许其中出现#或？，但是不得出现其它符号 运算符：在算式中不允许出现#或？作为运算符。 \identifier: generate javascript word, identifier, etc.

基本项 括号表达式 封闭在括号中的表达式。默认使用[]。 为什么选用[]作为sexpression的默认括号？一、代码即数据，[]是各种现代语言中列表数据类型的通用括号、二、[]击键方便，不需要shift 调用式 对象元素访问式 js表达式：在小括号内，由若干操作数通过若干运算符根据一定的优先级组合而成的式子。 冒号式：以冒号引导，以连词、分号或换行结束的若干项目的列表。 连词子句：以连词引导，以分号或换行结束的若干项目的列表

怀念中学时代 当我们刚离开稚气的童年，迈入活力的少年，中学老师教了我们很多的知识，也培养了我们很多习惯。然而，不同的老师总是传授不一样的内容，即使是看起来应该一样的事务。比如，英语老师告诉我们，a是英文的第一个字母，abc是入门的第一课。，而数学老师说a可以表示随便什么东西。abc表示a×b×c，3a+4b表示3×a+4×n。当然，我们的英文作业本上从来没有出现过3a之类的东西。如果出现了，也许老师会疯掉。后来，我们成长为青年，进入了大学，有的人接触了一个叫计算机的东西，教这门课的老师颠覆了我们的很多光年。a×b×c不能写成abc，也不能写成a×b×c。从来没有学生这样尝试过。也许老师看到学生有试图这样尝试的意图，就已经把他赶出了课堂。 那么，那个老师是正确的？哪个时代有更值得怀念？为了工作，我们总是坚持大学养成的习惯，而内心，却总是希望回到更快乐的少年时光。太极语言给我们一个不需要纠结的机会。

连词及其搭配

其它语法特征 续行符号 行尾紧跟\

太极语言在语法上带有两个小妖：一个叫精细鬼，一个叫伶俐虫:)，换句话说，太极语言拥有精细、灵活的语法控制能力。

其它语法特征 续行符号 行尾紧跟\ 回退缩进提示符号 行首为\

解析指令

3ab(1+2) sinAcosB a1+b2 A3+D5 AiHj

控制编译的与开发相关的指令 可以指定某一块在某种状态下不予编译。比如 (@quoteExpression =)@test quoteExpression = info { 表示只有在测试状态下才对该段代码予以编译，因为只有在测试状态下才需要探测某些私有成员

解析后的运算符变换

语法规则 列表综合 [expression for item in list if condition]

哈希综合 {expression for item in list if condition }

语法类型 gulp pipe stream: coffee copy clean mocha 否则 >> 应该表示向右移位运算 src >> coffee(...) >> dest

a = chan('a'), b = char('b') na na+ a *na* a! na, nb, na!, na!

w = literal('if') w!, spaces!, x=clauses!, colon!, then!

局部语法

单变量表达式（代数表达式）

数据块 [/ /]

怀念中学时代 当我们刚离开稚气的童年，迈入活力的少年，中学老师教了我们很多的知识，也培养了我们很多习惯。然而，不同的老师总是传授不一样的内容，即使是看起来应该一样的事务。比如，英语老师告诉我们，a是英文的第一个字母，abc是入门的第一课。，而数学老师说a可以表示随便什么东西。abc表示a×b×c，3a+4b表示3×a+4×n。当然，我们的英文作业本上从来没有出现过3a之类的东西。如果出现了，也许老师会疯掉。后来，我们成长为青年，进入了大学，有的人接触了一个叫计算机的东西，教这门课的老师颠覆了我们的很多光年。a×b×c不能写成abc，也不能写成a×b×c。从来没有学生这样尝试过。也许老师看到学生有试图这样尝试的意图，就已经把他赶出了课堂。 那么，那个老师是正确的？哪个时代有更值得怀念？为了工作，我们总是坚持大学养成的习惯，而内心，却总是希望回到更快乐的少年时光。太极语言给我们一个不需要纠结的机会。

# 本章结语

本章介绍了太极语言在语法以及解析方法（包括解析器及其词法解析）上的特点。理解这些特点对于定制和扩展太极语言会有很大的帮助。下一章我们就来介绍定制和扩展太极语言的方法。

元算符

太极语言提供了一组元算符，包括#，##，#/，#&，#/=，#&=，#-等。本章将解释这些算符的功能和用法。

元算符使用示例

# (1+1)

# (#(1+2) + #(3+4))

# (1+2) + #(3+4)

3+.#(1+1)

# ~ 1+1

## a=1

a#=1

# (a=1)

a#=1;#a

# if 1 then 1 else 2

if 1 then #1+2 else #3+4

if 1 then ##1+2 else ##3+4

compileNoOptimize if 1 then ##1+2 else ##3+4

array? #= (obj) -> ` (Object::toString.call(^obj) == '[object Array]')

array? # 1

array? # []

`# if 1 then a else b` 编译时报告错误：`fail to look up symbol from`environment:a

# #-{ print 1 } 编译结果是 `console.log(1)`

`#{ -> #- { print 1 }})()` 编译结果是 `[print, 1]`

# 元编译的用途

例子 在下面的例子中，text是解析器正在解析的整个文本，cursor是当前全局解析位置，parserLineno和parserRow分别是全局行号和列号。 考虑到处理换行符号，匹配任意文字的解析函数可以这样写 literal = (string) ->
start = cursor; lineno = parserLineno; row = parserRow for c in string if c==text[cur++] if c=='\n' then parserLineno++; parserRow = 0 else if c=='\r' then parserRow = 0; continue else parserRow++ else cursor = start; parserLineno = lineno; parserRow = row; return true

但是在实际应用中，绝大多数情况下我们都不需要考虑换行符号，因此使用如下的函数就可以了。 literal2 = (string) ->
start = cursor; row = parserRow
for c in string if c==text[cursor++] then parserRow++ else cursor = start; parserRow = row; return true

或者更快捷的实现 literal2 = (string) -> if text[cursor...cursor+length]==string then cursor += length; parserRow += length-1; true

下面是解析while语句的函数，作为例子，做了一些简化： whileStatement = > if literl('while') and (test = parser.clause())? and literal('then') and (body=parser.block() or parser.line()) return ['while', test, body]

出于速度的原因，我们应该将上面whileStatement实现代码中的literal替换成literal2。 对于这样一个功能简单的小问题引入两个函数，无疑会增加API的数量，从文档提供和库用户的学习使用的角度来讲都是一种负担。另外仔细分析，上面的通用文字处理函数literal还有优化的空间，由此引出了下面第二种方案。

第二种方案 literal = (string) -> i = 0; lineNumber = 0; length = string.length while c = string[i++] if c=='\n' then lineNumber++; row = 0 else if c=='\r' then _r = true; continue else row++ if lineNumber==0 if _r then -> if text[cursor...cursor+length]==string then cursor += length; parserRow += length-1; true else -> if text[cursor...cursor+length]==string then cursor += length; parserRow += length; true else -> if text[cursor...cursor+length]==string then cursor += length; parserLineno += lineNumber; parserRow += row; true

在此实现下，whileStatement的代码如下： whileStatement = > if literal('while')() and (test = parser.clause())? and literal('then')() and (body=parser.block() or parser.line()) return ['while', test, body] 然而，这样写并不能提升速度，反而会有性能损失，因为每次都必须要计算函数闭包。为了达到性能提升的目的，我们应该把对于的闭包计算提到函数之外： 因此应该这样写 while = *literal('while'); then* = literal('then')

whileStatement = > if while*() and (test = parser.clause())? and then*() and (body=parser.block() or parser.line()) return ['while', test, body]

元计算派上用场： 这种情况正好是太极语言的元计算能力的用武之地。 whileStatement = > if #(literal('while'))() and (test = parser.clause())? and #(literal('then'))() and (body=parser.block() or parser.line()) return ['while', test, body] 太极语言的元计算让我们鱼和熊掌兼得：同时拥有运行效率和编程效率。

非正式文档状态声明

本文件夹保含的所有文档都处于草稿状态，可能已过时，不完整，不准确甚至不正确。在发布新的声明之前，所有文档目前权当非正式的参考，不作为任何依据，阅读者敬请留意此声明。

呼神护卫

黑魔法 黑暗给了我黑色的眼睛，我却用它来寻找光明。 —— 顾城

“是守护神到来的时候了，但是这一次并没有人来帮忙，——突然他想通了——他明白了，他并没有见到他父亲——他看见的是——他自己” 他看见它低下头冲向成群的摄魂怪……现在它绕着地上的黑影疾驰，然后摄魂怪纷纷后退，散开，隐入黑暗中……他们不见了了。守护神转过身，穿过平静的湖面缓缓的跑回到哈利身边。它不是马，也不是独角兽，而是一头牡鹿，象天上的月光一般皎洁，向它在他走来…… —— J.K.罗琳《哈利.波特——阿兹卡班的囚徒》

# 包含和导入模块

# 比较太极语言和**lisp**的宏

宏扩展

# 本章结语

# 定制运算符

# 解析器算符

定制关键词语句

定制赋值语句

# 本章结语

# 太极之禅

有无相生

难易相成

多少善恶

是非因果

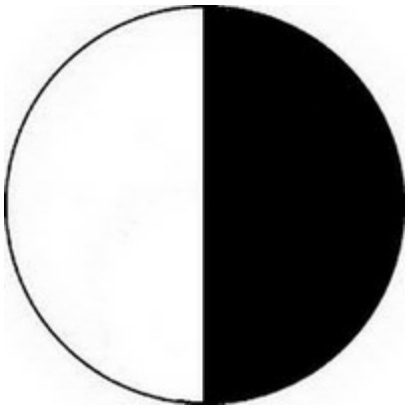
人机相应

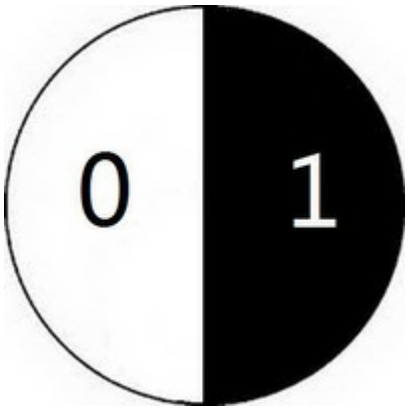
习惯自然

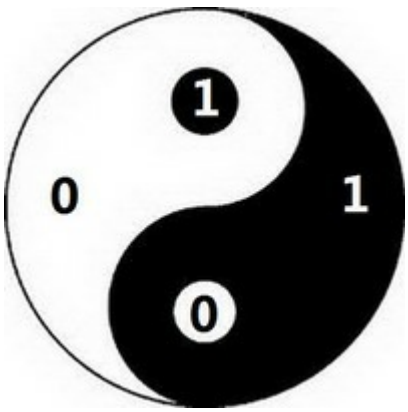
道名非常

一以贯之

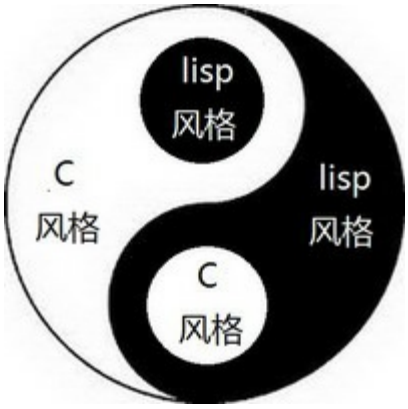
太极图解






C
风格
lisp
风格


lisp
风格
C
风格
lisp
风格
C
风格


代码
数据

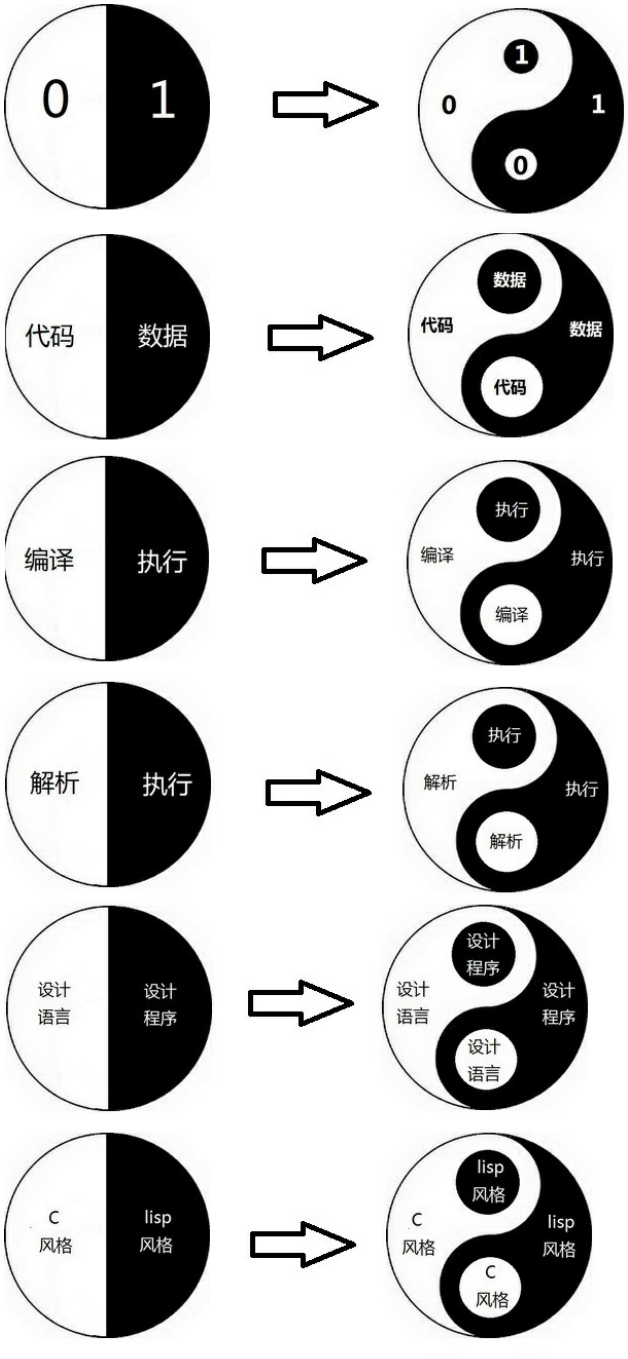

数据
代码
数据
代码


编译
执行


执行
编译
执行
编译


解析
执行


执行
解析
执行
解析


0
1
1
0
0
1
代码
数据
数据
代码
代码
数据
编译
执行
执行
编译
编译
执行
解析
执行
执行
解析
解析
执行
设计
语言
设计
程序
设计
程序
设计
语言
设计
语言
设计
程序
C
风格
lisp
风格
lisp
风格
C
风格
C
风格
lisp
风格

太极图解

# 常见问题列表 (FAQ)

- 为什么这门语言取名叫太极语言？

从太极图解可以看到部分答案。

- 有那么多语言了，为什么又来一门新语言？

实际上，编程语言不是够多了，而是太少了。日常生活中，人们说话从来不需要遵循僵死的语法规则，而是遵照活泼的历史和现实习惯。每个人都象尤达大师，他/她的表达习惯和句式或多或少总和别人有不一样之处。太极语言将便利人们创造新语言。每一个程序员都能创造新语言。

- 太极语言有什么用？

每种新东西出来，总是不可避免地被问到这个问题。提出这个问题，有的人是在认真地寻求答案，有的人只是不屑地表示怀疑。时间是这个问题的最好答案。真正有用的新东西最终会让那些不屑的怀疑论调显得很可笑。因为真正有用的新东西确实很少见，而且往往要经过很长很长的时间才能体现价值，所以，不经思考的怀疑总是廉价而且风险很小。

对此，我的具体答案是：太极语言可以让我们提高编程效率，以一种新的方式和思维来看待编程，完成更复杂的任务，编写更优化的程序，设定不同的优化方案。从下一个附录，我们还可以看到更多前景。

# 与coffee-script相同点的比较

## 相同点

- 编译到javascript
- 缩进语法
- 支持将for，while等语句作为表达式使用
- 支持隐式变量申明。对新变量赋值自动产生局部变量声明。
- @代表this，::代表prototype
- -> 和 => 定义函数
- 支持省略参数(x, y..., z) ->
- 支持默认参数(a, b, x=1, y=2) ->
- 支持自动返回最后语句的值
- 支持字符串插补, 在coffee-script中是"...#{expression} ..."。

## 不同点

- taijilang超越javascript；coffee-script就是javascript。

  太极语言的丰富的特征使得它不再与javas语言cript具有一一对应的关系，有点类似于C和汇编， 在语言金字塔上位于一个更高的位置。而coffee-script的口号是：coffee-script is just javascript, 它追求的是和javascript的一一对应。

- 太极语言支持lisp风格的宏，coffee-script没有宏功能。
- 太极语言支持元语言，或者说编译时间计算，coffee-script没有元语言功能。
- 太极语言支持动态解析时间计算，动态控制语法。coffee-script没有此功能。
- 太极语言支持根据空格改变算符优先级，coffee-script不支持。

  象 1+2 *3+4* 这样的算式，在太极语言中解释为*(1+2)*(3+4), 而在其它编程语言中都是解释为1+(2*3)+4。太极语言的处理顺应了普通人的阅读直觉。

- 太极语言中任何构造都是表达式，即使是break，continue，return。而这三种构造在coffee-script中不能作为表达式使用。
- 太极语言可以声明常量，coffee-script不能。
- 太极语言中，函数作用域内的赋值总是产生函数内部局部变量。而coffee-script则如果外层词法作用域中已经有同名变量，则将不产生局部变量，赋值将针对外层变量有效。coffee-script比javascript进步一层，但是，在某种情况下还是会出现因为无意中覆盖了外层变量而导致难以觉察的编程错误。太极语言中如果需要对外层变量赋值，必须显式使用@@varName。
- 类似于livescript，太极语言可以通过在->和=>前加“|”抑制自动返回最后一语句的值。coffee-script不支持此特征。
- 太极语言中任何字符串都可以是多行字符串。coffee-script中'...'和"..."是单行字符串。即使写在多行也会去除换行符合并成单行。
- coffee-script只支持用#{...}向字符串插补表达式，太极语言有多种方法，并且比coffee-script的更简捷：$expression, {}, [], ()。
- coffeescript支持literature programming，太极语言不支持。
- coffee-script支持#引导的行注释和###...###块注释。太极语言支持//行注释，/.引导的缩进块注释，/.../注释和/注释代码块

# 相关资源

## 项目地址

太极语言github项目：github.com/taijiweb/taijilang

发布在npm的包taiji：npmjs.org/package/taiji

# 交流

google 邮件组：google groups: taijilang

google+帐号：taiji lang

QQ群: 太极语言 194928684

# 将来的计划

- 自举

用太极语言实现太极语言。实现了自举的太极语言将更有利于实现下属两项目标：以其它语言作为目标语言，使得太极语言可以将元语言和目标语言配置为不同的语言。

- 各种语言作为目标语言

当前太极语言以javascript作为目标语言。可以考虑以其它语言作为目标语言。相对而言，动态语言更容易实现，比如python，ruby，perl，php等等。静态语言技术上困难要更多一些，但是也不存在无法克服的障碍。
Swift，Go，C++/C，Objective C等等都可以作为目标语言。太极语言的设计特点是有利于这项工作的，因为它的中间表示是通用的json和列表结构，这对于简化实现代码很有帮助。

- 元语言和目标语言配置为不同的语言

现在太极语言是以同一种语言作为元语言（编译时间语言）和目标语言（运行时间语言），具体说是javascript。这种方法当然有很多优点。但是，采用不同的语言作为元语言和目标语言也是可能的，也可能存在某些方面的优势。

- 用某种更快的语言重新实现太极语言，比如go或者C语言

太极语言现在的实现还没有充分考虑自身的优化。特别是解析过程，当前的手写解析器实现还存在很多的优化空间。另一个优化的途径是从动态语言切换到静态的编译型语言，比如用go或者C语言。也许用go是一个不错的选择，因为go语言开发效率高，而且也具有很高的运行时间效率。

- 基于太极语言的模板语言，比如html模板，xml模板，css预处理语言等

- 太极语言web框架

我启动太极语言项目的直接触发点是这样的：在实现基于web的自然语言编程系统的时候，我用到了angularjs，理解到它双向数据绑定的强大。因此，我产生了这样的想法：能否将这种能力扩展到服务器端，建立一个前后端统一的框架。由此，我开始查找相关项目和资料。我很熟悉jade模板语言，但是它无法提供我希望的功能。我探索其它的github项目，偶然遇到了lispyscript这一个项目。在我以前实现dao系统的基础上，lispyscript再次触动了我的灵感。

太极语言产生于web开发的经历中，太极语言的功能也让它很适合成为新一代web开发的工具，应该有可能用它实现我提到的上述web开发框架。

- 用太极语言实现更优化的库。比如underscore或lodash。

# 目录