**Itclx源码解析----刘淼堂**

[**【itclx源码解析01】 class方法**](http://www.cnblogs.com/nemolmt/p/6245822.html)

首先看itclx的命名空间，class classEnd oo New MsgTrust MsgSend Struct\_Init Dim Lassign 这么几个函数，一个一个解析。

namespace eval itclx {  
namespace export class classEnd oo New MsgTrust MsgSend Struct\_Init Dim Lassign  
variable \_PublicDebug 0 ;# 该值不为0时，所有通过oo的操作都为public权限  
}  
;

**class**

itclx解析入口 从demo开始。

demo代码：

[复制代码](javascript:void(0);)

1 proc CLASS-Father {} {}

2 class Father {

3 public variable name {NoOne}

4 public variable money {0}

5 public common people {0}

6 constructor {{mymoney 500}} {

7 set money $mymoney;

8 puts "has money $mymoney";

9 incr people

10 }

11 }

12 proc Father-::test1 {} {

13 puts "this is test1 method"

14 }

15 classEnd Father

16 proc test {args} {

17 set p [New Father {\*}$args]

18 puts "object,$p"

19 puts [.$p.money]

20 puts [.$p.people]

21 }

[复制代码](javascript:void(0);)

1、package require itclx

itclx是拓展开发的面向对象拓展包。

从里面看 首先 proc CLASS-Father {} {} 的作用是 定义为函数开始的标志，没有任何意义 只是作为类似于函数列表显示的作用。

首先第一个命令为class,找到itclx的函数定义：

[复制代码](javascript:void(0);)

1 proc itclx::class {className clsDefine} {

2 #puts "$className,$clsDefine"

3 set className [\_nmspFull $className]

4 #puts "find:[::itcl::find classes $className]"

5 if {[::itcl::find classes $className]!=""} {::itcl::delete class $className}

6 set ::itclx::\_ClassDefine($className) $clsDefine

7 set ::itclx::\_ClassBody($className) [\_class\_extBody $className $clsDefine]

8 #set ::itclx::\_ClassIfBuilded($className) 0

9 #puts "clsBody\_ext={$clsBody\_ext}"

10 #定义类的 命名空间 classname-

11 #puts "::itclx::\_ClassBody($className): $::itclx::\_ClassBody($className)"

12 uplevel 1 [list namespace eval "$className-" {}]

13 }

14 ;

[复制代码](javascript:void(0);)

demo的：

例如：class Father {xxxx}

className为Father clsDefine 为{XXXX}

5行查看 是否已存在Father类，有就删除类 调用itcl的类方法

6、7行给 数组赋值，重点关注7行\_class\_exBody函数自动拓展 类Body的内容，每个定义class的都添加相同内容，这个类就不多解释 很简单的语法可以puts一下出来就看到了。

[复制代码](javascript:void(0);)

1 # 自动扩展classBody

2 # Eval-：无限制的内部调试执行，不应该在实际代码中出现、只用于内部调试

3 # Get-: 获取任意属性的值(自动判定是否为数组变量)

4 # Set-: 对public级variable进行设置。(如需设置类变量，应该仅能通过方法去设置)

5 proc itclx::\_class\_extBody {className clsBody} {

6 set className [\_nmspFull $className]

7 append clsBody "\n"

8 append clsBody {

9 public variable \_InputKeys- ""

10 public method Get- {\_\_var} {

11 if [::array exists $\_\_var] {::array get $\_\_var} else {::set $\_\_var}

12 }

13 public method Set- {\_\_var \_\_val} {

14 set pubVars [itclx::getPubVars $this]

15 if {$\_\_var ni $pubVars} {error "unknown public variable {$\_\_var}, must in {$pubVars}"}

16

17 if [::array exists $\_\_var] {::array set $\_\_var $\_\_val} else {::set $\_\_var $\_\_val}

18 eval [$this info variable $\_\_var -config] ;# 执行pubVar的config部分

19 if [::array exists $\_\_var] {::array get $\_\_var} else {::set $\_\_var}

20 }

21 public method Eval- {\_\_script} {

22 ::eval $\_\_script

23 }

24 }

25 #append clsBody [::itclx::defineVar\_extBody $className]

26 return $clsBody

27 }

28 ;

[复制代码](javascript:void(0);)

经过class函数7行后，demo Father类 body自动拓展为：

[复制代码](javascript:void(0);)

1 public variable name {NoOne}

2 public variable money {0}

3 public common people {0}

4 constructor {{mymoney 500}} {

5 set money $mymoney;

6 puts "has money $mymoney";

7 incr people

8 }

9

10

11 public variable \_InputKeys- ""

12 public method Get- {\_\_var} {

13 if [::array exists $\_\_var] {::array get $\_\_var} else {::set $\_\_var}

14 }

15 public method Set- {\_\_var \_\_val} {

16 set pubVars [itclx::getPubVars $this]

17 if {$\_\_var ni $pubVars} {error "unknown public variable {$\_\_var}, must in {$pubVars}"}

18

19 if [::array exists $\_\_var] {::array set $\_\_var $\_\_val} else {::set $\_\_var $\_\_val}

20 eval [$this info variable $\_\_var -config] ;# 执行pubVar的config部分

21 if [::array exists $\_\_var] {::array get $\_\_var} else {::set $\_\_var}

22 }

23 public method Eval- {\_\_script} {

24 ::eval $\_\_script

25 }

[复制代码](javascript:void(0);)

12行 uplevel 1 [list namespace eval "$className-" {}] 的作用是 在上一级level即类所在那一级 执行 namespace eval "$className-" {} ,定义$className-命名空间。也就是类名的空间。

 class函数分析完毕。

[**【itclx源码解析02】 classEnd方法**](http://www.cnblogs.com/nemolmt/p/6251813.html)

**classEnd**

[复制代码](javascript:void(0);)

1 # create class(who define by class). and call "\_class\_extFCombo"

2 proc itclx::classEnd {className {fPre ""}} {

3 set className [\_nmspFull $className]

4

5 #classbody加入写在类外的方法名 作用是在类内声明方法 便于在类外进行方法的定义

6 \_class\_extFCombo $className

7

8 #使用itcl创建类 例如 itcl::class test {public variable aa 0}

9 uplevel 1 [list itcl::class $className $::itclx::\_ClassBody($className)]

10

11 \_class\_FMap $className

12 }

13 ;

[复制代码](javascript:void(0);)

还是那个demo

[复制代码](javascript:void(0);)

1 proc CLASS-Father {} {}

2 class Father {

3 public variable name {NoOne}

4 public variable money {0}

5 public common people {0}

6 constructor {{mymoney 500}} {

7 set money $mymoney;

8 puts "has money $mymoney";

9 incr people

10 }

11 }

12 proc Father-::test1 {} {

13 puts "this is test1 method"

14 }

15 classEnd Father

16 proc test {args} {

17 set p [New Father {\*}$args]

18 puts "object,$p"

19 puts [.$p.money]

20 puts [.$p.people]

21 }

[复制代码](javascript:void(0);)

demo中15行 classEnd Father 调用了classEnd方法

6行 \_class\_extFCombo $className 作用是在classbody加入写在类外的方法名 作用是在类内声明方法 便于在类外进行方法的定义与实现

接下来先看下 \_class\_extFCombo 函数

[复制代码](javascript:void(0);)

1 # extend class

2 # pubMethod:=> reg{^[A-Z].\*\_[A-Z].\*[^\_]$} or {^[A-Z][^\_]\*[^\_]$}

3 # proMethod:=> reg{^[A-Z].\*\_[a-z].\*[^\_]$} or {^[a-z].\*[^\_]$}

4 proc itclx::\_class\_extFCombo {className} {

5 set className [\_nmspFull $className]

6 set clsBody "" ;# $::itclx::\_ClassBody($className)

7

8 # proM: 小写开头、不以\_结尾

9 set proMethods [lsort -dictionary [info procs "$className-::\[a-z\]\*\[0-9a-zA-Z?\]"]]

10 set pubMethods ""

11 # 大写开头、不以\_结尾

12 foreach iMethod [lsort -dictionary [info procs "$className-::\[A-Z\]\*\[0-9a-zA-Z?\]"]] {

13 #获取所有符合正则条件的函数列表 例如 ::IPC-::DHCP\_CfgSer ::IPC-::DHCP\_CliCheckPkt ::IPC-::DHCP\_CliIcmp

14 set iMethod [namespace tail $iMethod] ;#获取函数名 例如 namespace tail "::IPC-::DHCP\_CfgSer" ==>> DHCP\_CfgSer

15 if {![string match {\*\_\*} $iMethod]} {

16 lappend pubMethods $iMethod ;# pubM: 大写开头、中间不带\_、不以\_结尾

17 } else {

18 if {[string match {\*\_[A-Z?]\*} $iMethod]} {

19 lappend pubMethods $iMethod ;# pubM: 大写开头、中间带\_大写开头、不以\_结尾

20 } else {

21 lappend proMethods $iMethod ;# proM: 大写开头、中间带\_小写开头、不以\_结尾

22 }

23 }

24 }

25

26 #以下代码 使用append将函数加入classbody 类体内

27

28 # 由于当前底层的itcl包未重写，故暂时用pub级别代替pro级别

29 set proLv public ;# protected

30 foreach proMethod $proMethods {

31 append clsBody "\n$proLv method [namespace tail $proMethod]"

32 }

33 foreach pubMethod $pubMethods {

34 append clsBody "\npublic method [namespace tail $pubMethod]"

35 }

36 foreach proProc [lsort [info procs "$className-::\[a-z\]\*\_"]] {

37 append clsBody "\n$proLv proc [namespace tail $proProc]"

38 }

39 foreach pubProc [lsort [info procs "$className-::\[A-Z\]\*\_"]] {

40 append clsBody "\npublic proc [namespace tail $pubProc]"

41 }

42

43

44 set ::itclx::\_ClassBody($className) "$clsBody\n$::itclx::\_ClassBody($className)"

45 puts "\_ClassBody\*\*\*\*$className:\n$::itclx::\_ClassBody($className)\n\*\*\*\*"

46 }

47 ;

[复制代码](javascript:void(0);)

经过此函数后demo Father类Body拓展为：

[复制代码](javascript:void(0);)

1 public method test1

2

3 public variable name {NoOne}

4 public variable money {0}

5 public common people {0}

6 constructor {{mymoney 500}} {

7 set money $mymoney;

8 puts "has money $mymoney";

9 incr people

10 }

11

12

13 public variable \_InputKeys- ""

14 public method Get- {\_\_var} {

15 if [::array exists $\_\_var] {::array get $\_\_var} else {::set $\_\_var}

16 }

17 public method Set- {\_\_var \_\_val} {

18 set pubVars [itclx::getPubVars $this]

19 if {$\_\_var ni $pubVars} {error "unknown public variable {$\_\_var}, must in {$pubVars}"}

20

21 if [::array exists $\_\_var] {::array set $\_\_var $\_\_val} else {::set $\_\_var $\_\_val}

22 eval [$this info variable $\_\_var -config] ;# 执行pubVar的config部分

23 if [::array exists $\_\_var] {::array get $\_\_var} else {::set $\_\_var}

24 }

25 public method Eval- {\_\_script} {

26 ::eval $\_\_script

27 }

[复制代码](javascript:void(0);)

经过此函数后 Father类body多了1行 public method test1

此行作为函数的类内声明。

classEnd函数的

9 uplevel 1 [list itcl::class $className $::itclx::\_ClassBody($className)]

 作用是 补充完成 类定义(不包含 类外定义的方法)，

其中 $::itclx::\_ClassBody($className) 数组存放的是 上面Body拓展贴的最新代码。  
  
classEnd 11行 \_class\_FMap $className 的含义：  
首先查看下\_class\_FMap 函数：

[复制代码](javascript:void(0);)

1 # "itclx::class $className fName" => auto map to "$className-::$fName"

2 # 将已经声明的函数 在类外 补充完整 例如 itcl::body ::Father::test1 {} {eval [::info body ::Father-::test1]}

3 proc itclx::\_class\_FMap {className} {

4 set className [\_nmspFull $className]

5 foreach fFull [info commands "$className-::\*"] { ;#获取所有的 函数列表 例如 ::IPC-::DHCP\_CfgSer ::IPC-::W522U\_CheckSsidInfo ::IPC-::HTTP\_CfgSer ::IPC-::PPPD\_Chk

6 set f [namespace tail $fFull] ;#获取 函数名 DHCP\_CfgSer

7 set fFull\_body "eval \[::info body $fFull\]"

8 set fFull\_argsNames [info args $fFull]

9 set fFull\_argsLs ""

10 foreach fFull\_arg $fFull\_argsNames {

11 if [info default $fFull $fFull\_arg fFull\_def] {

12 lappend fFull\_argsLs [list $fFull\_arg $fFull\_def]

13 } else {

14 lappend fFull\_argsLs $fFull\_arg

15 }

16 }

17

18 #puts "functions\_\_\_\_\_:\n itcl::body ${className}::$f \{$fFull\_argsLs\} {$fFull\_body} \nend\_\_\_\_\_"

19 uplevel #0 "itcl::body ${className}::$f \{$fFull\_argsLs\} {$fFull\_body}"

20 }

21 }

22 ;

[复制代码](javascript:void(0);)

\_class\_FMap Father 处理结果是：(因为只有一个类外定义方法test1，所以只有一个列表)

itcl::body ::Father::test1 {} {eval [::info body ::Father-::test1]}

其中eval [::info body ::Father-::test1]的作用是 获取test1的函数Body:  puts "this is test1 method"

classEnd分析完毕。

[**【itclx源码解析03】New**](http://www.cnblogs.com/nemolmt/p/6255394.html)

New 函数：创建对象

[复制代码](javascript:void(0);)

1 # ①用[::#auto]的方式创建对象。

2 # ②若在#0中执行New、或调试开关打开则创建全局对象，否则创建局部对象

3 proc itclx::New {className args} {

4 variable \_PublicDebug ;# 引用命名空间中的变量\_PublicDebug

5 if {![string match {::\*} $className]} {set className "::$className"}

6 set new\_pre ""

7 if {[info level]>1 && $\_PublicDebug==0} {set new\_pre "itclx::\_local "}

8 set objName [uplevel 1 "$new\_pre$className ::#auto $args"]

9 if {$\_PublicDebug!=0} {puts " /New/ $objName"}

10 return $objName

11 }

[复制代码](javascript:void(0);)

Demo还是同一个：

[复制代码](javascript:void(0);)

1 package require itclx

2

3 proc CLASS-Father {} {}

4 class Father {

5 public variable name {NoOne}

6 public variable money {0}

7 public common people {0}

8 constructor {{mymoney 500}} {

9 set money $mymoney;

10 puts "has money $mymoney";

11 incr people

12 }

13 }

14 proc Father-::test1 {} {

15 puts "this is test1 method"

16 }

17 classEnd Father

18 proc test {args} {

19 set p [New Father {\*}$args]

20 puts "object,$p"

21 puts [.$p.money]

22 puts [.$p.people]

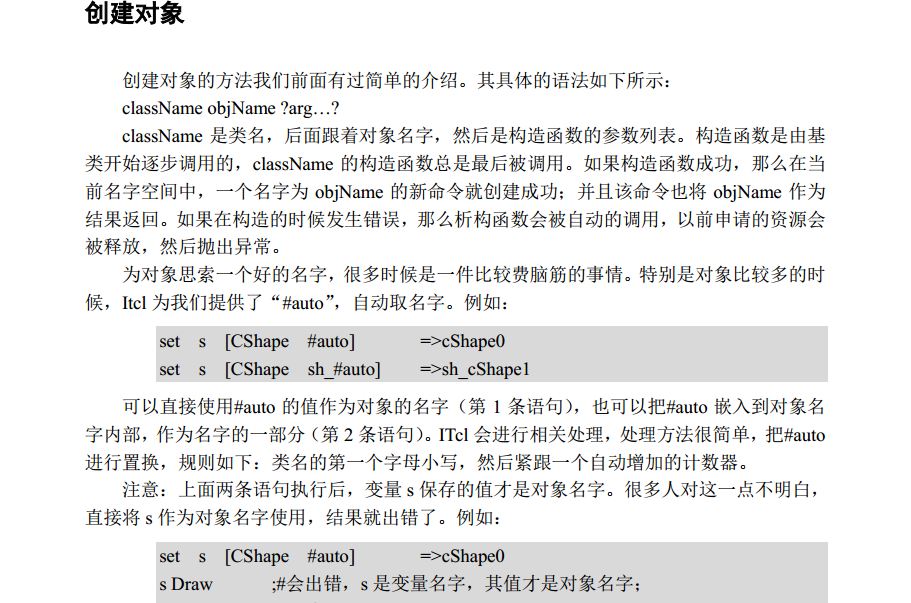
23 }

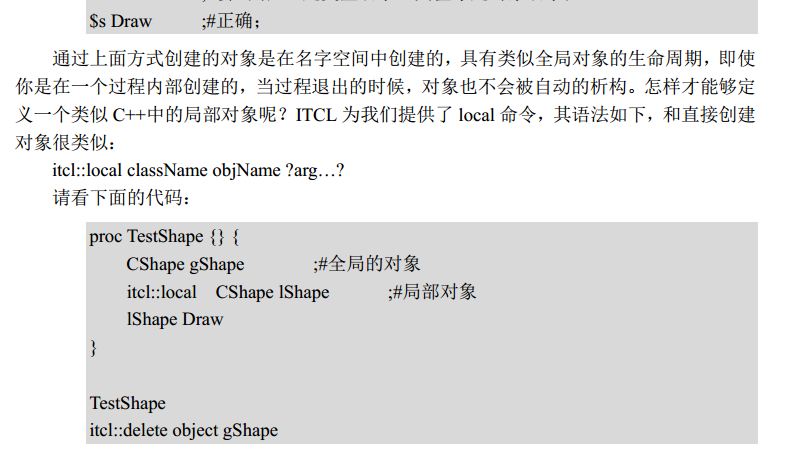
[复制代码](javascript:void(0);)

demo 19行 set p [New Father {\*}$args]

**其中$args前面的 {\*} 作用是$args 作为一个字符串，不管中间是否有空格间隔。**

 然后 为了解创建对象的问题，先插入关于Local创建对象的知识点 TCL、Python和软件测试自动化 136~137页







从介绍可以看到关于Local的定义不再多说。

因此 New函数： set objName [uplevel 1 "$new\_pre$className ::#auto $args"]的作用，事实上就是在调用类所在域创建一个以类名自动命名的对象objname，例如Father0,Father1,Father2...

下面我们看一下\_local函数：

[复制代码](javascript:void(0);)

1 # 模拟itcl::local

2 proc itclx::\_local {class name args} {

3 set ptr [uplevel [list $class $name] $args]

4 uplevel [list set itcl-local-($ptr) $ptr]

5 set cmd [uplevel namespace which -command $ptr]

6 uplevel [list trace variable itcl-local- u \

7 "::itcl::delete\_helper $cmd"]

8 return $ptr

9 }

10 ;

[复制代码](javascript:void(0);)

然后是库文件里面的itcl::local函数：

[复制代码](javascript:void(0);)

1 # ----------------------------------------------------------------------

2 # USAGE: local <className> <objName> ?<arg> <arg>...?

3 #

4 # Creates a new object called <objName> in class <className>, passing

5 # the remaining <arg>'s to the constructor. Unlike the usual

6 # [incr Tcl] objects, however, an object created by this procedure

7 # will be automatically deleted when the local call frame is destroyed.

8 # This command is useful for creating objects that should only remain

9 # alive until a procedure exits.

10 # ----------------------------------------------------------------------

11 proc ::itcl::local {class name args} {

12 set ptr [uplevel [list $class $name] $args]

13 uplevel [list set itcl-local-$ptr $ptr]

14 set cmd [uplevel namespace which -command $ptr]

15 uplevel [list trace variable itcl-local-$ptr u \

16 "::itcl::delete\_helper $cmd"]

17 return $ptr

18 }

[复制代码](javascript:void(0);)

为实现自动创建对象与析构对象，因此重写了itcl::local函数。

然后是::itcl::delete\_helper函数：

 1 proc ::itcl::delete\_helper { name args } { 2 ::itcl::delete object $name 3 }

作用是自动析构对象。

::itclx::\_local函数的作用是实现与itcl::local函数一样的作用。

[**【itclx源码解析04】oo 方法**](http://www.cnblogs.com/nemolmt/p/6266134.html)

首先不要问为什么，先修改tcl安装目录下例如 E:\tclSpace\Tcl\lib\tcl8.6 找到init.tcl文件

修改proc unknown args {}函数 在最后的return之前加入以下代码：

1 set name [lindex $args 0]

2 if {[info commands ::itclx::oo]!="" && [string match {.\*} $name]} {

3 return [uplevel 1 [string replace $args 0 0 {::itclx::oo }]]

4 }

这点先留个小悬念，待会再说明。。

 demo还是沿用那个:

[复制代码](javascript:void(0);)

1 package require itclx

2

3 proc CLASS-Father {} {}

4 class Father {

5 public variable name {NoOne}

6 public variable money {0}

7 public common people {0}

8 constructor {{mymoney 500}} {

9 set money $mymoney;

10 puts "has money $mymoney";

11 incr people

12 }

13 }

14 proc Father-::test1 {} {

15 puts "this is test1 method"

16 }

17 classEnd Father

18 proc test {args} {

19 set p [New Father {\*}$args]

20 puts "object,$p"

21 puts [.$p.money]

22 puts [.$p.people]

23 .$p.test1

24 }

[复制代码](javascript:void(0);)

**如demo所示22、23、24行类似.$p.money之类的用法，是itcl所没有的，是预期类似C++的用法而设计出来的机制。也是itclx的最核心目标。**

分析 英文字符 . 开头的命令，在未定义之前，在tcl和itcl都是未识别指令。

开头加入init.tcl的几行代码中：

2、info commands ::itclx::oo 用于判断 ::itclx::oo函数是否存在，即包是否已经引入。

   string match {.\*} $name 判断命令名称是否 以 . 开头 ，若以 . 开头这对命令执行条件语句里面的内容

3、在调用unknown函数层，即解释器#0层，将 . 替换为{::itclx::oo }，特别注意后面的空格。

例如：.$p.money会变为::itclx::oo $p.money，此时方法是 可识别的方法。

接下来解析的内容只需要解析 ::itclx::oo $p.money即可。

::itclx::oo函数代码：

[复制代码](javascript:void(0);)

1 # ex0.1: [oo var]==[itclx::\_get var]; ex0.2: [oo var = val]==[itclx::\_set var val]

2 # ex1: [oo obj1.objN.var]==[[obj1 Get- objN] Get- var]

3 # ex2: [oo obj1.objN.var = val]==[[obj1 Get- objN] configure -var val]

4 # ex3: [oo obj1.objN.var ...]==[[obj1 Get- objN] configure -var val]

5 # 别用oo执行函数，会使得函数栈变得混乱. (例外：\_PublicDebug模式下，全在#0栈调试，不担心栈混乱)

6 proc itclx::oo {args} {

7 set argLen [llength $args]

8 lassign $args \_\_namePath \_\_input \_\_val ;# ex2: obj1.objN.var = val

9 set nameLs [split $\_\_namePath "."] ;# ex2: {obj1 objN var}

10 set nameNum [llength $nameLs] ;# ex2: 3

11

12 # 类的内部相互调用。在全局调试时配合::this变量很有用(方法内的代码可以直接进行调试)

13 if {$nameNum==1} {

14 set \_\_obj [set thisUp [uplevel 1 "set this"]]

15 set \_\_nameEnd $nameLs

16 if {$\_\_nameEnd in [itclx::getMethods $\_\_obj]} {

17 set levelUp 1

18 #set levelUp [expr min(2,[::info level])]

19 # 【待完善】PublicDebug调试开关打开时，所有的itclx面向对象权限都改为pub。 否则在#0调试模式下无法调用非pub函数。

20

21 # 通过Eval-Debug-间接执行时，可会多占用1层函数栈

22 return [uplevel $levelUp "{$\_\_obj} $args"]

23 } elseif {$argLen==1} { # ex0.1: [oo var]==[itclx::\_get var]

24 return [$thisUp Get- $nameLs]

25 } elseif {$argLen==3 && $\_\_input=="="} { #ex0.2: [oo var = val]==[itclx::\_set var val]

26 return [$thisUp Eval- "if \[array exists {$nameLs}] \

27 {array set {$nameLs} {$\_\_val}} else {set {$nameLs} {$\_\_val}} \

28 "]

29 } else {

30 error "itclx::oo $args"

31 }

32 }

33

34 set \_\_obj [itclx::\_nmspFull [lindex $nameLs 0]]

35 for {set i 1} {$i<$nameNum-1} {incr i} {

36 set \_\_obj [itclx::\_nmspFull [$\_\_obj Get- [lindex $nameLs $i]]]

37 }

38 set \_\_nameEnd [lindex $nameLs end]

39

40 #if {$::itclx::\_PublicDebug} {puts "argLen,\_\_obj,\_\_nameEnd: {$argLen} {$\_\_obj} {$\_\_nameEnd}"}

41 if {$\_\_nameEnd in [itclx::getMethods $\_\_obj]} {

42 set ret [uplevel 1 "{$\_\_obj} {$\_\_nameEnd} [lrange $args 1 end]" ]

43 } elseif {$argLen==1} {

44 set ret [$\_\_obj Get- $\_\_nameEnd]

45 } elseif {$argLen==3 && $\_\_input=="="} {

46 set ret [$\_\_obj Set- $\_\_nameEnd $\_\_val]

47 } else {

48 error "itclx::oo $args"

49 }

50 return $ret

51 }

52 ;

[复制代码](javascript:void(0);)

分2种情况处理：

1、不带obj的情况(只有一个.) 例如 .money , .money = 100 ,

1)、16行判断 函数是否为 对象的成员函数、22行在调用层 执行函数，例如 .test1

2)、23行调用Get-获取 成员变量的值 例如 .money

3)、25行 先判断变量是否为 数组变量，根据结果对 成员变量赋值 例如 .money = 1

2、带2个.以上的情况。例如 .$p.money = 1  .$p.test1

参考1的情况分析。

重点关注 45、46与1.2）的区别。

分这两种情况，9、10行对参数进行了分解。

13行条件 对1个.的情况进行处理，接下来就对2个.以上的情况进行处理。

**this定义：类似于C++的this，在调试的时候 作用相当大。**

**14行 比较难理解解释下，set this 事实上包含2个含义：若this不存在，则报错，提示this不存在，若在执行oo函数前，将this赋值为对象，例如 set this $p**

**则将\_\_obj赋值为this对象**

其中 涉及到的 Set- 、Get-、 Eval- 函数均在 拓展类Body的时候已经 在类内，具体可以回看class那一节。

oo函数还建立了自测机制，若是失败会直接导致itclx引入失败。

包最后的一句 eval $itclx::TEST\_oo

下面是 TEST\_oo 自测代码：

[复制代码](javascript:void(0);)

1 set itclx::TEST\_oo {

2 class Test1\_oo {

3 public common ObjName ""

4 public method Eval cmd {eval $cmd}

5 }

6 classEnd Test1\_oo

7 class Test2\_oo {

8 public variable var 0

9 protected variable var2 var2

10 public method f {args} {return "f {$args}"}

11 protected method f2 {args} {return "f {$args}"}

12 constructor {args} {set Test1\_oo::ObjName $this; puts "[itclx::\_fUpInfo 0]: args={$args}"}

13 }

14 classEnd Test2\_oo

15 set t1 [New Test1\_oo]

16 set t2 [New Test2\_oo]

17 #puts "$Test1\_oo::ObjName"

18 set testInfo "

19 {.$t2.var = 1} 1 {.$t2.var} 1 {catch {.$t2.var2 = 1}} 1 {catch {.$t2.var2}} 0

20 {.$t1.ObjName.var} 1 {.$t1.ObjName.var = 2} 2 {.$t1.ObjName.var} 2

21 {.$t1.ObjName.f} {f {}} {.$t1.ObjName.f 1} {f {1}} {.$t1.ObjName.f 1 2} {f {1 2}}

22 {.$t2.f} {f {}} {.$t2.f 1} {f {1}} {.$t2.f 1 2} {f {1 2}}

23 {catch {.$t2.f2}} 1

24 "

25 #puts "$testInfo"

26 foreach {itest iexpect} $testInfo {

27

28 set iout [eval $itest]

29 puts "$itest,$iexpect###$iout,$iexpect"

30 if {$iout!=$iexpect} {error "{$iout}!={$iexpect} , {$itest}"}

31 }

32 proc itclx::test\_oo {cmd} {eval $cmd}

33 foreach {itest iexpect} $testInfo {

34 set iout [itclx::test\_oo $itest]

35 if {$iout!=$iexpect} {error "{$iout}!={$iexpect} , {$itest}"}

36 }

37 puts "TEST\_oo success 11111"

38 }

[复制代码](javascript:void(0);)

其中涉及到的oo的所有情况、以及每种情况在itclx框架下的 预期正确执行效果。

set testInfo "  
{.$t2.var = 1} 1

{.$t2.var} 1

{catch {.$t2.var2 = 1}} 1

{catch {.$t2.var2}} 0  
{.$t1.ObjName.var} 1

{.$t1.ObjName.var = 2} 2

{.$t1.ObjName.var} 2  
{.$t1.ObjName.f} {f {}}

{.$t1.ObjName.f 1} {f {1}}

{.$t1.ObjName.f 1 2} {f {1 2}}  
{.$t2.f} {f {}}

{.$t2.f 1} {f {1}}

{.$t2.f 1 2} {f {1 2}}  
{catch {.$t2.f2}} 1  
"

若 此自测出错，包将引入失败，因此也是作为保证itclx包引入成功与包生效的一个机制。

oo解释完毕。

oo函数需要大量练习才可以深入理解。

# [【itclx源码解析05】Struct\_Init、Dim、Lassign 方法](http://www.cnblogs.com/nemolmt/p/6278579.html)

[复制代码](javascript:void(0);)

1 # 初始化"结构体"型的对象是否初始化正常，仅建议在构造函数中使用

2 # 所做的操作：①以args形式接收构造函数中的值，args输入格式为ini形式，然后转为字典格式。

3 # 并按字典形式初始化(字典的key须为pubVar，且首先字母必须大写、且不带\_后缀)

4 # ②在类声明中未给定初值的为必选参数，初始化时若不带则报错

5 # ③若有参数Expe，则值必须为全大写的PASS或FAIL

6 # (注: 所有涉及到跨函数栈执行的特殊形式，其中使用的变量名都采用特殊形式以免与上层函数栈

7 # 中的变量名产生冲突，每跨1层函数加1个"-"后缀)

8 proc itclx::Struct\_Init {} {

9 uplevel 1 {

10 ;# 将形如[a=1 {b=1 2 }]的ini格式转为形如[a 1 b {1 2 }]的dict格式

11 set args [itclx::ini2dict $args]

12 if {[llength $args]%2 != 0} {error "参数集须为字典形式"}

13 set pubLs- [itclx::getPubVars $this] ;#获取当前对象的 public属性的成员变量

14 dict for {key- value-} $args {

15 if {[set key-] in [set pubLs-]} {

16 if {![string match {[A-Z]\*[a-zA-Z0-9]} [set key-]]} {

17 error "错误的对象变量名格式{[set key-]}，须首字母大写且不带\_后缀，形如{Var}"

18 }

19 set [set key-] [set value-] ;#等价于 set $key- $value- 给字典的 每个Key赋值为 = 号后面的值 value 例如a=1 ===》 set a 1

20 lappend \_InputKeys- [set key-]

21 } else {

22 error "错误的参数{[set key-]}，须属于{[set pubLs-]}"

23 }

24 }

25 foreach key- [set pubLs-] { ;#规范参数输入 必须为 a=1 或者 {b=1 2 }的情况

26 if {[catch {set [set key-]} err1-] && [catch {array get [set key-]} err2-]} {

27 error "缺少必须参数{[set key-]}, err1={[set err1-]} err2={[set $err2-]}"

28 }

29 }

30 if {"Expe" in [set pubLs-] && ([set Expe] ni "$::TSPASS $::TSFAIL")} { ;# Expe作为 预期执行结果，初始值必须为 PASS或者FAIL 不能有其他值

31 error "错误的参数Expe值={[set Expe]}，须为属于{$::TSPASS $::TSFAIL}"

32 }

33 }

34 }

35 ;

[复制代码](javascript:void(0);)

以上为Struct\_Init函数。

**itclx致力于构建类似 C语言里面结构体的 内存结构，而Struct\_Init说白了就是给结构体定规则的。**

关于此函数的所有解释在函数前注释与内容注释已经解释的很清楚，不再赘述。

需要理解的是 constructor {args}  {xxx}函数生效是在实例化对象的时候，也就是New函数的时候。

demo

[复制代码](javascript:void(0);)

1 #pppoe客户端拨号连接

2 class IPC/PPPOE\_CfgCli@ {

3 public variable Auth {chap} ; #认证方式

4 public variable User {tenda} ; #用户名

5 public variable Pwd {tenda} ; #密码

6 public variable Intf {eth1} ; #服务器接口

7 public variable Mppe {} ; #Mppe加密类型{40,128,both}

8 public variable Debug {1} ; #Debug开关

9 public variable Expe $::TSPASS ; #预期结果

10

11 constructor {args} {

12 Struct\_Init

13 Dim Debug Bool

14 }

15 }

16 classEnd IPC/PPPOE\_CfgCli@

[复制代码](javascript:void(0);)

demo中19行 set p [New  IPC/PPPOE\_CfgCli@  {\*}$args]，Struct\_Init的args就是从这里来。

其中args前面的 {\*} 作用是：当args为 多个参数例如  {a=1 b=2}列表时，将args作为一个整体的字符串处理。

Struct\_Init函数介绍完毕。

**以下是Dim函数**： Dim函数 给成员变量定规则。必须符合此条件才可完成constructor.

[复制代码](javascript:void(0);)

1 ;# 定义属性变量的类型。 若不满足类型，则会报错(但操作还是无法阻止)

2 ;# [一个命名空间/类中，不能有同名的dim变量]

3 ;# 原理(除Arr/Array类型外)： trace add variable $varname write f\_dim\_proc

4 ;# f\_dim\_proc {varNm element opt {codes ""}} { ... }

5 ;# 用法： 形如 [dim Int {x0 x1}] [dim ]

6

7 proc itclx::Dim {args} {

8 #

9 set argsLen [llength $args]

10 if {$argsLen==2} {

11 set dimBody ""

12 lassign $args varNames type

13 } elseif {$argsLen==3} {

14 lassign $args varNames type dimBody

15 } else {

16 error "错误的args长度，须为2或3"

17 }

18 set typeLs {Int Bool Dict IntRange InLs NiLs InClass NiClass Custom}

19 switch $type {

20 Int {

21 set dimEval {

22 if {![string is integer $\_\_val]} {

23 error "错误的$\_\_var, 需为Int型的整数。但设置值={$\_\_val}"

24 }

25 }

26 }

27 IntRange {

28 lassign [split $dimBody "-"] min max

29 if {![string is integer $min] || ![string is integer $max]} {

30 error "错误的dimBody，需为IntRange型的范围整数，形如1-3。但设置值={$dimBody}"

31 }

32 set dimEval "

33 if {!(\[string is integer \$\_\_val] && \$\_\_val<=$max && \$\_\_val>=$min)} {

34 error \"错误的\$\_\_var, 需为$min-$max的IntRange型范围整数。但设置值={\$\_\_val}\"

35 }

36 "

37 }

38 Bool {

39 set dimEval {

40 if {$\_\_val ni "0 1"} {

41 error "错误的变量$\_\_var, 需为Boolean型、即0或1。但设置值={$\_\_val}"

42 }

43 }

44 ;

45 }

46 Dict {

47 set dimEval {

48 if {![string is list $\_\_val] && [llength $\_\_val]%2 != 0} {

49 error "错误的变量$\_\_var, 需为Dict型、即llength%2==0。但设置值={$\_\_val}"

50 }

51 }

52 ;

53 }

54 InLs {

55 set dimBodyLower [string tolower $dimBody]

56 set dimEval "

57 if {!(\[string tolower \$\_\_val\] in {$dimBodyLower})} {

58 error \"错误的变量\$\_\_var, 需InList型、属于集合{$dimBodyLower}。但设置值={\$\_\_val}\"

59 }

60 "

61 }

62 NiLs {

63 set dimBodyLower [string tolower $dimBody]

64 set dimEval "

65 if {!(\[string tolower \$\_\_val\] ni {$dimBodyLower})} {

66 error \"错误的变量\$\_\_var, 需NiList型、不属于集合{$dimBodyLower}。但设置值={\$\_\_val}\"

67 }

68 "

69 }

70 InClass {

71 if {[itcl::find classes "::${dimBody}"]=={}} {error "dim InClass: {$dimBody}须为一个类"}

72 set dimEval "

73 if {\[itcl::find objects -isa $dimBody \$\_\_val\]=={}} {

74 error \"错误的变量\$\_\_var, 需为类{$dimBody}的对象。但设置值={\$\_\_val}\"

75 }

76 "

77 }

78 Custom {

79 set dimEval "

80 if {$dimBody} {} else {

81 append err \"错误的变量\$\_\_var, 需满足表达式\"

82 append err \{\{$dimBody\}\}

83 append err \"。但设置值={\$\_\_val}\"

84 error \$err

85 }

86 "

87 #error \"错误的变量\$varName，需满足表达式{$dimBody}。但设置值={\$var}\"

88 }

89 default { error "错误的类型{$type}，需属于{$typeLs}" }

90 }

91 #

92

93 foreach varName $varNames {

94 uplevel 1 "set \_\_var $varName; set \_\_val \$$varName"

95 uplevel 1 $dimEval

96 }

97 ;

98 }

99 ;

[复制代码](javascript:void(0);)

其他语法都不难 重点解释93~95行：

94 在 constructor函数中 给 \_\_var，\_\_val 赋值为初始值，目的是给95行uplevel 1 $dimEval 信息中 给对应的提示信息赋值。

switch语句的19~90行 目的是判断args是否符合对应的条件 并且给dimEval变量赋值。

95行 则在 调用层将 dimEval 变量输出。

Dim函数只是作为判断参数赋值（成员变量的初始值、以及New对象时候进行判断）是否符合条件。

请特别留意Dim函数仅建议在constructor函数中使用。

**Lassign函数：**

[复制代码](javascript:void(0);)

1 proc itclx::Lassign {lsInfo obj varLs} {

2 foreach var $varLs val $lsInfo {

3 if {$var == ""} {continue}

4 .$obj.$var = $val

5 }

6 }

7 ;

[复制代码](javascript:void(0);)

从代码理解就可以知道此函数的作用是将 lsInfo列表元素 逐个赋值给 obj的成员变量。而其中varLs列表元素存放的是 obj成员变量。

因此在处理的时候 IsInfo与varLs的顺序应该对应，否则会出赋值出错。

MsgTrust MsgSend Lassign这三个函数使用的频率相当低，Lassign只见过一次，而另外2个没有使用过，因此不解析。