|  |  |  |  |
| --- | --- | --- | --- |
| **深圳国泰安教育技术股份有限公司** | **版本** | **密级** | **页数** |
| **V1.1** | **机密** | **共21页** |
| **文档编号:** **GTA\_CMMI\_CodeStandard\_Matlab** | | |

**国泰安软件编码规范Matlab**

|  |  |  |  |
| --- | --- | --- | --- |
| **作者** | 张富铭 | **编制日期** | 2014-02-10 |
| **审核** |  | **审核日期** |  |
| **批准** |  | **批准日期** |  |



深圳国泰安教育技术股份有限公司

版权所有 侵权必究

**文档修改记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **版号** | **修改日期** | **修改内容** | **修改人** |
| V1.0 | 2014-02-10 | 根据CMMI过程改进要求编制该文档 | 张富铭 |
| V1.1 | 2014-04-17 | 根据A线要求修改该文档 | 刘兵 |
|  |  |  |  |

**目录**

[1. 简介 3](#_Toc385868965)

[2. 适用范围 3](#_Toc385868966)

[3. 文体 3](#_Toc385868967)

[4. 排版部分 3](#_Toc385868968)

[4.1. 缩进 3](#_Toc385868969)

[4.2. Tab 4](#_Toc385868970)

[4.3. 语句长度 4](#_Toc385868971)

[4.4. 对齐格式 5](#_Toc385868972)

[4.5. 函数长度 5](#_Toc385868973)

[4.6. 空行 5](#_Toc385868974)

[5. 2 注释部分 6](#_Toc385868975)

[5.1. 基本约定 6](#_Toc385868976)

[5.1.1. 注释量 6](#_Toc385868977)

[5.1.2. 注释时间 6](#_Toc385868978)

[5.1.3. 注释内容 6](#_Toc385868979)

[5.1.4. 注释缩写 7](#_Toc385868980)

[5.1.5. 注释缩进 7](#_Toc385868981)

[5.1.6. 注释意图 7](#_Toc385868982)

[5.1.7. 注释格式 7](#_Toc385868983)

[5.1.8. 注释排版 7](#_Toc385868984)

[5.2. 文件头注释 7](#_Toc385868985)

[5.3. 代码行注释 8](#_Toc385868986)

[5.4. 变量注释 8](#_Toc385868987)

[5.5. 条件注释 9](#_Toc385868988)

[5.6. 代码块注释 10](#_Toc385868989)

[6. 标识符部分 11](#_Toc385868990)

[6.1. 基本约定 11](#_Toc385868991)

[6.2. 标识符变量部分 12](#_Toc385868992)

[7. 程序编写部分 13](#_Toc385868993)

[8. 函数、过程部分 15](#_Toc385868994)

[9. 效率部分 17](#_Toc385868995)

[10. 利用内部或外部工具部分 18](#_Toc385868996)

[10.1. Matlab内部工具 18](#_Toc385868997)

[11. 函数和类调用部分 20](#_Toc385868998)

# 简介

本规范为一套编写高效可靠的Matlab代码的标准、约定和指南。它以安全可靠的软件工程原则为基础，使代码易于理解、维护和增强，提高生产效率。同时，将带来更大的一致性，使软件开发团队的效率明显提高。

# 适用范围

本规范适用于公司所有的Matlab源代码，为详细设计，代码编写和代码审核提供参考和依据。

# 文体

本规范中的建议分为四种：**要，建议，避免，不要**，表示需要遵循的级别。文档中会以粗体表示。对于应遵循的规范，前面会以“Ö”来表示，对不好的做法前面会以“´”来表示：

**要**：描述必须遵循的规范。例如：

Ö 异常类**要**以“Exception”做为后缀；

**建议**：描述在一般情况下应该遵循的规范，但如果完全理解规范背后的道理，并有很好的理由不遵循它时，也不畏惧打破常规。例如：

Ö 强制类型转换时，在类型和变量之间**建议**加一空格。

**不要**：描述一些几乎绝对绝不应该违反的规范。例如：

´ 每个函数有效代码（不包括注释和空行）长度**不要**超过50行。

**避免**：与**建议**相对，一般情况下应该遵循，但有很好的理由时也可以打破。例如：

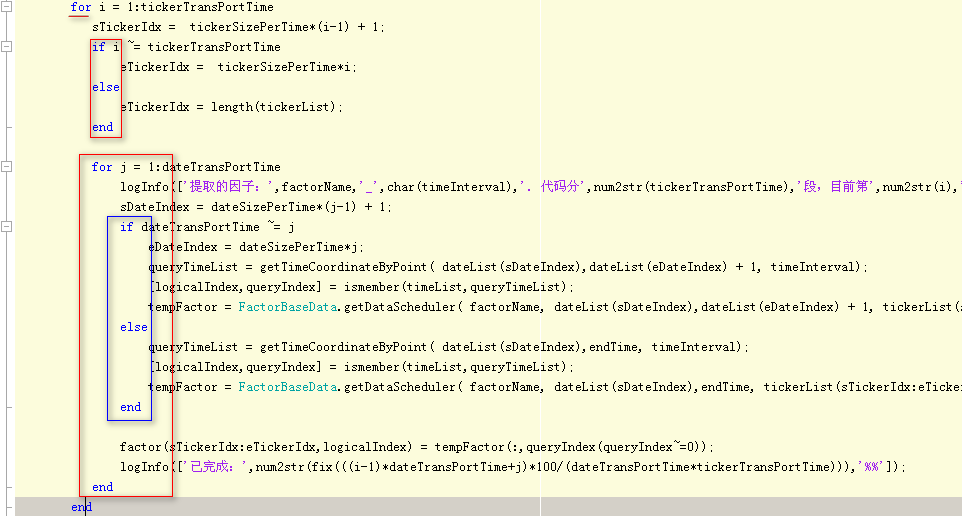
´ **避免**块内部的变量与它外部的变量名相同。

对一些规范内容一并提供了示例代码。

# 排版部分

## 缩进

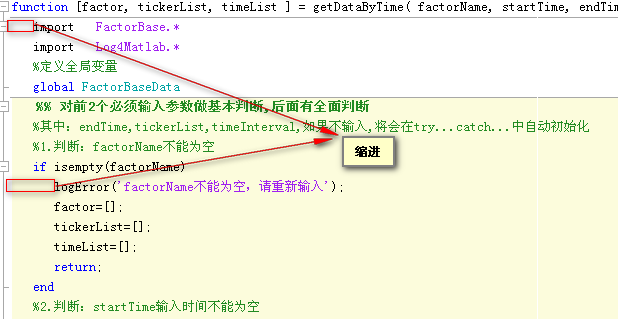
* 程序块要采用缩进风格编写，缩进的空格数**要**为4个
* 函数或过程的开始、结构的定义及循环、判断等语句中的代码都**要**采用缩进风格，case语句下的情况处理语句也**要**遵从语句缩进要求。



## Tab

说明：对于由开发工具自动生成的代码可以有不一致。

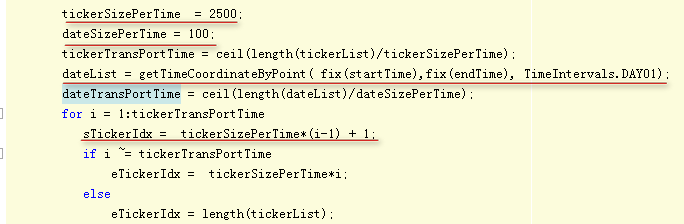
在MATLAB环境下，直接按TAB键或者全选代码，同时按Ctrl+i



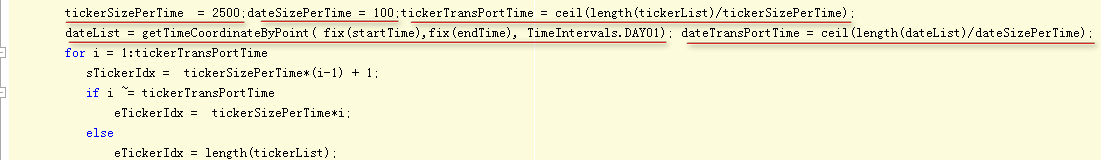
## 语句长度

* 较长的语句(>80字符)**要**分成多行书写，长表达式**要**在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行**要**进行适当的缩进，使排版整齐，语句可读。
* 循环、判断等语句中若有较长的表达式或语句，则**要**进行适应的划分，长表达式**要**在低优先级操作符处划分新行，操作符放在新行之首。
* 若函数或过程中的参数较长，则**要**进行适当的划分。
* **避免**把多个短语句写在一行中，即一行只写一条语句。

**【规范】**



**【不规范】**



## 对齐格式

对齐**要**使用空格键，**不要**使用TAB键。

说明：为避免不同的编辑器阅读程序时，因TAB键所设置的空格数目不同而造成程序布局不整齐，将缩进变乱。

## 函数长度

一个函数或脚本不要写得过长,过长则要考虑抽象出子模块。

## 空行

* 建议适当增加空行，来增加代码的可读性
* 方法内的功能逻辑部分之间

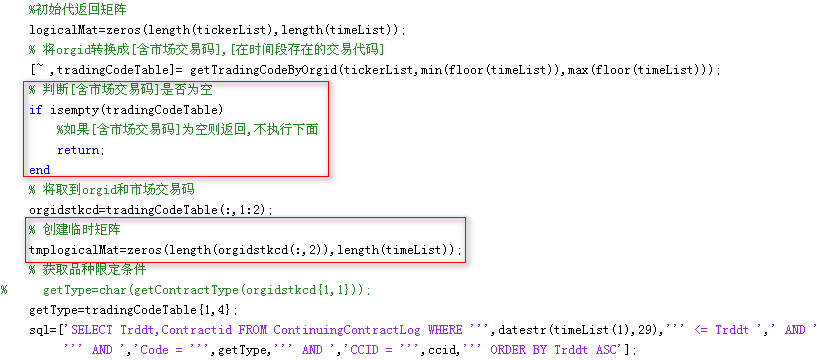
# 2 注释部分

## 基本约定

### 注释量

一般情况下，源程序有效注释量**必须**在20％以上。

说明：注释的原则是有助于对程序的阅读理解，在该加的地方都加了；注释也不宜太多也不能太少，注释语言必须准确、易懂、简洁。



### 注释时间

前期编码时要做到边写代码边注释，后期修改代码要同时修改相应的注释，以保证注释与代码的一致性。没有用的注释要删除。



### 注释内容

注释的内容要清楚、明了，含义准确，防止注释二义性。

总之：要用两句话才能说明的事，就不要用一句话去说明;但能用一句话说明的，就不需要用两句话去阐述。

### 注释缩写

在注释中要对简写变量做必要的说明，特别是不常用缩写。

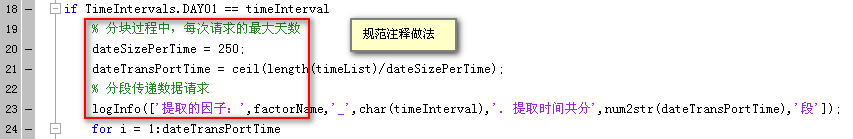


### 注释缩进

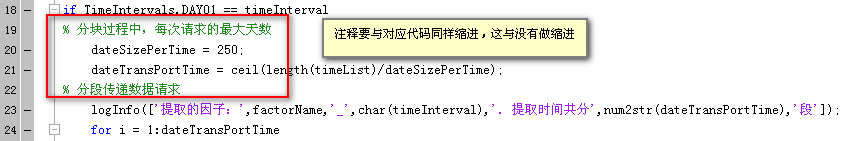
注释与所描述内容，必须是同样的缩排。

说明：可使程序排版整齐，并方便注释的阅读与理解。

**【规范】**



**【不规范】**



### 注释意图

在代码的功能、意图层次上进行注释，提供有用、额外的信息。

说明：注释的目的是解释代码的目的、功能和采用的方法，提供代码以外的信息，帮助读者理解代码，防止没必要的重复注释信息。

### 注释格式

注释格式尽量统一，建议使用%, 不建议使用%{}%方式整块注释,因为这样有些编辑器如source insight无法识别, 导致阅读上的困难.

### 注释排版

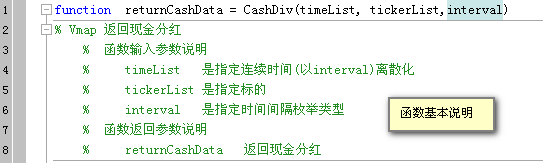
注释应考虑程序易读及外观排版的因素，使用的语言若是中、英兼有的，建议多使用中文，除非能用非常流利准确的英文表达。

说明：注释语言不统一，影响程序易读性和外观排版，出于对维护人员的考虑，建议使用中文。

## 文件头注释

文件头部应进行必要说明，包括：函数功能简要说明，入参和出参说明，创建日期和作者

示例：下面这段头文件的头注释比较标准，当然，并不局限于此格式，但上述信息建议要包含在内。



## 代码行注释

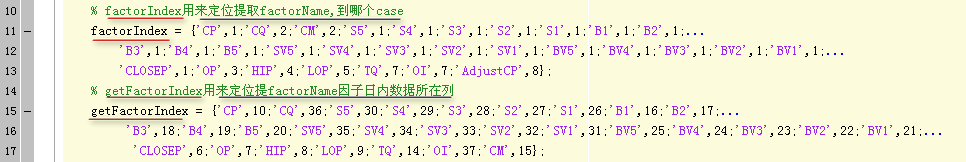
代码注释必须与其描述的代码相近，放在代码上方或右方(对单条语句的注释)相邻位置，切不可放在下面，如放于上方则需与其上面的代码用空行隔开。



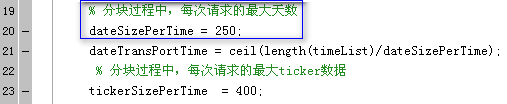
## 变量注释

* 对于所有物理含义的变量、常量，如果其命名不是充分自注释的，在声明时都必须加以注释，说明其物理含义，注释应放在其上方相邻位置或右方。 如果是关键性临时变量，也需要做一定的注释说明

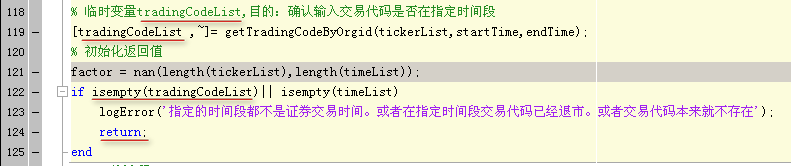
**【变量】**



**【常量】**



**【临时变量】**



* 全局变量要有较详细的注释，包括对其功能、取值范围、注意事项等的说明。

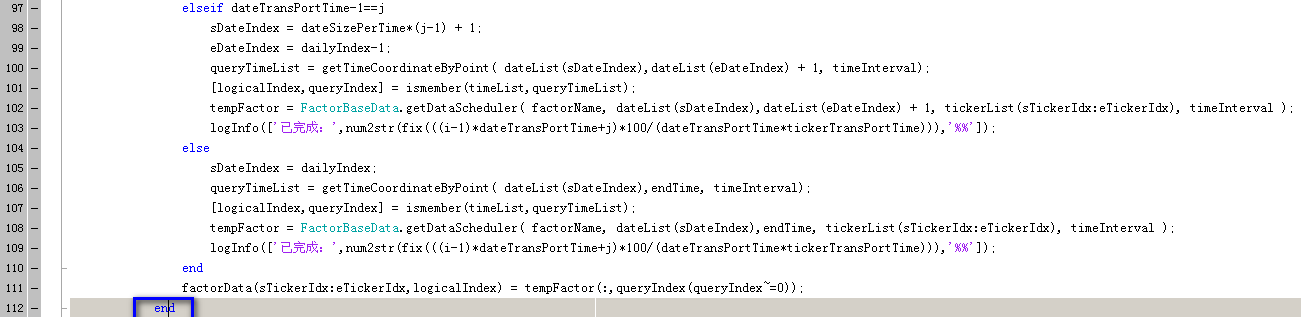
****

## 条件注释

* 对分支语句(条件分支、循环语句等)必须增加注释。

说明：这些语句往往是程序实现某一特定功能的关键，对于维护人员来说，良好的注释帮助更好的理解程序，有时甚至优于看设计文档。





* 对于switch语句下的case语句，如果因为特殊情况需要处理完一个case后进入下一个case处理，必须在该case语句处理完、下一个case语句前加上明确的注释。

说明：这样比较清楚程序编写者的意图，有效防止无故遗漏break语句。

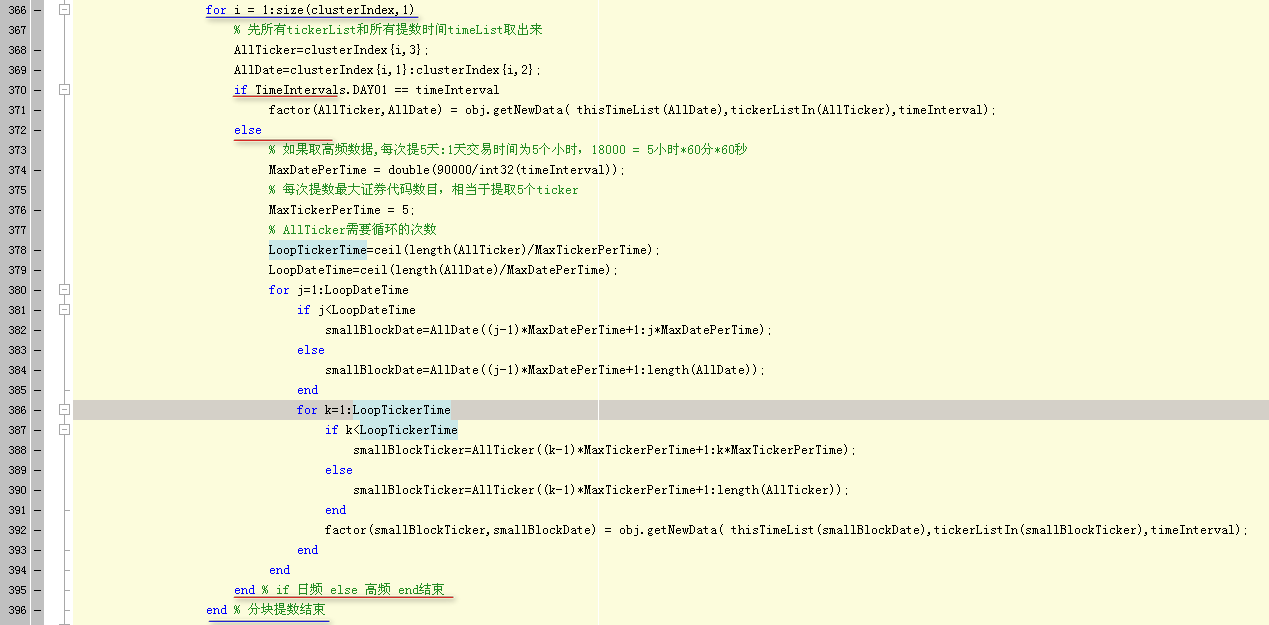
## 代码块注释

* 通过对函数或过程、变量、结构等正确的命名以及合理地组织代码的结构，使代码成为自注释的。

说明：清晰准确的函数、变量等的命名，可增加代码可读性，并减少不必要的注释。

* 在程序块的结束行右方加注释标记，以表明某程序块的结束。

说明：针对当代码段较长，特别是多重嵌套时，这样做可以使代码更清晰，更便于阅读。



# 标识符部分

## 基本约定

* 标识符的命名要清晰、明了，有明确含义，同时使用完整的单词或大家基本可以理解的缩写，避免使人产生误解。

(无论是函数标识符，还是变量标识符及常量标识符都必须做到这一点)

说明：较短的单词可通过去掉“元音”形成缩写；较长的单词可取单词的头几个字母形成缩写；一些单词有大家公认的缩写。

示例：如下单词的缩写能够被大家基本认可。

temp 可缩写为 tmp ;通常用来表达临时含义

flag 可缩写为 flg ; 通常用来表达是否存在含义

statistic 可缩写为 stat ; 通常用来表达是静态含义

increment 可缩写为 inc ; 通常用来表达是增长含义

message 可缩写为 msg ; 通常用来表达是某信息含义

* 命名中若使用特殊约定或缩写，则要有注释说明。

说明：应该在源文件的开始之处，对文件中所使用的缩写或约定，特别是特殊的缩写，进行必要的注释说明。

* 自己特有的命名风格，要自始至终保持一致，不可来回变化。

说明：个人的命名风格，在符合所在项目组或产品组的命名规则的前提下，才可使用。(即命名规则中没有规定到的地方才可有个人命名风格)。

* 对于变量命名，禁止取单个字符(如a, b, c,…i、j、k...)，建议要有具体含义外，最好还能表明其变量类型、数据类型等，但i、j、k作局部循环变量是允许的。

说明：变量，尤其是局部变量，如果用单个字符表示，很容易敲错(如i写成j)，而编译时又检查不出来，有可能为了这个小小的错误而花费大量的查错时间。

这样可以防止局部变量与全局变量重名。

* 命名规范必须与所使用的系统风格保持一致，并在同一项目中统一，比如采用UNIX的全小写加下划线的风格或大小写混排的方式，不要使用大小写与下划线混排的方式，用作特殊标识如标识成员变量或全局变量的m\_和g\_，其后加上大小写混排的方式是允许的。特殊说明:全局变量强制要求前面加上g\_然后是简要描述.

示例： Add\_User不允许，add\_user、AddUser、m\_AddUser允许。

* 除非必要，不要用数字或较奇怪的字符(如数字)来定义标识符,尽量使用有意义的单词

不规范**:使用较奇怪的字符，即使该软件允许。**譬如：gta123456789=45,tmp\*f=12。

* 在同一软件产品内，应规划好接口部分标识符(全局变量、函数)的命名，防止运行时出现调用错误

说明：对接口部分的标识符应该有更严格限制，防止冲突。如可规定接口部分的变量与常量之前加上“模块”标识等。

* 用正确的反义词组命名具有互斥意义的变量或相反动作的函数,但一定不能与MATLAB原有的函数命名相冲突。

说明：下面是一些在软件中常用的反义词组。

add / remove begin / end create / destroy

insert / delete first / last get / set

increment / decrement add / delete lock / unlock

open / close min / max old / new

start / stop next / previous source / target

show / hide send / receive source / destination

cut / paste up / down

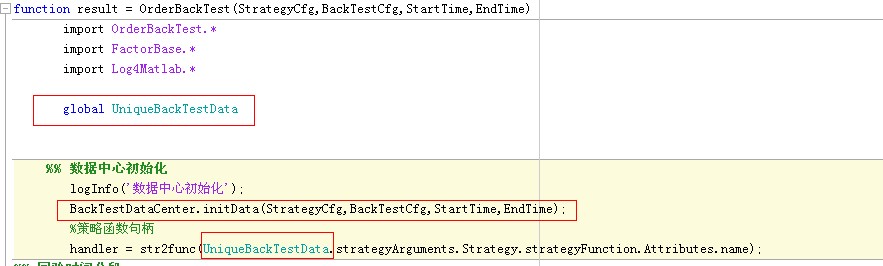
## 标识符变量部分

* 避免滥用全局变量，并在各接口内到处使用。

说明：全局变量是增大模块间耦合的原因之一，考虑到效率问题，可以内部使用，尽量对其它模块屏蔽，以降低模块间的耦合度。

* 仔细定义并明确全局变量的含义、作用、取值范围及全局变量间的关系。

说明：在对变量声明的同时，应对其含义、作用及取值范围进行注释说明，同时若有必要还应说明与其它变量的关系。

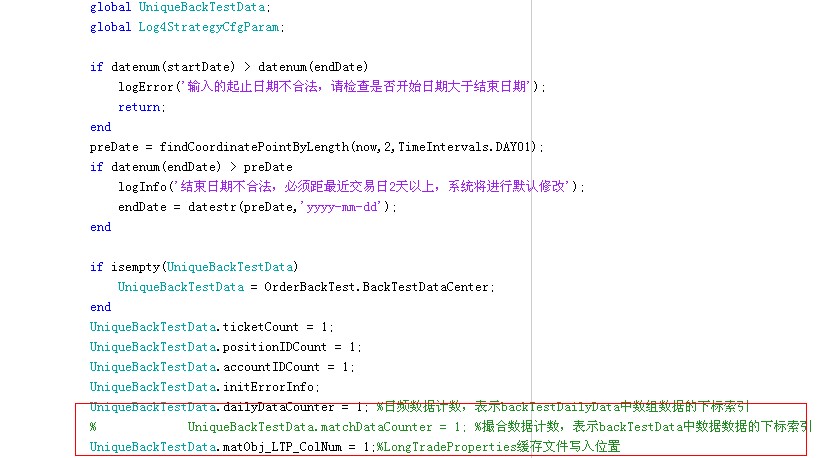


这里的全局变量所含的信息较大，而且全局变量的初始化在其它函数，因此有在这对全局变量做个大概说明。

* 明确全局变量与操作此全局变量的函数或过程的关系，如访问、修改及创建等。

说明：明确过程操作变量的关系后，将有利于程序的进一步优化、单元测试、系统联调以及代码维护等。这种关系的说明可在注释或文档中描述。

如上例在应该在BackTestDataCenter.innitDataCenter.innitData中做些说明。



* 当向全变量传递数据时，要十分小心，防止赋与不合理的值或越界等现象发生。

说明：对全局变量赋值时，若有必要应进行合法性检查，以提高代码的可靠性、稳定性。

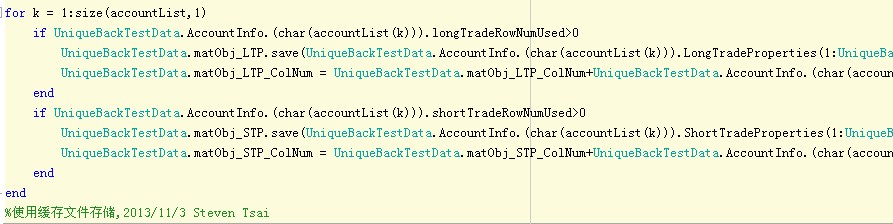
* 防止局部变量与全局变量同名。

说明：若使用了较好的命名规则，那么此问题可自动消除。

* 复杂变量如cell,struct的功能要单一，是针对一种事务的抽象,而非常灵活，关系过于复杂的cell或struct会导致处理上的误解和操作困难，增加代码的难度和稳定度。

说明：设计结构时应力争使结构代表一种现实事务的抽象，而不是同时代表多种。结构中的各元素应代表同一事务的不同侧面，而不应把描述没有关系或关系很弱的不同事务的元素放到同一结构中。

对于又是全局变量又是结构体的变量，功能复杂，最好对结构体的全局变量做出单独说明，进行统一管理。下面是一个全局结构体调用全局结构体函数，变量等。



* 仔细设计cell, struct中元素的布局与排列顺序，使结构容易理解、节省占用空间，并减少引起误用现象。

# 程序编写部分

* 注意运算符的优先级，并用括号明确表达式的操作顺序，避免使用默认优先级。

说明：防止阅读程序时产生误解，防止因默认的优先级与设计思想不符而导致程序出错。Matlab运算符优先级中，括号最高，其次乘方，加减乘除，冒号，关系运算，单逻辑与，或，标量值捷径与(&&)，标量值捷径或(||)。

逻辑运算中，如果次级运算要优先，则先括号括起来，如果有多个双目运算则可把优先级高的括起来以便于阅读。

示例：下列语句中的表达式

if ((a | b) && (a & c)) (1)

if ((a | b) < (c & d)) (2)

如果书写为

a | b && a & c

a | b < c & d

由于

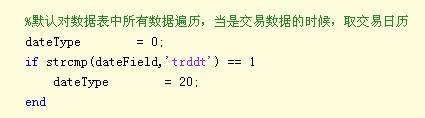
a | b && a & c = (a | b) && (a & c)，

(1)不会出错，但语句不易理解；

a | b < c & d = a | (b < c) & d，(2)造成了判断条件出错。

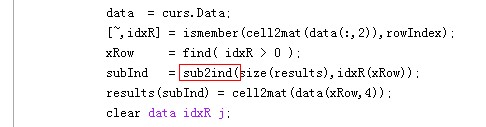
* 避免代码中直接使用不易理解的数字，必须要进行注释说明,说明其业务要求,必要时用枚举来代替。

示例：如下的程序可读性差。如下面的: dateType=20需要做出注释，否则维护者不知道深层含义。



* 不要使用难懂的技巧性很高的语句，除非很有必要时,但必须要有详细的注释说明。

说明：高技巧语句不等于高效率的程序，实际上程序的效率关键在于算法。如下面的sub2ind()函数



# 函数、过程部分

* 明确函数功能，精确(而不是近似)地实现函数设计,特别是用户API，不能设计太灵活。
* 在同一项目组应明确规定对接口函数参数的合法性检查应由函数的调用者负责还是由接口函数本身负责，缺省是由函数调用者负责。

说明：对于模块间接口函数的参数的合法性检查这一问题，往往有两个极端现象，即：要么是调用者和被调用者对参数均不作合法性检查，结果就遗漏了合法性检查这一必要的处理过程，造成问题隐患；要么就是调用者和被调用者均对参数进行合法性检查，这种情况虽不会造成问题，但产生了冗余代码，降低了效率。

* 函数的规模尽量限制在500行以内。

说明：不包括注释和空格行。

* 一个函数仅完成一件功能。
* 为简单功能编写函数。

说明：虽然为仅用一两行就可完成的功能去编函数好象没有必要，但用函数可使功能明确化，增加程序可读性，亦可方便维护、测试。

* 不要设计多用途面面俱到的函数。

说明：多功能集于一身的函数，很可能使函数的理解、测试、维护等变得困难。

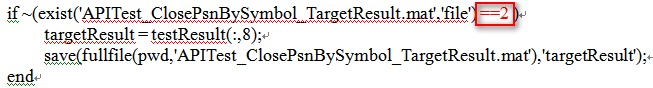
* 函数的功能应该是可以预测的，也就是只要输入数据相同就应产生同样的输出。
* 尽量不要编写依赖于其他函数内部实现的函数。

说明：此条为函数独立性的基本要求。

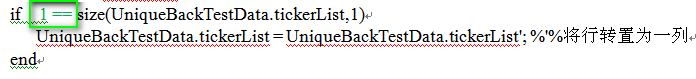
* if分支判断＝＝时，强制要求使用左值，即常量或常量表达式放在左边，严禁使用右值判断。

说明：右值判断，如果只输入了＝号，可能永远为真，尽管MATLAB做了检查，但这一习惯还是要在编程时养成。

【**不规范**】：

【**规范**】：

%判断一下tickerList的结构，如果是行，转成一列



* 检查函数所有参数输入的有效性。
* 检查函数所有非参数输入的有效性，如数据文件、公共变量等。

说明：函数的输入主要有两种：一种是参数输入；另一种是全局变量、数据文件的输入，即非参数输入。函数在使用输入之前，应进行必要的检查。

* 函数名应准确描述函数的功能。
* 使用动宾词组为执行某操作的函数命名。如果是OOP方法，可以只有动词(名词是对象本身)，但一定不能与MATLAB本身的函数冲突。
* 【建议】避免使用无意义或含义不清的动词为函数命名。

说明：避免用含义不清的动词等为函数命名，因为这些动词并没有说明要具体做什么。

* 【建议】函数的返回值要清楚、明了，让使用者不容易忽视错误情况。

说明：函数的每种出错返回值的意义要清晰、明了、准确，防止使用者误用、理解错误或忽视错误返回码。

* 让函数在调用点显得易懂、容易理解。
* 避免函数中不必要语句，防止程序中的垃圾代码。

说明：程序中的垃圾代码不仅占用额外的空间，而且还常常影响程序的功能与性能，很可能给程序的测试、维护等造成不必要的麻烦。

* 如果多段代码重复做同一件事情，那么在函数的划分上可能存在问题。

说明：若此段代码各语句之间有实质性关联并且是完成同一件功能的，那么可考虑把此段代码构造成一个新的函数。

* 功能不明确较小的函数，特别是仅有一个上级函数调用它时，应考虑把它合并到上级函数中，而不必单独存在。

说明：模块中函数划分的过多，一般会使函数间的接口变得复杂。所以过小的函数，特别是扇入很低的或功能不明确的函数，不值得单独存在。

* 设计高扇入、合理扇出(小于7)的函数。

说明：扇出是指一个函数直接调用(控制)其它函数的数目，而扇入是指有多少上级函数调用它。

扇出过大，表明函数过分复杂，需要控制和协调过多的下级函数；而扇出过小，如总是1，表明函数的调用层次可能过多，这样不利程序阅读和函数结构的分析，并且程序运行时会对系统资源如堆栈空间等造成压力。函数较合理的扇出(调度函数除外)通常是3-5。扇出太大，一般是由于缺乏中间层次，可适当增加中间层次的函数。扇出太小，可把下级函数进一步分解多个函数，或合并到上级函数中。当然分解或合并函数时，不能改变要实现的功能，也不能违背函数间的独立性。

扇入越大，表明使用此函数的上级函数越多，这样的函数使用效率高，但不能违背函数间的独立性而单纯地追求高扇入。公共模块中的函数及底层函数应该有较高的扇入。

较良好的软件结构通常是顶层函数的扇出较高，中层函数的扇出较少，而底层函数则扇入到公共模块中。

* 减少函数本身或函数间的递归调用。

说明：递归调用特别是函数间的递归调用(如A->B->C->A)，影响程序的可理解性；递归调用一般都占用较多的系统资源(如栈空间)；递归调用对程序的测试有一定影响。故除非为某些算法或功能的实现方便，应减少没必要的递归调用。

* 仔细分析模块的功能及性能需求，并进一步细分，同时若有必要画出有关数据流图，据此来进行模块的函数划分与组织。

说明：函数的划分与组织是模块的实现过程中很关键的步骤，如何划分出合理的函数结构，关系到模块的最终效率和可维护性、可测性等。根据模块的功能图或/及数据流图映射出函数结构是常用方法之一。

* 改进模块中函数的结构，降低函数间的耦合度，并提高函数的独立性以及代码可读性、效率和可维护性。优化函数结构时，要遵守以下原则：

(1)不能影响模块功能的实现。

(2)仔细考查模块或函数出错处理及模块的性能要求并进行完善。

(3)通过分解或合并函数来改进软件结构。

(4)考查函数的规模，过大的要进行分解。

(5)降低函数间接口的复杂度。

(6)不同层次的函数调用要有较合理的扇入、扇出。

(7)函数功能应可预测。

(8)提高函数内聚。(单一功能的函数内聚最高)

说明：对初步划分后的函数结构应进行改进、优化，使之更为合理。

# 效率部分

* 编程时要经常注意代码的效率。

说明：代码效率分为全局效率、局部效率、时间效率及空间效率。全局效率是站在整个系统的角度上的系统效率；局部效率是站在模块或函数角度上的效率；时间效率是程序处理输入任务所需的时间长短；空间效率是程序所需内存空间，如机器代码空间大小、数据空间大小、栈空间大小等。

* 在保证软件系统的正确性、稳定性、可读性及可测性的前提下，提高代码效率。

说明：不能一味地追求代码效率，而对软件的正确性、稳定性、可读性及可测性造成影响。

* 局部效率应为全局效率服务，不能因为提高局部效率而对全局效率造成影响。
* 尽量使用矩阵运算和MATLAB本身己有的处理函数，以及对程序算法的优化来提高代码效率。

说明：这种方式是解决软件空间效率的根本办法。

* 如有必要，减少循环代码的使用，或者至少循环体内工作量最小化。

说明：应仔细考虑循环体内的语句是否可以放在循环体之外，使循环体内工作量最小，从而提高程序的时间效率。

* 仔细分析有关算法，并进行优化。
* 对模块中函数的划分及组织方式进行分析、优化，改进模块中函数的组织结构，提高程序效率。

说明：软件系统的效率主要与算法、处理任务方式、系统功能及函数结构有很大关系，仅在代码上下功夫一般不能解决根本问题。

* 编程时，要随时留心代码效率；优化代码时，要考虑周全。
* 不应花过多的时间拼命地提高调用不很频繁的函数代码效率。

说明：对代码优化可提高效率，但若考虑不周很有可能引起严重后果。

* 在多重循环中，应将最忙的循环放在最内层。

说明：减少CPU切入循环层的次数。

* 尽量减少循环嵌套层次。
* 避免循环体内含判断语句，应将循环语句置于判断语句的代码块之中。

说明：目的是减少判断次数。循环体中的判断语句是否可以移到循环体外，要视程序的具体情况而言，一般情况，与循环变量无关的判断语句可以移到循环体外，而有关的则不可以。

# 利用内部或外部工具部分

目的：方便自己编程定位。在任务多，同时相应模块功能没有完善(如基本功能已经实现，但是效率很差)，为日后完善提供保障

当然Reports功能还可以做其他事情：如冗余代码分析提示，生成相应report等

## Matlab内部工具

部分:TODO,FIXME,XXX

* **TODO:详细说明**

如果代码中有该标识，说明在标识处有功能代码待编写，待实现的功能在说明中会简略说明

如：%TODO:此处函数fetchDailyData()取功能没有实现

* **FIXME: 详细说明**

如果代码中有该标识，说明标识处代码需要修正，甚至代码是错误的，不能工作，需要修复，如何修正会在说明中简略说明。

如：% **FIXME**:此处函数fetchDailyData()取数函数，已经编写完成，但程序内部逻辑有错

* **XXX: 详细说明**

如果代码中有该标识，说明标识处代码虽然实现了功能，但是实现的方法有待商榷，希望将来能改进，要改进的地方会在说明中简略说明。

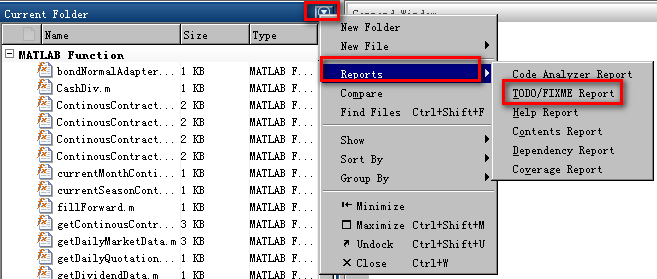
如：% **XXX**:此处函数fetchDailyData()取数功能已经正确取到，但耗费时间非常长，不符合QIA要求，要进一步优化

* 在MATLAB查找TODO，**FIXME ，XXX，请看图解过程。**

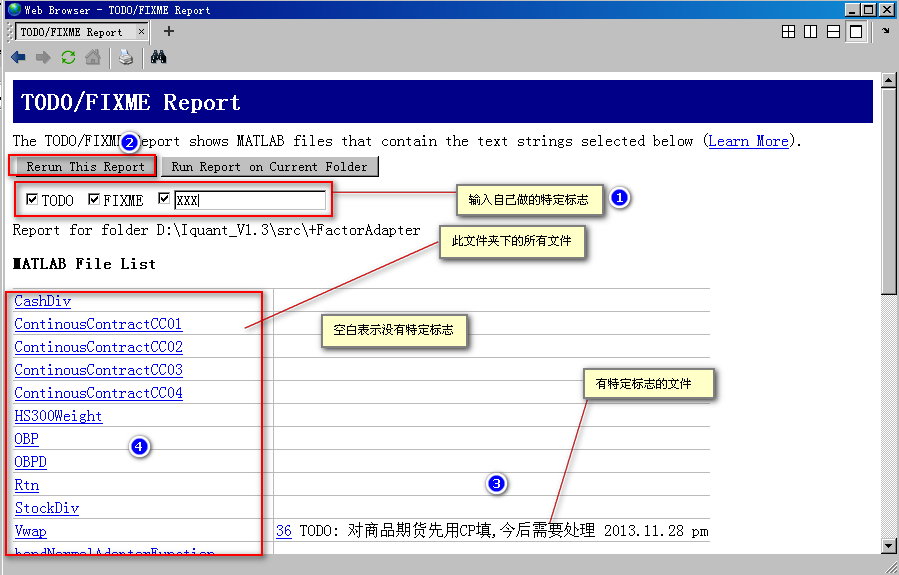
**第一步：进入相应的文件夹**



**第二步：单击小圈圈，并定位到RePorts，找到相应的Report**

****

**第三步：输入特定标志和查看结果**

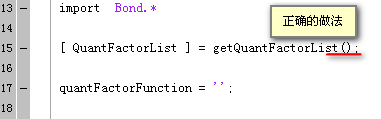


# 函数和类调用部分

* 【建议】函数或类应用时，必须带上括号

函数调用和类调用中，如果遇到该函数或该类没有参数输入，在调用中我们必须带上()，即使Matlab识别不带参数函数直接输入函数名。

**【规范】**



**【不规范】**

