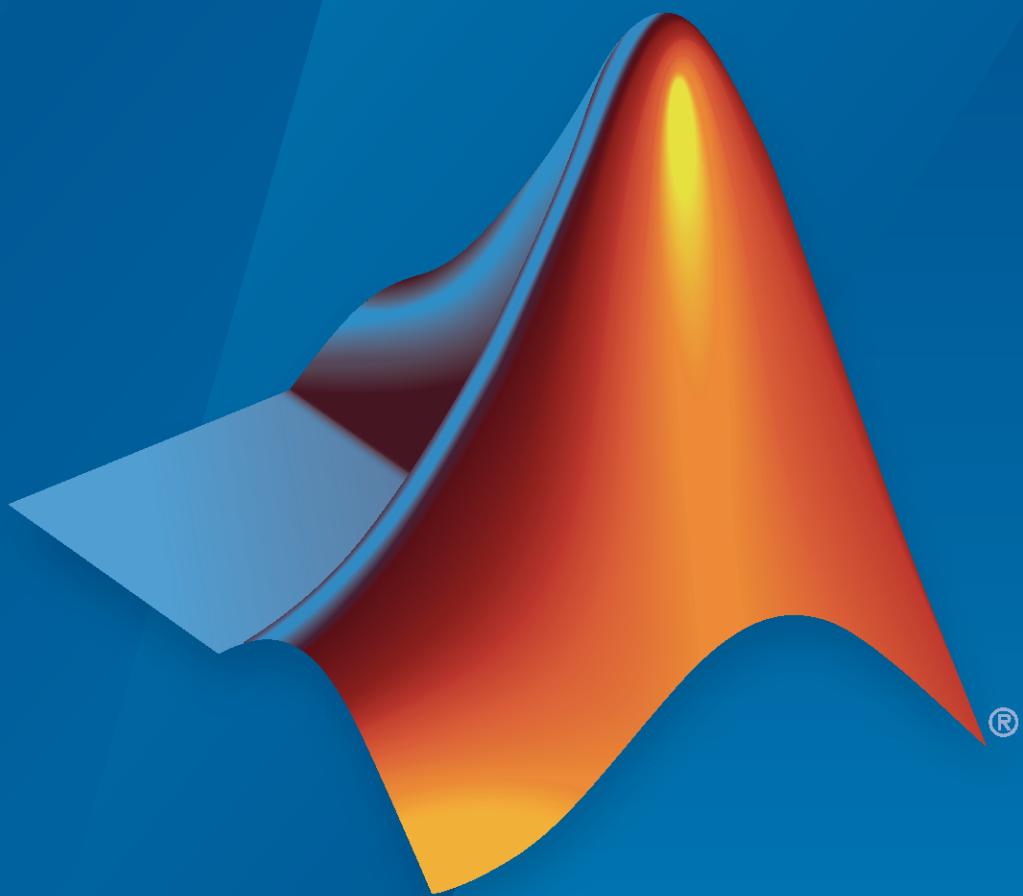


MATLAB®

编程基础



MATLAB®

R2019b

 MathWorks®

# 如何联系 MathWorks



最新动态: [www.mathworks.com](http://www.mathworks.com)  
销售和服务: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
用户社区: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
技术支持: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



电话: 010-59827000



迈斯沃克软件(北京)有限公司  
北京市朝阳区望京东园四区6号楼  
北望金辉大厦16层1604

## MATLAB 编程基础

© COPYRIGHT 1984–2019 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

**FEDERAL ACQUISITION:** This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

## 商标

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

## 专利

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## 修订历史记录

2004 年 6 月	第一次印刷	MATLAB 7.0 (版本 14) 中的新增内容
2004 年 10 月	仅限在线版本	MATLAB 7.0.1 (版本 14SP1) 中的修订内容
2005 年 3 月	仅限在线版本	MATLAB 7.0.4 (版本 14SP2) 中的修订内容
2005 年 6 月	第二次印刷	MATLAB 7.0.4 中的微小修订内容
2005 年 9 月	仅限在线版本	MATLAB 7.1 (版本 14SP3) 中的修订内容
2006 年 3 月	仅限在线版本	MATLAB 7.2 (版本 2006a) 中的修订内容
2006 年 9 月	仅限在线版本	MATLAB 7.3 (版本 2006b) 中的修订内容
2007 年 3 月	仅限在线版本	MATLAB 7.4 (版本 2007a) 中的修订内容
2007 年 9 月	仅限在线版本	MATLAB 7.5 (版本 2007b) 中的修订内容
2008 年 3 月	仅限在线版本	MATLAB 7.6 (版本 2008a) 中的修订内容
2008 年 10 月	仅限在线版本	MATLAB 7.7 (版本 2008b) 中的修订内容
2009 年 3 月	仅限在线版本	MATLAB 7.8 (版本 2009a) 中的修订内容
2009 年 9 月	仅限在线版本	MATLAB 7.9 (版本 2009b) 中的修订内容
2010 年 3 月	仅限在线版本	MATLAB 7.10 (版本 2010a) 中的修订内容
2010 年 9 月	仅限在线版本	MATLAB 7.11 (版本 2010b) 中的修订内容
2011 年 4 月	仅限在线版本	MATLAB 7.12 (版本 2011a) 中的修订内容
2011 年 9 月	仅限在线版本	MATLAB 7.13 (版本 2011b) 中的修订内容
2012 年 3 月	仅限在线版本	MATLAB 7.14 (版本 2012a) 中的修订内容
2012 年 9 月	仅限在线版本	MATLAB 8.0 (版本 2012b) 中的修订内容
2013 年 3 月	仅限在线版本	MATLAB 8.1 (版本 2013a) 中的修订内容
2013 年 9 月	仅限在线版本	MATLAB 8.2 (版本 2013b) 中的修订内容
2014 年 3 月	仅限在线版本	MATLAB 8.3 (版本 2014a) 中的修订内容
2014 年 10 月	仅限在线版本	MATLAB 8.4 (版本 2014b) 中的修订内容
2015 年 3 月	仅限在线版本	MATLAB 8.5 (版本 2015a) 中的修订内容
2015 年 9 月	仅限在线版本	MATLAB 8.6 (版本 2015b) 中的修订内容
2015 年 10 月	仅限在线版本	MATLAB 8.5.1 (版本 2015aSP1) 中的再发布内容
2016 年 3 月	仅限在线版本	MATLAB 9.0 (版本 2016a) 中的修订内容
2016 年 9 月	仅限在线版本	MATLAB 9.1 (版本 2016b) 中的修订内容
2017 年 3 月	仅限在线版本	MATLAB 9.2 (版本 2017a) 中的修订内容
2017 年 9 月	仅限在线版本	MATLAB 9.3 (版本 2017b) 中的修订内容
2018 年 3 月	仅限在线版本	MATLAB 9.4 (版本 2018a) 中的修订内容
2018 年 9 月	仅限在线版本	MATLAB 9.5 (版本 2018b) 中的修订内容
2019 年 3 月	仅限在线版本	MATLAB 9.6 (版本 2019a) 中的修订内容
2019 年 9 月	仅限在线版本	MATLAB 9.7 (版本 2019b) 中的修订内容



## 语言

### 语法基础知识

1

在多行上延续长语句 .....	1-2
调用函数 .....	1-3
忽略函数输出 .....	1-4
变量名称 .....	1-5
有效名称 .....	1-5
与函数名称冲突 .....	1-5
大小写和空格敏感性 .....	1-6
命令与函数语法 .....	1-7
命令与函数语法 .....	1-7
避免常见的语法错误 .....	1-7
MATLAB 如何识别命令语法 .....	1-8
调用函数时的常见错误 .....	1-10
函数名称和变量名称冲突 .....	1-10
未定义的函数或变量 .....	1-10

### 程序组件

2

MATLAB 运算符和特殊字符 .....	2-2
算术运算符 .....	2-2
关系运算符 .....	2-2
逻辑运算符 .....	2-2
特殊字符 .....	2-3
字符串和字符格式化 .....	2-16
数组与矩阵运算 .....	2-19
简介 .....	2-19
数组运算 .....	2-19
矩阵运算 .....	2-21

<b>基本运算的兼容数组大小</b>	2-23
大小兼容的输入	2-23
大小不兼容的输入	2-25
示例	2-25
<b>使用关系运算符进行数组比较</b>	2-27
数组比较	2-27
逻辑语句	2-29
<b>运算符优先级</b>	2-30
AND 和 OR 运算符的优先级	2-30
覆盖默认优先级	2-30
<b>使用容差为类似数据点求平均值</b>	2-32
<b>使用容差为散点数据分组</b>	2-34
<b>按位运算</b>	2-36
<b>执行循环冗余校验</b>	2-41
<b>条件语句</b>	2-44
<b>循环控制语句</b>	2-46
<b>正则表达式</b>	2-48
什么是正则表达式?	2-48
构建表达式的步骤	2-49
运算符和字符	2-51
<b>正则表达式中的前向断言</b>	2-58
前向断言	2-58
重叠匹配项	2-58
逻辑 AND 条件	2-59
<b>正则表达式中的标文</b>	2-61
简介	2-61
多个标文	2-63
不匹配的标文	2-63
替代文本中的标文	2-64
命名捕获	2-64
<b>动态正则表达式</b>	2-67
简介	2-67
动态匹配表达式 - (??expr)	2-68
修改匹配表达式的命令 - (??@cmd)	2-68
满足功能性需求的命令 - (?@cmd)	2-69
替代表达式中的命令 - \${cmd}	2-71
<b>逗号分隔的列表</b>	2-73
什么是逗号分隔的列表?	2-73
生成逗号分隔的列表	2-73
从逗号分隔的列表分配输出	2-75
为逗号分隔的列表赋值	2-75
如何使用逗号分隔的列表	2-76

快速傅里叶变换示例	2-78
-----------	------

<b>eval 函数的替代方法</b>	2-80
为什么要避免使用 eval 函数?	2-80
带有序列名称的变量	2-80
带有序列名称的文件	2-81
变量中的函数名称	2-81
变量中的字段名称	2-81
错误的处理方式	2-82

## 类 (数据类型)

3

### MATLAB 类概述

<b>MATLAB 基础类</b>	3-2
-------------------	-----

4

### 数值类

<b>整数</b>	4-2
整数类	4-2
创建整数数据	4-2
整数类的算术运算	4-3
整数类的最大值和最小值	4-4

<b>浮点数</b>	4-6
双精度浮点	4-6
单精度浮点	4-6
创建浮点数据	4-6
浮点数的算术运算	4-7
浮点类的最大值和最小值	4-8
浮点数据的精度	4-9
避免浮点算术运算出现常见问题	4-10

<b>创建复数</b>	4-13
-------------	------

<b>无穷大和 NaN</b>	4-14
无穷大	4-14
NaN	4-14

<b>确定数值类</b>	4-16
--------------	------

<b>数值的显示格式</b>	4-17
----------------	------

<b>整数算术运算</b>	4-19
---------------	------

**逻辑类****5**

查找符合条件的数组元素 .....	5-2
将逻辑数组约简为单个值 .....	5-6

**字符和字符串****6**

字符串数组和字符数组中的文本 .....	6-2
创建字符串数组 .....	6-5
<b>字符向量元胞数组</b> .....	6-12
创建字符向量元胞数组 .....	6-12
访问元胞数组中的字符向量 .....	6-12
将元胞数组转换为字符串数组 .....	6-13
分析字符串数组的文本数据 .....	6-15
测试空字符串和缺失值 .....	6-20
<b>格式化文本</b> .....	6-24
格式化操作符的字段 .....	6-24
设置字段宽度和精度 .....	6-28
使用标识符的限制 .....	6-29
比较文本 .....	6-31
搜索和替换文本 .....	6-36
<b>从数值转换为字符数组</b> .....	6-41
函数摘要 .....	6-41
将数字转换为字符代码 .....	6-41
将数字表示为文本 .....	6-41
转换为特定基数 .....	6-42
<b>从字符数组转换为数值</b> .....	6-43
函数摘要 .....	6-43
从字符代码转换 .....	6-43
转换代表数值的文本 .....	6-44
从特定基数转换 .....	6-44
<b>十六进制和二进制值</b> .....	6-45

<b>有关字符串数组的常见问题解答</b>	6-48
为什么在命令形式中使用字符串会返回错误?	6-48
为什么元胞数组中的字符串返回错误?	6-48
为什么使用 <code>length()</code> 调用字符串会返回 1?	6-49
为什么 <code>isempty("")</code> 返回 0?	6-50
为什么使用方括号追加字符串返回多个字符串?	6-51
<b>更新您的代码以接受字符串</b>	6-52
什么是字符串数组?	6-52
在旧 API 中采用字符串类型的建议方法	6-52
如何在旧 API 中采用字符串数组	6-53
在新代码中采用字符串的建议方法	6-54
如何在新代码中保持兼容性	6-55
如何手动转换输入参数	6-55
如何检查参数数据类型	6-56
字符和字符串数组的术语	6-57
<b>函数摘要</b>	6-59

## 日期和时间

# 7

<b>表示 MATLAB 中的日期和时间</b>	7-2
<b>指定时区</b>	7-5
<b>将日期和时间转换为儒略日期或 POSIX 时间</b>	7-7
<b>设置日期和时间显示格式</b>	7-10
单个日期和持续时间数组的格式	7-10
<code>datetime</code> 显示格式	7-10
<code>duration</code> 显示格式	7-11
<code>calendarDuration</code> 显示格式	7-11
默认 <code>datetime</code> 格式	7-12
<b>生成日期与时间的序列</b>	7-13
两个端点间已知步长的日期时间或持续时间值的序列	7-13
添加持续时间或日历持续时间以创建日期的序列	7-15
指定日期或持续时间序列的长度和端点	7-16
使用日历规则的日期时间值的序列	7-16
<b>在不同区域设置之间共享代码和数据</b>	7-19
编写独立于区域设置的日期和时间代码	7-19
用其他语言编写日期	7-20
读取以其他语言编写的日期	7-20
<b>提取或分配日期时间数组的日期和时间分量</b>	7-21
<b>合并来自各自变量的日期和时间</b>	7-24
<b>日期和时间算术运算</b>	7-26

<b>比较日期和时间</b>	7-31
<b>绘制日期和持续时间图</b>	7-34
绘制日期线图	7-34
绘制持续时间线图	7-35
用日期和持续时间绘制散点图	7-37
支持日期和持续时间的绘图	7-38
<b>支持日期和时间数组的核心函数</b>	7-39
<b>在日期时间数组、数值和文本之间转换</b>	7-40
概述	7-40
在日期时间和字符向量之间转换	7-40
在日期时间和字符串数组之间转换	7-41
在日期时间和日期向量之间转换	7-42
将日期序列值转换为日期时间	7-43
将日期时间数组转换为数值	7-43
<b>日期向量和字符串结转</b>	7-45
<b>转换日期向量返回意外输出</b>	7-46

## 分类数组

# 8

<b>创建分类数组</b>	8-2
<b>将表变量中的文本转换为分类数组</b>	8-6
<b>对分类数据绘图</b>	8-10
<b>比较分类数组元素</b>	8-16
<b>合并分类数组</b>	8-19
<b>使用乘法合并分类数组</b>	8-22
<b>使用分类数组访问数据</b>	8-24
按类别选择数据	8-24
使用分类数组访问数据的常见方法	8-24
<b>使用受保护的分类数组</b>	8-30
<b>使用分类数组的好处</b>	8-34
分类数据的自然表示形式	8-34
字符向量的数学排序	8-34
降低内存要求	8-34
<b>有序分类数组</b>	8-36
类别的顺序	8-36
如何创建有序分类数组	8-36

使用有序分类数组 . . . . .	8-37
<b>支持分类数组的核心函数 . . . . .</b>	<b>8-39</b>

## 表

**9**

<b>创建和使用表 . . . . .</b>	<b>9-2</b>
添加和删除表行 . . . . .	9-11
添加、删除和重新排列表变量 . . . . .	9-14
清除表中的杂乱数据和缺失数据 . . . . .	9-20
修改单位、说明和表变量名称 . . . . .	9-25
向表和时间表中添加自定义属性 . . . . .	9-28
<b>访问表中的数据 . . . . .</b>	<b>9-33</b>
表索引语法一览 . . . . .	9-33
包含指定的行和变量的表 . . . . .	9-35
使用圆点表示法和逻辑值提取数据 . . . . .	9-38
圆点表示法支持任意变量名称或表达式 . . . . .	9-41
从指定的行和变量中提取数据 . . . . .	9-43
对表执行计算 . . . . .	9-45
将数据拆分为不同组并计算统计量 . . . . .	9-48
拆分表数据变量并应用函数 . . . . .	9-51
使用表的好处 . . . . .	9-55
<b>对变量分组以拆分数据 . . . . .</b>	<b>9-60</b>
分组变量 . . . . .	9-60
组定义 . . . . .	9-60
拆分-应用-合并工作流 . . . . .	9-60
缺少组值 . . . . .	9-61
R2016b 中对 DimensionNames 属性的更改 . . . . .	9-62

## 时间表

**10**

创建时间表 . . . . .	10-2
对时间表中的数据进行重采样和聚合 . . . . .	10-5

合并时间表并同步其数据 . . . . .	10-8
使用不同的方法对时间表变量重设时间并进行同步 . . . . .	10-14
按行时间和变量类型选择时间表数据 . . . . .	10-18
清理包含缺失、重复或不均匀时间的时间表 . . . . .	10-23
在表和时间表运算中使用行标签 . . . . .	10-31
洛马普列塔地震分析 . . . . .	10-36
使用 timetable 预处理和探索时间戳数据 . . . . .	10-46

## 结构体

# 11

创建结构体数组 . . . . .	11-2
访问结构体数组中的数据 . . . . .	11-5
访问标量结构体中的数据 . . . . .	11-5
通过对结构体数组进行索引来访问数据 . . . . .	11-6
串联结构体 . . . . .	11-9
基于变量生成字段名称 . . . . .	11-11
访问嵌套结构体中的数据 . . . . .	11-12
访问非标量结构体数组的元素 . . . . .	11-14
结构体数组中数据的组织方法 . . . . .	11-16
平面组织 . . . . .	11-16
按元素组织 . . . . .	11-17
结构体数组的内存要求 . . . . .	11-18

## 元胞数组

# 12

什么是元胞数组? . . . . .	12-2
创建元胞数组 . . . . .	12-3
访问元胞数组中的数据 . . . . .	12-5
将元胞添加到元胞数组 . . . . .	12-8

删除元胞数组中的数据 . . . . .	12-9
合并元胞数组 . . . . .	12-10
将元胞数组的内容传递给函数 . . . . .	12-11
为元胞数组预分配内存 . . . . .	12-15
元胞数组与结构体数组 . . . . .	12-16
访问部分元胞的多级索引 . . . . .	12-20

## 函数句柄

# 13

<b>创建函数句柄 . . . . .</b>	13-2
什么是函数句柄? . . . . .	13-2
创建函数句柄 . . . . .	13-2
匿名函数 . . . . .	13-3
由函数句柄组成的数组 . . . . .	13-4
保存和加载函数句柄 . . . . .	13-4
<b>将一个函数传递到另一个函数 . . . . .</b>	13-5
<b>使用函数句柄调用局部函数 . . . . .</b>	13-6
<b>比较函数句柄 . . . . .</b>	13-8

## 映射容器

# 14

<b>映射数据结构体概述 . . . . .</b>	14-2
<b>Map 类的说明 . . . . .</b>	14-3
Map 类的属性 . . . . .	14-3
Map 类的方法 . . . . .	14-3
<b>创建映射对象 . . . . .</b>	14-5
构造空的映射对象 . . . . .	14-5
构造已初始化的映射对象 . . . . .	14-5
组合映射对象 . . . . .	14-6
<b>检查映射的内容 . . . . .</b>	14-7
<b>使用键索引读取和写入 . . . . .</b>	14-8
从映射中读取 . . . . .	14-8
添加键/值对组 . . . . .	14-9
使用串联构建映射 . . . . .	14-9

<b>修改映射中的键和值</b> . . . . .	14-11
从映射中删除键和值 . . . . .	14-11
修改值 . . . . .	14-11
修改键 . . . . .	14-12
修改映射副本 . . . . .	14-12
<b>映射到不同值类型</b> . . . . .	14-13
映射到结构体数组 . . . . .	14-13
映射到元胞数组 . . . . .	14-14

## 15

### 合并不同类

<b>不同类的有效合并</b> . . . . .	15-2
<b>合并不同的整数类型</b> . . . . .	15-3
概述 . . . . .	15-3
合并不同大小的整数的示例 . . . . .	15-3
合并有符号与无符号整数的示例 . . . . .	15-3
<b>合并整数与非整数数据</b> . . . . .	15-5
<b>合并元胞数组与非元胞数组</b> . . . . .	15-6
<b>空矩阵</b> . . . . .	15-7
<b>串联示例</b> . . . . .	15-8
合并单精度与双精度类型值 . . . . .	15-8
合并整数与双精度类型值 . . . . .	15-8
合并字符与双精度类型值 . . . . .	15-8
合并逻辑值与双精度类型值 . . . . .	15-8

## 16

### 使用对象

<b>对象行为</b> . . . . .	16-2
两种复制行为 . . . . .	16-2
句柄对象复制 . . . . .	16-2
值对象复制行为 . . . . .	16-2
句柄对象复制行为 . . . . .	16-3
测试句柄或值类 . . . . .	16-5

**脚本和函数****脚本**

<b>创建脚本</b> .....	18-2
<b>向程序中添加注释</b> .....	18-3
<b>代码节</b> .....	18-5
将您的文件分为多个代码节 .....	18-5
执行代码节 .....	18-5
在文件中的各代码节之间导航 .....	18-6
执行代码节的示例 .....	18-6
更改代码节的外观 .....	18-8
同时使用代码节与控制语句和函数 .....	18-9
<b>脚本与函数</b> .....	18-12
<b>向脚本中添加函数</b> .....	18-13
添加局部函数 .....	18-13
访问帮助 .....	18-13
运行代码 .....	18-14
在实时脚本中添加和运行节 .....	18-14

**实时脚本和函数**

<b>什么是实时脚本或实时函数?</b> .....	19-2
与纯代码脚本和函数的差异 .....	19-3
要求 .....	19-4
不支持的功能 .....	19-4
<b>在实时编辑器中创建实时脚本</b> .....	19-6
创建实时脚本 .....	19-6
添加代码 .....	19-6
运行代码 .....	19-8
显示输出 .....	19-8
设置文本格式 .....	19-9
<b>在实时脚本中运行节</b> .....	19-11
将文件划分为不同节 .....	19-11

<b>执行节</b>	<b>19-11</b>
<b>在实时编辑器中调试代码</b>	<b>19-13</b>
显示输出	19-13
使用运行到此行进行调试	19-14
在调试时查看变量值	19-15
暂停运行文件	19-16
结束调试会话	19-16
步入函数	19-16
添加断点并运行	19-17
<b>修改实时脚本中的图窗</b>	<b>19-20</b>
探查数据	19-20
更改图窗时更新代码	19-22
添加格式设置和注释	19-22
添加和修改多个子图	19-24
保存和打印图窗	19-28
<b>在实时编辑器中格式化文件</b>	<b>19-29</b>
自动格式设置	19-30
<b>将方程插入实时编辑器中</b>	<b>19-33</b>
以交互方式插入方程	19-33
插入 LaTeX 方程	19-34
<b>将交互式控件添加到实时脚本</b>	<b>19-41</b>
插入控件	19-41
使用多个交互式控件创建实时脚本	19-42
共享实时脚本	19-43
<b>将交互式任务添加到实时脚本中</b>	<b>19-45</b>
什么是实时编辑器任务?	19-45
插入任务	19-45
运行任务及其周围的代码	19-48
修改输出参数名称	19-49
查看和编辑生成的代码	19-49
<b>创建实时函数</b>	<b>19-51</b>
创建实时函数	19-51
添加代码	19-51
添加帮助	19-51
运行实时函数	19-52
<b>为实时函数添加帮助</b>	<b>19-54</b>
<b>共享实时脚本和函数</b>	<b>19-57</b>
共享前隐藏代码	19-57
<b>实时代码文件格式 (.mlx)</b>	<b>19-59</b>
实时代码文件格式的好处	19-59
源代码管理	19-59
<b>实时编辑器介绍</b>	<b>19-60</b>
<b>使用实时编辑器加快探索编程速度</b>	<b>19-65</b>

使用实时编辑器创建交互式记叙脚本	19-69
使用实时编辑器创建交互式课程材料	19-76
使用实时编辑器创建示例	19-82
使用实时编辑器创建交互式表单	19-83
使用实时编辑器创建实时控制板	19-86

## 20

### 函数基础知识

<b>在文件中创建函数</b>	20-2
函数定义语法	20-2
函数和文件的内容	20-3
End 语句	20-3
<b>为程序添加帮助</b>	20-5
<b>在编辑器中运行函数</b>	20-7
<b>基础和函数工作区</b>	20-8
<b>在工作区之间共享数据</b>	20-9
简介	20-9
最佳做法：传递参数	20-9
嵌套函数	20-9
持久变量	20-10
全局变量	20-10
在另一工作区中计算	20-11
<b>在编辑器中检查变量作用域</b>	20-13
使用自动函数和变量高亮显示功能	20-13
使用自动函数和变量高亮显示功能的示例	20-13
<b>函数类型</b>	20-16
文件中的局部和嵌套函数	20-16
子文件夹中的私有函数	20-17
无需文件的匿名函数	20-17
<b>匿名函数</b>	20-19
什么是匿名函数？	20-19
表达式中的变量	20-19
多个匿名函数	20-20
不带输入的函数	20-21
带有多个输入或输出的函数	20-21
匿名函数的数组	20-22
<b>局部函数</b>	20-24

<b>嵌套函数</b> . . . . .	20-26
什么是嵌套函数? . . . . .	20-26
嵌套函数的要求 . . . . .	20-26
在父函数与嵌套函数之间共享变量 . . . . .	20-26
使用句柄存储函数参数 . . . . .	20-27
嵌套函数的可见性 . . . . .	20-30
<b>嵌套函数和匿名函数中的变量</b> . . . . .	20-32
<b>私有函数</b> . . . . .	20-33
<b>函数优先顺序</b> . . . . .	20-34
函数优先顺序规则的更改 . . . . .	20-35
<b>针对 R2019b 对函数优先顺序的更改更新代码</b> . . . . .	20-36
标识符在一个函数内只能用于一个目的。 . . . . .	20-36
没有显式声明的标识符可能不被视为变量 . . . . .	20-36
变量无法在父函数和嵌套函数之间隐式共享 . . . . .	20-37
基于通配符导入的优先级发生更改 . . . . .	20-37
完全限定的导入函数不能与嵌套函数同名 . . . . .	20-38
完全限定的导入函数会遮蔽同名的外层作用域定义 . . . . .	20-38
未发现导入时的错误处理 . . . . .	20-39
嵌套函数从父函数继承 import 语句 . . . . .	20-39
复合名称解析优先级的更改 . . . . .	20-40
匿名函数可以同时包含已解析和未解析的标识符 . . . . .	20-40
<b>对函数调用结果进行索引</b> . . . . .	20-41
示例 . . . . .	20-41
支持的语法 . . . . .	20-41

## 21

### 函数参数

<b>查找函数参数的数量</b> . . . . .	21-2
<b>支持可变数量的输入</b> . . . . .	21-4
<b>支持可变数量的输出</b> . . . . .	21-5
<b>验证函数参数的数量</b> . . . . .	21-6
<b>嵌套函数中的参数检查</b> . . . . .	21-8
<b>忽略函数输入</b> . . . . .	21-10
<b>通过 validateattributes 检查函数输入</b> . . . . .	21-11
<b>解析函数输入</b> . . . . .	21-13
<b>输入解析器验证函数</b> . . . . .	21-16

<b>调试 MATLAB 程序 . . . . .</b>	<b>22-2</b>
设置断点 . . . . .	22-2
运行文件 . . . . .	22-2
暂停运行文件 . . . . .	22-3
查找并解决问题 . . . . .	22-3
逐步执行文件 . . . . .	22-5
结束调试会话 . . . . .	22-5
<b>设置断点 . . . . .</b>	<b>22-6</b>
标准断点 . . . . .	22-6
条件断点 . . . . .	22-7
错误断点 . . . . .	22-8
匿名函数中的断点 . . . . .	22-8
无效断点 . . . . .	22-9
禁用断点 . . . . .	22-9
清除断点 . . . . .	22-9
<b>调试时检验值 . . . . .</b>	<b>22-11</b>
选择工作区 . . . . .	22-11
查看变量值 . . . . .	22-11

<b>发布和共享 MATLAB 代码 . . . . .</b>	<b>23-2</b>
在实时编辑器中创建和共享实时脚本 . . . . .	23-2
发布 MATLAB 代码 . . . . .	23-2
添加帮助和创建文档 . . . . .	23-4
<b>发布标记 . . . . .</b>	<b>23-5</b>
标记概述 . . . . .	23-5
节和节标题 . . . . .	23-7
文本格式设置 . . . . .	23-8
项目符号和编号列表 . . . . .	23-9
文本和代码块 . . . . .	23-9
外部文件内容 . . . . .	23-10
外部图形 . . . . .	23-11
图像快照 . . . . .	23-12
LaTeX 方程 . . . . .	23-13
超链接 . . . . .	23-14
HTML 标记 . . . . .	23-17
LaTeX 标记 . . . . .	23-17
<b>用于发布的输出预设 . . . . .</b>	<b>23-20</b>
如何编辑发布选项 . . . . .	23-20
指定输出文件 . . . . .	23-21
在发布过程中运行代码 . . . . .	23-21
在发布输出时操作图形 . . . . .	23-23

保存发布设置 . . . . .	23-25
管理发布配置 . . . . .	23-26

## 编码及工作效率的提示

# 24

<b>在编辑器中打开和保存文件</b> . . . . .	24-2
打开现有文件 . . . . .	24-2
保存文件 . . . . .	24-3
<b>检查代码中的错误和警告</b> . . . . .	24-5
在编辑器中自动检查代码 - 代码分析器 . . . . .	24-5
创建代码分析器消息报告 . . . . .	24-8
调整代码分析器消息指示标记和消息 . . . . .	24-8
了解包含隐藏消息的代码 . . . . .	24-10
了解代码分析的局限 . . . . .	24-11
启用 MATLAB 编译器部署消息 . . . . .	24-13
<b>提高代码可读性</b> . . . . .	24-14
缩进代码 . . . . .	24-14
右侧文本限制指示器 . . . . .	24-15
代码折叠 - 展开和折叠代码构造 . . . . .	24-16
代码重构 - 自动将选定的代码转换为函数 . . . . .	24-18
<b>在文件中查找并替换文本</b> . . . . .	24-19
在当前文件中查找任何文本 . . . . .	24-19
在当前文件中查找和替换函数或变量 . . . . .	24-19
自动对文件中的所有函数或变量进行重命名 . . . . .	24-20
查找和替换任何文本 . . . . .	24-21
查找多个文件名或文件中的文本 . . . . .	24-21
用于查找文本的备选函数 . . . . .	24-22
在编辑器中执行增量搜索 . . . . .	24-22
转到文件中的位置 . . . . .	24-22
<b>在文件中添加提醒</b> . . . . .	24-26
使用 TODO/FIXME 报告 . . . . .	24-26
<b>MATLAB 代码分析器报告</b> . . . . .	24-28
运行代码分析器报告 . . . . .	24-28
根据代码分析器消息更改代码 . . . . .	24-29
访问代码分析器消息的其他方法 . . . . .	24-29
<b>MATLAB 代码兼容性报告</b> . . . . .	24-30
生成代码兼容性报告 . . . . .	24-30
编程用法 . . . . .	24-31

## 25

<b>确定程序依赖项</b>	25-2
简单显示程序文件依赖项	25-2
详细显示程序文件依赖项	25-2
文件夹中的依赖项	25-2
<b>保护您的源代码</b>	25-6
使用 P 代码构建掩盖内容的格式	25-6
构建独立的可执行文件	25-7
<b>创建运行函数的超链接</b>	25-8
运行单个函数	25-8
运行多个函数	25-9
提供命令选项	25-9
包括特殊字符	25-9
<b>创建和共享工具箱</b>	25-10
创建工具箱	25-10
共享工具箱	25-14

## 函数参数验证

## 26

<b>函数参数验证</b>	26-2
参数验证简介	26-2
何时使用参数验证	26-2
arguments 代码块语法	26-3
参数验证示例	26-4
参数的种类	26-6
必需和可选位置参数	26-6
重复参数	26-8
名称-值参数	26-10
基于类属性的名称-值参数	26-12
类方法中的参数验证	26-13
参数验证的顺序	26-14
避免类和大小转换	26-14
参数验证中的 nargin	26-16
变量和函数访问的限制	26-17
<b>参数验证函数</b>	26-19
定义验证函数	26-20
<b>解析函数输入的方法</b>	26-21
函数参数验证	26-21
validateattributes	26-21
inputParser	26-21
<b>MATLAB 代码中的透明</b>	26-22
编写透明代码	26-22

<b>MATLAB 应用程序中的异常处理</b>	27-2
概述	27-2
在命令行处遇到异常	27-2
在您的程序代码中遇到异常	27-3
产生新的异常	27-3
<b>捕获有关异常的信息</b>	27-4
概述	27-4
MException 类	27-4
MException 类的属性	27-5
MException 类的方法	27-10
<b>引发异常</b>	27-11
关于如何引发异常的建议	27-11
<b>对异常作出响应</b>	27-13
概述	27-13
try/catch 语句	27-13
有关如何处理异常的建议	27-14
<b>在函数结束后清理</b>	27-16
概述	27-16
退出时清理程序的示例	27-17
检索有关清理例程的信息	27-18
使用 onCleanup 与 try/catch	27-19
脚本中的 onCleanup	27-19
<b>引发警告和错误</b>	27-20
引发警告	27-20
引发错误	27-20
向您的警告和错误中添加运行时参数	27-21
向警告和错误中添加标识符	27-21
<b>隐蔽警告</b>	27-23
开启和关闭警告	27-23
<b>恢复警告</b>	27-25
禁用和恢复特定警告	27-25
禁用和恢复多个警告	27-26
<b>更改警告的显示方式</b>	27-27
启用 Verbose 警告	27-27
显示对特定警告的堆栈跟踪	27-27
<b>使用 try/catch 处理错误</b>	27-28

<b>使用计时器安排命令的执行</b>	28-2
概述	28-2
示例：显示消息	28-2
<b>计时器回调函数</b>	28-4
将命令与计时器对象事件关联	28-4
创建回调函数	28-5
指定回调函数属性的值	28-5
<b>处理计时器队列冲突</b>	28-7
Drop 模式（默认值）	28-7
错误模式	28-8
Queue 模式	28-9

<b>测量程序性能</b>	29-2
性能计时函数概述	29-2
计时函数	29-2
计算部分代码的时间	29-2
Cputime 函数与 tic/toc 和 timeit	29-2
有关测量性能的提示	29-3
<b>探查以提升性能</b>	29-4
什么是探查功能？	29-4
探查过程及准则	29-4
使用探查器	29-4
探查摘要报告	29-6
探查详细信息报告	29-7
<b>使用探查器可确定代码覆盖率</b>	29-9
<b>提升性能的方法</b>	29-11
环境	29-11
代码结构	29-11
针对性能的编程做法	29-11
有关特定 MATLAB 函数的提示	29-12
<b>预分配</b>	29-13
预分配非双精度类型的矩阵	29-13
<b>向量化</b>	29-15
向量化的应用	29-15
数组运算	29-16
逻辑数组运算	29-17
矩阵运算	29-18
排序、设置和计数运算	29-18

## 30

### 内存使用率

高效使用内存的策略 . . . . .	30-2
使用适当的数据存储 . . . . .	30-2
避免临时性的数据副本 . . . . .	30-3
回收使用的内存 . . . . .	30-4
解决“内存不足”错误 . . . . .	30-5
利用 tall 数组 . . . . .	30-5
利用多台计算机的内存 . . . . .	30-6
仅加载需要的数据 . . . . .	30-6
增加系统交换空间 . . . . .	30-7
设置 Linux 系统上的进程限制 . . . . .	30-7
在 Linux 系统上禁用 Java VM . . . . .	30-7
MATLAB 如何分配内存 . . . . .	30-9
为数组分配内存 . . . . .	30-9
数据结构体和内存 . . . . .	30-11
避免不必要的数据副本 . . . . .	30-15
将值传递给函数 . . . . .	30-15
为什么要使用传值语义 . . . . .	30-16
句柄对象 . . . . .	30-16

## 31

### 自定义帮助和文档

为类创建帮助 . . . . .	31-2
doc 命令显示的帮助文本 . . . . .	31-2
自定义帮助文本 . . . . .	31-3
检查哪些程序具有帮助 . . . . .	31-8
创建帮助摘要文件 - Contents.m . . . . .	31-10
什么是 Contents.m 文件? . . . . .	31-10
创建 Contents.m 文件 . . . . .	31-10
检查现有的 Contents.m 文件 . . . . .	31-11
自定义代码建议和自动填充 . . . . .	31-12
函数对象 . . . . .	31-12
签名对象 . . . . .	31-13
参数对象 . . . . .	31-13
创建函数签名文件 . . . . .	31-16
多个签名 . . . . .	31-18

<b>显示自定义文档</b> .....	31-20
概述 .....	31-20
创建 HTML 帮助文件 .....	31-20
创建 info.xml 文件 .....	31-21
创建 helptoc.xml 文件 .....	31-22
编译搜索数据库 .....	31-24
解决 info.xml 文件的验证错误 .....	31-24
<b>显示自定义示例</b> .....	31-26
如何显示示例 .....	31-26
demos.xml 文件的元素 .....	31-27

## 工程

# 32

<b>创建工程</b> .....	32-2
什么是工程? .....	32-2
创建工程 .....	32-2
打开工程 .....	32-2
设置工程 .....	32-2
将文件添加到工程 .....	32-5
创建工程的其他方法 .....	32-6
<b>自动执行启动和关闭任务</b> .....	32-8
指定工程路径 .....	32-8
设置启动文件夹 .....	32-8
指定启动和关闭文件 .....	32-8
<b>管理工程文件</b> .....	32-10
重命名、删除或移除文件时自动更新 .....	32-11
<b>查找工程文件</b> .....	32-12
对工程文件进行分组和排序 .....	32-12
搜索和筛选工程文件 .....	32-12
搜索工程文件中的内容 .....	32-12
<b>创建常见任务的快捷方式</b> .....	32-14
运行快捷方式 .....	32-14
创建快捷方式 .....	32-14
整理快捷方式 .....	32-14
<b>将标签添加到工程文件</b> .....	32-16
添加标签 .....	32-16
查看和编辑标签数据 .....	32-16
创建标签 .....	32-17
<b>创建自定义任务</b> .....	32-18
创建自定义任务函数 .....	32-18
运行自定义任务 .....	32-18
保存自定义任务报告 .....	32-18

<b>大型工程组件化</b>	32-20
添加或删除对工程的引用	32-20
查看、编辑或运行引用工程文件	32-20
提取文件夹以创建引用工程	32-20
使用检查点管理引用工程中的更改	32-21
<b>共享工程</b>	32-22
创建导出配置文件	32-24
<b>升级工程</b>	32-26
运行升级工程工具	32-26
检查升级工程报告	32-27
<b>分析工程依存关系</b>	32-29
运行依存关系分析	32-29
调查和解决问题	32-30
查找必需的工具箱	32-31
查找文件依存关系	32-32
查找需求文档	32-33
保存、打开和比较依存关系分析结果	32-33
<b>对工程使用源代码管理</b>	32-35
设置源代码管理	32-35
执行源代码管理操作	32-37
处理工程中的派生文件	32-42
查找具有未保存更改的工程文件	32-42
关闭工程时管理打开的文件	32-43
<b>以编程方式创建和编辑工程</b>	32-44
<b>了解示例工程</b>	32-51

## 源代码管理接口

# 33

<b>关于 MathWorks 源代码管理集成</b>	33-2
典型和分布式源代码管理	33-2
<b>选择或禁用源代码管理系统</b>	33-4
选择源代码管理系统	33-4
禁用源代码管理	33-4
<b>创建新的存储库</b>	33-5
在您的本地系统上创建 Git 存储库	33-5
创建 SVN 存储库	33-5
<b>在源代码管理中审核更改</b>	33-7
<b>标记文件以添加到源代码管理</b>	33-8
<b>解决源代码管理冲突</b>	33-9
检查和解决冲突	33-9

解决冲突 . . . . .	33-9
合并文本文件 . . . . .	33-9
提取冲突标记 . . . . .	33-10
<b>将已修改文件提交到源代码管理 . . . . .</b>	<b>33-12</b>
<b>在源代码管理中还原更改内容 . . . . .</b>	<b>33-13</b>
还原本地更改内容 . . . . .	33-13
将文件还原到指定修订版本 . . . . .	33-13
<b>设置 SVN 源代码管理 . . . . .</b>	<b>33-14</b>
SVN 源代码管理选项 . . . . .	33-14
使用 SVN 注册二进制文件 . . . . .	33-14
标准存储库结构 . . . . .	33-16
文件的标签版本 . . . . .	33-17
强制实施编辑前锁定文件 . . . . .	33-17
共享 Subversion 存储库 . . . . .	33-17
<b>从 SVN 存储库签出 . . . . .</b>	<b>33-19</b>
检索存储库的标记版本 . . . . .	33-19
<b>更新 SVN 文件状态和修订版本 . . . . .</b>	<b>33-21</b>
刷新文件状态 . . . . .	33-21
更新文件的修订版本 . . . . .	33-21
<b>获取 SVN 文件锁 . . . . .</b>	<b>33-22</b>
管理 SVN 存储库锁 . . . . .	33-22
<b>设置 Git 源代码管理 . . . . .</b>	<b>33-23</b>
关于 Git 源代码管理 . . . . .	33-23
安装命令行 Git 客户端 . . . . .	33-24
在 Git 中注册二进制文件 . . . . .	33-24
添加 Git 子模块 . . . . .	33-26
<b>从 Git 存储库克隆 . . . . .</b>	<b>33-27</b>
故障排除 . . . . .	33-27
<b>更新 Git 文件状态和修订版本 . . . . .</b>	<b>33-28</b>
刷新文件状态 . . . . .	33-28
更新文件的修订版本 . . . . .	33-28
<b>Git 的分支和合并 . . . . .</b>	<b>33-29</b>
创建分支 . . . . .	33-29
切换分支 . . . . .	33-30
比较分支 . . . . .	33-31
合并分支 . . . . .	33-31
还原到 HEAD . . . . .	33-31
删除分支 . . . . .	33-32
<b>使用 Git 取回、推送和提取文件 . . . . .</b>	<b>33-33</b>
取回和推送 . . . . .	33-33
提取和合并 . . . . .	33-34
使用 Git 暂存文件 . . . . .	33-34
<b>移动、重命名或删除源代码管理下的文件 . . . . .</b>	<b>33-36</b>

自定义外部源代码管理以使用 MATLAB 执行差异分析和合并 . . . . .	33-37
<b>MSSCCI 源代码管理接口 . . . . .</b>	<b>33-39</b>
<b>设置 MSSCCI 源代码管理 . . . . .</b>	<b>33-40</b>
在源代码管理系统中创建工程 . . . . .	33-40
通过 MATLAB 软件指定源代码管理系统 . . . . .	33-41
向 MATLAB 软件中注册源代码管理工程 . . . . .	33-42
向源代码管理中添加文件 . . . . .	33-44
<b>从 MSSCCI 源代码管理中签入和签出文件 . . . . .</b>	<b>33-45</b>
将文件签入到源代码管理系统 . . . . .	33-45
签出源代码管理系统中的文件 . . . . .	33-45
撤消签出 . . . . .	33-46
<b>其他 MSSCCI 源代码管理操作 . . . . .</b>	<b>33-47</b>
获取最新版本的文件以便查看或编译 . . . . .	33-47
删除源代码管理系统中的文件 . . . . .	33-47
显示文件历史记录 . . . . .	33-48
将工作副本与源代码管理中的最新版本进行比较 . . . . .	33-49
查看文件的源代码管理属性 . . . . .	33-50
启动源代码管理系统 . . . . .	33-51
<b>从编辑器访问 MSSCCI 源代码管理 . . . . .</b>	<b>33-52</b>
<b>MSSCCI 源代码管理问题故障排除 . . . . .</b>	<b>33-53</b>
源代码管理错误：提供程序不存在或安装不当 . . . . .	33-53
针对 @ 字符的限制 . . . . .	33-53
“添加到源代码管理”是唯一可用操作 . . . . .	33-54
针对源代码管理问题的更多解决方案 . . . . .	33-54

## 单元测试

# 34

<b>使用实时脚本编写测试 . . . . .</b>	<b>34-3</b>
<b>编写基于脚本的单元测试 . . . . .</b>	<b>34-5</b>
<b>使用局部函数编写基于脚本的测试 . . . . .</b>	<b>34-10</b>
<b>扩展基于脚本的测试 . . . . .</b>	<b>34-13</b>
测试套件创建 . . . . .	34-13
测试选择 . . . . .	34-13
以编程方式访问测试诊断 . . . . .	34-14
测试运行程序自定义 . . . . .	34-14
<b>在编辑器中运行测试 . . . . .</b>	<b>34-16</b>
<b>编写基于函数的单元测试 . . . . .</b>	<b>34-19</b>
创建测试函数 . . . . .	34-19
运行测试 . . . . .	34-21
分析结果 . . . . .	34-21

<b>使用函数编写简单测试用例</b>	<b>34-22</b>
<b>使用设置和拆解函数编写测试</b>	<b>34-26</b>
<b>扩展基于函数的测试</b>	<b>34-31</b>
设置和拆解代码的脚手架	34-31
测试日志记录和详细程度	34-32
测试套件创建	34-32
测试选择	34-32
测试运行	34-33
以编程方式访问测试诊断	34-33
测试运行程序自定义	34-33
<b>在 MATLAB 中编写基于类的单元测试</b>	<b>34-35</b>
测试类定义	34-35
单元测试	34-35
高级测试类的其他功能	34-36
<b>使用类来编写简单测试用例</b>	<b>34-37</b>
<b>使用类来编写设置代码和拆解代码</b>	<b>34-41</b>
测试脚手架	34-41
包含方法级设置代码的测试用例	34-41
包含类级设置代码的测试用例	34-42
<b>验证、断言及其他验证一览表</b>	<b>34-44</b>
<b>标记单元测试</b>	<b>34-46</b>
标记测试	34-46
选择并运行测试	34-47
<b>使用共享脚手架编写测试</b>	<b>34-50</b>
<b>创建基本自定义脚手架</b>	<b>34-53</b>
<b>创建高级自定义脚手架</b>	<b>34-55</b>
<b>创建基本参数化测试</b>	<b>34-60</b>
<b>创建高级参数化测试</b>	<b>34-64</b>
<b>在参数化测试中使用外部参数</b>	<b>34-70</b>
<b>创建简单测试套件</b>	<b>34-74</b>
<b>为各个工作流运行测试</b>	<b>34-76</b>
设置示例测试	34-76
在类或函数中运行所有测试	34-76
在类或函数中运行单个测试	34-76
按名称运行测试套件	34-77
从测试数组运行测试套件	34-77
使用自定义测试运行程序运行测试	34-78
<b>以编程方式访问测试诊断</b>	<b>34-79</b>

<b>向测试运行程序添加插件</b>	<b>34-80</b>
<b>编写插件以扩展 TestRunner</b>	<b>34-82</b>
自定义插件概述	34-82
扩展测试会话级别插件方法	34-82
扩展测试套件级别插件方法	34-83
扩展测试类级别插件方法	34-83
扩展测试级别插件方法	34-83
<b>创建自定义插件</b>	<b>34-85</b>
<b>使用自定义插件并行运行测试</b>	<b>34-90</b>
<b>编写用于保存诊断详细信息的插件</b>	<b>34-98</b>
<b>用于生成自定义测试输出格式的插件</b>	<b>34-102</b>
<b>分析测试用例结果</b>	<b>34-105</b>
<b>分析失败的测试结果</b>	<b>34-108</b>
<b>重新运行失败的测试</b>	<b>34-110</b>
<b>动态筛选的测试</b>	<b>34-113</b>
测试方法	34-113
方法设置和拆解代码	34-115
类设置和拆解代码	34-117
<b>创建自定义约束</b>	<b>34-119</b>
<b>创建自定义布尔约束</b>	<b>34-122</b>
<b>创建自定义容差</b>	<b>34-125</b>
<b>App 测试框架概述</b>	<b>34-129</b>
App 测试	34-129
UI 组件的手势支持	34-129
为 App 编写测试	34-130
<b>为 App 编写测试</b>	<b>34-133</b>
<b>编写使用 App 测试和模拟框架的测试</b>	<b>34-137</b>
创建 App	34-137
在手动干预下测试 App	34-138
创建全自动测试	34-139
<b>性能测试框架概述</b>	<b>34-142</b>
确定已测量代码的范围	34-142
计时试验的类型	34-142
编写具有测量范围的性能测试	34-143
运行性能测试	34-143
理解无效测试结果	34-144
<b>使用脚本或函数测试性能</b>	<b>34-145</b>

使用类测试性能 . . . . .	34-149
测量快速执行的测试代码 . . . . .	34-155
创建 Mock 对象 . . . . .	34-158
指定 Mock 对象行为 . . . . .	34-164
定义 Mock 方法的行为 . . . . .	34-164
定义 Mock 属性的行为 . . . . .	34-165
定义重复行为和后续行为 . . . . .	34-166
行为汇总 . . . . .	34-167
验证 Mock 对象交互 . . . . .	34-169
验证 Mock 方法交互 . . . . .	34-169
验证 Mock 属性交互 . . . . .	34-170
使用 Mock 对象约束 . . . . .	34-171
验证概述 . . . . .	34-172

## System object 用法和编写

# 35

何谓 System object? . . . . .	35-2
运行 System object . . . . .	35-3
System object 函数 . . . . .	35-3
System object 与 MATLAB 函数 . . . . .	35-5
System object 与 MATLAB 函数 . . . . .	35-5
使用仅包含 MATLAB 函数的代码处理音频数据 . . . . .	35-5
使用 System object 处理音频数据 . . . . .	35-6
使用 System object 在 MATLAB 中进行系统设计 . . . . .	35-7
在 MATLAB 中进行系统设计和仿真 . . . . .	35-7
创建单个组件 . . . . .	35-7
配置组件 . . . . .	35-8
同时创建和配置组件 . . . . .	35-8
将组件组合到系统中 . . . . .	35-9
运行系统 . . . . .	35-9
重新配置对象 . . . . .	35-10
定义基本 System object . . . . .	35-11
创建 System object 类 . . . . .	35-11
定义算法 . . . . .	35-11
更改输入数目 . . . . .	35-13
验证属性和输入值 . . . . .	35-16
验证单个属性 . . . . .	35-16
验证相互依赖的属性 . . . . .	35-16
验证输入 . . . . .	35-16
完整的类示例 . . . . .	35-16
初始化属性并设置一次性计算 . . . . .	35-18

<b>构造时设置属性值</b>	35-20
<b>重置算法并释放资源</b>	35-22
重置算法状态	35-22
释放 System object 资源	35-22
<b>定义属性特性</b>	35-24
将属性指定为不可调属性	35-24
将属性指定为离散状态属性	35-24
具有各种属性特性的类示例	35-24
<b>隐藏非活动属性</b>	35-26
指定非活动属性	35-26
包含非活动属性方法的完整类定义文件	35-26
<b>将属性值限制为有限列表</b>	35-28
使用 mustBeMember 进行属性验证	35-28
枚举属性	35-28
创建 Whiteboard System object	35-28
<b>处理调整后的属性</b>	35-32
<b>定义复合的 System object</b>	35-34
<b>定义有限源对象</b>	35-36
使用 FiniteSource 类并指定源的结尾	35-36
包含有限源的完整类定义文件	35-36
<b>保存和加载 System object</b>	35-38
保存 System object 和子对象	35-38
加载 System object 和子对象	35-38
包含保存和加载的完整类定义文件	35-38
<b>定义 System object 信息</b>	35-41
<b>处理输入设定更改</b>	35-43
响应输入设定更改	35-43
限制输入设定更改	35-43
<b>调用序列摘要</b>	35-45
setup 调用序列	35-45
运行对象或 step 调用序列	35-45
reset 方法调用序列	35-46
release 方法调用序列	35-47
<b>详细的调用序列</b>	35-48
setup 调用序列	35-48
运行对象或 step 调用序列	35-48
reset 调用序列	35-49
release 调用序列	35-49
<b>定义 System object 的技巧</b>	35-50
常规	35-50
输入和输出	35-50
在方法定义中使用 ~ 作为输入参数	35-50

属性 .....	35-50
文本比较 .....	35-51
Simulink .....	35-51
代码生成 .....	35-51
<b>使用 MATLAB 编辑器插入 System object 代码 .....</b>	<b>35-52</b>
通过代码插入定义 System object .....	35-52
创建温度枚举属性 .....	35-54
为冻结点创建自定义属性 .....	35-55
添加方法来验证输入 .....	35-56
<b>分析 System object 代码 .....</b>	<b>35-57</b>
查看并导航 System object 代码 .....	35-57
示例：使用分析器转至 StepImpl 方法 .....	35-57
<b>在 System object 中使用全局变量 .....</b>	<b>35-59</b>
MATLAB 中的 System object 全局变量 .....	35-59
Simulink 中的 System object 全局变量 .....	35-59



# 语言



# 语法基础知识

---

- “在多行上延续长语句” (第 1-2 页)
- “调用函数” (第 1-3 页)
- “忽略函数输出” (第 1-4 页)
- “变量名称” (第 1-5 页)
- “大小写和空格敏感性” (第 1-6 页)
- “命令与函数语法” (第 1-7 页)
- “调用函数时的常见错误” (第 1-10 页)

## 在多行上延续长语句

此示例说明了如何使用省略号 (...) 将语句延续到下一行。

```
s = 1 - 1/2 + 1/3 - 1/4 + 1/5 ...
    - 1/6 + 1/7 - 1/8 + 1/9;
```

通过将多个较短的向量串联在一起生成一个长的字符向量：

```
mytext = ['Accelerating the pace of' ...
          'engineering and science'];
```

字符向量的起始引号和结束引号必须出现在同一行中。例如，以下代码会返回错误，因为每一行仅包含一个引号：

```
mytext = 'Accelerating the pace of ...
          engineering and science'
```

所引用的文本外部的省略号等效于空格。例如，

```
x = [1.23...
      4.56];
```

与下列语句相同

```
x = [1.23 4.56];
```

# 调用函数

以下示例演示了如何调用 MATLAB 函数。要运行这些示例，您必须先创建数值数组 A 和 B，例如：

```
A = [1 3 5];  
B = [10 6 4];
```

将函数输入括入括号：

```
max(A)
```

用逗号分隔多个输入：

```
max(A,B)
```

将函数赋值给变量，以此方式来存储函数的输出：

```
maxA = max(A)
```

将多个输出括入方括号：

```
[maxA, location] = max(A)
```

对于不需要任何输入，也不返回任何输出的函数，只需键入此类函数的函数名，即可调用它们：

```
clc
```

将文本输入放在单引号中：

```
disp('hello world')
```

## 另请参阅

## 相关示例

- “忽略函数输出”（第 1-4 页）

## 忽略函数输出

此示例说明如何使用波浪号 (~) 运算符请求函数的特定输出。

请求 `fileparts` 函数的所有三个可能的输出。

```
helpFile = which('help');  
[helpPath,name,ext] = fileparts(helpFile);
```

当前工作区现在包含 `fileparts` 的三个变量：`helpPath`、`name` 和 `ext`。在本例中，这些变量很小。但是，某些函数返回的结果会使用很多的内存。如果您不需要这些变量，它们就会浪费系统上的空间。

如果不使用波浪号运算符，您只能请求函数的前  $N$  个输出（其中  $N$  小于或等于可能的输出数），并忽略剩余的任何输出。例如，仅请求第一个输出，而忽略第二个和第三个输出。

```
helpPath = fileparts(helpFile);
```

如果要请求多个输出，请将变量名称括入方括号 []。以下代码忽略输出参数 `ext`。

```
[helpPath,name] = fileparts(helpFile);
```

要忽略参数列表中任何位置的函数输出，请使用波浪号运算符。例如，使用波浪号忽略第一个输出。

```
[~,name,ext] = fileparts(helpFile);
```

您可以使用波浪号运算符忽略任意数量的函数输出。用逗号分隔连续的波浪号。例如：

```
[~,~,ext] = fileparts(helpFile);
```

## 另请参阅

### 详细信息

- “忽略函数输入”（第 21-10 页）

# 变量名称

## 本节内容

- “有效名称”（第 1-5 页）
- “与函数名称冲突”（第 1-5 页）

## 有效名称

有效的变量名称以字母开头，后跟字母、数字或下划线。MATLAB 区分大小写，因此 A 和 a 不是同一变量。变量名称的最大长度为 `namelengthmax` 命令返回的值。

您不能定义与 MATLAB 关键字同名的变量（例如 `if` 或 `end`）。要获取关键字的完整列表，请运行 `iskeyword` 命令。

有效名称示例：

```
x6
lastValue
n_factorial
```

无效名称示例：

```
6x
end
n!
```

## 与函数名称冲突

定义变量时应避免创建与函数同名的变量，例如 `i`、`j`、`mode`、`char`、`size` 和 `path`。一般情况下，变量名称优先于函数名称。如果您创建的变量使用了某个函数的名称，则有时会获得意外的结果。

使用 `exist` 或 `which` 函数检查拟用名称是否已被使用。如果不存在与拟用名称同名的变量、函数或其他工件，`exist` 将返回 0。例如：

```
exist checkname
ans =
0
```

如果您无意中创建了名称存在冲突的变量，请使用 `clear` 函数将该变量从内存中删除。

当您定义调用 `load` 或 `eval`（或类似函数），以将变量添加到工作区时，可能会出现另一个潜在的名称冲突源。在某些情况下，`load` 或 `eval` 会添加与函数同名的变量。除非在调用 `load` 或 `eval` 之前这些变量已经存在于函数工作区中，否则 MATLAB 解析器会将变量名称解释为函数名称。有关详细信息，请参阅：

- “在函数中加载变量时出现意外结果”
- “`eval` 函数的替代方法”（第 2-80 页）

## 另请参阅

`clear` | `exist` | `iskeyword` | `isvarname` | `namelengthmax` | `which`

## 大小写和空格敏感性

MATLAB 代码对大小写敏感，对空白空格不敏感（定义数组时除外）。

### 大写字母和小写字母

在 MATLAB 代码中使用变量、文件和函数时，请按照其定义的大小写，应用精确匹配。例如，如果您有一个变量 `a`，则不能将该变量称为 `A`。最佳做法是仅在命名函数时使用小写字母。这在您同时使用 Microsoft® Windows® 和 UNIX®<sup>1</sup> 平台时特别有用，因为它们的文件系统在大小写方面具有不同的表现。

当您使用 `help` 函数时，`help` 会以全大写字母形式显示某些函数名称（例如 `PLOT`），目的仅仅是为了区分函数名称与其余的文本。某些用于连接 Oracle® Java® 软件的函数则在混用大小写，命令行帮助和文档也准确地反映了这一点。

### 空格

运算符（例如 `-`、`:` 和 `( )`）周围的空白空格是可选的，它们可以改善代码的可读性。例如，MATLAB 在解释下面两个语句时将它们视为相同的语句。

```
y = sin (3 * pi) / 2
y=sin(3*pi)/2
```

抛开这种情况，空白空格还可充当水平串联的分隔符。定义行向量时，您可以交替使用空格和逗号来分隔元素：

```
A = [1, 0 2, 3 3]
```

```
A =
```

```
1 0 2 3 3
```

由于这种灵活性，请一定仔细检查代码以确保 MATLAB 存储正确的值。例如，语句 `[1 sin (pi) 3]` 生成的结果与 `[1 sin(pi) 3]` 生成的结果截然不同。

```
[1 sin (pi) 3]
```

```
Error using sin
Not enough input arguments.
```

```
[1 sin(pi) 3]
```

```
ans =
```

```
1.0000 0.0000 3.0000
```

---

1. UNIX is a registered trademark of The Open Group in the United States and other countries.

# 命令与函数语法

## 命令与函数语法

在 MATLAB 中，以下语句是等效的：

```
load durer.mat      % Command syntax
load('durer.mat')  % Function syntax
```

这种等效有时称为命令-函数二元性。

所有函数都支持以下标准函数语法：

```
[output1, ..., outputM] = functionName(input1, ..., inputN)
```

如果您不需要函数的任何输出，并且所有输入都是字符向量（即置于单引号内的文本），则可以使用以下更为简单的命令语法：

```
functionName input1 ... inputN
```

通过命令语法，您可以用空格而不是逗号来分隔输入，并且不需要将输入参数括入括号。命令语法始终将输入作为字符向量传递。要使用字符串作为输入，请使用函数语法。如果字符向量包含空格，请使用函数语法。例如：

当函数输入为变量时，您必须使用函数语法来将值传递给函数。命令语法始终将输入作为字符向量传递，不能传递变量值。例如，创建一个变量并通过函数语法调用 `disp` 函数，以传递该变量的值：

```
A = 123;
disp(A)
```

该代码返回预期的结果，

```
123
```

您不能使用命令语法来传递 A 的值，因为此调用

```
disp A
```

等效于

```
disp('A')
```

并返回

```
A
```

## 避免常见的语法错误

假定您的工作区包含以下变量：

```
filename = 'accounts.txt';
A = int8(1:8);
B = A;
```

下表说明了常见的命令语法误用情况。

以下命令...	等同于...	传递值的正确语法
<b>open filename</b>	<b>open('filename')</b>	<b>open(filename)</b>
<b>isequal A B</b>	<b>isequal('A','B')</b>	<b>isequal(A,B)</b>
<b>strcmp class(A) int8</b>	<b>strcmp('class(A)', 'int8')</b>	<b>strcmp(class(A), 'int8')</b>
<b>cd matlabroot</b>	<b>cd('matlabroot')</b>	<b>cd(matlabroot)</b>
<b>isnumeric 500</b>	<b>isnumeric('500')</b>	<b>isnumeric(500)</b>
<b>round 3.499</b>	<b>round('3.499') , 等效于 round([51 46 52 57 57])</b>	<b>round(3.499)</b>
<b>disp hello world</b>	<b>disp('hello','world')</b>	<b>disp('hello world')</b>
<b>disp "string"</b>	<b>disp("string")</b>	<b>disp("string")</b>

### 传递变量名称

某些函数期望获取变量名称的字符向量，例如 **save**、**load**、**clear** 和 **whos**。例如，

```
whos -file durer.mat X
```

请求示例文件 **durer.mat** 中有关变量 **X** 的信息。此命令等同于

```
whos('-file','durer.mat','X')
```

## MATLAB 如何识别命令语法

以下面可能具有多义性的语句为例：

```
ls ./d
```

该语句可能是调用 **ls** 函数并将文件夹 **./d** 作为其参数，也可能是使用变量 **d** 作为除数请求对数组 **ls** 执行按元素除法。

如果您在命令行中发出此类命令，MATLAB 可能会访问当前工作区和路径，以确定 **ls** 和 **d** 是函数还是变量。但是，某些组件（例如代码分析器和编辑器/调试器）在运行时不需要引用路径或工作区。在这些情况下，MATLAB 会使用语法规则来确定表达式是否为使用命令语法的函数调用。

通常，当 MATLAB 识别出一个标识符（可能用来命名一个函数或变量）时，它会分析该标识符后面的字符，以确定表达式的类型，如下所示：

- 等号 (=) 表示赋值。例如：

```
ls =d
```

- 标识符之后的左括号表示函数调用。例如：

```
ls('./d')
```

- 标识符后面的空格（但不在潜在的运算符之后）表示使用命令语法的函数调用。例如：

```
ls ./d
```

- 潜在运算符的两侧带有空格，或者该运算符任意一侧没有空格，均表示变量运算。例如，以下语句是等效的：

```
ls ./ d
```

```
ls./d
```

因此，可能具有多义性的语句 `ls ./d` 是使用命令语法调用 `ls` 函数。

最佳做法是避免定义与公共函数冲突的变量名称，以防止出现任何多义性。

## 调用函数时的常见错误

### 本节内容

“函数名称和变量名称冲突”（第 1-10 页）

“未定义的函数或变量”（第 1-10 页）

### 函数名称和变量名称冲突

如果变量和函数具有相同的名称，且没有提供足够的信息以供 MATLAB 用来解决冲突，MATLAB 将会引发错误。您可能会看到与以下类似的错误消息：

```
Error: <functionName> was previously used as a variable,
conflicting with its use here as the name of a function
or command.
```

其中，`<functionName>` 为函数的名称。

有时使用 `eval` 和 `load` 函数也可能在变量名称和函数名称之间产生类似冲突。有关详细信息，请参阅：

- “与函数名称冲突”（第 1-5 页）
- “在函数中加载变量时出现意外结果”
- “`eval` 函数的替代方法”（第 2-80 页）

### 未定义的函数或变量

当您在 MATLAB 中使用函数或变量时，可能会遇到以下错误消息或类似的消息：

未定义函数或变量 '`x`'。

这些错误通常指示 MATLAB 在当前目录或搜索路径中找不到特定的变量或 MATLAB 程序文件。其根本原因可能是下面这些原因中的一个：

- 函数的名称拼写错误。
- 函数名称和包含函数的文件的名称不相同。
- 未安装函数所属的工具箱。
- 函数的搜索路径已更改。
- 函数是您没有许可证的工具箱的一部分。

遇到此类错误时，请按照本节所述步骤来解决问题。

### 验证函数名称的拼写

最常见的错误之一是函数名称拼写错误。尤其对于很长的函数名称或包含相似字符（例如，字母 `I` 和数字 `一`）的名称，则很容易犯不易发现的错误。

如果您拼错 MATLAB 函数的名称，命令行窗口中将显示一个建议的函数名称。例如，由于以下命令在函数名称中包含大写字母，因此该命令将会失败。

`accumArray`

Undefined function or variable '`accumArray`'.

Did you mean:  
 >> accumarray

按 **Enter** 执行建议的命令，或按 **Esc** 取消该命令。

### 确保函数名称与文件名匹配

当您编写代码行以定义函数时，需要确定函数的名称。此名称应始终与用来保存该函数的文件的名称相匹配。例如，如果您创建一个名为 **curveplot** 的函数，

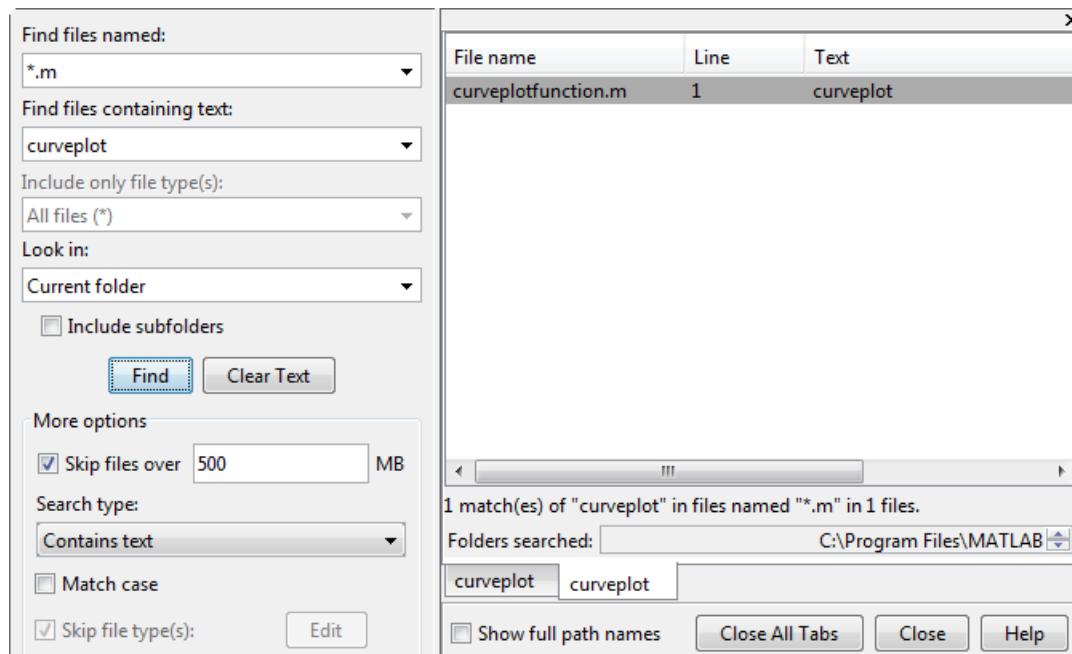
```
function curveplot(xVal, yVal)
- program code -
```

则应将包含该函数的文件命名为 **curveplot.m**。如果您为该函数创建一个 **pcode** 文件，则将此文件命名为 **curveplot.p**。如果函数名称和文件名冲突，文件名将覆盖为函数指定的名称。在此示例中，如果您将 **curveplot** 函数保存到名为 **curveplotfunction.m** 的文件中，则尝试使用函数名称调用该函数将会失败：

```
curveplot
未定义函数或变量 'curveplot'.
```

如果您遇到此问题，请更改函数名称或文件名，使它们保持相同。如果您找不到使用该函数的文件，请使用 MATLAB **查找文件** 实用工具，如下所示：

- 1 在主页选项卡上的文件部分中，点击  **查找文件**。
- 2 在**查找具有以下名称的文件:** 下输入 **\*.m**
- 3 在**查找包含以下文本的文件:** 下输入函数名称。
- 4 点击**查找**按钮



### 确保安装了工具箱

如果您无法使用 MATLAB 或其工具箱中的内置函数，请确保安装了该函数。

如果您不知道哪个工具箱支持所需的函数，请在 <https://www.mathworks.com/help> 中搜索函数文档。工具箱名称显示在函数参考页的顶部。

当您知道函数所属的工具箱后，请使用 `ver` 函数查看在运行 MATLAB 的系统上安装了哪些工具箱。`ver` 函数会显示当前已安装的所有 MathWorks® 产品的列表。如果您可以在 `ver` 显示的输出中找到所需的工具箱，则表明该工具箱已安装。有关安装 MathWorks 产品的帮助，请参阅《安装指南》文档。

如果您未看到该工具箱，但又认为该工具箱已安装，则 MATLAB 路径设置可能不正确。请继续进入下一部分。

## 验证访问函数所用的路径

此步骤将路径重置为默认值。

由于 MATLAB 将工具箱信息存储在缓存文件中，因此您需要先更新此缓存，然后再重置路径。要完成此操作，请执行以下步骤：

1 在主页选项卡上的环境部分中，点击  预设。

此时将显示“预设项”对话框。

2 在 MATLAB > 常规节点下，点击 **更新工具箱路径缓存** 按钮。

3 在主页选项卡上的环境部分中，点击  设置路径....。

此时将打开“设置路径”对话框。

4 点击默认。

此时将打开一个很小的对话框，警告您如果继续，将会丢失当前路径设置。如果您决定继续，请点击是。

(如果您已在 MATLAB 中添加了任何自定义路径，则需要在以后还原这些路径)

重新运行 `ver` 以查看工具箱是否已安装。如果未安装，则您可能需要重新安装此工具箱以使用该函数。有关安装工具箱的详细信息，请参阅 How do I install additional toolboxes into an existing installation of MATLAB。

当 `ver` 显示您的工具箱后，请运行以下命令以查看能否找到相应函数：

`which -all <functionname>`

将 `<functionname>` 替换为函数的名称。您应该会看到函数文件的路径。如果您收到消息指示函数名称未找到，则可能需要重新安装该工具箱以激活函数。

# 程序组件

---

- “MATLAB 运算符和特殊字符” (第 2-2 页)
- “数组与矩阵运算” (第 2-19 页)
- “基本运算的兼容数组大小” (第 2-23 页)
- “使用关系运算符进行数组比较” (第 2-27 页)
- “运算符优先级” (第 2-30 页)
- “使用容差为类似数据点求平均值” (第 2-32 页)
- “使用容差为散点数据分组” (第 2-34 页)
- “按位运算” (第 2-36 页)
- “执行循环冗余校验” (第 2-41 页)
- “条件语句” (第 2-44 页)
- “循环控制语句” (第 2-46 页)
- “正则表达式” (第 2-48 页)
- “正则表达式中的前向断言” (第 2-58 页)
- “正则表达式中的标文” (第 2-61 页)
- “动态正则表达式” (第 2-67 页)
- “逗号分隔的列表” (第 2-73 页)
- “eval 函数的替代方法” (第 2-80 页)

## MATLAB 运算符和特殊字符

此页面包含所有 MATLAB 运算符、符号和特殊字符的完整列表。

### 算术运算符

符号	角色	更多信息
+	加法	<code>plus</code>
+	一元加法	<code>uplus</code>
-	减法	<code>minus</code>
-	一元减法	<code>uminus</code>
.*	按元素乘法	<code>times</code>
*	矩阵乘法	<code>mtimes</code>
./	按元素右除	<code>rdivide</code>
/	矩阵右除	<code>mrdivide</code>
.\	按元素左除	<code>ldivide</code>
\	矩阵左除 (也称为反斜杠)	<code>mldivide</code>
.^	按元素求幂	<code>power</code>
^	矩阵幂	<code>mpower</code>
!	转置	<code>transpose</code>
'	复共轭转置	<code>ctranspose</code>

### 关系运算符

符号	角色	更多信息
==	等于	<code>eq</code>
~=	不等于	<code>ne</code>
>	大于	<code>gt</code>
>=	大于或等于	<code>ge</code>
<	小于	<code>lt</code>
<=	小于或等于	<code>le</code>

### 逻辑运算符

符号	角色	更多信息
&	逻辑 AND	<code>and</code>
	逻辑 OR	<code>or</code>
&&	逻辑 AND (具有短路功能)	<code>Logical Operators: Short-Circuit &amp;&amp;   </code>

符号	角色	更多信息
	逻辑 OR (具有短路功能)	
~	逻辑非	not

## 特殊字符

@	<p><b>名称:</b> at 符号</p> <p><b>用法:</b></p> <ul style="list-style-type: none"> <li>• 函数句柄构造和引用</li> <li>• 调用超类方法</li> </ul> <p><b>说明:</b> @ 符号可为跟在 @ 符号后面的命名函数或跟在 @ 符号后面的匿名函数构造函数句柄。您也可以使用 @ 从子类中调用超类方法。</p> <p><b>示例</b></p> <p>创建命名函数的函数句柄:</p> <pre>fhandle = @myfun</pre> <p>创建匿名函数的函数句柄:</p> <pre>fhandle = @(x,y) x.^2 + y.^2;</pre> <p>从子类中调用 MySuper 的 disp 方法:</p> <pre>disp@MySuper(obj)</pre> <p>使用正在构造的对象从子类中调用超类构造函数:</p> <pre>obj = obj@MySuper(arg1,arg2,...)</pre> <p><b>更多信息:</b></p> <ul style="list-style-type: none"> <li>• “<a href="#">创建函数句柄</a>” (第 13-2 页)</li> <li>• “<a href="#">对子类对象调用超类方法</a>”</li> </ul>
---	---

.	<p><b>名称:</b> 句点或点</p> <p><b>用法:</b></p> <ul style="list-style-type: none"><li>• 小数点</li><li>• 按元素运算</li><li>• 结构体字段访问</li><li>• 对象属性或方法设定符</li></ul> <p><b>说明:</b> 句点字符可分隔一个数（例如 3.1415）的整数部分和小数部分。包含句点的 MATLAB 运算符会始终按元素执行运算。通过句点字符，还可以访问结构体中的字段以及对象的属性和方法。</p> <p><b>示例</b></p> <p>小数点: <code>102.5543</code></p> <p>按元素运算: <code>A.*B</code> <code>A.^2</code></p> <p>结构体字段访问: <code>myStruct.f1</code></p> <p>对象属性设定符: <code>myObj.PropertyName</code></p> <p><b>更多信息</b></p> <ul style="list-style-type: none"><li>• “数组与矩阵运算” (第 2-19 页)</li><li>• “结构体”</li><li>• “访问属性值”</li></ul>
---	---

...	<p><b>名称:</b> 三个点或省略号</p> <p><b>用法:</b> 续行</p> <p><b>说明:</b> 行末尾的三个或更多个句点表示当前命令延续到下一行。如果行末尾之前存在三个或更多个句点，则 MATLAB 会忽略该行的其余部分而直接延续到下一行。这实际上相当于将当前行上三个句点之后的任何内容作为注释。</p> <hr/> <p><b>注意</b> MATLAB 将省略号解释为空格字符。因此，多行命令与将省略号替换为空格字符的单行命令等效。</p> <p><b>示例</b></p> <p>将函数调用延续到下一行：</p> <pre>sprintf(['The current value '... 'of %s is %d'],vname,value)</pre> <p>在多行上分解字符串并将这些行串联在一起：</p> <pre>S = [If three or more periods occur before the '... 'end of a line, then the rest of that line is ' ... 'ignored and MATLAB continues to the next line']</pre> <p>要将多行命令中的一行注释掉，请在该行的开头使用 ... 以确保命令保持完整。如果您使用 % 将某一行注释掉，则会产生错误：</p> <pre>y = 1 +...     2 +... % 3 +...     4;</pre> <p>但是，以下代码可正常运行，因为第三行不会在命令中产生间断。</p> <pre>y = 1 +...     2 +... ... 3 +...     4;</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"> <li>“在多行上延续长语句”（第 1-2 页）</li> </ul>
-----	--

,	<p><b>名称:</b> 逗号</p> <p><b>用法:</b> 分隔符</p> <p><b>说明:</b> 使用逗号可分隔数组中的行元素、数组下标、函数输入和输出参数以及同一行中输入的命令。</p> <p><b>示例</b></p> <p>分隔行元素以创建数组:</p> <pre>A = [12,13; 14,15]</pre> <p>分隔下标:</p> <pre>A(1,2)</pre> <p>分隔函数调用中的输入和输出参数:</p> <pre>[Y,I] = max(A,[],2)</pre> <p>分隔同一行中的多个命令 (显示输出) :</p> <pre>figure, plot(sin(-pi:0.1:pi)), grid on</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"><li>• <code>horzcat</code></li></ul>
---	--

:	<p><b>名称:</b> 冒号</p> <p><b>用法:</b></p> <ul style="list-style-type: none"><li>• 创建向量</li><li>• 索引</li><li>• For 循环迭代</li></ul> <p><b>说明:</b> 使用冒号运算符可创建规律间隔的向量、对数组进行索引以及定义 for 循环的边界。</p> <p><b>示例</b></p> <p>创建向量:</p> <pre>x = 1:10</pre> <p>创建以 3 为单位递增的向量:</p> <pre>x = 1:3:19</pre> <p>将矩阵重构为列向量:</p> <pre>A(:)</pre> <p>分配新元素而不更改数组的形状:</p> <pre>A = rand(3,4); A(:) = 1:12;</pre> <p>对特定维度中某个范围的元素进行索引:</p> <pre>A(2:5,3)</pre> <p>对特定维度中的所有元素进行索引:</p> <pre>A(:,3)</pre> <p><b>for</b> 循环边界:</p> <pre>x = 1; for k = 1:25     x = x + x^2; end</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"><li>• colon</li><li>• “创建、串联和扩展矩阵”</li></ul>
---	---

;	<p><b>名称:</b> 分号</p> <p><b>用法:</b></p> <ul style="list-style-type: none"><li>• 表示行尾</li><li>• 禁止代码行输出</li></ul> <p><b>说明:</b> 使用分号可分隔数组创建命令中的各行，也可禁止代码行的输出显示。</p> <p><b>示例</b></p> <p>分隔各行以创建数组：</p> <pre>A = [12,13; 14,15]</pre> <p>禁止代码输出：</p> <pre>Y = max(A);</pre> <p>在单行中分隔多个命令（禁止输出）：</p> <pre>A = 12.5; B = 42.7, C = 1.25; B =     42.7000</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"><li>• <code>vertcat</code></li></ul>
---	--

( )	<p><b>名称:</b> 圆括号</p> <p><b>用法:</b></p> <ul style="list-style-type: none"><li>运算符优先级</li><li>括起函数参数</li><li>索引</li></ul> <p><b>说明:</b> 使用圆括号可指定运算优先级、括住函数输入参数以及对数组进行索引。</p> <p><b>示例</b></p> <p>运算优先级:</p> <pre>(A.*(B./C)) - D</pre> <p>括起函数参数:</p> <pre>plot(X,Y,'r*') C = union(A,B)</pre> <p>索引:</p> <pre>A(3,:) A(1,2) A(1:5,1)</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"><li>“运算符优先级” (第 2-30 页)</li><li>“数组索引”</li></ul>
-----	--

[ ]	<p><b>名称:</b> 方括号</p> <p><b>用法:</b></p> <ul style="list-style-type: none"><li>• 数组构造</li><li>• 数组串联</li><li>• 空矩阵和数组元素删除</li><li>• 多输出参数赋值</li></ul> <p><b>说明:</b> 方括号可以构造和串联数组、创建空矩阵、删除数组元素以及捕获函数返回的值。</p> <p><b>示例</b></p> <p>构造三元素向量:</p> <pre>X = [10 12 -3]</pre> <p>在矩阵底部添加新的一行:</p> <pre>A = rand(3); A = [A; 10 20 30]</pre> <p>创建空矩阵:</p> <pre>A = []</pre> <p>删除矩阵列:</p> <pre>A(:,1) = []</pre> <p>从函数捕获三个输出参数:</p> <pre>[C,iA,iB] = union(A,B)</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"><li>• “创建、串联和扩展矩阵”</li><li>• <b>horzcat</b></li><li>• <b>vertcat</b></li></ul>
-----	---

{ }	<p><b>名称:</b> 花括号</p> <p><b>用法:</b> 元胞数组赋值和内容</p> <p><b>说明:</b> 使用花括号可构造元胞数组，也可访问元胞数组中特定元胞的内容。</p> <p><b>示例</b></p> <p>要构造元胞数组，请将数组的所有元素括在花括号中：</p> <pre>C = {[2.6 4.7 3.9], rand(8)*6, 'C. Coolidge'};</pre> <p>通过将所有索引括在花括号中来对特定元胞数组元素进行索引：</p> <pre>A = C{4,7,2}</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"> <li>“元胞数组”</li> </ul>
%	<p><b>名称:</b> 百分号</p> <p><b>用法:</b></p> <ul style="list-style-type: none"> <li>注释</li> <li>转换设定符</li> </ul> <p><b>说明:</b> 百分号最常用于指示程序主体中不可执行的文本。此文本通常的作用是在代码中加入注释。</p> <p>某些函数还将百分号解释为转换设定符。</p> <p>根据“代码节”（第 18-5 页）中的描述，两个百分号 %% 用作元胞分隔符。</p> <p><b>示例</b></p> <p>在代码块中添加注释：</p> <pre>% The purpose of this loop is to compute % the value of ...</pre> <p>将转换设定符与 sprintf 结合使用：</p> <pre>sprintf('"%s = %d"', name, value)</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"> <li>“向程序中添加注释”（第 18-3 页）</li> </ul>

<code>%{ %}</code>	<p><b>名称:</b> 百分号加花括号</p> <p><b>用法:</b> 块注释</p> <p><b>说明:</b> <code>%{</code> 和 <code>%}</code> 符号将超出一行的注释块括起来。</p> <p><b>注意</b> 除空白字符之外，<code>%{</code> 和 <code>%}</code> 运算符必须在帮助文本块之前和之后另起一行，单独显示。不要在这些行中包括任何其他文本。</p> <p><b>示例</b></p> <p>用百分号后跟左花括号或右花括号将多行注释括起来：</p> <pre>%{ The purpose of this routine is to compute the value of ... %}</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"> <li>“向程序中添加注释”（第 18-3 页）</li> </ul>
<code>!</code>	<p><b>名称:</b> 感叹号</p> <p><b>用法:</b> 操作系统命令</p> <p><b>说明:</b> 感叹号位于要从 MATLAB 内部执行的操作系统命令的前面。</p> <p>在 MATLAB Online™ 中不可用。</p> <p><b>示例</b></p> <p>感叹号启动 shell 转义函数。此类函数将直接由操作系统执行：</p> <pre>!rmdir oldtests</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"> <li>“Shell Escape Function Example”</li> </ul>
<code>?</code>	<p><b>名称:</b> 问号</p> <p><b>用法:</b> MATLAB 类的元类</p> <p><b>说明:</b> 问号可检索特定类名的 <code>meta.class</code> 对象。<code>?</code> 运算符仅适用于类名而不是对象。</p> <p><b>示例</b></p> <p>检索类 <code>inputParser</code> 的 <code>meta.class</code> 对象：</p> <pre>?inputParser</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"> <li><code>metaclass</code></li> </ul>

"	<p><b>名称:</b> 单引号</p> <p><b>用法:</b> 字符数组构造符号</p> <p><b>说明:</b> 使用单引号可创建 <code>char</code> 类的字符向量。</p> <p><b>示例</b></p> <p>创建字符向量：</p> <pre>chr = 'Hello, world'</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"> <li>“字符串数组和字符数组中的文本”（第 6-2 页）</li> </ul>
""	<p><b>名称:</b> 双引号</p> <p><b>用法:</b> 字符串构造符号</p> <p><b>说明:</b> 使用双引号可创建 <code>string</code> 类的字符串标量。</p> <p><b>示例</b></p> <p>创建字符串标量：</p> <pre>S = "Hello, world"</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"> <li>“字符串数组和字符数组中的文本”（第 6-2 页）</li> </ul>
N/A	<p><b>名称:</b> 空格字符</p> <p><b>用法:</b> 分隔符</p> <p><b>说明:</b> 使用空格字符可分隔数组构造符号内的行元素，也可分隔函数返回的值。在这些上下文中，空格字符和逗号是等效的。</p> <p><b>示例</b></p> <p>分隔行元素以创建数组：</p> <pre>% These statements are equivalent A = [12 13; 14 15] A = [12,13; 14,15]</pre> <p>分隔函数调用中的输出参数：</p> <pre>% These statements are equivalent [Y I] = max(A) [Y,I] = max(A)</pre>

N/A	<p><b>名称:</b> 换行符</p> <p><b>用法:</b> 分隔符</p> <p><b>说明:</b> 使用换行符分隔数组构造语句中的多个行。在这种情况下，换行符和分号是等效的。</p> <p><b>示例</b></p> <p>在数组创建命令中分隔行：</p> <pre>% These statements are equivalent A = [12 13       14 15] A = [12 13; 14 15]</pre>
~	<p><b>名称:</b> 波浪号</p> <p><b>用法:</b></p> <ul style="list-style-type: none"> <li>逻辑非</li> <li>参数占位符</li> </ul> <p><b>说明:</b> 使用波浪号可表示逻辑非，也可禁止特定输入或输出参数。</p> <p><b>示例</b></p> <p>计算矩阵的逻辑 NOT：</p> <pre>A = eye(3); ~A</pre> <p>确定 A 的元素在哪些位置不等于 B 的元素：</p> <pre>A = [1 -1; 0 1] B = [1 -2; 3 2] A~=B</pre> <p>仅返回 union 的第三个输出值：</p> <pre>[~,~,iB] = union(A,B)</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"> <li><b>not</b></li> <li>“忽略函数输入” (第 21-10 页)</li> <li>“忽略函数输出” (第 1-4 页)</li> </ul>

=	<p><b>名称:</b> 等号</p> <p><b>用法:</b> 赋值</p> <p><b>说明:</b> 使用等号可为变量赋值。语法 <math>B = A</math> 用于将 <math>A</math> 的元素存储在变量 <math>B</math> 中。</p> <p><b>注意</b> = 字符用于赋值，而 == 字符用于比较两个数组中的元素。有关详细信息，请参阅 <a href="#">eq</a>。</p> <p><b>示例</b></p> <p>创建矩阵 <math>A</math>。将 <math>A</math> 中的值赋给新变量 <math>B</math>。最后，将一个新值赋给 <math>B</math> 中的第一个元素。</p> <pre>A = [1 0; -1 0]; B = A; B(1) = 200;</pre>
< &	<p><b>名称:</b> 左尖括号和 &amp; 符号</p> <p><b>用法:</b> 指定超类</p> <p><b>说明:</b> 在类定义中指定一个或多个超类</p> <p><b>示例</b></p> <p>定义从一个超类派生的类：</p> <pre>classdef MyClass &lt; MySuperclass ... end</pre> <p>定义从多个超类派生的类：</p> <pre>classdef MyClass &lt; Superclass1 &amp; Superclass2 &amp; ... ... end</pre> <p><b>更多信息:</b></p> <ul style="list-style-type: none"> <li>“子类语法”</li> </ul>

.?	<p><b>名称:</b> 点问号</p> <p><b>用法:</b> 指定名称-值结构体的字段</p> <p><b>说明:</b></p> <p>使用函数参数验证时，可以将名称-值结构体的字段定义为类的所有可写属性的名称。</p> <p><b>示例</b></p> <p>将 propArgs 结构体的字段名称指定为 matlab.graphics.primitive.Line 类的可写属性。</p> <pre>function f(propArgs)     arguments         propArgs.?matlab.graphics.primitive.Line     end     % Function code     ... end</pre> <p><b>更多信息:</b></p> <ul style="list-style-type: none"> <li>“基于类属性的名称-值参数”（第 26-12 页）</li> </ul>
----	---

## 字符串和字符格式化

某些特殊字符只能在字符向量或字符串的文本中使用。您可以使用这些特殊字符来插入换行符或回车符、指定文件夹路径以及执行更多操作。

使用下表中的特殊字符可通过字符向量或字符串来指定文件夹路径。

/ \	<p><b>名称:</b> 斜杠和反斜杠</p> <p><b>用法:</b> 文件或文件夹路径分隔</p> <p><b>说明:</b> 除了用作数学运算符以外，斜杠和反斜杠字符还可分隔路径或文件夹的元素。在基于 Microsoft Windows 的系统上，斜杠和反斜杠具有相同的效果。在基于 The Open Group UNIX 的系统上，只能使用斜杠。</p> <p><b>示例</b></p> <p>在 Windows 系统上，您可以使用反斜杠或斜杠：</p> <pre>dir([matlabroot '\toolbox\matlab\elmat\shiftdim.m']) dir([matlabroot '/toolbox/matlab/elmat/shiftdim.m'])</pre> <p>在 UNIX 系统上，只能使用正斜杠：</p> <pre>dir([matlabroot '/toolbox/matlab/elmat/shiftdim.m'])</pre>
-----	--

..	<p><b>名称:</b> 二连点</p> <p><b>用法:</b> 父文件夹</p> <p><b>说明:</b> 连续的两个点表示当前文件夹的父文件夹。使用此字符可指定相对于当前文件夹的文件夹路径。</p> <p><b>示例</b></p> <p>要在文件夹树中上移两层进入 test 文件夹, 请使用:</p> <pre>cd ..\..\test</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"> <li>• cd</li> </ul>
*	<p><b>名称:</b> 星号</p> <p><b>用法:</b> 通配符</p> <p><b>说明:</b> 除了用作矩阵乘法的符号以外, 星号 * 还可用作通配符。</p> <p>通配符一般用于对多个文件或文件夹执行的文件操作。MATLAB 会精确匹配名称中的所有字符, 但通配符 * 除外, 该字符可以与任何一个或多个字符匹配。</p> <p><b>示例</b></p> <p>查找名称以 january_ 开头并且文件扩展名为 .mat 的所有文件:</p> <pre>dir(january_*.mat)</pre>
@	<p><b>名称:</b> at 符号</p> <p><b>用法:</b> 类文件夹指示符</p> <p><b>说明:</b> @ 符号指示类文件夹的名称。</p> <p><b>示例</b></p> <p>表示类文件夹:</p> <pre>\@myClass\get.m</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"> <li>• “类和路径文件夹”</li> </ul>

<b>+</b>	<p><b>名称:</b> 加</p> <p><b>用法:</b> 包目录指示符</p> <p><b>说明:</b> + 符号指示包文件夹的名称。</p> <p><b>示例</b></p> <p>包文件夹始终以 + 字符开头：</p> <pre>+mypack +mypack/pkfcn.m % a package function +mypack/@myClass % class folder in a package</pre> <p><b>更多信息</b></p> <ul style="list-style-type: none"> <li>“包命名空间”</li> </ul>
----------	---

您无法将某些特殊字符作为普通文本输入，必须使用具有唯一性的字符序列来表示它们。您可单独使用下表中的符号或结合格式设置函数（如 `compose`、`sprintf` 和 `error`）来设置字符串和字符向量的格式。有关详细信息，请参阅“格式化文本”（第 6-24 页）。

符号	文本效果
''	单引号
%%	单个百分号
\\"	单个反斜杠
\a	警报
\b	退格符
\f	换页符
\n	换行符
\r	回车符
\t	水平制表符
\v	垂直制表符
\xN	十六进制数 N
\N	八进制数 N

## 另请参阅

### 详细信息

- “数组与矩阵运算”（第 2-19 页）
- “使用关系运算符进行数组比较”（第 2-27 页）
- “基本运算的兼容数组大小”（第 2-23 页）
- “运算符优先级”（第 2-30 页）
- “查找符合条件的数组元素”（第 5-2 页）
- “图文本中的希腊字母和特殊字符”

# 数组与矩阵运算

## 本节内容

- “简介” (第 2-19 页)
- “数组运算” (第 2-19 页)
- “矩阵运算” (第 2-21 页)

## 简介

MATLAB 具有两种不同类型的算术运算：数组运算和矩阵运算。您可以使用这些算术运算来执行数值计算，例如两数相加、计算数组元素的给定次幂或两个矩阵相乘。

矩阵运算遵循线性代数的法则。与之不同，数组运算则是执行逐元素运算并支持多维数组。句点字符 (.) 将数组运算与矩阵运算区别开来。但是，由于矩阵运算和数组运算在加法和减法的运算上相同，因此没有必要使用字符组合 .+ 和 .-。

## 数组运算

数组运算可针对向量、矩阵和多维数组的对应元素执行逐元素运算。如果操作数的大小相同，则第一个操作数中的每个元素都会与第二个操作数中同一位置的元素匹配。如果操作数的大小兼容，则每个输入都会根据需要进行隐式扩展以匹配另一个输入的大小。有关详细信息，请参阅“基本运算的兼容数组大小”(第 2-23 页)。

举一个简单的示例，您可以添加两个大小相同的向量。

```
A = [1 1 1]
```

```
A =
```

```
1 1 1
```

```
B = [1 2 3]
```

```
B =
```

```
1 2 3
```

```
A+B
```

```
ans =
```

```
2 3 4
```

如果一个操作数是标量，而另一个操作数不是标量，则 MATLAB 会将该标量隐式扩展为与另一个操作数具有相同的大小。例如，您可以计算一个标量和一个矩阵的按元素乘积。

```
A = [1 2 3; 1 2 3]
```

```
A =
```

```
1 2 3
1 2 3
```

```
3.*A
```

```
ans =  
3   6   9  
3   6   9
```

如果从一个  $3 \times 3$  矩阵中减去一个  $1 \times 3$  向量，隐式扩展仍然会起作用，因为它们的大小是兼容的。当您执行减法运算时，该向量将隐式扩展为一个  $3 \times 3$  矩阵。

```
A = [1 1 1; 2 2 2; 3 3 3]
```

```
A =
```

```
1   1   1  
2   2   2  
3   3   3
```

```
m = [2 4 6]
```

```
m =
```

```
2   4   6
```

```
A - m
```

```
ans =
```

```
-1   -3   -5  
0   -2   -4  
1   -1   -3
```

行向量和列向量的大小兼容。如果您将一个  $1 \times 3$  向量与一个  $2 \times 1$  向量相加，则每个向量都会在 MATLAB 执行按元素加法之前隐式扩展为一个  $2 \times 3$  矩阵。

```
x = [1 2 3]
```

```
x =
```

```
1   2   3
```

```
y = [10; 15]
```

```
y =
```

```
10  
15
```

```
x + y
```

```
ans =
```

```
11   12   13  
16   17   18
```

如果两个操作数的大小不兼容，则您将会收到错误消息。

```
A = [8 1 6; 3 5 7; 4 9 2]
```

```
A =
```

```
8   1   6
```

```
3   5   7
4   9   2
```

**m = [2 4]**

**m =**

```
2   4
```

**A - m**

Matrix dimensions must agree.

下表概述了 MATLAB 中的数组算术运算符。有关函数特定的信息，请点击最后一列中的函数参考页的链接。

运算符	用途	说明	参考页
+	加法	$A+B$ 表示将 $A$ 和 $B$ 加在一起。	<b>plus</b>
+	一元加法	$+A$ 表示返回 $A$ 。	<b>uplus</b>
-	减法	$A-B$ 表示从 $A$ 中减去 $B$	<b>minus</b>
-	一元减法	$-A$ 表示对 $A$ 的元素求反。	<b>uminus</b>
.*	按元素乘法	$A.*B$ 表示 $A$ 和 $B$ 的逐元素乘积。	<b>times</b>
.^	按元素求幂	$A.^B$ 表示包含元素 $A(i,j)$ 的 $B(i,j)$ 次幂的矩阵。	<b>power</b>
.	数组右除	$A./B$ 表示包含元素 $A(i,j)/B(i,j)$ 的矩阵。	<b>rdivide</b>
.	数组左除	$A.\B$ 表示包含元素 $B(i,j)/A(i,j)$ 的矩阵。	<b>ldivide</b>
'	数组转置	$A.'$ 表示 $A$ 的数组转置。对于复矩阵，这不涉及共轭。	<b>transpose</b>

## 矩阵运算

矩阵运算遵循线性代数的法则，与多维数组不兼容。运算双方所需输入的大小和形状取决于所执行的运算。对于非标量输入，矩阵运算符与对应的数组运算符计算出的答案通常不同。

例如，如果您使用矩阵右除运算符  $/$  来除两个矩阵，这两个矩阵必须具有相同的列数。但是，如果您使用矩阵乘法运算符  $*$  来乘两个矩阵，则这两个矩阵必须具有共同的内部维度。也即，第一个输入的列数必须等于第二个输入的行数。矩阵乘法运算符使用以下公式计算两个矩阵的乘积：

$$C(i, j) = \sum_{k=1}^n A(i, k)B(k, j).$$

要了解这一点，您可以计算两个矩阵的乘积。

**A = [1 3;2 4]**

**A =**

```
1   3
2   4
```

**B = [3 0;1 5]**

**B =**

```
3 0
1 5
```

A\*B

ans =

```
6 15
10 20
```

前一个矩阵乘积与下面的按元素乘积不相等。

A.\*B

ans =

```
3 0
2 20
```

下表概述了 MATLAB 中的矩阵算术运算符。有关函数特定的信息，请点击最后一列中的函数参考页的链接。

运算符	用途	说明	参考页
*	矩阵乘法	$C = A * B$ 表示矩阵 A 和 B 的线性代数乘积。A 的列数必须与 B 的行数相等。	<a href="#">mtimes</a>
\	矩阵左除	$x = A \backslash B$ 是方程 $Ax = B$ 的解。矩阵 A 和 B 必须拥有相同的行数。	<a href="#">mldivide</a>
/	矩阵右除	$x = B / A$ 是方程 $xA = B$ 的解。矩阵 A 和 B 必须拥有相同的列数。用左除运算符表示的话， $B / A = (A \backslash B)'$ 。	<a href="#">mrdivide</a>
^	矩阵幂	$A ^ B$ 表示 A 的 B 次幂（如果 B 为标量）。对于 B 的其他值，计算包含特征值和特征向量。	<a href="#">mpower</a>
'	复共轭转置	$A'$ 表示 A 的线性代数转置。对于复矩阵，这是复共轭转置。	<a href="#">ctranspose</a>

## 另请参阅

### 详细信息

- “基本运算的兼容数组大小”（第 2-23 页）
- “MATLAB 运算符和特殊字符”（第 2-2 页）
- “运算符优先级”（第 2-30 页）

## 基本运算的兼容数组大小

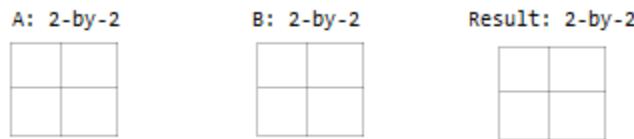
MATLAB 中的大多数二元（两个输入）运算符和函数都支持具有兼容大小的数值数组。对于每个维度，如果两个输入的维度大小相同或其中一个为 1，则这些输入将具有兼容的大小。以最简单的情况为例，如果两个数组大小完全相同或其中一个为标量，则这两个数组大小是兼容的。执行按元素运算或函数时，MATLAB 会将大小兼容的数组隐式扩展为相同的大小。

### 大小兼容的输入

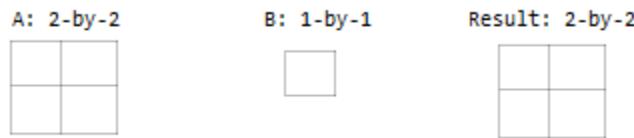
#### 二维输入

以下是一些具有兼容大小的标量、向量和矩阵的组合：

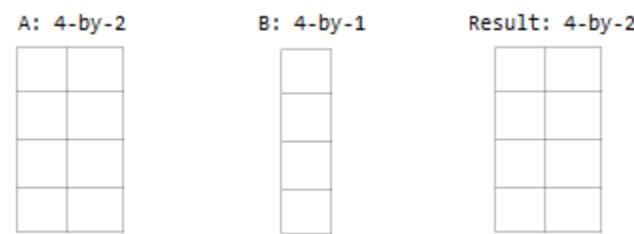
- 两个大小完全相同的输入。



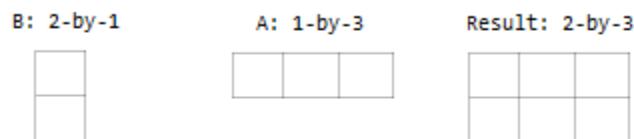
- 一个输入是标量。



- 一个输入是矩阵，另一个输入是具有相同行数的列向量。



- 一个输入是列向量，另一个输入是行向量。

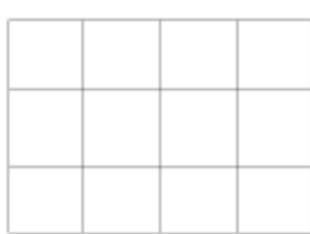


## 多维数组

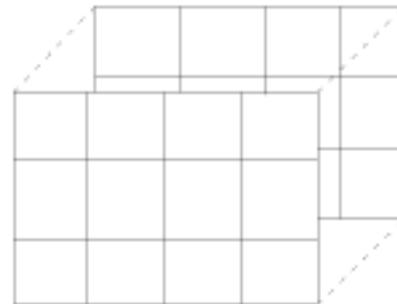
MATLAB 中的每个数组都具有大小为 1 的尾部维度。对于多维数组，这意味着  $3 \times 4$  矩阵与大小为  $3 \times 4 \times 1$  的矩阵相同。具有兼容大小的多维数组的示例如下：

- 一个输入是矩阵，另一个输入是具有相同行数和列数的三维数组。

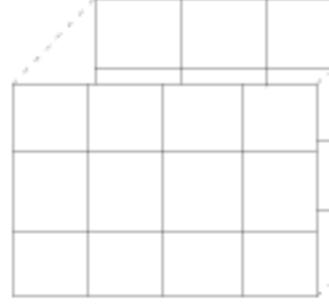
A: 3-by-4



B: 3-by-4-by-2



Result: 3-by-4-by-2

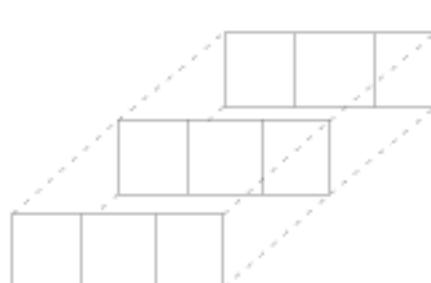


- 一个输入是矩阵，另一个输入是三维数组。这些维度要么都相同，要么其中一个为 1。

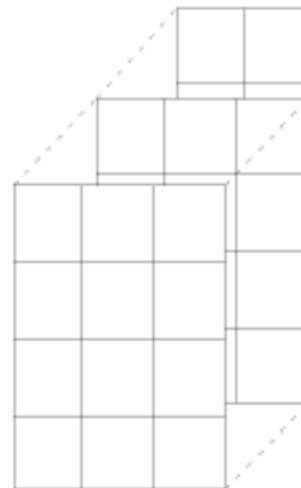
A: 4-by-3



B: 1-by-3-by-3



Result: 4-by-3



## 空数组

对于空数组或维度大小为零的数组，规则是相同的。不等于 1 的维度大小确定输出的大小。这意味着，大小为零的维度必须与另一个数组中大小为 1 或 0 的维度进行配对，并且输出的维度大小为 0。

```
A: 1-by-0
B: 3-by-1
Result: 3-by-0
```

## 大小不兼容的输入

不兼容的输入的大小无法隐式扩展为相同的大小。例如：

- 其中一个维度大小不相等，并且维度大小均不为 1。

```
A: 3-by-2
B: 4-by-2
```

- 两个长度不相同的非标量行向量。

```
A: 1-by-3
B: 1-by-4
```

## 示例

### 从矩阵减去向量

要简化向量-矩阵运算，请对维函数（例如 `sum`、`mean`、`min` 以及其他）使用隐式扩展。

例如，计算矩阵中每列的均值，然后从每个元素中减去均值。

```
A = magic(3)
```

```
A =
```

8	1	6
3	5	7
4	9	2

```
C = mean(A)
```

```
C =
```

5	5	5
---	---	---

```
A - C
```

```
ans =
```

3	-4	1
-2	0	2
-1	4	-3

### 行向量和列向量相加

行向量和列向量的大小兼容，当您对它们执行运算时，结果为一个矩阵。

例如，将行向量和列向量相加。结果与 `bsxfun(@plus,a,b)` 相同。

```
a = [1 2 3 4]
```

```
ans =
```

1	2	3	4
---	---	---	---

```
b = [5; 6; 7]
```

```
ans =
```

```
5  
6  
7
```

```
a + b
```

```
ans =
```

```
6   7   8   9  
7   8   9   10  
8   9   10  11
```

### 另请参阅

[bsxfun](#)

### 详细信息

- “数组与矩阵运算” (第 2-19 页)
- “MATLAB 运算符和特殊字符” (第 2-2 页)

# 使用关系运算符进行数组比较

## 本节内容

“数组比较” (第 2-27 页)

“逻辑语句” (第 2-29 页)

关系运算符使用“小于”、“大于”和“不等于”等运算符对操作数进行定量比较。关系比较的结果是一个逻辑数组，指示关系为 true 的位置。

这些是 MATLAB 中的关系运算符。

符号	等效函数	说明
<	lt	小于
<=	le	小于或等于
>	gt	大于
>=	ge	大于或等于
==	eq	等于
~=	ne	不等于

## 数组比较

### 数值数组

关系运算符会在两个数组之间执行按元素比较。数组的大小必须兼容以便于执行运算。执行计算时，具有兼容大小的数组会隐式扩展为相同的大小。以最简单的情况为例，两个操作数为大小相同的数组，或者其中一个操作数为标量。有关详细信息，请参阅“基本运算的兼容数组大小” (第 2-23 页)。

例如，如果您比较两个大小相同的矩阵，则结果将是大小相同且其元素指示关系为 true 的位置的逻辑矩阵。

A = [2 4 6; 8 10 12]

A =

2	4	6
8	10	12

B = [5 5 5; 9 9 9]

B =

5	5	5
9	9	9

A < B

ans =

1	1	0
1	0	0

同样，您也可以将某一个数组与标量进行比较。

**A > 7**

ans =

```
0 0 0  
1 1 1
```

如果您将一个  $1 \times N$  行向量与一个  $M \times 1$  列向量进行比较，则 MATLAB 会在执行比较之前将每个向量都扩展为一个  $M \times N$  矩阵。生成的矩阵包含这些向量中元素的每个组合的比较结果。

**A = 1:3**

A =

```
1 2 3
```

**B = [2; 3]**

B =

```
2  
3
```

**A >= B**

ans =

```
0 1 1  
0 0 1
```

### 空数组

关系运算符可用于其中有任一维度大小为零的数组，只要两个数组的大小兼容。这意味着，如果一个数组的维度大小为 0，则另一个数组中对应维度的大小必须为 1 或 0，并且输出中该维度的大小为 0。

```
A = ones(3,0);  
B = ones(3,1);  
A == B
```

ans =

```
Empty matrix: 3-by-0
```

但是，类似下面的表达式

```
A == []
```

在 A 不是  $0 \times 0$  或  $1 \times 1$  矩阵时将返回错误。此行为与所有其他二元运算符（例如 +、-、>、<、&、| 等）的行为一致。

要验证数组是否为空数组，请使用 `isempty(A)`。

### 复数

- 运算符  $>$ 、 $<$ 、 $>=$  和  $<=$  在执行比较时仅使用操作数的实部。
- 运算符  $==$  和  $\sim=$  会同时检验操作数的实部和虚部。

## Inf、NaN、NaT 和 undefined 元素比较

- Inf 值与其他 Inf 值相等。
- NaN 值与任何其他数值（包括其他 NaN 值）都不相等。
- NaT 值与任何其他日期时间值（包括其他 NaT 值）都不相等。
- 未定义的分类元素不等于任何其他分类值，包括其他未定义的元素。

## 逻辑语句

将关系运算符与逻辑运算符 A & B (AND)、A | B (OR)、xor(A,B) (XOR) 和 ~A (NOT) 结合使用可以构建更为复杂的逻辑语句。

例如，您可以找到负数元素出现在两个数组中的位置。

**A = [2 -1; -3 10]**

A =

2	-1
-3	10

**B = [0 -2; -3 -1]**

B =

0	-2
-3	-1

**A<0 & B<0**

ans =

0	1
1	0

有关更多示例，请参阅“查找符合条件的数组元素”（第 5-2 页）。

## 另请参阅

**eq | ge | gt | le | lt | ne**

## 详细信息

- “数组与矩阵运算”（第 2-19 页）
- “基本运算的兼容数组大小”（第 2-23 页）
- “MATLAB 运算符和特殊字符”（第 2-2 页）

## 运算符优先级

您可以构建使用算术运算符、关系运算符和逻辑运算符的任意组合的表达式。优先级别用来确定 MATLAB 计算表达式时的运算顺序。处于同一优先级别的运算符具有相同的运算优先级，将从左至右依次进行计算。下表显示了 MATLAB 运算符的优先级规则，顺序从最高优先级别到最低优先级别：

- 1 括号 ()
  - 2 转置 (.)、幂 (.^)、复共轭转置 (')、矩阵幂 (^)
  - 3 带一元减法 (^-)、一元加法 (^+) 或逻辑求反 (^~) 的幂，以及带一元减法 (^-)、一元加法 (^+) 或逻辑求反 (^~) 的矩阵幂。
- 
- 注意** 尽管大多数运算符都从左至右运行，但 (^-)、(^-)、(^+)、(^+)、(^~) 和 (^~) 按从右至左顺序从第二个运行。建议您使用括号显式指定包含这些运算符组合的语句的期望优先级。
- 4 一元加法 (+)、一元减法 (-)、逻辑求反 (~)
  - 5 乘法 (.\* )、右除 (/.)、左除 (\.)、矩阵乘法 (\*)、矩阵右除 (/)、矩阵左除 (\)
  - 6 加法 (+)、减法 (-)
  - 7 冒号运算符 (:)
  - 8 小于 (<)、小于或等于 (<=)、大于 (>)、大于或等于 (>=)、等于 (==)、不等于 (~=)
  - 9 按元素 AND (&)
  - 10 按元素 OR (|)
  - 11 短路 AND (&&)
  - 12 短路 OR (||)

## AND 和 OR 运算符的优先级

MATLAB 始终将 & 运算符的优先级指定为高于 | 运算符。尽管 MATLAB 通常按从左到右的顺序计算表达式，但表达式  $a|b\&c$  按  $a|(b\&c)$  形式计算。对于包含 & 和 | 组合的语句，比较好的做法是使用括号显式指定期望的语句优先级。

该优先级规则同样适用于 && 和 || 运算符。

## 覆盖默认优先级

可以使用括号覆盖默认优先级，如以下示例中所示：

```
A = [3 9 5];
B = [2 1 5];
C = A./B.^2
C =
    0.7500   9.0000   0.2000

C = (A./B).^2
C =
    2.2500  81.0000  1.0000
```

## 另请参阅

### 详细信息

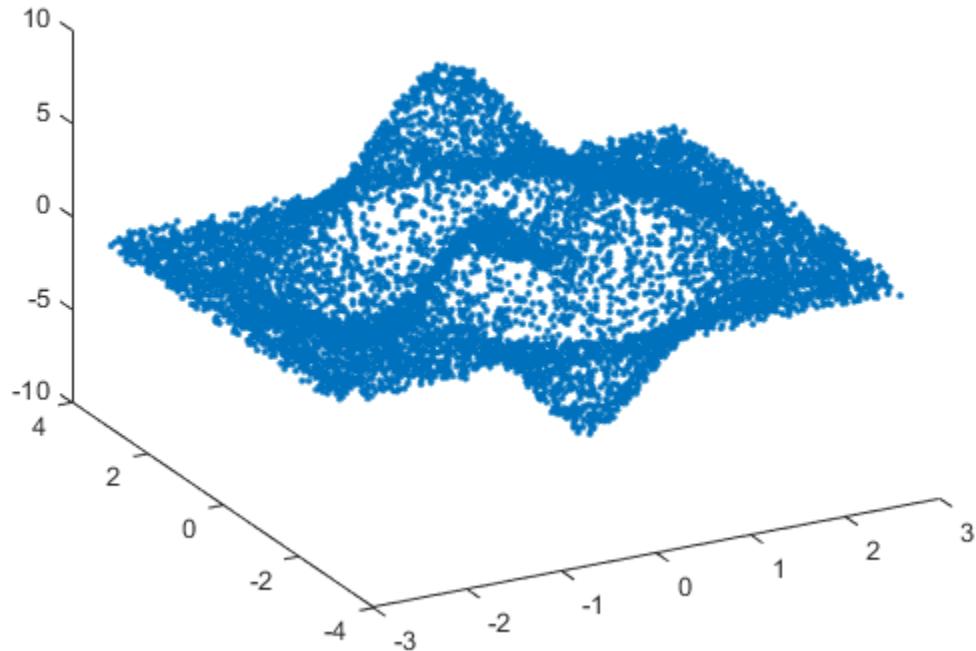
- “数组与矩阵运算” (第 2-19 页)
- “基本运算的兼容数组大小” (第 2-23 页)
- “使用关系运算符进行数组比较” (第 2-27 页)
- “MATLAB 运算符和特殊字符” (第 2-2 页)

## 使用容差为类似数据点求平均值

此示例说明如何使用 `uniquetol` 来为具有类似（在容差范围内）的 x 和 y 坐标的三维点求平均 z 坐标。

使用通过 `peaks` 函数在  $[-3, 3] \times [-3, 3]$  域内选取的随机点作为数据集。向数据中添加少量干扰。

```
xy = rand(10000,2)*6-3;
z = peaks(xy(:,1),xy(:,2)) + 0.5-rand(10000,1);
A = [xy z];
plot3(A(:,1), A(:,2), A(:,3), 'o')
view(-28,32)
```



使用以下选项，通过 `uniquetol` 查找具有类似 x 和 y 坐标的点：

- 将 `ByRows` 指定为 `true`，因为 A 的行包含点坐标。
- 将 `OutputAllIndices` 指定为 `true`，以返回在彼此容差范围内的所有点的索引。
- 将 `DataScale` 指定为 `[1 1 Inf]`，以将绝对容差用于 x 和 y 坐标，同时忽略 z 坐标。

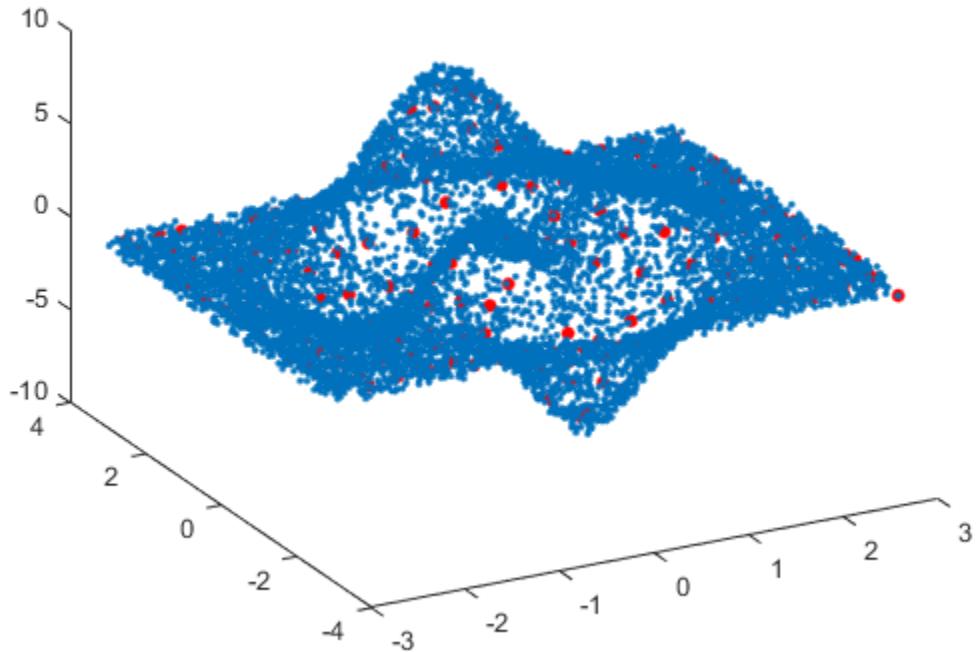
```
DS = [1 1 Inf];
[C,ia] = uniquetol(A, 0.3, 'ByRows', true, ...
    'OutputAllIndices', true, 'DataScale', DS);
```

为在容差范围内的每组点求平均值（包括 z 坐标），从而生成仍保持原始数据大致形状的约简数据集。

```
for k = 1:length(ia)
    aveA(k,:) = mean(A(ia{k},:),1);
end
```

在原始数据的上方标绘生成的平均点。

```
hold on  
plot3(aveA(:,1), aveA(:,2), aveA(:,3), '.r', 'MarkerSize', 15)
```



## 另请参阅

[uniquetol](#)

## 详细信息

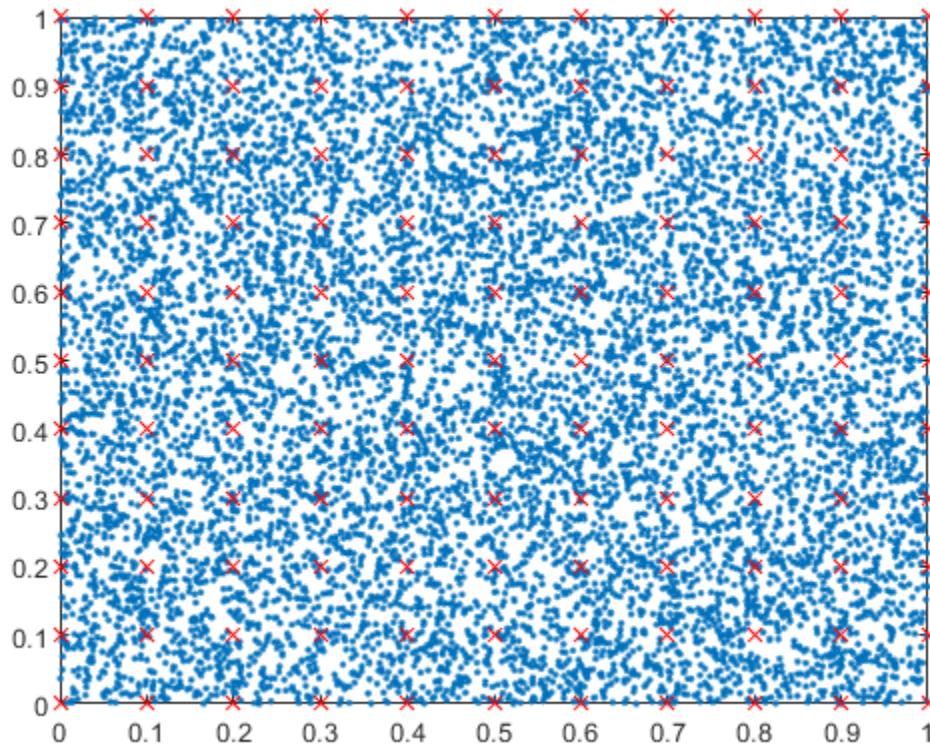
- “使用容差为散点数据分组” (第 2-34 页)

## 使用容差为散点数据分组

本示例展示如何根据散点数据点与相关点的邻近度对其进行分组。

创建一个随机二维点集。然后创建并在随机数据的上方绘制等距点网格。

```
x = rand(10000,2);
[a,b] = meshgrid(0:0.1:1);
gridPoints = [a(:), b(:)];
plot(x(:,1), x(:,2), '.')
hold on
plot(gridPoints(:,1), gridPoints(:,2), 'xr', 'MarkerSize', 6)
```



使用 `ismembertol`，在 `x` 内查找处于 `gridPoints` 中的网格点容差范围内的数据点。将以下选项与 `ismembertol` 配合使用：

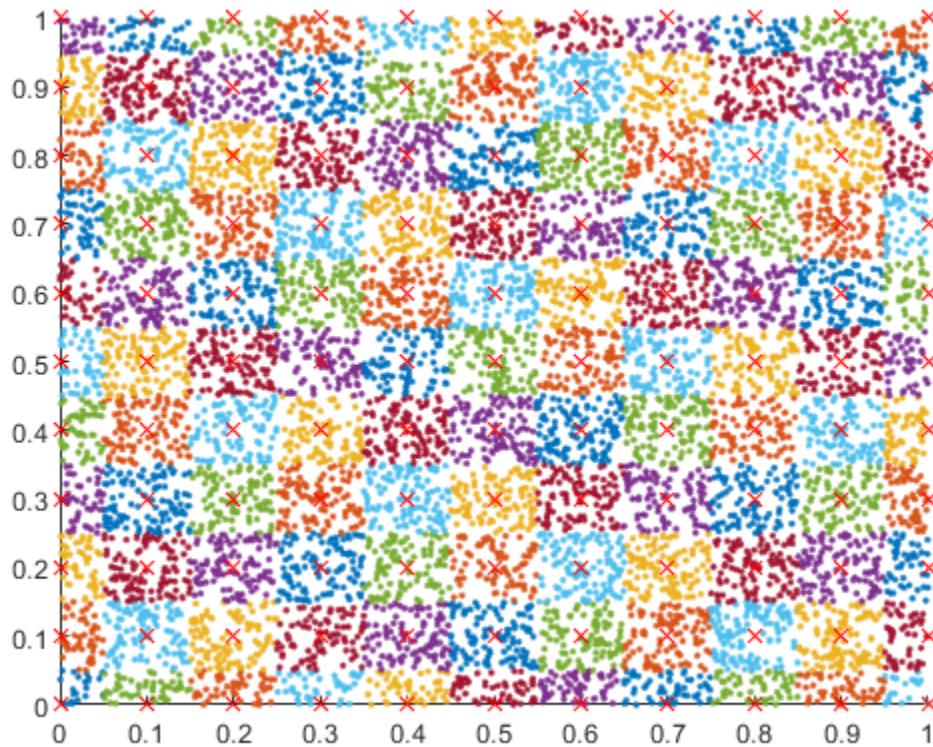
- 将 `ByRows` 指定为 `true`，因为点坐标在 `x` 的行中。
- 将 `OutputAllIndices` 指定为 `true`，以返回 `x` 中处于 `gridPoints` 对应行容差范围内的行的所有索引。

```
[LIA,LocB] = ismembertol(gridPoints, x, 0.05, ...
    'ByRows', true, 'OutputAllIndices', true);
```

针对每个网格点，标绘 `x` 中处于该网格点容差范围内的点。

```
figure
hold on
```

```
for k = 1:length(LocB)
    plot(x(LocB{k},1), x(LocB{k},2), 'r')
end
plot(gridPoints(:,1), gridPoints(:,2), 'xr', 'MarkerSize', 6)
```



## 另请参阅

[ismembertol](#)

## 详细信息

- “使用容差为类似数据点求平均值” (第 2-32 页)

## 按位运算

本主题说明如何在 MATLAB® 中使用按位运算来操作数字的位。大多数现代处理器直接支持位运算。在许多情况下，以这种方式操作数字的位比执行除法或乘法等算术运算更快。

### 数值表示

任何数值都可以用位来表示（也称为二进制位）。数值的二进制（即基数为 2）形式包含 1 和 0，以此表示数值中都存在 2 的哪些次幂。例如，7 的 8 位二进制形式是

```
00000111
```

8 个位的集合也称为 1 个字节。在二进制表示中，位从右向左计数，因此该表示中的第一个位是 1。此数值表示 7，因为

$$2^2 + 2^1 + 2^0 = 7.$$

当您在 MATLAB 中键入数值时，它假设数值为双精度（64 位二进制表示）。但是，您也可以指定单精度数（32 位二进制表示）和整数（有符号或无符号，从 8 到 64 位）。例如，要存储数值 7，最节省内存的方法是使用 8 位无符号整数：

```
a = uint8(7)
a = uint8
    7
```

您甚至可以直接使用前缀 **0b** 后跟二进制数字来指定二进制形式（有关详细信息，请参阅“十六进制和二进制值”（第 6-45 页））。MATLAB 以位数最少的整数格式存储数值。您只需指定最左边的 1 和它右边的所有位，而不必指定所有位。该位左边的位是无效零。因此数值 7 是：

```
b = 0b111
b = uint8
    7
```

MATLAB 使用 2 的补码存储负整数。以 8 位有符号整数 -8 为例。要找到该数值的 2 的补码位模式，请执行以下操作：

- 1 首先找到该数值对应的正数 8 的位模式：00001000。
- 2 接下来，翻转所有位：11110111。
- 3 最后，对结果加 1：11111000。

得到的 11111000 是 -8 的位模式：

```
n = 0b11111000s8
n = int8
    -8
```

MATLAB 并不主动显示数值的二进制格式。为此，您可以使用 **dec2bin** 函数，该函数会返回正整数的二进制数字字符串。同样，此函数只返回不包含无效零的位。

```
dec2bin(b)
```

```
ans =
'111'
```

您可以使用 **bin2dec** 在这两种格式之间切换。例如，您可以使用以下命令将二进制数字 **10110101** 转换为十进制格式

```
data = [1 0 1 1 0 1 0 1];
dec = bin2dec(num2str(data))

dec = 181
```

**cast** 和 **typecast** 函数也可用于不同数据类型之间的切换。这些函数是相似的，但它们在如何处理数值的底层存储方面有所不同：

- **cast** - 改变量的基础数据类型。
- **typecast** - 转换数据类型而不更改基础位。

由于 MATLAB 不直接显示二进制数的位，您在进行按位运算时必须注意数据类型。有些函数以字符串形式返回二进制数字 (**dec2bin**)，有些函数返回十进制数 (**bitand**)，还有一些函数返回由位本身组成的向量 (**bitget**)。

### 用逻辑运算符进行位掩码

使用 MATLAB 中的一些函数，您可以对以等长二进制表示的两个数值中的位执行逻辑运算，此种运算称为位掩码：

- **bitand** - 如果两个位均为 1，则结果位也是 1。否则，结果位为 0。
- **bitor** - 如果任一位是 1，则结果位也是 1。否则，结果位为 0。
- **bitxor** - 如果位不同，则结果位为 1。否则，结果位为 0。

除了这些函数之外，您还可以使用 **bitcmp** 进行按位补码，但这是一元运算，一次只能翻转一个数值中的位。

位掩码的一个用途是查询特定位的状态。例如，如果对二进制数 **00001000** 进行按位 AND 运算，您可以查询第四个位的状态。然后，您可以将该位移至第一个位置，以便 MATLAB 返回 0 或 1（下一节将更详细地说明位移）。

```
n = 0b10111001;
n4 = bitand(n,0b1000);
n4 = bitshift(n4,-3)

n4 = uint8
    1
```

按位运算有时可以发挥意想不到的作用。例如，以下是数值  $n = 8$  的 8 位二进制表示：

**00001000**

8 是 2 的幂，因此它的二进制表示只包含一个 1。现在考虑数值  $(n - 1) = 7$ ：

**00000111**

由于减去了 1，从最右边的 1 开始的所有位都会翻转。因此，当  $n$  是 2 的幂时， $n$  和  $(n - 1)$  的对应位始终不同，使得按位 AND 返回零。

```
n = 0b1000;
bitand(n,n-1)

ans = uint8
    0
```

但是，如果  $n$  不是 2 的幂，则最右边的 1 表示  $2^0$  位，因此  $n$  和  $(n - 1)$  除了  $2^0$  位之外，其他位都相同。在这种情况下，按位 AND 返回一个非零数值。

```
n = 0b101;
bitand(n,n-1)

ans = uint8
    4
```

受上述运算启发，我们可以编写一个简单的函数对给定的输入数值执行位运算，以判断它是否为 2 的幂：

```
function tf = isPowerOfTwo(n)
    tf = n && ~bitand(n,n-1);
end
```

使用短路 AND 运算符 `&&` 检查以确保  $n$  不为零。如果为零，则该函数不需要计算 `bitand(n,n-1)` 即可知道正确答案是 `false`。

### 移位

由于按位逻辑运算比较两个数值中对应的位，因此，能够按需移位以便于比较对应位就显得非常重要。您可以使用 `bitshift` 执行此操作：

- `bitshift(A,N)` 将  $A$  的位向左移动  $N$  位。这等效于将  $A$  和  $2^N$  相乘。
- `bitshift(A,-N)` 将  $A$  的位向右移动  $N$  位。这等效于将  $A$  除以  $2^N$ 。

上述操作有时写作  $A \ll N$  (左移) 和  $A \gg N$  (右移)，但 MATLAB 没有将 `<<` 和 `>>` 运算符用于此目的。

当数值的位发生移动时，数值会从末尾丢弃一些位，并引入 0 或 1 来填充新腾出的空间。当您向左移动位时，右端发生位填充；当您向右移动位时，左端发生位填充。

例如，如果将数值 8 (二进制：1000) 右移一位，则得到 4 (二进制：100)。

```
n = 0b1000;
bitshift(n,-1)

ans = uint8
    4
```

同样，如果将数值 15 (二进制：1111) 左移两位，则得到 60 (二进制：111100)。

```
n = 0b1111;
bitshift(15,2)

ans = 60
```

当您移动负数的位时，`bitshift` 会保留有符号位。例如，如果有符号整数 -1 (二进制：11111101) 向右移动 2 位，则得到 -1 (二进制：11111111)。在这些情况下，`bitshift` 在左端填充 1 而不是 0。

```
n = 0b11111101s8;
bitshift(n,-2)

ans = int8
    -1
```

## 写入位

您可以使用 **bitset** 函数来更改数值中的位。例如，将数值 8 的第一个位更改为 1（相当于将该数值加 1）：

```
bitset(8,1)
```

```
ans = 9
```

默认情况下，**bitset** 将位翻转为 on 或 1。您可以选择使用第三个输入参数来指定位值。

**bitset** 不会一次更改多个位；要更改多个位，您需要使用 **for** 循环。因此，您更改的位可以是连续的，也可以是非连续的。例如，更改二进制数 1000 的前两位：

```
bits = [1 2];
c = 0b1000;
for k = 1:numel(bits)
    c = bitset(c,bits(k));
end
dec2bin(c)
```

```
ans =
'1011'
```

**bitset** 的另一个常见用途是将二进制数字向量转换为十进制格式。例如，使用循环来设置整数 11001101 的各个位。

```
data = [1 1 0 0 1 1 0 1];
n = length(data);
dec = 0b0u8;
for k = 1:n
    dec = bitset(dec,n+1-k,data(k));
end
dec

dec = uint8
205

dec2bin(dec)

ans =
'11001101'
```

## 读取连续位

位移的另一个用途是隔离位的连续部分。例如，读取 16 位数值 0110000010100000 中的最后四位。前面提到，最后四位位于二进制表示的左端。

```
n = 0b0110000010100000;
dec2bin(bitshift(n,-12))

ans =
'110'
```

要隔离该数值中间的连续位，可以结合使用位移和逻辑掩码。例如，要提取第 13 位和第 14 位，可以向右移动 12 位，然后用 0011 对所得的 4 位进行掩码。由于 **bitand** 的输入必须为相同的整数数据类型，您可以使用 **0b11u16** 将 0011 指定为无符号 16 位整数。如果没有 **-u16** 后缀，MATLAB 会将数值存储为无符号 8 位整数。

```
m = 0b11u16;
dec2bin(bitand(bitshift(n,-12),m))

ans =
'10'
```

读取连续位的另一种方法是使用 **bitget**，它从数值中读取指定的位。您可以使用冒号表示法指定要读取的几个连续位。例如，读取 **n** 的最后 8 位。

```
bitget(n,16:-1:8)

ans = 1x9 uint16 row vector

0 1 1 0 0 0 0 0 1
```

### 读取非连续位

您也可以使用 **bitget** 从数值中读取彼此不相邻的位。例如，从 **n** 中读取第 5、8 和 14 位。

```
bits = [14 8 5];
bitget(n,bits)

ans = 1x3 uint16 row vector

1 1 0
```

### 另请参阅

**bitand** | **bitcmp** | **bitget** | **bitor** | **bitset** | **bitshift** | **bitxor**

### 详细信息

- “整数” (第 4-2 页)
- “执行循环冗余校验” (第 2-41 页)
- “十六进制和二进制值” (第 6-45 页)

## 执行循环冗余校验

此示例说明如何对数值的位执行循环冗余校验 (CRC)。CRC 用于检测数字系统中数据传输中的错误。发送数据时，会对数据附加一个短校验值。该校验值通过用数据中的位进行多项式除法获得。当接收到数据时，重复执行多项式除法，并将结果与校验值进行比较。如果结果不同，则数据在传输过程中被破坏。

### 手动计算校验值

从一个 16 位二进制数开始，这是要传输的报文：

**1101100111011010**

要获得校验值，请将该数除以多项式  $x^3 + x^2 + x + 1$ 。您可以用其系数来表示此多项式：1111。

除法是分步骤进行的，每步后多项式除数都与数值中最左边的 1 对齐。由于除以四项多项式的结果有三位（通常除以长度为  $n + 1$  的多项式会产生长度为  $n$  的校验值），请对该数值追加 000 以计算余数。每一步，结果都对正在操作的四个位进行按位 XOR，所有其他位保持不变。

第一个除法是

**1101100111011010 000**

1111

—————

**0010100111011010 000**

每个后续除法运算都基于前一步的结果，因此第二个除法是

**0010100111011010 000**

1111

—————

**0001010111011010 000**

一旦被除数全为零，则除法完成。整个除法运算过程（包括上述两步在内）如下

**1101100111011010 000**

1111

**0010100111011010 000**

1111

**0001010111011010 000**

1111

**0000101111011010 000**

1111

**0000010011011010 000**

1111

**0000001101011010 000**

1111

**0000000010011010 000**

1111

**0000000001101010 000**

1111

**000000000010010 000**

1111

**000000000001100 000**

1111

**000000000000111 000**

11 11

**000000000000000 110**

余数位 110 是该报文的校验值。

### 以编程方式计算校验值

在 MATLAB® 中，您可以通过按位运算执行上述运算来获得校验值。首先，为报文和多项式除数定义变量。使用无符号 32 位整数，以便有额外的位可供余数使用。

```
message = 0b1101100111011010u32;
messageLength = 16;
divisor = 0b1111u32;
divisorDegree = 3;
```

接下来，初始化多项式除数。使用 `dec2bin` 显示结果的位。

```
divisor = bitshift(divisor,messageLength-divisorDegree-1);
dec2bin(divisor)
```

```
ans =
'1111000000000000'
```

现在，移动除数和报文，使它们具有正确的位数（报文为 16 位，余数为 3 位）。

```
divisor = bitshift(divisor,divisorDegree);
remainder = bitshift(message,divisorDegree);
dec2bin(divisor)
```

```
ans =
'1111000000000000'
dec2bin(remainder)
```

```
ans =
'1101100111011010000'
```

使用 `for` 循环执行循环冗余校验的除法步骤。`for` 循环每步都会前进一位，因此请包含一项校验来查看当前位置是否为 1。如果当前位置是 1，则执行除法步骤；否则，循环前进一位并继续。

```
for k = 1:messageLength
    if bitget(remainder,messageLength+divisorDegree)
        remainder = bitxor(remainder,divisor);
    end
    remainder = bitshift(remainder,1);
end
```

将余数的位向右移动，以获得运算的校验值。

```
CRC_check_value = bitshift(remainder,-messageLength);
dec2bin(CRC_check_value)

ans =
'110'
```

### 检查报文完整性

您可以重复上述除法运算以使用校验值来验证报文的完整性。但是，不要使用余数 000 开始，而应使用校验值 110。如果报文没有错误，则除法的结果将为零。

重置余数变量，并使用按位 OR 将 CRC 校验值添加到余数位。使用 `bitset` 翻转一个位值，以在报文中引入一个错误。

```

remainder = bitshift(message,divisorDegree);
remainder = bitor(remainder,CRC_check_value);
remainder = bitset(remainder,6);
dec2bin(remainder)

ans =
'1101100111011110110'

```

执行 CRC 除法运算，然后检查结果是否为零。

```

for k = 1:messageLength
    if bitget(remainder,messageLength+divisorDegree)
        remainder = bitxor(remainder,divisor);
    end
    remainder = bitshift(remainder,1);
end
if remainder == 0
    disp('Message is error free.')
else
    disp('Message contains errors.')
end

```

Message contains errors.

## 参考

- [1] Sklar, Bernard. *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice Hall, 1988.
- [2] Wicker, Stephen B. *Error Control Systems for Digital Communication and Storage*. Upper Saddle River, NJ: Prentice Hall, 1995.

## 另请参阅

[bitshift](#) | [bitxor](#)

## 详细信息

- “按位运算”（第 2-36 页）
- “十六进制和二进制值”（第 6-45 页）

## 条件语句

条件语句可用于在运行时选择要执行的代码块。最简单的条件语句为 **if** 语句。例如：

```
% Generate a random number
a = randi(100, 1);

% If it is even, divide by 2
if rem(a, 2) == 0
    disp('a is even')
    b = a/2;
end
```

通过使用可选关键字 **elseif** 或 **else**, **if** 语句可以包含备用选项。例如：

```
a = randi(100, 1);

if a < 30
    disp('small')
elseif a < 80
    disp('medium')
else
    disp('large')
end
```

再者，当您希望针对一组已知值测试相等性时，请使用 **switch** 语句。例如：

```
[dayNum, dayString] = weekday(date, 'long', 'en_US');

switch dayString
    case 'Monday'
        disp('Start of the work week')
    case 'Tuesday'
        disp('Day 2')
    case 'Wednesday'
        disp('Day 3')
    case 'Thursday'
        disp('Day 4')
    case 'Friday'
        disp('Last day of the work week')
    otherwise
        disp('Weekend!')
end
```

对于 **if** 和 **switch**, MATLAB 执行与第一个 **true** 条件相对应的代码，然后退出该代码块。每个条件语句都需要 **end** 关键字。

一般而言，如果您具有多个可能的离散已知值，读取 **switch** 语句比读取 **if** 语句更容易。但是，无法测试 **switch** 和 **case** 值之间的不相等性。例如，无法使用 **switch** 实现以下类型的条件：

```
yourNumber = input('Enter a number: ');

if yourNumber < 0
    disp('Negative')
elseif yourNumber > 0
    disp('Positive')
else
```

```
    disp('Zero')
end
```

### 另请参阅

end | if | return | switch

## 循环控制语句

通过循环控制语句，您可以重复执行代码块。循环有两种类型：

- **for** 语句：循环特定次数，并通过递增的索引变量跟踪每次迭代。

例如，预分配一个 10 元素向量并计算五个值：

```
x = ones(1,10);
for n = 2:6
    x(n) = 2 * x(n - 1);
end
```

- **while** 语句：只要条件仍然为 true 就进行循环。

例如，计算使 **factorial(n)** 成为 100 位数的第一个整数 **n**：

```
n = 1;
nFactorial = 1;
while nFactorial < 1e100
    n = n + 1;
    nFactorial = nFactorial * n;
end
```

每个循环都需要 **end** 关键字。

最好对循环进行缩进处理以便于阅读，特别是使用嵌套循环时（也即一个循环包含另一个循环）：

```
A = zeros(5,100);
for m = 1:5
    for n = 1:100
        A(m, n) = 1/(m + n - 1);
    end
end
```

您可以使用 **break** 语句以编程方式退出循环，也可以使用 **continue** 语句跳到循环的下一次迭代。例如，计算 **magic** 函数的帮助中的行数（也即空行之前的所有注释行）：

```
fid = fopen('magic.m','r');
count = 0;
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line)
        break
    elseif ~strcmp(line,'% ',1)
        continue
    end
    count = count + 1;
end
fprintf('%d lines in MAGIC help\n',count);
fclose(fid);
```

---

**提示** 如果意外创建了一个无限循环（永远不会自行结束的循环），请按 **Ctrl+C** 停止执行循环。

**另请参阅**

`break` | `continue` | `end` | `for` | `while`

## 正则表达式

### 本节内容

- “什么是正则表达式？”（第 2-48 页）
- “构建表达式的步骤”（第 2-49 页）
- “运算符和字符”（第 2-51 页）

### 什么是正则表达式？

正则表达式是一串用于定义某种模式的字符。在有些情况下（例如，在解析程序输入或处理文本块时），您通常会使用正则表达式在文本中搜索与该模式匹配的一组单词。

字符串向量 '**Joh?n\w\***' 是一个正则表达式的示例。该字符串定义的模式以字母 **Jo** 开头，后面可跟，也可不跟字母 **h**（由 '**h?**' 指示），然后跟有字母 **n**，最后以任意数量的单词字符结尾（由 '**\w\***' 指示），此处的单词字符指字母、数字或下划字符。该模式与以下任意条目匹配：

**Jon, John, Jonathan, Johnny**

正则表达式提供了一种在大量文本中搜索特定字符子集的独特方式。正则表达式使您能够查找特定模式的字符，而不必像使用 **strfind** 等函数那样查找字符的完全匹配项。

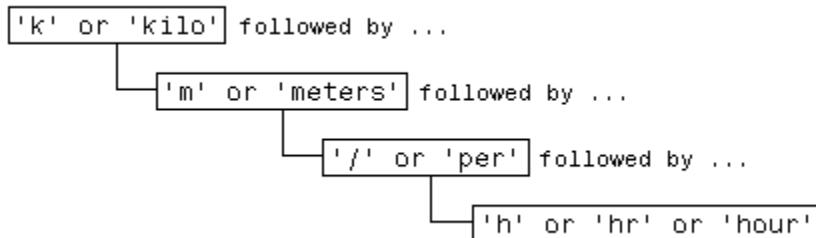
例如，有多种表示公制速率的方式：

**km/h  
km/hr  
km/hour  
kilometers/hour  
kilometers per hour**

您可以通过发出五个单独的搜索命令在文本中查找以上任意词汇：

```
strfind(text, 'km/h');
strfind(text, 'km/hour');
% etc.
```

当然，为了提高效率，您也可以构建一个适用于所有这些搜索词汇的短语：



将此短语转换为正则表达式（本部分后面将会介绍），您将具有如下表达式：

```
pattern = 'k(ilo)?m(eters)?(/|\sper\s)h(r|our)?';
```

现在仅使用一个命令来查找一个或多个词汇：

```

text = ["The high-speed train traveled at 250 ', ...
    'kilometers per hour alongside the automobile ', ...
    'travelling at 120 km/h.'];
regexp(text, pattern, 'match')

ans =
1×2 cell array

{'kilometers per hour'}  {'km/h'}

```

有四个 MATLAB 函数支持使用正则表达式搜索和替换字符。前三个函数在接受的输入值和返回的输出值方面类似。有关详细信息，请点击函数参考页的链接。

函数	说明
<b>regexp</b>	匹配正则表达式。
<b>regexpi</b>	匹配正则表达式并忽略大小写。
<b>regexp替換</b>	使用正则表达式替換部分文本。
<b>regexptranslate</b>	将文本转换为正则表达式。

调用前三个函数中的任何一个函数时，请在前两个输入参数中传递要解析的文本以及正则表达式。调用 **regexp替換** 时，还需再传递一个额外的输入，该输入是一个表达式，用于指定替代的模式。

## 构建表达式的步骤

使用正则表达式在文本中搜索特定词涉及以下三个步骤：

**1 确定字符串中的独特模式（第 2-50 页）**

这需要根据字符形式的类同情况对要搜索的文本进行拆分。这些字符形式可以是一系列小写字母、一个美元符号后跟三个数字，然后跟有一个小数点等。

**2 将每种模式表示为正则表达式（第 2-50 页）**

使用本文档中所述的元字符和运算符将搜索模式的每个段表示为正则表达式。然后，将这些表达式段组合成单个表达式以在搜索时使用。

**3 - 调用合适的搜索函数（第 2-50 页）**

将要解析的文本传递给其中一个搜索函数（例如 **regexp** 或 **regexpi**），或者传递给文本替代函数 **regexp替換**。

此部分中显示的示例搜索包含群组中五个朋友的联系信息的记录。该信息包括每个人的姓名、电话号码、居住地和电子邮件地址。目标是从文本中提取特定信息。

```

contacts = {
'Harry 287-625-7315 Columbus, OH hparker@hmail.com'; ...
'Janice 529-882-1759 Fresno, CA jan_stephens@horizon.net'; ...
'Mike 793-136-0975 Richmond, VA sue_and_mike@hmail.net'; ...
'Nadine 648-427-9947 Tampa, FL nadine_berry@horizon.net'; ...
'Jason 697-336-7728 Montrose, CO jason_blake@mymail.com'};

```

此示例的第一部分构建一个表示标准电子邮件地址的格式的正则表达式。通过该表达式，此示例就可以在信息中搜索群组中一个朋友的电子邮件地址。Janice 的联系信息位于 **contacts** 元胞数组的第 2 行：

```
contacts{2}
```

```
ans =
'Janice 529-882-1759 Fresno, CA jan_stephens@horizon.net'
```

### 步骤 1 - 确定文本中的独特模式

典型的电子邮件地址包含以下标准部分：用户的帐户名称（后跟 @ 符号）、用户的 Internet 服务提供商 (ISP) 的名称、点（句点）以及该 ISP 所属的域。下表在左列中列出了这些部分，在右列中概述了每部分的格式。

电子邮件地址的独特模式	每种模式的一般说明
以帐户名称开头 jan_stephens ...	一个或多个小写字母和下划线
添加 '@' jan_stephens@ ...	@ 符号
添加 ISP jan_stephens@horizon ...	一个或多个小写字母，无下划线
添加点（句点） jan_stephens@horizon. ...	点（句点）字符
以域结尾 jan_stephens@horizon.net	com 或 net

### 步骤 2 - 将每种模式表示为正则表达式

在此步骤中，您将步骤 1 中得到的一般格式转换为正则表达式的段。然后，将这些段添加到一起以构成整个表达式。

下表在最左侧的列中显示了每种字符模式的广义格式说明。（这是从步骤 1 的表的右列中继承过来的。）第二列显示表示字符模式的运算符或元字符。

每个段的说明	模式
一个或多个小写字母和下划线	[a-z_] +
@ 符号	@
一个或多个小写字母，无下划线	[a-z] +
点（句点）字符	\.
com 或 net	(com   net)

将这些模式组合成一个字符向量可得到完整的表达式：

```
email = '[a-z_] +@[a-z] +\.(com|net);
```

### 步骤 3 - 调用合适的搜索函数

在此步骤中，您使用步骤 2 中得到的正则表达式来匹配群组中其中一个朋友的电子邮件地址。使用 regexp 函数执行搜索。

下面列出了此部分先前显示的联系信息。每个人的记录占用 contacts 元胞数组的一行：

```
contacts = {
'Harry 287-625-7315 Columbus, OH hparker@hmail.com'; ...
'Janice 529-882-1759 Fresno, CA jan_stephens@horizon.net'; ...}
```

```
'Mike 793-136-0975 Richmond, VA sue_and_mike@hmail.net'; ...
'Nadine 648-427-9947 Tampa, FL nadine_berry@horizon.net'; ...
'Jason 697-336-7728 Montrose, CO jason_blake@mymail.com');
```

以下是步骤 2 中得到的表示电子邮件地址的正则表达式：

```
email = '[a-z_]+@[a-z]+\.(com|net)';
```

调用 `regexp` 函数，并传递 `contacts` 元胞数组的第 2 行以及 `email` 正则表达式。这将返回 Janice 的电子邮件地址。

```
regexp(contacts{2}, email, 'match')
```

```
ans =
```

```
1×1 cell array
```

```
{'jan_stephens@horizon.net'}
```

MATLAB 从左至右解析字符串向量，并随着解析的进行“消减”该向量。如果找到匹配的字符，`regexp` 将会记录相应位置并继续解析字符串向量（仅从最新匹配项结尾之后开始）。

执行相同调用，但这次是针对列表中的第五个人进行调用：

```
regexp(contacts{5}, email, 'match')
```

```
ans =
```

```
1×1 cell array
```

```
{'jason_blake@mymail.com'}
```

您也可以使用整个元胞数组作为输入参数，以搜索列表中每个人的电子邮件地址：

```
regexp(contacts, email, 'match');
```

## 运算符和字符

正则表达式可包含用于指定要匹配的模式的字符、元字符、运算符、标文和标志，如下面各部分所述：

- “元字符”（第 2-52 页）
- “字符表示”（第 2-52 页）
- “限定符”（第 2-52 页）
- “分组运算符”（第 2-53 页）
- “定位点”（第 2-54 页）
- “环顾断言”（第 2-54 页）
- “逻辑和条件运算符”（第 2-54 页）
- “标文运算符”（第 2-55 页）
- “动态表达式”（第 2-56 页）
- “注释”（第 2-56 页）
- “搜索标志”（第 2-56 页）

## 元字符

元字符表示字母、字母范围、数字和空格字符。使用它们来构造广义的字符模式。

元字符	说明	示例
.	任何单个字符，包括空白	'..ain' 与以 'ain' 结尾的五个连续字符序列匹配。
[c <sub>1</sub> c <sub>2</sub> c <sub>3</sub> ]	包含在方括号中的任意字符。下列字符将按字面意义进行处理：\$   . * + ? 和 - (不用于指示范围时)。	'[rp.]ain' 与 'rain'、'pain' 或 '.ain' 匹配。
[^c <sub>1</sub> c <sub>2</sub> c <sub>3</sub> ]	未包含在方括号中的任意字符。下列字符将按字面意义进行处理：\$   . * + ? 和 - (不用于指示范围时)。	'[^*rp]ain' 与以 'ain' 结尾的所有由四个字母组成的序列 ('rain'、'pain' 和 '*ain' 除外) 匹配。例如，它与 'gain'、'lain' 或 'vain' 匹配。
[c <sub>1</sub> -c <sub>2</sub> ]	c <sub>1</sub> 到 c <sub>2</sub> 范围中的任意字符	'[A-G]' 与 A 到 G 范围中的单个字符匹配。
\w	任意字母、数字或下划线字符。对于英语字符集，\w 等同于 [a-zA-Z_0-9]	'\w*' 标识一个单词。
\W	字母、数字或下划线之外的任意字符。对于英语字符集，\W 等同于 [^a-zA-Z_0-9]	'\W*' 标识非单词项。
\s	任意空白字符；等同于 [\f\n\r\t\v]	'\w*\n\s' 与以字母 n 结尾且后跟空白字符的单词匹配。
\S	任意非空白字符；等同于 [^\f\n\r\t\v]	'\d\S' 与数字 (后跟任意非空白字符) 匹配。
\d	任意数字；等同于 [0-9]	'\d*' 与任意数量的连续数字匹配。
\D	任意非数字字符；等同于 [^0-9]	'\w*\D>' 与不以数字结尾的单词匹配。
\o{N} 或 \o{N}	八进制值 N 的字符	'\o{40}' 与八进制 40 定义的空格字符匹配。
\x{N} 或 \x{N}	十六进制值 N 的字符	'\x{2C}' 与十六进制 2C 定义的逗号字符匹配。

## 字符表示

运算符	说明
\a	警报 (蜂鸣)
\b	退格符
\f	换页符
\n	换行符
\r	回车符
\t	水平制表符
\v	垂直制表符
\char	正则表达式中您要从字面上匹配 (例如，使用 \\ 匹配单个反斜杠) 的具有特殊含义的任意字符。

## 限定符

限定符指定某个模式必须出现在匹配文本中的次数。

限定符	在出现以下次数时匹配表达式	示例
expr*	0 次或连续多次。	'\w*' 与任意长度的单词匹配。

限定符	在出现以下次数时匹配表达式	示例
<code>expr?</code>	0 次或 1 次。	'\w*(\..m)?' 与单词或以扩展名 .m 结尾（此条件为可选条件）的单词匹配。
<code>expr+</code>	1 次或连续多次。	'' 与 <img> HTML 标记匹配（当文件名包含一个或多个字符时）。
<code>expr{m,n}</code>	至少 m 次，但不超过连续 n 次。 <code>{0,1}</code> 等效于 ?。	'\S{4,8}' 与四到八个非空白字符匹配。
<code>expr{m,}</code>	至少连续 m 次。 <code>{0,}</code> 和 <code>{1,}</code> 分别等效于 * 和 +。	'<a href="\w{1,}\.html">' 与 <a> HTML 标记匹配（当文件名包含一个或多个字符时）。
<code>expr{n}</code>	恰好连续 n 次。 等效于 <code>{n,n}</code> 。	'\d{4}' 与四个连续数字匹配。

限定符可以以三种模式显示，如下表所述。q 表示上表中的任意限定符。

模式	说明	示例
<code>exprq</code>	积极表达式：与尽可能多的字符匹配。	给定文本 '<tr><td><p>text</p></td>'，表达式 '</?t.*>' 与介于 <tr> 和 <td> 之间的所有字符匹配： '<tr><td><p>text</p></td>'
<code>exprq?</code>	消极表达式：与所需的尽可能少的字符匹配。	给定文本 '<tr><td><p>text</p></td>'，表达式 '</?t.*?>' 在第一次出现右尖括号 (>) 时结束每个匹配项： '<tr>' '<td>' '</td>'
<code>exprq+</code>	主动表达式：最大程度地匹配，但不重新扫描文本的任何部分。	给定文本 '<tr><td><p>text</p></td>'，表达式 '</?t.*+>' 不返回任何匹配项，这是因为右尖括号是使用 .* 捕获的且不进行重新扫描。

## 分组运算符

分组运算符允许您捕获标文，将一个运算符应用于多个元素或在特定组中禁止追溯。

分组运算符	说明	示例
<code>(expr)</code>	将表达式元素分组并捕获标文。	'Joh?n\s(\w*)' 捕获一个标文，该标文包含名字为 John 或 Jon 的任何人的姓氏。
<code>(?:expr)</code>	分组但不捕获标文。	'(?:[aeiou][^aeiou])\{2\}' 与两个连续的元音后跟非元音（例如 'anon'）的模式匹配。 不进行分组时，'[aeiou][^aeiou]\{2\}' 与元音后跟两个非元音匹配。
<code>(?&gt;expr)</code>	以原子方式分组。不在组中追溯以完成匹配，并且不捕获标文。	'A(?>.* )Z' 与 'AtoZ' 不匹配，但 'A(?:.* )Z' 与其匹配。使用原子组时，Z 将使用 .* 进行捕获并且不进行重新扫描。

分组运算符	说明	示例
(expr1 expr2)	匹配表达式 expr1 或表达式 expr2。 如果存在与 expr1 匹配的项，则将忽略 expr2。 您可以在左括号后包括 ?: 或 ?> 以禁用标文或以原子方式分组。	'(let tel)\w+' 匹配以 let 或 tel 开头的单词。

## 定位点

表达式中的定位点与字符向量或单词的开头或结尾匹配。

定位点	与以下项匹配	示例
^expr	输入文本的开头。	'^M\w*' 与以 M 作为文本开头的单词匹配。
expr\$	输入文本的结尾。	'\w*m\$' 与以 m 作为文本结尾的单词匹配。
\<expr	单词开头。	'\<n\w*' 与以 n 开头的任何单词匹配。
expr\>	单词结尾。	'\w*\e\>' 与以 e 结尾的任何单词匹配。

## 环顾断言

环顾断言查找紧邻预期匹配项前后但并非该匹配项一部分的模式。

指针停留在当前位置，并且将放弃或不捕获对应于 test 表达式的字符。因此，前向断言可匹配重叠字符组。

环顾断言	说明	示例
expr(?=test)	向前查找与 test 匹配的字符。	'\w*(?=ing)' 匹配后跟 ing 的词汇，例如输入文本 'Flying, not falling.' 中的 'Fly' 和 'fall'。
expr(?!=test)	向前查找与 test 不匹配的字符。	'i(?!=ng)' 匹配字母 i 的不后跟 ng 的实例。
(?<=test)expr	向后查找与 test 匹配的字符。	'(?<=re)\w*' 匹配紧跟 're' 的词汇，例如输入文本 'renew, reuse, recycle' 中的 'new'、'use' 和 'cycle'
(?<!test)expr	向后查找与 test 不匹配的字符。	'(?<!\\d)(\\d)(?!\\d)' 与一位数字匹配（不紧随其他数字前后的数字）。

如果您在表达式之前指定前向断言，则运算等同于逻辑 AND。

运算	说明	示例
(?=test)expr	同时与 test 和 expr 匹配。	'(?=[a-z])[^aeiou]' 与辅音匹配。
(?!=test)expr	匹配 expr，但不匹配 test。	'(?![aeiou])[a-z]' 与辅音匹配。

有关详细信息，请参阅“正则表达式中的前向断言”（第 2-58 页）。

## 逻辑和条件运算符

逻辑和条件运算符允许您测试给定条件的状态，然后使用结果确定哪个模式（如果有）与下一条件匹配。这些运算符支持逻辑 OR、if 或 if/else 条件。（对于 AND 条件，请参阅“环顾断言”（第 2-54 页）。）

条件可以是标文（第 2-55 页）、环顾断言（第 2-54 页）或 (@cmd) 形式的动态表达式（第 2-56 页）。动态表达式必须返回逻辑值或数值。

条件运算符	说明	示例
<code>expr1   expr2</code>	匹配表达式 <code>expr1</code> 或表达式 <code>expr2</code> 。 如果存在与 <code>expr1</code> 匹配的项，则将忽略 <code>expr2</code> 。	'(let   tel)\w+' 匹配以 let 或 tel 开头的单词。
<code>(?(cond)expr)</code>	如果条件 <code>cond</code> 为 true，则匹配 <code>expr</code> 。	'(?(?(@iscpc)[A-Z]:\\))' 匹配驱动器名称，例如 C:\ (在 Windows 系统上运行时)。
<code>(?(cond)expr1   expr2)</code>	如果条件 <code>cond</code> 为 true，则匹配 <code>expr1</code> 。否则，匹配 <code>expr2</code> 。	'Mr(s?)\..*?(?(1)her   his) \w*' 匹配包含 her 的文本（当文本以 Mrs 开头时），或包含 his 的文本（当文本以 Mr 开头时）。

### 标文运算符

标文是您通过将正则表达式部分括在括号中而定义的匹配文本的部分。您可以按标文在文本中的顺序引用该标文（顺序标文），或将名称分配给标文以便于代码维护和使输出更易于阅读。

顺序标文运算符	说明	示例
<code>(expr)</code>	在标文中捕获与括起来的表达式匹配的字符。	'John\n\s(\w*)' 捕获一个标文，该标文包含名字为 John 或 Jon 的任何人的姓氏。
<code>\N</code>	匹配第 N 个标文。	'<(\w+).*>.*</\1>' 从文本 '<title>Some text</title>' 捕获 HTML 标记的标文，例如 'title'。
<code>(?(N)expr1   expr2)</code>	如果找到第 N 个标文，则匹配 <code>expr1</code> 。否则，匹配 <code>expr2</code> 。	'Mr(s?)\..*?(?(1)her   his) \w*' 匹配包含 her 的文本（当文本以 Mrs 开头时），或包含 his 的文本（当文本以 Mr 开头时）。

命名标文运算符	说明	示例
<code>(?&lt;name&gt;expr)</code>	在命名标文中捕获与括起来的表达式匹配的字符。	'(?<month>\d+)-(?<day>\d+)-(?<yr>\d+)' 在 mm-dd-yy 形式的输入日期中创建命名月、日和年标文。
<code>\k&lt;name&gt;</code>	匹配 name 引用的标文。	'<(?<tag>\w+).*>.*</\k<tag>>' 从文本 '<title>Some text</title>' 捕获 HTML 标记的标文，例如 'title'。
<code>(?(name)expr1   expr2)</code>	如果找到命名标文，则匹配 <code>expr1</code> 。否则，匹配 <code>expr2</code> 。	'Mr(?<sex>s?)\..*?(?(sex)her   his) \w*' 匹配包含 her 的文本（当文本以 Mrs 开头时），或包含 his 的文本（当文本以 Mr 开头时）。

**注意** 如果表达式具有嵌套括号，则 MATLAB 捕获对应于最外层括号的标文。例如，给定搜索模式 '(and(y | rew))'，MATLAB 将为 'andrew' 但不为 'y' 或 'rew' 创建一个标文。

有关详细信息，请参阅“正则表达式中的标文”（第 2-61 页）。

### 动态表达式

动态表达式允许您执行 MATLAB 命令或正则表达式以确定要匹配的文本。

将动态表达式括起来的括号不创建捕获组。

运算符	说明	示例
(??expr)	解析 <code>expr</code> 并将得到的项包括在匹配表达式中。 解析后， <code>expr</code> 必须对应于完整的效果正则表达式。使用反斜杠转义字符 (\) 的动态表达式需要两个反斜杠：一个用于 <code>expr</code> 的初始解析，一个用于完整匹配。	'^(\\d+)((??\\w\{\$1}))' 通过读取匹配项开头的数字确定匹配的字符数。动态表达式括在另一组括号中，以便在标文中捕获生成的匹配项。例如，匹配 '5XXXXX' 将捕获 '5' 和 'XXXXX' 的标文。
(??@cmd)	执行 <code>cmd</code> 表示的 MATLAB 命令，并将该命令返回的输出包括在匹配表达式中。	'(.{2,}).?(??@fliplr(\$1))' 查找长度至少为四个字符的回文，例如 'abba'。
(?@cmd)	执行 <code>cmd</code> 表示的 MATLAB 命令，但放弃该命令返回的任何输出。（对诊断正则表达式有帮助。）	'\\w*?(\\w)(?@disp(\$1))\\1\\w*' 匹配包括双字母（例如 pp）的单词并显示中间结果。

在动态表达式中，使用下列运算符定义替代项。

替代运算符	说明
\$& 或 \$0	当前作为匹配项的输入文本部分
\$`	位于当前匹配项之前的输入文本部分
\$'	紧随当前匹配项的输入文本部分（使用 \$" 表示 \$'）
\$N	第 N 个标文
\$<name>	命名标文
\${cmd}	在 MATLAB 执行命令 <code>cmd</code> 时返回的输出

有关详细信息，请参阅“动态正则表达式”（第 2-67 页）。

### 注释

通过 `comment` 运算符，可以在代码中插入注释，以使代码更易于维护。根据输入文本进行匹配时，MATLAB 会忽略注释的文本。

字符	说明	示例
(?#comment)	在正则表达式中插入注释。匹配输入时将忽略注释文本。	'(?# Initial digit)\\<\\d\\w+' 包括一个注释，并匹配以一个数字开头的单词。

### 搜索标志

搜索标志修改匹配表达式的行为。

标志	说明
(?-i)	匹配字母大小写（ <code>regexp</code> 和 <code>regexp替</code> 的默认值）。

标志	说明
(?i)	不匹配字母大小写（ <code>regexpi</code> 的默认值）。
(?s)	将模式中的点（.）与任意字符匹配（默认值）。
(?-s)	将模式中的点与并非换行符的任意字符匹配。
(?-m)	匹配文本开头和结尾的 ^ 和 \$ 元字符（默认值）。
(?m)	匹配行开头和结尾的 ^ 和 \$ 元字符。
(?-x)	在匹配时包括空格字符和注释（默认值）。
(?x)	在匹配时忽略空格字符和注释。使用 '\' 和 '\#' 匹配空格和 # 字符。

该标志修改的表达式可显示在括号后，例如

`(?i)\w*`

或显示在括号内并使用冒号（:）与该标志分隔开，例如

`(?i:\w*)`

后面的语法允许您更改较大表达式的一部分的行为。

## 另请参阅

`regexp` | `regexpi` | `regexpexpr` | `regexptranslate`

## 详细信息

- “正则表达式中的前向断言”（第 2-58 页）
- “正则表达式中的标文”（第 2-61 页）
- “动态正则表达式”（第 2-67 页）

## 正则表达式中的前向断言

### 本节内容

- “前向断言”（第 2-58 页）
- “重叠匹配项”（第 2-58 页）
- “逻辑 AND 条件”（第 2-59 页）

### 前向断言

正则表达式有两种类型的环顾断言：前向断言和后向断言。在这两种情况下，断言是一个必须满足才能返回表达式匹配项的条件。

前向断言的形式为 `(?=test)`，可以显示在正则表达式中的任意位置。MATLAB 从文本的当前位置向前查找测试条件。如果 MATLAB 与测试条件匹配，它将会继续处理表达式的其余部分以找到匹配项。

例如，在指定路径的字符向量中向前查找以找到包含程序文件（此示例中为 `fileread.m`）的文件夹的名称。

```
chr = which('fileread')
chr =
'matlabroot\toolbox\matlab\iofun\fileread.m'
regexp(chr,'w+(?=\w+\.[mp])','match')
ans =
1×1 cell array
{'iofun'}
```

匹配表达式 `\w+` 会搜索一个或多个字母数字或下划线字符。每次 `regexp` 找到与此条件匹配的项时，它都会向前查找反斜杠（通过两个反斜杠 `\`` 指定），后跟一个带有 `.m` 或 `.p` 扩展名 (`\.[mp]`) 的文件名 (`\w+`)。`regexp` 函数返回符合前向条件的匹配项，即文件夹名称 `iofun`。

### 重叠匹配项

前向断言不消减文本中的任何字符。因此，您可以使用它们来查找重叠的字符序列。

例如，使用前向搜索并通过匹配后跟五个其他字符的初始字符，在字符向量中查找每一个由六个非空白字符组成的序列：

```
chr = 'Locate several 6-char. phrases';
startIndex = regexpi(chr,'^S(?\=\S{5})')
startIndex =
1   8   9   16  17  24  25
起始索引与以下短语对应：
Locate severa everal 6-char -char. phrase hrases
```

如果没有前向运算符，MATLAB 将从左至右解析字符向量，并在解析过程中消减该向量。如果找到匹配的字符，**regexp** 将会记录相应位置并从最新匹配项的位置开始继续解析字符向量。此过程不存在字符重叠。

```
chr = 'Locate several 6-char. phrases';
startIndex = regexpi(chr,'^S{6}')
```

startIndex =

1 8 16 24

起始索引与以下短语对应：

Locate severa 6-char phrase

## 逻辑 AND 条件

使用前向运算的另一种方式是在两个条件之间执行逻辑 AND。此示例最初会尝试找到字符数组（由 **normest** 函数帮助的前 50 个字符组成）中的所有小写辅音字母。

```
helptext = help('normest');
chr = helptext(1:50)
```

chr =

```
' NORMEST Estimate the matrix 2-norm.
NORMEST(S'
```

仅仅搜索非元音字母 (`[^aeiou]`) 不会返回预期的答案，因为输出包含大写字母、空格字符和标点符号：

```
c = regexp(chr,['^aeiou'],'match')
```

c =

1×43 cell array

Columns 1 through 14

```
{ } {N} {O} {R} {M} {E} {S} {T} { } {E} {s} {t} {m} {t}
```

Columns 15 through 28

```
{ } {t} {h} { } {m} {t} {r} {x} { } {2} {-} {n} {r} {m}
```

Columns 29 through 42

```
{ } {-} { } { } { } {N} {O} {R} {M} {E} {S} {T} { }
```

Column 43

```
{S}
```

重新尝试此操作，并使用前向运算符创建以下 AND 条件：

**(lowercase letter) AND (not a vowel)**

这一次的结果是正确的：

```
c = regexp(chr,'(?=[a-z])[^aeiou]','match')  
c =  
1×13 cell array  
's' 't' 'm' 't' 't' 'h' 'm' 't' 'r' 'x' 'n' 'r' 'm'
```

请注意，使用前向运算符执行 AND 时，您需要将匹配表达式 `expr` 放在测试表达式 `test` 的后面：

`(?=test)expr or (?!=test)expr`

### 另请参阅

`regexp` | `regexpi` | `regexpexpr`

### 详细信息

- “正则表达式”（第 2-48 页）

# 正则表达式中的标文

## 本节内容

- “简介”（第 2-61 页）
- “多个标文”（第 2-63 页）
- “不匹配的标文”（第 2-63 页）
- “替代文本中的标文”（第 2-64 页）
- “命名捕获”（第 2-64 页）

## 简介

正则表达式中使用的括号不仅将该表达式的元素分组在一起，而且会为找到的与该组条件匹配的任何匹配项指定标文。您可以使用标文匹配同一文本的其他部分。使用标文的一个好处是，标文会记住所匹配的内容，因此您可以在搜索或替换过程中重新调用和重用匹配的文本。

表达式中的每个标文都被分配一个数字，从 1 开始，从左至右顺序递增。要在表达式的后面引用某个标文，请使用反斜杠并后跟标文编号进行引用。例如，引用表达式中第三组括号生成的标文时，请使用 \3。

举一个简单的例子，如果您要搜索字符数组中相同顺序的字母，可以捕获第一个字母作为标文，然后搜索紧随其后的匹配字符。在以下所示的表达式中，只要 regexp 与字符数组中的任何非空白字符匹配，(\S) 短语就会创建一个标文。表达式的第二部分 '\1' 查找同一字符第一个实例后面紧跟的第二个实例。

```
poe = ['While I nodded, nearly napping, ' ...
'suddenly there came a tapping,'];

[mat,tok,ext] = regexp(poe, '(\S)\1', 'match',...
    'tokens','tokenExtents');
mat

mat =
1×4 cell array
{'dd'}  {'pp'}  {'dd'}  {'pp'}
```

元胞数组 tok 包含多个元胞数组，其中每个元胞数组都包含一个标文。

```
tok{};

ans =
1×1 cell array
{'d'};

ans =
1×1 cell array
{'p'}
```

```
ans =
1×1 cell array
{'d'}
```

```
ans =
1×1 cell array
{'p'}
```

元胞数组 `ext` 包含多个数值数组，其中每个数值数组都包含一个标文的开始索引和结束索引。

```
ext{:}
ans =
11 11
```

```
ans =
26 26
```

```
ans =
35 35
```

```
ans =
57 57
```

另一个示例捕获匹配的 HTML 标记对组（例如 `<a>` 和 `</a>`）以及这些标记之间的文本。用于此示例的表达式为

```
expr = '<(\w+).*?>.*?</\1>';
```

该表达式的第一部分 '`<(\w+)`' 匹配左尖括号 (`<`)，后跟一个或多个字母、数字或下划线字符。封闭的圆括号捕获左尖括号后面的标文字符。

该表达式的第二部分 '`.*?>.*?`' 匹配此 HTML 标记的其余部分（一直到 `>` 的字符），以及可能位于下一个左尖括号之前的任何字符。

最后一部分 '`</\1>`' 匹配结尾的 HTML 标记中的所有字符。此标记由 `</tag>` 序列组成，其中 `tag` 表示捕获为标文的任何字符。

```
hstr = '<!comment><a name="752507"></a><b>Default</b><br>';
expr = '<(\w+).*?>.*?</\1>';

[mat,tok] = regexp(hstr, expr, 'match', 'tokens');
mat{:,}
```

```
ans =
```

```
'<a name="752507"></a>'
```

```
ans =
'<b>Default</b>'

tok{:}

ans =
1×1 cell array

{'a'}
```

```
ans =
1×1 cell array

{'b'}
```

## 多个标文

下面是一个如何为标文分配值的示例。假定您要搜索以下文本：

**andy ted bob jim andrew andy ted mark**

您选择使用以下搜索模式来搜索上述文本：

**and(y|rew)|(t)e(d)**

此模式包含三个生成标文的带括号表达式。当您最后执行搜索时，系统将为每个匹配项生成以下标文。

匹配项	标文 1	标文 2
<b>andy</b>	y	
<b>ted</b>	t	d
<b>andrew</b>	rew	
<b>andy</b>	y	
<b>ted</b>	t	d

只会使用最高级别的括号。例如，如果搜索模式 **and(y|rew)** 找到文本 **andrew**，则会为标记 1 分配值 **rew**。但是，如果使用搜索模式 **(and(y|rew))**，则会为标记 1 分配值 **andrew**。

## 不匹配的标文

对于正则表达式中指定的标文，如果在待查文本中没有匹配项，**regexp** 和 **regexpI** 会返回一个空字符串向量 ("") 作为标文输出，同时还会返回一个范围，用来标记标文在字符串中预期出现的位置。

此处所示的示例会对一个字符串执行 **regexp**，该字符串指定了从 MATLAB **tempdir** 函数返回的路径。正则表达式 **expr** 包括六个标文设定符，每个设定符代表路径的一部分。第三个设定符 **[a-z]+** 在字符串中没有匹配项，因为此部分的路径 **Profiles** 是以大写字母开头的：

```
chr = tempdir
```

```
chr =
'C:\WINNT\Profiles\bpascal\LOCALS~1\Temp\'  

expr = [([A-Z]:)\((WINNT)\)([a-z]+)?.*\| ...
'([a-z]+)\([A-Z]+\d)\((Temp)\)\|';  

[tok, ext] = regexp(chr, expr, 'tokens', 'tokenExtents');
```

如果在文本中未找到标文, `regexp` 会返回一个空字符向量 ('') 作为标文以及一个包含标文范围的数值数组。而范围的第一个数字为字符串索引, 用于标记标文预期出现的位置; 范围的第二个数字为第一个数字减 1。

在此示例中, 表达式中指定的第三个标文为空标文, 因此返回的第三个标文为空:

```
tok{:}  

ans =  

1×6 cell array  

{'C:'}  {'WINNT'}  {0×0 char}  {'bpascal'}  {'LOCALS~1'}  {'Temp'}
```

变量 `ext` 中返回的第三个标文范围的起始索引设置为 10, 这是不含匹配项的词 `Profiles` 在路径中的起始位置。范围结束索引设置为起始索引减 1, 也就是 9:

```
ext{:}  

ans =  

1   2  

4   8  

10  9  

19  25  

27  34  

36  39
```

## 替代文本中的标文

在替代文本中使用标文时, 请使用 `$1`、`$2` 等符号, 而不要使用 `\1`、`\2` 等符号来引用标文。此示例捕获两个标文并颠倒它们的顺序。第一个标文 `$1` 为 '`Norma Jean`', 第二个标文 `$2` 为 '`Baker`'。请注意, `regexp替`换返回修改后的文本, 而不是起始索引的向量。

```
regexp替('Norma Jean Baker', '(\w+\s\w+)\s(\w+)', '$2, $1')
ans =
'Baker, Norma Jean'
```

## 命名捕获

如果您在表达式中使用许多标文, 则为这些标文分配名称可能会很有帮助, 而不必跟踪为哪个标文分配了哪个标文编号。

引用表达式中的命名标文时, 请使用语法 `\k<name>`, 而不要使用 `\1`、`\2` 等数字:

```
poe = ['While I nodded, nearly napping, ' ...
'suddenly there came a tapping,'];
```

```
regexp(poe, '(?<anychar>.)\k<anychar>', 'match')
```

```
ans =
```

```
1×4 cell array
```

```
{'dd'} {'pp'} {'dd'} {'pp'}
```

命名标文还有助于标记 MATLAB 正则表达式函数的输出。当您处理许多文本段时尤其如此。

例如，从多个字符向量中解析街道地址的不同部分。为表达式中的每个标文分配了一个短名称：

```
chr1 = '134 Main Street, Boulder, CO, 14923';
chr2 = '26 Walnut Road, Topeka, KA, 25384';
chr3 = '847 Industrial Drive, Elizabeth, NJ, 73548';
```

```
p1 = '(?<adrs>\d+\s\S+\s(Road|Street|Avenue|Drive))';
p2 = '(?<city>[A-Z][a-z]+)';
p3 = '(?<state>[A-Z]{2})';
p4 = '(?<zip>\d{5})';
```

```
expr = [p1 ' ' p2 ' ' p3 ' ' p4];
```

正如以下结果所展示的，您可以通过命名标文使输出更易于使用：

```
loc1 = regexp(chr1, expr, 'names')
```

```
loc1 =
```

```
struct with fields:
```

```
adrs: '134 Main Street'
city: 'Boulder'
state: 'CO'
zip: '14923'
```

```
loc2 = regexp(chr2, expr, 'names')
```

```
loc2 =
```

```
struct with fields:
```

```
adrs: '26 Walnut Road'
city: 'Topeka'
state: 'KA'
zip: '25384'
```

```
loc3 = regexp(chr3, expr, 'names')
```

```
loc3 =
```

```
struct with fields:
```

```
adrs: '847 Industrial Drive'
city: 'Elizabeth'
state: 'NJ'
zip: '73548'
```

**另请参阅**

`regexp | regexpi | regexpexpr`

**详细信息**

- “正则表达式”（第 2-48 页）

# 动态正则表达式

本节内容
“简介” (第 2-67 页)
“动态匹配表达式 - (??expr)” (第 2-68 页)
“修改匹配表达式的命令 - (??@cmd)” (第 2-68 页)
“满足功能性需求的命令 - (@cmd)” (第 2-69 页)
“替代表达式中的命令 - \${cmd}” (第 2-71 页)

## 简介

在动态表达式中，您可以要求 `regexp` 进行匹配的模式随输入文本的内容而动态变化。通过这种方式，您可以更紧密地匹配所解析的文本中的不同输入模式。此外，也可以在替代项中使用动态表达式以用于 `regexprep` 函数。这样，您就有办法令替代文本更好地适应所解析的输入。

您可以在以下命令的 `match_expr` 或 `replace_expr` 参数中包含任意数量的动态表达式：

```
regexp(text, match_expr)
regexpi(text, match_expr)
regexprep(text, match_expr, replace_expr)
```

以一个动态表达式为例，下面的 `regexprep` 命令将术语 `internationalization` 正确替换为其缩写形式 `i18n`。但是，要对其他术语（例如 `globalization`）使用该命令，您必须使用一个不同的替代表达式：

```
match_expr = '(^\w)(\w*)(\w$)';
replace_expr1 = '$118$3';
regexprep('internationalization', match_expr, replace_expr1)

ans =
'i18n'

replace_expr2 = '$111$3';
regexprep('globalization', match_expr, replace_expr2)

ans =
'g11n'
```

通过动态表达式  `${num2str(length($2))}`，您可以将替代表达式建立在输入文本基础之上，这样您就不必每次更改该表达式。此示例使用动态替代语法  `${cmd}`。

```
match_expr = '(^\w)(\w*)(\w$)';
replace_expr = '$1${num2str(length($2))}$3';

regexprep('internationalization', match_expr, replace_expr)

ans =
'i18n'

regexprep('globalization', match_expr, replace_expr)
```

```
ans =
```

```
'g11n'
```

解析后，动态表达式必须与完整的有效正则表达式对应。此外，使用反斜杠转义字符 (\) 的动态匹配表达式还需要两个反斜杠：一个用于表达式的初始解析，一个用于完整匹配。将动态表达式括起来的括号不创建捕获组。

有三种形式的动态表达式可用作匹配表达式，还有一种形式的动态表达式可用作替代表达式，下面分别对这几种动态表达式进行了介绍。

## 动态匹配表达式 - (??expr)

(??expr) 运算符解析表达式 expr，并将结果插回到匹配表达式中。然后，MATLAB 会计算修改后的匹配表达式。

下面是一个使用此运算符的表达式类型的示例：

```
chr = {'5XXXXXX', '8XXXXXXXXX', '1X'};
regexp(chr, '^(\d+)(??X{$1})$', 'match', 'once');
```

此特殊命令的目的是在输入元胞数组存储的每个字符向量中定位一系列连续的 X 字符。但是请注意，X 的数量在每个字符向量中有所不同。如果数量没有变化，您可以使用表达式 X{n} 指示要匹配这些字符中的 n 个字符。但是，n 为常量值并不适用于此示例。

这里使用的解决方法是：捕获标文中的前导计数（例如元胞数组的第一个字符向量中的 5），然后在动态表达式中使用该计数。此示例中的动态表达式为 (??X{\$1})，其中 \$1 是标文 \d+ 捕获的值。运算符 {\$1} 为该标文值创建限定符。由于该表达式是动态的，因此同一模式适用于元胞数组中的所有三个输入向量。对于第一个输入字符向量，regexp 查找五个 X 字符；对于第二个输入字符串，该命令查找八个字符，而对于第三个输入字符串，仅查找一个字符：

```
regexp(chr, '^(\d+)(??X{$1})$', 'match', 'once')
```

```
ans =
```

```
1×3 cell array
```

```
{'5XXXXXX'}  {'8XXXXXXXXX'}  {'1X'}
```

## 修改匹配表达式的命令 - (??@cmd)

MATLAB 使用 (??@cmd) 运算符将 MATLAB 命令的结果纳入匹配表达式中。此命令必须返回可在匹配表达式中使用的项。

例如，使用动态表达式 (??@fliplr(\$1)) 查找嵌入到较大的字符向量中的回文 “Never Odd or Even” 。

首先，创建输入字符串。确保所有字母都是小写字母，并删除所有非单词字符。

```
chr = lower(...  
'Find the palindrome Never Odd or Even in this string');  
  
chr = regexprep(chr, '\W*', '')  
  
chr =  
'findthepalindromeneveroddoreveninthisstring'
```

使用以下动态表达式查找字符串中的回文：

```
palindrome = regexp(chr, '(.{3,}).?(??@fliplr($1))', 'match')
palindrome =
1×1 cell array
{'neveroddoreven'}
```

该动态表达式颠倒构成字符串的字母的顺序，然后尝试匹配尽可能多的逆序字符串。这需要一个动态表达式，因为 \$1 的值依赖于标文 (.{}3,) 的值。

MATLAB 中的动态表达式有权访问当前活动的工作区。这意味着，只需更改工作区中的变量，即可更改动态表达式中使用的任何函数或变量。重复执行以上示例的最后一个命令，但这次使用基础工作区中存储的函数句柄定义要在表达式中调用的函数：

```
fun = @fliplr;
palindrome = regexp(chr, '(.{3,}).?(??@fun($1))', 'match')
palindrome =
1×1 cell array
{'neveroddoreven'}
```

## 满足功能性需求的命令 - (?@cmd)

(?@cmd) 运算符用于指定 `regexp` 或 `regexpexpr` 要在解析整个匹配表达式时运行的 MATLAB 命令。与 MATLAB 中的其他动态表达式不同，此运算符不会更改其所在的表达式的内容。相反，您可以使用此功能让 MATLAB 仅报告在解析您的其中一个正则表达式的内容时所采取的步骤。此功能可用于诊断您的正则表达式。

以下示例解析由零个或多个字符后跟两个相同字符再后跟零个或多个字符组成的单词：

```
regexp('mississippi', '\w*(\w)\1\w*', 'match')
ans =
1×1 cell array
{'mississippi'}
```

为跟踪 MATLAB 在确定匹配项时采取的确切步骤，此示例在表达式中插入一个简短脚本 (?@`disp($1)`)，以显示最终构成匹配项的字符。由于此示例使用积极限制符，因此 MATLAB 尝试匹配尽可能多的字符串。这样，即使 MATLAB 在字符串的开头处找到匹配项，它也会继续查找更多匹配项，直至到达字符串的最末尾处。从该处开始，它会备份从字母 i 到 p 以及接下来的 p，并停止在该位置，因为匹配项最终符合要求：

```
regexp('mississippi', '\w*(\w)(?@disp($1))\1\w*', 'match')
i
p
p
ans =
```

```
1×1 cell array
```

```
{'mississippi'}
```

现在，重新尝试同一示例，这一次将第一个限定符设置为消极限定符 (\*?)。同样，MATLAB 生成相同的匹配项：

```
regexp('mississippi', '\w*?(\\w)\\1\\w*', 'match')
```

```
ans =
```

```
1×1 cell array
```

```
{'mississippi'}
```

但是，通过插入动态脚本，这一次可以查看匹配项，MATLAB 以完全不同的方式与文本进行了匹配。在本例中，MATLAB 使用可找到的第一个匹配项，甚至不会考虑文本的其余部分：

```
regexp('mississippi', '\w*?(\\w)(?@disp($1))\\1\\w*', 'match')
```

```
m  
i  
s
```

```
ans =
```

```
1×1 cell array
```

```
{'mississippi'}
```

为了演示此类型的动态表达式的灵活性，请尝试下面的示例。当 MATLAB 以迭代方式解析输入文本时，该示例会逐步设置一个元胞数组。在表达式结尾找到的 (?)! 运算符实际上是一个空的前向运算符，该运算符强制在每次迭代时失败。如果您要跟踪 MATLAB 在处理表达式时采取的步骤，则这种强制失败是必要的。

MATLAB 通过输入文本进行多次传递，每次都尝试另一个字母组合，以查看能否找到比上一个匹配项更好的匹配项。在未找到匹配项的任何传递中，测试将生成一个空字符向量。动态脚本 (?@if(~isempty(\$&))) 用于省略 matches 元胞数组中的空字符向量：

```
matches = {};  
expr = [(Euler\\s)?(Cauchy\\s)?(Boole)?(?@if(~isempty($&)), ' ...  
'matches{end+1}=$;&;end)(?!)];  
  
regexp('Euler Cauchy Boole', expr);  
  
matches  
  
matches =  
  
1×6 cell array  
  
{'Euler Cauchy Bo...'} {'Euler Cauchy '} {'Euler '} {'Cauchy Boole'} {'Cauchy '} {'Boole'}
```

运算符 \$& (或与之等效的 \$0)、\$` 和 \$' 分别指代输入文本中的当前匹配项部分、当前匹配项前面的所有字符以及当前匹配项后面的所有字符。在处理动态表达式 (尤其是使用 (?@cmd) 运算符的动态表达式) 时，这些运算符有时会很有用。

下面的示例解析输入文本以查找字母 g。在每次迭代扫描该文本时，`regexp` 都会将当前字符与 g 进行比较，如果没有找到匹配项，则会前进到下一个字符。此示例通过 ^ 字符标记要解析的当前位置，跟踪在该文本中扫描的进度。

(\$` 和 \$` 运算符捕获位于当前解析位置之前和之后的文本部分。当序列 \$` 出现在文本中时，您需要使用两个单引号 (\$") 来表示它。)

```
chr = 'abcdefghijkl';
expr = '(?@disp(sprintf("starting match: [%s^%s]",$,,$"))))g';

regexp(chr, expr, 'once');

starting match: [^abcdefghijkl]
starting match: [a^abcdefghijkl]
starting match: [ab^abcdefghijkl]
starting match: [abc^abcdefghijkl]
starting match: [abcd^efghij]
starting match: [abcde^fghij]
starting match: [abcdef^ghij]
```

## 替表达式中的命令 - \${cmd}

`${cmd}` 运算符修改正则表达式替代模式的内容，使得该模式适用于输入文本中可能因用法而异的参数。与 MATLAB 中使用的其他动态表达式一样，您也可以在整个替表达式中包含任意数量的这些表达式。

在此处显示的 `regexprep` 调用中，替代模式为 ' `${convertMe($1,$2)}`'。在本例中，整个替代模式是一个动态表达式：

```
regexprep('This highway is 125 miles long', ...
    '(\d+\.\?\d*)\W(\w+)', '${convertMe($1,$2)}');
```

该动态表达式指示 MATLAB 使用两个派生自所匹配的文本的标文 `(\d+\.\?\d*)` 和 `(\w+)`，作为 `convertMe` 调用的参数，执行一个名为 `convertMe` 的函数。由于 `$1` 和 `$2` 的值是在运行时生成的，因此替代模式需要动态表达式。

以下示例定义了一个名为 `convertMe` 的函数，该函数将测量值从英制单位转换为公制单位。

```
function valout = convertMe(valin, units)
switch(units)
    case 'inches'
        fun = @(in)in .* 2.54;
        uout = 'centimeters';
    case 'miles'
        fun = @(mi)mi .* 1.6093;
        uout = 'kilometers';
    case 'pounds'
        fun = @(lb)lb .* 0.4536;
        uout = 'kilograms';
    case 'pints'
        fun = @(pt)pt .* 0.4731;
        uout = 'litres';
    case 'ounces'
        fun = @(oz)oz .* 28.35;
        uout = 'grams';
end
val = fun(str2num(valin));
```

```
valout = [num2str(val) ' ' uout];
end
```

在命令行上，通过 `regexprep` 调用 `convertMe` 函数，并传入要转换的数量值和英制单位名称：

```
regexprep('This highway is 125 miles long', ...
    '(\d+\.?\d*)\W(\w+)', '${convertMe($1,$2)})')

ans =

'This highway is 201.1625 kilometers long'

regexprep('This pitcher holds 2.5 pints of water', ...
    '(\d+\.?\d*)\W(\w+)', '${convertMe($1,$2)})')

ans =

'This pitcher holds 1.1828 litres of water'

regexprep('This stone weighs about 10 pounds', ...
    '(\d+\.?\d*)\W(\w+)', '${convertMe($1,$2)})')

ans =

'This stone weighs about 4.536 kilograms'
```

与先前部分中讨论的 `(??@)` 运算符一样，`{}$` 运算符有权访问当前活动的工作区中的变量。以下 `regexprep` 命令使用基础工作区中定义的数组 `A`：

```
A = magic(3)

A =

 8   1   6
 3   5   7
 4   9   2

regexprep('The columns of matrix _nam are _val', ...
    {'_nam', '_val'}, ...
    {'A', '${sprintf("%d%d%d ", A)}'})

ans =

'The columns of matrix A are 834 159 672'
```

### 另请参阅

`regexp` | `regexpI` | `regexprep`

### 详细信息

- “正则表达式”（第 2-48 页）

# 逗号分隔的列表

## 本节内容

- “什么是逗号分隔的列表？”（第 2-73 页）
- “生成逗号分隔的列表”（第 2-73 页）
- “从逗号分隔的列表分配输出”（第 2-75 页）
- “为逗号分隔的列表赋值”（第 2-75 页）
- “如何使用逗号分隔的列表”（第 2-76 页）
- “快速傅里叶变换示例”（第 2-78 页）

## 什么是逗号分隔的列表？

键入一系列逗号分隔的数字，即为所谓的逗号分隔的列表。MATLAB 逐一返回每个值：

```
1,2,3
```

```
ans =
```

```
1
```

```
ans =
```

```
2
```

```
ans =
```

```
3
```

此类列表本身并不是非常有用。但是，当与更复杂的大型数据结构体（如 MATLAB 结构体）和元胞数组一起使用时，逗号分隔的列表能够帮助您简化 MATLAB 代码。

## 生成逗号分隔的列表

此部分介绍如何从元胞数组或 MATLAB 结构体生成逗号分隔的列表。

### 从元胞数组生成列表

从元胞数组提取多个元素即可得到一个逗号分隔的列表。以下面所示的 4x6 元胞数组为例

```
C = cell(4,6);
for k = 1:24
    C{k} = k*2;
end
C =
[2]  [10]  [18]  [26]  [34]  [42]
[4]  [12]  [20]  [28]  [36]  [44]
[6]  [14]  [22]  [30]  [38]  [46]
[8]  [16]  [24]  [32]  [40]  [48]
```

提取第五列将生成以下逗号分隔的列表：

```
C{:,:5}
```

```
ans =
```

```
34
```

```
ans =
```

```
36
```

```
ans =
```

```
38
```

```
ans =
```

```
40
```

这一结果与显式键入以下内容所得到的列表相同

```
C{1,5},C{2,5},C{3,5},C{4,5}
```

### 从结构体生成列表

对于结构体，提取结构体在某一维度上存在的字段即可得到一个逗号分隔的列表。

首先将前面使用的元胞数组转换为一个带有六个字段的 4x1 MATLAB 结构体，这六个字段为：f1 到 f6。读取所有行的字段 f5，MATLAB 会返回一个字段分隔的列表：

```
S = cell2struct(C,{['f1','f2','f3','f4','f5','f6'],2);  
S.f5
```

```
ans =
```

```
34
```

```
ans =
```

```
36
```

```
ans =
```

```
38
```

```
ans =
```

```
40
```

这一结果与显式键入以下内容所得到的列表相同

```
S(1).f5,S(2).f5,S(3).f5,S(4).f5
```

## 从逗号分隔的列表分配输出

您可以通过一个简单的赋值语句将逗号分隔列表的任何或所有连续元素分配给变量。使用前一部分中的元胞数组 C，将第一行分配给 c1 到 c6 变量：

```
C = cell(4,6);
for k = 1:24
    C{k} = k*2;
end
[c1,c2,c3,c4,c5,c6] = C{1,1:6};
c5

c5 =
34
```

如果您指定的输出变量的数目少于表达式返回的输出数目，MATLAB 会将前 N 个输出分配给 N 个变量，然后放弃其余的任何输出。在接下来的示例中，MATLAB 将 C{1,1:3} 分配给变量 c1、c2 和 c3，然后放弃 C{1,4:6}：

```
[c1,c2,c3] = C{1,1:6};
```

您可以按照相同的方式分配结构体输出：

```
S = cell2struct(C,{'f1','f2','f3','f4','f5','f6'},2);
[sf1,sf2,sf3] = S.f5;
sf3

sf3 =
38
```

此外，也可以使用 deal 函数来实现此目的。

## 为逗号分隔的列表赋值

将多个值分配给逗号分隔的列表的最简单方法是使用 deal 函数。此函数将其所有输入参数分发给逗号分隔的列表的元素。

此示例使用 deal 覆盖逗号分隔的列表中的每个元素。首先创建一个列表。

```
c{1} = [31 07];
c{2} = [03 78];
c{:}

ans =
31    7
```

```
ans =
```

```
3    78
```

使用 deal 覆盖列表中的每个元素。

```
[c{:}] = deal([10 20],[14 12]);
c{:}
```

```
ans =  
10 20
```

```
ans =  
14 12
```

以下示例与上述示例执行相同的操作，不同的是将值赋给了结构体字段中以逗号分隔的向量列表：

```
s(1).field1 = [31 07];  
s(2).field1 = [03 78];  
s.field1
```

```
ans =  
31 7
```

```
ans =  
3 78
```

使用 **deal** 覆盖结构体字段。

```
[s.field1] = deal([10 20],[14 12]);  
s.field1
```

```
ans =  
10 20
```

```
ans =  
14 12
```

## 如何使用逗号分隔的列表

逗号分隔的列表的常见用途如下

- “构造数组”（第 2-76 页）
- “显示数组”（第 2-77 页）
- “串联”（第 2-77 页）
- “用作函数调用参数”（第 2-77 页）
- “用作函数返回值”（第 2-78 页）

以下部分提供了将逗号分隔的列表与元胞数组一起使用的示例。其中每个示例也适用于 MATLAB 结构体。

### 构造数组

构造矩阵或数组时，您可以使用逗号分隔的列表来输入一组元素。请注意当您插入元素列表而不是添加元胞本身时会发生什么情况。

当您使用 `C{:, 5}` 指定元素列表时，MATLAB 将会插入四个单独的元素：

```
A = {'Hello',C{:,5},magic(4)}
```

```
A =
```

```
'Hello' [34] [36] [38] [40] [4x4 double]
```

当您指定 C 元胞本身时，MATLAB 将会插入整个元胞数组：

```
A = {'Hello',C,magic(4)}
```

```
A =
```

```
'Hello' {4x6 cell} [4x4 double]
```

### 显示数组

使用列表显示整个或部分结构体或元胞数组：

```
A{:}
```

```
ans =
```

```
Hello
```

```
ans =
```

```
[2] [10] [18] [26] [34] [42]
[4] [12] [20] [28] [36] [44]
[6] [14] [22] [30] [38] [46]
[8] [16] [24] [32] [40] [48]
```

```
ans =
```

```
16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1
```

### 串联

将逗号分隔的列表放在方括号内，可从列表中提取指定的元素并将这些元素进行串联：

```
A = [C{:,5:6}]
```

```
A =
```

```
34 36 38 40 42 44 46 48
```

### 用作函数调用参数

编写函数调用代码时，可以列表形式输入各输入参数，并将每个参数用逗号进行分隔。如果您将这些参数存储在某个结构体或元胞数组中，则可以从该结构体或元胞数组生成整个或部分参数列表。传入数量可变的参数时，这会特别有用。

以下示例将多个属性-值参数传递给 **plot** 函数：

```
X = -pi:pi/10:pi;
Y = tan(sin(X)) - sin(tan(X));
```

```
C = cell(2,3);
C{1,1} = 'LineWidth';
C{2,1} = 2;
C{1,2} = 'MarkerEdgeColor';
C{2,2} = 'k';
C{1,3} = 'MarkerFaceColor';
C{2,3} = 'g';
figure
plot(X,Y,'-rs',C{})
```

### 用作函数返回值

MATLAB 函数也可以向调用方返回多个值。这些值以列表形式返回，并且每个值用逗号分隔。您可以将逗号分隔的列表与结构体或元胞数组一起使用，而不用列出每个返回值。对于返回值的数量可变的函数，这一用法更为实用。

以下示例向一个元胞数组返回三个值：

```
C = cell(1,3);
[C{:}] = fileparts('work/mytests/strArrays.mat')

C =
    'work/mytests'    'strArrays'    '.mat'
```

### 快速傅里叶变换示例

`fftshift` 函数交换数组的每个维度的左半部分和右半部分。对于简单向量（例如 [0 2 4 6 8 10]），输出将为 [6 8 10 0 2 4]。对于多维数组，`fftshift` 沿每个维度执行此交换。

`fftshift` 使用索引向量来执行交换。对于上面显示的向量，索引 [1 2 3 4 5 6] 经过重新排列以形成新索引 [4 5 6 1 2 3]。然后，该函数使用此索引向量重新定位各元素。对于多维数组，`fftshift` 必须为每个维度构造一个索引向量。逗号分隔的列表使该任务变得很简单。

下面是 `fftshift` 函数：

```
function y = fftshift(x)
    numDims = ndims(x);
    idx = cell(1,numDims);
    for k = 1:numDims
        m = size(x,k);
        p = ceil(m/2);
        idx{k} = [p+1:m 1:p];
    end
    y = x(idx{:});
end
```

该函数将索引向量存储在元胞数组 `idx` 中。构建此元胞数组相对比较简单。对于 N 维中的每个维度，请确定该维度的大小并计算最接近中点的整数索引。然后，构造一个交换该维度左右部分的向量。

通过使用元胞数组存储索引向量以及使用逗号分隔的列表进行索引运算，`fftshift` 只需执行单个运算 `y = x(idx{:})` 即可移动任意维度的数组。如果您要使用显式索引，则需要为希望函数处理的每个维度都编写一条 `if` 语句：

```
if ndims(x) == 1
    y = x(index1);
else if ndims(x) == 2
```

```
y = x(index1,index2);
end
end
```

在没有逗号分隔的列表时处理这种情况的另一种方法是对每个维度进行循环，每次转换一个维度并移动数据。通过逗号分隔的列表，您只需将数据移动一次。逗号分隔的列表使得将交换运算推广到任意数量的维度变得非常容易。

## eval 函数的替代方法

### 本节内容

- “为什么要避免使用 eval 函数？”（第 2-80 页）
- “带有序列名称的变量”（第 2-80 页）
- “带有序列名称的文件”（第 2-81 页）
- “变量中的函数名称”（第 2-81 页）
- “变量中的字段名称”（第 2-81 页）
- “错误的处理方式”（第 2-82 页）

### 为什么要避免使用 eval 函数？

虽然 eval 函数非常强大和灵活，但它并不总是编程问题的最佳解决方案。与使用其他函数或语言构造的代码相比，调用 eval 的代码通常效率较低，而且难以阅读和调试。例如：

- MATLAB 会在您首次运行代码时对代码进行编译，以增强以后运行的性能。但是，由于 eval 语句中的代码可在运行时更改，因此无法进行编译。
- eval 语句中的代码可能会意外创建变量或为当前工作区中已存在的变量赋值，并覆盖现有的数据。
- eval 语句中的串联字符串通常难以阅读。其他语言构造方式可以简化您代码中的语法。

对于 eval 的许多常见用法，有一些首选的替代方法，下面以示例形式介绍了这些方法。

### 带有序列名称的变量

eval 函数的常见用法是创建 A1、A2、...、An 之类的变量集，但此方法不使用 MATLAB 的数组处理功能，因此建议不要使用。首选方法是将相关数据存储在单个数组中。如果数据集具有不同的类型或大小，请使用结构体或元胞数组。

例如，创建一个包含 10 个元素的元胞数组，其中每个元素都是数值数组：

```
numArrays = 10;
A = cell(numArrays,1);
for n = 1:numArrays
    A{n} = magic(n);
end
```

通过花括号创建索引来访问元胞数组中的数据。例如，显示 A 的第五个元素：

```
A{5}
```

```
ans =
 17  24   1   8  15
 23   5   7  14  16
  4   6  13  20  22
 10  12  19  21   3
 11  18  25   2   9
```

赋值语句 A{n} = magic(n) 要比下面调用 eval 的语法更为简洁高效：

```
eval(['A', int2str(n), '= magic(n)']) % Not recommended
```

有关详细信息，请参阅：

- “创建元胞数组”（第 12-3 页）
- “创建结构体数组”（第 11-2 页）

## 带有序列名称的文件

相互关联的数据文件通常具有一个共同的带整数索引的根名称，例如 `myfile1.mat` 到 `myfileN.mat`。  
`eval` 函数的一个常见（但不建议）用法是使用命令语法构造每个文件名并将它们传递给某个函数，例如

```
eval(['save myfile',int2str(n),'.mat']) % Not recommended
```

最佳做法是使用函数语法，该语法允许您将变量作为输入传递。例如：

```
currentFile = 'myfile1.mat';
save(currentFile)
```

您可以使用 `sprintf` 函数（通常比 `int2str` 更有效）在循环内构造文件名，然后调用 `save` 函数，而无需使用 `eval`。此代码在当前文件夹中创建 10 个文件：

```
numFiles = 10;
for n = 1:numFiles
    randomData = rand(n);
    currentFile = sprintf('myfile%d.mat',n);
    save(currentFile,'randomData')
end
```

有关详细信息，请参阅：

- “命令与函数语法”（第 1-7 页）
- “导入或导出一系列文件”

## 变量中的函数名称

`eval` 的一个常见用法是当函数的名称位于可变字符串中时执行函数。有两种方式可计算变量中的函数，每一种都比使用 `eval` 更为高效：

- 使用 @ 符号或使用 `str2func` 函数创建函数句柄。例如，从元胞数组中存储的列表运行函数：

```
examples = {@odedemo,@sunspots,@fitdemo};
n = input('Select an example (1, 2, or 3): ');
examples{n}()
```

- 使用 `feval` 函数。例如，使用您在运行时指定的数据调用绘图函数（例如 `plot`、`bar` 或 `pie`）：

```
plotFunction = input('Specify a plotting function: ','s');
data = input('Enter data to plot: ');
feval(plotFunction,data)
```

## 变量中的字段名称

通过将字段表达式括入括号中，使用变量字段名称访问结构体中的数据。例如：

```
myData.height = [67, 72, 58];
myData.weight = [140, 205, 90];

fieldName = input('Select data (height or weight): ','s');
dataToUse = myData.(fieldName);
```

如果您在输入提示符处输入 `weight`, 则可以使用以下命令查找最小 `weight` 值。

```
min(dataToUse)  
ans =  
    90
```

有关其他示例, 请参阅 “基于变量生成字段名称” (第 11-11 页)。

## 错误的处理方式

MATLAB 中处理错误的首选方法是使用 `try`, `catch` 语句。例如:

```
try  
    B = A;  
catch exception  
    disp('A is undefined')  
end
```

如果您的工作区不包含变量 `A`, 则该代码返回:

```
A is undefined
```

`eval` 函数以前版本的文档中包含 `eval(expression,catch_expr)` 语法介绍。如果计算 `expression` 输入返回错误, 则 `eval` 会计算 `catch_expr`。但是, 在 `eval` 语句中, 显式的 `try/catch` 明显比隐式的 `catch` 更清晰。建议不要使用隐式的 `catch`。

# 类 (数据类型)



# MATLAB 类概述

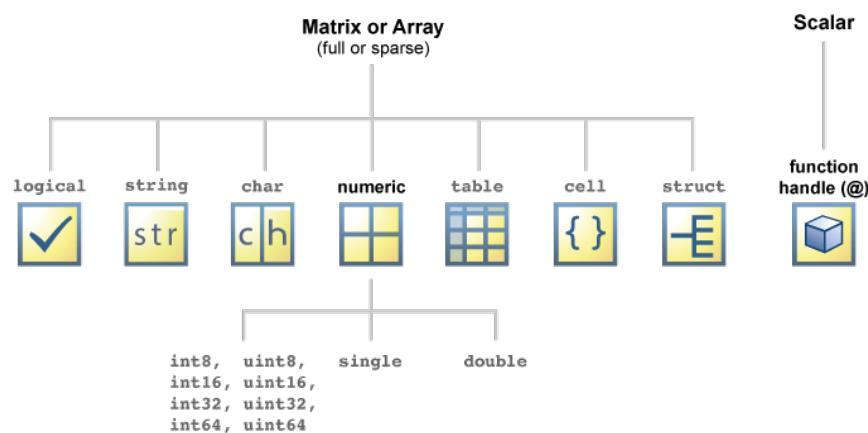
---

## MATLAB 基础类

您可以在 MATLAB 软件中使用许多不同的数据类型或类。您可以构建浮点和整数数据、字符和字符串以及逻辑 `true` 和 `false` 状态的矩阵与数组。函数句柄可将您的代码与任何 MATLAB 函数连接在一起，而与当前范围无关。表格、结构体和元胞数组提供了一种将不同类型的数据存储在同一容器中的方法。

MATLAB 中有 16 个基础类。其中每个类都采用矩阵或数组的形式存在。除了函数句柄之外，矩阵或数组的最小大小为  $0 \times 0$ ，并且可以扩展到任意大小的  $n$  维数组。函数句柄始终为标量 ( $1 \times 1$ )。

下图显示了所有的 MATLAB 基础类：



MATLAB 软件中的数值类包括有符号和无符号整数、单精度和双精度浮点数。默认情况下，MATLAB 以双精度浮点形式存储所有数值。（您不能更改默认类型和精度。）您可以选择以整数或单精度形式存储任何数值或数值数组。与双精度相比，以整数数组和单精度数组存储数据更节省内存。

所有数值类型都支持基本的数组运算，例如添加下标、重构和数学运算。

您可以使用以下两种存储格式之一创建二维的 `double` 和 `logical` 矩阵：`full` 或 `sparse`。对于元素大部分是零值的矩阵，即稀疏矩阵，需要的存储空间是与之相同大小的满矩阵的一部分。稀疏矩阵会调用专为求解稀疏问题而设计的方法。

这些类需要不同的存储量，最小的类为 `logical` 值或 8 位整数，仅需要 1 个字节。如果您正在处理用小于 8 位的精度写入的文件中的数据，请务必记住这一最小大小。

下表详细地介绍了这些基础类。

类名	文档	预期用途
<code>double</code> 、 <code>single</code>	浮点数（第 4-6 页）	<ul style="list-style-type: none"> <li>分数据必需。</li> <li>双精度（第 4-6 页）和单精度（第 4-6 页）。</li> <li>使用 <code>realmin</code> 和 <code>realmax</code> 显示值的范围（第 4-8 页）。</li> <li>二维数组可以为稀疏数组。</li> <li>MATLAB 中的默认数值类型。</li> </ul>

类名	文档	预期用途
<code>int8</code> 、 <code>uint8</code> 、 <code>int16</code> 、 <code>uint16</code> 、 <code>int32</code> 、 <code>uint32</code> 、 <code>int64</code> 、 <code>uint64</code>	整数 (第 4-2 页)	<ul style="list-style-type: none"> <li>用于有符号和无符号整数。</li> <li>更有效地使用内存。 (第 30-2 页)</li> <li>使用 <code>intmin</code> 和 <code>intmax</code> 显示值的范围 (第 4-4 页)。</li> <li>从 4 个大小 (8 位、16 位、32 位和 64 位) 中选择。</li> </ul>
<code>char</code> 、 <code>string</code>	“字符和字符串”	<ul style="list-style-type: none"> <li>文本的数据类型。</li> <li>本机或 Unicode®。</li> <li>转换为数值/从数值进行转换。</li> <li>与正则表达式 (第 2-48 页) 一起使用。</li> <li>对于多个字符数组，使用元胞数组。</li> <li>从 R2016b 开始，您还可以将文本存储在字符串数组中。有关详细信息，请参阅 <code>string</code>。</li> </ul>
<code>logical</code>	“逻辑运算”	<ul style="list-style-type: none"> <li>用于关系条件或用于测试状态。</li> <li>可以具有以下两个值之一： <code>true</code> 或 <code>false</code>。</li> <li>也用于数组索引。</li> <li>二维数组可以为稀疏数组。</li> </ul>
<code>function_handle</code>	“函数句柄”	<ul style="list-style-type: none"> <li>函数的指针。</li> <li>支持将一个函数传递给另一个函数</li> <li>也可以在通常的范围之外调用函数。</li> <li>用于指定图形回调函数。</li> <li>保存到 MAT 文件并在以后恢复。</li> </ul>
<code>table</code>	“表”	<ul style="list-style-type: none"> <li>混合型列向数据的矩形容器。</li> <li>用行名称和变量名称标识内容。</li> <li>使用表的属性来存储变量单位之类的元数据。</li> <li>元素处理与数值或逻辑数组类似。</li> <li>通过数值或命名索引访问数据。</li> <li>可以选择数据子集并保留表格容器，也可以从表格中提取数据。</li> </ul>
<code>struct</code>	“结构体”	<ul style="list-style-type: none"> <li>用字段存储不同类、不同大小的数组。</li> <li>通过单一操作访问一个或所有字段/索引。</li> <li>用字段名称标识内容。</li> <li>传递函数参数的方法。</li> <li>用于逗号分隔的列表 (第 2-73 页)。</li> <li>开销需要较多内存</li> </ul>
<code>cell</code>	“元胞数组”	<ul style="list-style-type: none"> <li>用元胞存储不同类、不同大小的数组。</li> <li>可以根据需要自由打包数据。</li> <li>元素处理与数值或逻辑数组类似。</li> <li>传递函数参数的方法。</li> <li>用于逗号分隔的列表。</li> <li>开销需要较多内存</li> </ul>

#### 另请参阅

#### 详细信息

- “不同类的有效合并” (第 15-2 页)

# 数值类

---

- “整数” (第 4-2 页)
- “浮点数” (第 4-6 页)
- “创建复数” (第 4-13 页)
- “无穷大和 NaN” (第 4-14 页)
- “确定数值类” (第 4-16 页)
- “数值的显示格式” (第 4-17 页)
- “整数算术运算” (第 4-19 页)
- “单精度运算” (第 4-26 页)

## 整数

本节内容
“整数类” (第 4-2 页)
“创建整数数据” (第 4-2 页)
“整数类的算术运算” (第 4-3 页)
“整数类的最大值和最小值” (第 4-4 页)

### 整数类

MATLAB 具有四个有符号整数类和四个无符号整数类。有符号类型使您能够处理负整数以及正整数，但表示的数字范围不如无符号类型广泛，因为有一个位用于指定数字的正号或负号。无符号类型提供了更广泛的数字范围，但这些数字只能为零或正数。

MATLAB 支持以 1 字节、2 字节、4 字节和 8 字节几种形式存储整数数据。如果您使用可容纳您的数据的最小整数类型来存储数据，则可以节省程序内存和执行时间。例如，您不需要使用 32 位整数来存储值 100。

下面列出了八个整数类、使用每种类型可存储的值范围以及创建该类型所需的 MATLAB 转换函数：

类	值的范围	转换函数
有符号 8 位整数	-2 <sup>7</sup> 到 2 <sup>7</sup> -1	<code>int8</code>
有符号 16 位整数	-2 <sup>15</sup> 到 2 <sup>15</sup> -1	<code>int16</code>
有符号 32 位整数	-2 <sup>31</sup> 到 2 <sup>31</sup> -1	<code>int32</code>
有符号 64 位整数	-2 <sup>63</sup> 到 2 <sup>63</sup> -1	<code>int64</code>
无符号 8 位整数	0 到 2 <sup>8</sup> -1	<code>uint8</code>
无符号 16 位整数	0 到 2 <sup>16</sup> -1	<code>uint16</code>
无符号 32 位整数	0 到 2 <sup>32</sup> -1	<code>uint32</code>
无符号 64 位整数	0 到 2 <sup>64</sup> -1	<code>uint64</code>

### 创建整数数据

MATLAB 默认情况下以双精度浮点形式 (`double`) 存储数值数据。要以整数形式存储数据，您需要从 `double` 转换为所需的整数类型。使用上表中所示的转换函数之一。

例如，如果要以 16 位有符号整数形式存储赋给变量 `x` 的值 325，请键入

```
x = int16(325);
```

如果要转换为整数的数值带有小数部分，MATLAB 将舍入到最接近的整数。如果小数部分正好是 0.5，则 MATLAB 会从两个同样临近的整数中选择绝对值更大的整数：

```
x = 325.499;
int16(x)
ans =
```

```
int16
```

325

```
x = x + .001;
int16(x)
ans =
```

int16

326

如果您需要使用非默认舍入方案对数值进行舍入，MATLAB 提供了以下四种舍入函数：**round**、**fix**、**floor** 和 **ceil**。**fix** 函数使您能够覆盖默认的舍入方案，并朝零舍入（如果存在非零的小数部分）：

```
x = 325.9;
int16(fix(x))
ans =
```

int16

325

同时涉及整数和浮点数的算术运算始终生成整数数据类型。MATLAB 会在必要时根据默认的舍入算法对结果进行舍入。以下示例生成 1426.75 的确切答案，然后 MATLAB 将该数值舍入到下一个最高的整数：

```
int16(325) * 4.39
ans =
```

int16

1427

在将其他类（例如字符串）转换为整数时，这些整数转换函数也很有用：

```
str = 'Hello World';
```

```
int8(str)
ans =
```

1×11 int8 row vector

72 101 108 108 111 32 87 111 114 108 100

如果您将 NaN 值转换为整数类，则结果为该整数类中的 0 值。例如，

```
int32(NaN)
ans =
```

int32

0

## 整数类的算术运算

MATLAB 可以对以下类型的数据执行整数算术运算：

- 整数或具有相同整数数据类型的整数数组。此运算生成的结果与操作数具有相同的数据类型：

```
x = uint32([132 347 528]) .* uint32(75);
class(x)
ans =
uint32
```

- 整数或整数数组以及双精度标量浮点数。此运算生成的结果与整数操作数具有相同的数据类型：

```
x = uint32([132 347 528]) .* 75.49;
class(x)
ans =
uint32
```

对于一个操作数为整数数据类型（64 位整数除外）的数组，另一个操作数为双精度标量的所有二进制运算，MATLAB 会使用按元素双精度算法来执行运算，然后将结果重新转换为原始的整数数据类型。对于涉及 64 位整数数组和双精度标量的二进制运算，MATLAB 会使用扩展精度（比如 80 位扩展精度）算法来执行运算，以防止精度损失。

不支持涉及具有整数类型的复数的运算。

## 整数类的最大值和最小值

每种整数数据类型都存在可以用该类型表示的最大数和最小数。“整数”（第 4-2 页）中显示的表在“值的范围”一列中列出了每种整数数据类型的最大值和最小值。

您也可以通过 `intmax` 和 `intmin` 函数获取这些值：

```
intmax('int8')
ans =
int8
127

intmin('int8')
ans =
int8
-128
```

如果您将大于某个整数数据类型的最大值的数值转换为该类型，MATLAB 会将其设置为最大值。同样，如果您转换小于该整数数据类型的最小值的数值，MATLAB 会将其设置为最小值。例如，

```
x = int8(300)
x =
int8
127

x = int8(-300)
x =
int8
-128
```

此外，当涉及整数的算术运算的结果超出该数据类型的最大值（或最小值）时，MATLAB 也会将其设置为最大值（或最小值）：

```
x = int8(100) * 3  
x =
```

```
int8
```

```
127
```

```
x = int8(-100) * 3  
x =
```

```
int8
```

```
-128
```

## 浮点数

### 本节内容

- “双精度浮点”（第 4-6 页）
- “单精度浮点”（第 4-6 页）
- “创建浮点数据”（第 4-6 页）
- “浮点数的算术运算”（第 4-7 页）
- “浮点类的最大值和最小值”（第 4-8 页）
- “浮点数据的精度”（第 4-9 页）
- “避免浮点算术运算出现常见问题”（第 4-10 页）

MATLAB 以双精度或单精度格式表示浮点数。默认为双精度，但您可以通过一个简单的转换函数将任何数值转换为单精度数值。

### 双精度浮点

MATLAB 根据适用于双精度的 IEEE® 754 标准来构造双精度（即 `double`）数据类型。以 `double` 形式存储的任何值都需要 64 位，并按照下表所示进行格式化：

位	用法
63	符号（0 = 正号、1 = 负号）
62 到 52	指数，偏差为 1023
51 到 0	数值 1.f 的小数 f

### 单精度浮点

MATLAB 根据适用于单精度的 IEEE 754 标准来构造单精度（即 `single`）数据类型。以 `single` 形式存储的任何值都需要 32 位，并按照下表所示进行格式化：

位	用法
31	符号（0 = 正号、1 = 负号）
30 到 23	指数，偏差为 127
22 到 0	数值 1.f 的小数 f

由于 MATLAB 使用 32 位来存储 `single` 类型的数值，因此与使用 64 位的 `double` 类型的数值相比，前者需要的内存更少。但是，由于它们是使用较少的位存储的，因此 `single` 类型的数值所呈现的精度要低于 `double` 类型的数值。

### 创建浮点数据

一般使用双精度来存储大于  $3.4 \times 10^{38}$  或约小于  $-3.4 \times 10^{38}$  的值。对于位于这两个范围之间的数值，您可以使用双精度，也可以使用单精度，但单精度需要的内存更少。

#### 创建双精度数据

由于 MATLAB 的默认数值类型为 `double`，因此您可以通过一个简单的赋值语句来创建 `double` 值：

```
x = 25.783;
```

**whos** 函数显示，MATLAB 已为您刚在 **x** 中存储的值创建了一个 **double** 类型的 1x1 数组：

<b>whos x</b>	
Name	Size
<b>x</b>	1x1
	8 double

如果您只想验证 **x** 是否为浮点数，请使用 **isfloat**。如果输入为浮点数，此函数将返回逻辑值 1 (**true**)，否则返回逻辑值 0 (**false**)：

```
isfloat(x)
ans =
logical
```

```
1
```

您可以使用 MATLAB 函数 **double** 将其他数值数据、字符或字符串以及逻辑数据转换为双精度值。以下示例将有符号整数转换为双精度浮点数：

```
y = int64(-589324077574); % Create a 64-bit integer
x = double(y) % Convert to double
x =
-5.8932e+11
```

### 创建单精度数据

由于 MATLAB 默认情况下以 **double** 形式存储数值数据，因此您需要使用 **single** 转换函数来创建单精度数：

```
x = single(25.783);
```

**whos** 函数在结构体中返回变量 **x** 的属性。此结构体的 **bytes** 字段显示，当以 **single** 形式存储 **x** 时，该变量仅需要 4 字节，而以 **double** 形式存储则需要 8 字节：

```
xAttrib = whos('x');
xAttrib.bytes
ans =
4
```

您可以使用 **single** 函数将其他数值数据、字符或字符串以及逻辑数据转换为单精度值。以下示例将有符号整数转换为单精度浮点数：

```
y = int64(-589324077574); % Create a 64-bit integer
x = single(y) % Convert to single
x =
single
-5.8932e+11
```

## 浮点数的算术运算

此部分介绍您可以在算术运算中将哪些类与浮点数一起使用。

## 双精度运算

您可以使用 **double** 和以下的任何其他类来执行基本算术运算。如果一个或多个操作数为整数（标量或数组），则 **double** 操作数必须为标量。运算结果默认为 **double** 类型，除非另有说明：

- **single** - 结果为 **single** 类型
- **double**
- **int\*** 或 **uint\*** - 结果与整数操作数具有相同的数据类型
- **char**
- **logical**

以下示例对 **char** 和 **double** 类型的数据执行算术运算。结果为 **double** 类型：

```
c = 'uppercase' - 32;
```

```
class(c)
ans =
    double
```

```
char(c)
ans =
    UPPERCASE
```

## 单精度运算

您可以使用 **single** 和以下的任何其他类来执行基本算术运算。运算结果始终为 **single**：

- **single**
- **double**
- **char**
- **logical**

在以下示例中，7.5 默认为 **double** 类型，结果为 **single** 类型：

```
x = single([1.32 3.47 5.28]) .* 7.5;
```

```
class(x)
ans =
    single
```

## 浮点类的最大值和最小值

**double** 和 **single** 类都存在可以用该类型表示的最大数和最小数。

### 最大和最小双精度值

MATLAB 函数 **realmax** 和 **realmin** 分别返回可以用 **double** 数据类型表示的最大值和最小值：

```
str = 'The range for double is:\n\t%g to %g and\n\t%g to %g';
sprintf(str, -realmax, -realmin, realmin, realmax)
```

```
ans =
The range for double is:
```

```
-1.79769e+308 to -2.22507e-308 and
2.22507e-308 to 1.79769e+308
```

大于 `realmax` 或小于 `-realmax` 的数分别被赋予正无穷大和负无穷大的值：

```
realmax + .0001e+308
```

```
ans =
```

```
Inf
```

```
-realmax - .0001e+308
```

```
ans =
```

```
-Inf
```

### 最大和最小单精度值

在用参数 '`'single'`' 调用 MATLAB 函数 `realmax` 和 `realmin` 时，这两个函数会分别返回可以用 `single` 数据类型表示的最大值和最小值：

```
str = 'The range for single is:\n\t%g to %g and\n\t %g to %g';
sprintf(str, -realmax('single'), -realmin('single'), ...
```

```
realmin('single'), realmax('single'))
```

```
ans =
```

```
The range for single is:
```

```
-3.40282e+38 to -1.17549e-38 and
```

```
1.17549e-38 to 3.40282e+38
```

大于 `realmax('single')` 或小于 `-realmax('single')` 的数分别被赋予正无穷大和负无穷大的值：

```
realmax('single') + .0001e+038
```

```
ans =
```

```
single
```

```
Inf
```

```
-realmax('single') - .0001e+038
```

```
ans =
```

```
single
```

```
-Inf
```

### 浮点数据的精度

如果浮点算术计算的结果不如预期的精确，可能是由于您的计算机硬件的限制所致。由于硬件缺乏足够的位而无法呈现具有完美精度的结果，计算机可能会将结果值截断，使得结果值不够准确。

#### 双精度数的精度

由于双精度数的数量有限，因此您无法在双精度存储中表示所有数值。在任何计算机上，每个双精度数和下一个更大的双精度数之间都存在一个较小的间隔。您可以使用 `eps` 函数确定此间隔的大小，该大小限制了您的结果的精度。例如，要计算 5 和下一个更大的双精度数之间的间距，请输入

```
format long
```

```
eps(5)
```

```
ans =
8.881784197001252e-16
```

这表明， $5$  和  $5 + \text{eps}(5)$  之间不存在任何双精度数。如果某个双精度计算返回答案  $5$ ，则结果仅精确到  $\text{eps}(5)$  之内。

$\text{eps}(x)$  的值取决于  $x$ 。以下示例显示，当  $x$  变得更大时， $\text{eps}(x)$  也会变得更大：

```
eps(50)
ans =
7.105427357601002e-15
```

如果您输入不带输入参数的  $\text{eps}$ ，MATLAB 将返回  $\text{eps}(1)$  的值（从  $1$  到下一个更大的双精度数之间的间距）。

### 单精度数的精度

同样，两个单精度数之间也存在间隔。如果  $x$  的类型为  $\text{single}$ ，则  $\text{eps}(x)$  返回  $x$  和下一个更大的单精度数之间的间距。例如，

```
x = single(5);
eps(x)
```

返回

```
ans =
single
4.7684e-07
```

请注意，此结果大于  $\text{eps}(5)$ 。由于单精度数的数量少于双精度数的数量，因此单精度数之间的间距也大于双精度数之间的间距。这意味着，单精度算术运算的结果精度要低于双精度算术运算的结果精度。

对于类型为  $\text{double}$  的双精度数  $x$ ， $\text{eps}(\text{single}(x))$  的计算值即为将  $x$  从  $\text{double}$  转换为  $\text{single}$  时的舍入上限。例如，当您将双精度数  $3.14$  转换为  $\text{single}$  时，它会通过以下方式进行舍入

```
double(single(3.14) - 3.14)
ans =
1.0490e-07
```

$3.14$  舍入的数量小于

```
eps(single(3.14))
ans =
single
2.3842e-07
```

## 避免浮点算术运算出现常见问题

MATLAB 中几乎所有运算都是通过符合 IEEE 754 标准的双精度算术运算执行的。由于计算机仅将数值表示为有限精度（双精度需要 52 个尾数位），因此计算有时会生成数学上的非预期结果。务必注意，这些结果并非 MATLAB 中的错误。

使用以下示例来帮助您确定这些情况：

### 示例 1 - 舍入或您所获得的不是所期望的

十进制数  $4/3$  不能精确表示为二进制分数。为此，以下计算的结果不是零，而是显示数量 `eps`。

```
e = 1 - 3*(4/3 - 1)
```

```
e =
2.2204e-16
```

同样， $0.1$  也不能精确表示为二进制数。因此，您会发现以下非预期行为：

```
a = 0.0;
for i = 1:10
    a = a + 0.1;
end
a == 1
ans =
```

```
logical
```

```
0
```

请注意，计算中运算的顺序会很重要：

```
b = 1e-16 + 1 - 1e-16;
c = 1e-16 - 1e-16 + 1;
b == c
ans =
```

```
logical
```

```
0
```

浮点数之间存在间隔。当数值变得越大时，间隔也会变得越大，如以下所示：

```
(2^53 + 1) - 2^53
```

```
ans =
0
```

由于 `pi` 实际上不是  $\pi$ ，因此 `sin(pi)` 不精确为零并不足为奇：

```
sin(pi)
```

```
ans =
1.224646799147353e-16
```

### 示例 2 - 具有灾难性后果的取消操作

对几乎相等的操作数执行减法时，有时可能会发生意外取消。以下是因淹没（使加法没有意义的精度损失）导致的取消的示例。

```
sqrt(1e-16 + 1) - 1
```

```
ans =
0
```

MATLAB 中的某些函数（例如 `expm1` 和 `log1p`）可用于弥补这种灾难性取消所造成的影响。

### 示例 3 - 浮点运算和线性代数

解决线性代数的问题时，舍入、取消和浮点算术运算的其他一些特性结合起来时，可能产生令人意想不到的运算。MATLAB 会警告下面的矩阵  $A$  是病态的，因此即便是细微的扰动，都可能对方程组  $Ax = b$  产生很大的影响：

```
A = diag([2 eps]);
b = [2; eps];
y = A\b;
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 1.110223e-16.
```

这些只是几个例子，用于说明 IEEE 浮点算术运算如何影响 MATLAB 中的计算。请注意，IEEE 754 算术运算中执行的所有计算都会受到影响，这包括用 C 或 FORTRAN 编写的应用程序以及 MATLAB。

### 参考

- [1] Moler, Cleve. "Floating Points." *MATLAB News and Notes*. Fall, 1996.
- [2] Moler, Cleve. *Numerical Computing with MATLAB*. Natick, MA: The MathWorks, Inc., 2004.

## 创建复数

复数由两个单独的部分组成：实部和虚部。基本虚数单位等于 -1 的平方根。这在 MATLAB 中通过以下两个字母之一表示：i 或 j。

以下语句显示了一种在 MATLAB 中创建复数值的方法。变量 x 被赋予了一个复数值，该复数的实部为 2，虚部为 3：

```
x = 2 + 3i;
```

创建复数的另一种方法是使用 **complex** 函数。此函数将两个数值输入组合成一个复数输出，并使第一个输入成为实部，使第二个输入成为虚部：

```
x = rand(3) * 5;
y = rand(3) * -8;

z = complex(x, y)
z =
4.7842 -1.0921i  0.8648 -1.5931i  1.2616 -2.2753i
2.6130 -0.0941i  4.8987 -2.3898i  4.3787 -3.7538i
4.4007 -7.1512i  1.3572 -5.2915i  3.6865 -0.5182i
```

您可以使用 **real** 和 **imag** 函数分解复数，捕获其实部和虚部：

```
zr = real(z)
zr =
4.7842  0.8648  1.2616
2.6130  4.8987  4.3787
4.4007  1.3572  3.6865

zi = imag(z)
zi =
-1.0921 -1.5931 -2.2753
-0.0941 -2.3898 -3.7538
-7.1512 -5.2915 -0.5182
```

## 无穷大和 NaN

### 本节内容

“无穷大” (第 4-14 页)  
“NaN” (第 4-14 页)

## 无穷大

MATLAB 用特殊值 `inf` 表示无穷大。除以零和溢出等运算会生成无穷大，从而导致结果因太大而无法表示为传统的浮点值。MATLAB 还提供了一个称为 `inf` 的函数，该函数以 `double` 标量值形式返回正无穷大的 IEEE 算术表示。

下面显示了在 MATLAB 中返回正无穷大或负无穷大的多个语句示例。

<code>x = 1/0</code>	<code>x = 1.e1000</code>
<code>x =</code>	<code>x =</code>
<code>Inf</code>	<code>Inf</code>
<code>x = exp(1000)</code>	<code>x = log(0)</code>
<code>x =</code>	<code>x =</code>
<code>Inf</code>	<code>-Inf</code>

使用 `isinf` 函数验证 `x` 是否为正无穷大或负无穷大：

```
x = log(0);
isinf(x)
ans =
1
```

## NaN

MATLAB 使用一个称为 `NaN` (代表 “非数值” ) 的特殊值来表示不是实数或复数的值。`0/0` 和 `inf/inf` 之类的表达式会生成 `NaN`，就像执行涉及 `NaN` 的任何算术运算一样：

```
x = 0/0
x =
NaN
```

您也可以通过以下方式创建 `NaN`：

```
x = NaN;
whos x
  Name      Size            Bytes  Class
  x            1x1              8  double
```

`NaN` 函数将 `NaN` 的一个 IEEE 算术表示形式作为 `double` 标量值返回。此 `NaN` 值的按位十六进制精确表示形式为：

```
format hex
x = NaN
```

```
x =  
ffff8000000000000000
```

始终使用 `isnan` 函数来校验数组中的元素是否为 NaN:

```
isnan(x)  
ans =
```

```
1
```

MATLAB 保留其他 NaN 表示形式的“非数值”状态，并同等对待 NaN 的所有不同表示形式。但是，在某些特殊情形中（可能由于硬件限制），MATLAB 在整个计算过程中不保留其他 NaN 表示形式的精确位模式，而是使用上文定义的标准 NaN 位模式。

### 对 NaN 执行逻辑运算

由于两个 NaN 彼此不相等，因此与 NaN 相关的逻辑运算始终返回 `false`，但测试是否不相等 (`Nan ~= Nan`) 除外：

```
Nan > Nan  
ans =  
0
```

```
Nan ~= Nan  
ans =  
1
```

## 确定数值类

您可以使用以下任意命令检查变量  $x$  的数据类型。

命令	运算
<code>whos x</code>	显示 $x$ 的数据类型。
<code>xType = class(x);</code>	将 $x$ 的数据类型赋予变量。
<code>isnumeric(x)</code>	确定 $x$ 是否为数值类型。
<code>isa(x, 'integer')</code> <code>isa(x, 'uint64')</code> <code>isa(x, 'float')</code> <code>isa(x, 'double')</code> <code>isa(x, 'single')</code>	确定 $x$ 是否为指定的数值类型。（此处显示了任意整数、无符号的 64 位整数、任意浮点数、双精度数和单精度数的示例）。
<code>isreal(x)</code>	确定 $x$ 是实数还是复数。
<code>isnan(x)</code>	确定 $x$ 是否不是数值 (NaN)。
<code>isinf(x)</code>	确定 $x$ 是否为无限值。
<code>isfinite(x)</code>	确定 $x$ 是否为有限值。

# 数值的显示格式

默认情况下，MATLAB 使用 5 位短格式显示数值。例如，

```
x = 4/3
```

```
x =
```

```
1.3333
```

您可以使用 **format** 函数更改在命令行窗口或编辑器中的显示。

```
format long
```

```
x
```

```
x =
```

```
1.333333333333333
```

**format** 函数仅设置当前 MATLAB 会话中的格式。要设置后续会话中的格式，请在环境部分的主页选项卡上，点击 预设。选择 MATLAB > 命令行窗口，然后选择数值格式选项。

下表总结了数值输出格式选项。

Style	结果	示例
<b>short (default)</b>	短固定十进制小数点格式，小数点后包含 4 位数。	3.1416
<b>long</b>	长固定十进制小数点格式， <b>double</b> 值的小数点后包含 15 位数， <b>single</b> 值的小数点后包含 7 位数。	3.141592653589793
<b>shortE</b>	短科学记数法，小数点后包含 4 位数。	3.1416e+00
<b>longE</b>	长科学记数法， <b>double</b> 值的小数点后包含 15 位数， <b>single</b> 值的小数点后包含 7 位数。	3.141592653589793e+00
<b>shortG</b>	短固定十进制小数点格式或科学记数法（取更紧凑的一个），总共 5 位。	3.1416
<b>longG</b>	长固定十进制小数点格式或科学记数法（取更紧凑的一个），对于 <b>double</b> 值，总共 15 位；对于 <b>single</b> 值，总共 7 位。	3.14159265358979
<b>shortEng</b>	短工程记数法，小数点后包含 4 位数，指数为 3 的倍数。	3.1416e+000
<b>longEng</b>	长工程记数法，包含 15 位有效位数，指数为 3 的倍数。	3.14159265358979e+000
<b>+</b>	正/负格式，对正、负和零元素分别显示 +、- 和空白字符。	+
<b>bank</b>	货币格式，小数点后包含 2 位数。	3.14
<b>hex</b>	二进制双精度数字的十六进制表示形式。	400921fb54442d18
<b>rat</b>	小整数的比率。	355/113

显示格式只影响数值的显示方式，不影响它们在 MATLAB 中的存储方式。

**另请参阅**

`format`

### 相关示例

- “设置输出格式”

# 整数算术运算

此示例说明如何对表示信号和图像的整数数据执行算术运算。

## 加载整数信号数据

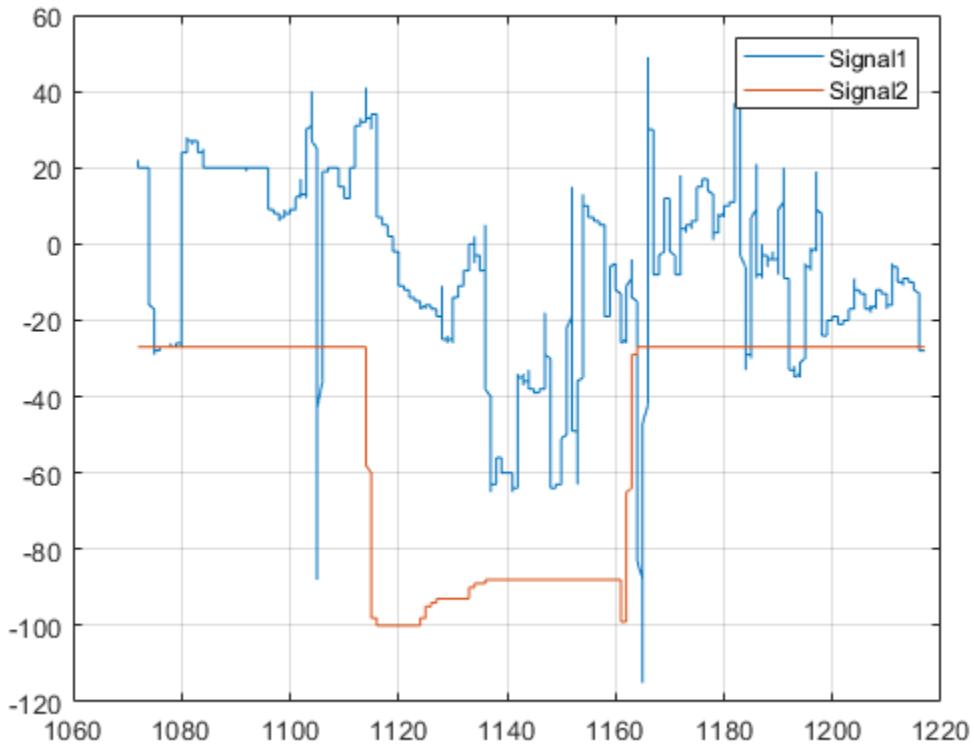
加载由四种乐器发出的信号组成的测量数据集，并将数据中 8 位和 16 位的 A 至 D 结果保存为 `int8`、`int16` 和 `uint16`。时间存储为 `uint16`。

```
load integersignal  
  
% Look at variables  
whos Signal1 Signal2 Signal3 Signal4 Time1  
  
Name      Size      Bytes Class    Attributes  
Signal1   7550x1    7550  int8  
Signal2   7550x1    7550  int8  
Signal3   7550x1    15100 int16  
Signal4   7550x1    15100 uint16  
Time1     7550x1    15100 uint16
```

## 对数据绘图

首先，对两个信号绘图以查看信号范围。

```
plot(Time1, Signal1, Time1, Signal2);  
grid;  
legend('Signal1','Signal2');
```



此时可以看到 int8 的值。可能需要对这些值进行缩放以计算信号代表的实际物理值，例如电压。

### 处理数据

可以对整数执行标准算术运算，例如 +、-、\* 和 /。假设要计算 Signal1 和 Signal2 的和。

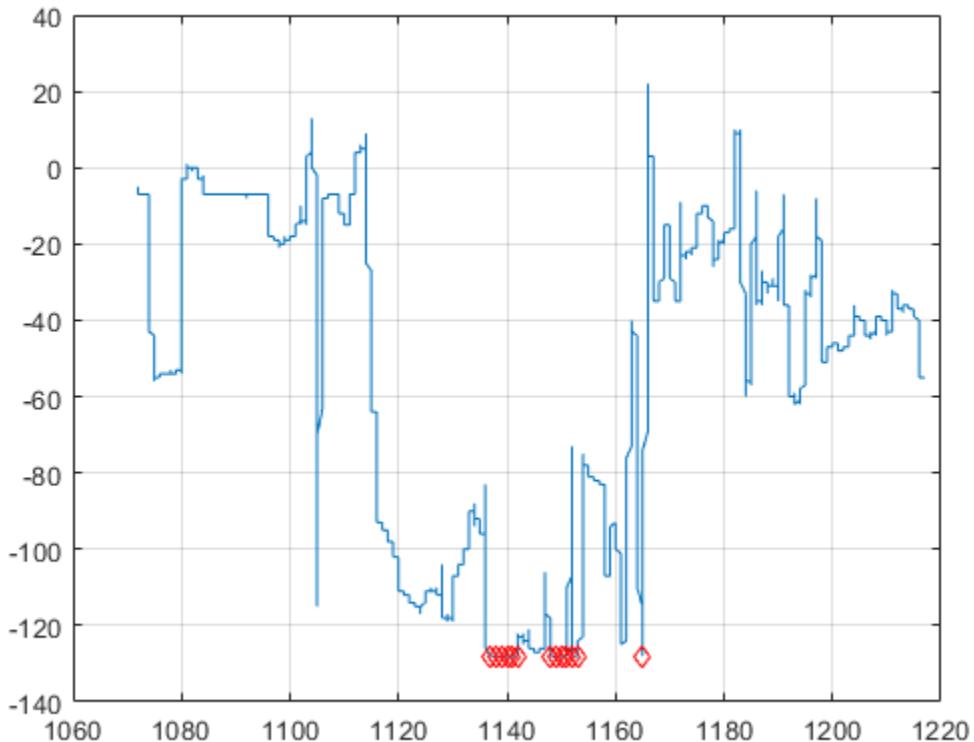
`SumSig = Signal1 + Signal2; % Here we sum the integer signals.`

现在，对和信号绘图并查看饱和位置。

```

cla;
plot(Time1, SumSig);
hold on
Saturated = (SumSig == intmin('int8')) | (SumSig == intmax('int8')); % Find where it has saturated
plot(Time1(Saturated),SumSig(Saturated),'rd')
grid
hold off

```



标记所示即信号饱和的位置。

### 加载整数图像数据

接下来，看一下对一些图像数据执行的算术运算。

```
street1 = imread('street1.jpg'); % Load image data
street2 = imread('street2.jpg');
whos street1 street2
```

Name	Size	Bytes	Class	Attributes
street1	480x640x3	921600	uint8	
street2	480x640x3	921600	uint8	

可以看出，图像为 24 位颜色，存储为三个 uint8 数据平面。

### 显示图像

显示第一个图像。

```
cla;
image(street1); % Display image
axis equal
axis off
```



显示第二个图像

```
image(street2); % Display image  
axis equal  
axis off
```



## 缩放图像

可以按一个双精度常量缩放图像，但仍保持以整数形式存储图像。例如，

```
duller = 0.5 * street2; % Scale image with a double constant but create an integer  
whos duller
```

Name	Size	Bytes	Class	Attributes
duller	480x640x3	921600	uint8	

```
subplot(1,2,1);  
image(street2);  
axis off equal tight  
title('Original'); % Display image  
  
subplot(1,2,2);  
image(duller);  
axis off equal tight  
title('Duller'); % Display image
```



### 添加图像

现在，将两个街道图像叠加在一起，并对重影结果绘图。

```
combined = street1 + duller; % Add | uint8 | images
subplot(1,1,1)
cla;
image(combined); % Display image
title('Combined');
axis equal
axis off
```

Combined



## 单精度运算

此示例说明如何对单精度数据执行算术运算和线性代数运算。此外，还说明了如何根据输入相应地按单精度或双精度计算结果。

### 创建双精度数据

首先创建一些数据，默认情况下为双精度。

```
Ad = [1 2 0; 2 5 -1; 4 10 -1]
```

$Ad = 3 \times 3$

1	2	0
2	5	-1
4	10	-1

### 转换为单精度

可以使用 `single` 函数将数据转换为单精度。

```
A = single(Ad); % or A = cast(Ad,'single');
```

### 创建单精度零和一

此外，也可以分别使用函数创建单精度零和一。

```
n = 1000;
Z = zeros(n,1,'single');
O = ones(n,1,'single');
```

看一下工作区中的变量。

```
whos A Ad O Z n
```

Name	Size	Bytes Class	Attributes
A	3x3	36 single	
Ad	3x3	72 double	
O	1000x1	4000 single	
Z	1000x1	4000 single	
n	1x1	8 double	

可以看到，部分变量的类型为 `single`，变量 A (Ad 的单精度版本) 需要一半的内存字节数用于存储，因为单精度仅需要四字节 (32 位)，而双精度需要 8 字节 (64 位)。

### 算术运算和线性代数运算

可以对单精度数据执行标准算术运算和线性代数运算。

```
B = A' % Matrix Transpose
```

$B = 3 \times 3$  single matrix

1	2	4
2	5	10
0	-1	-1

```
whos B
```

Name	Size	Bytes	Class	Attributes
B	3x3	36	single	

可以看出，此操作的结果 B 为单精度。

```
C = A * B % Matrix multiplication
```

C = 3x3 single matrix

5	12	24
12	30	59
24	59	117

```
C = A .* B % Elementwise arithmetic
```

C = 3x3 single matrix

1	4	0
4	25	-10
0	-10	1

```
X = inv(A) % Matrix inverse
```

X = 3x3 single matrix

5	2	-2
-2	-1	1
0	-2	1

```
I = inv(A) * A % Confirm result is identity matrix
```

I = 3x3 single matrix

1	0	0
0	1	0
0	0	1

```
I = A \ A % Better way to do matrix division than inv
```

I = 3x3 single matrix

1	0	0
0	1	0
0	0	1

```
E = eig(A) % Eigenvalues
```

E = 3x1 single column vector

3.7321
0.2679
1.0000

```
F = fft(A(:,1)) % FFT
```

F = 3x1 single column vector

```
7.0000 + 0.0000i  
-2.0000 + 1.7321i  
-2.0000 - 1.7321i
```

```
S = svd(A) % Singular value decomposition
```

S = 3x1 single column vector

```
12.3171  
0.5149  
0.1577
```

```
P = round(poly(A)) % The characteristic polynomial of a matrix
```

P = 1x4 single row vector

```
1 -5 5 -1
```

```
R = roots(P) % Roots of a polynomial
```

R = 3x1 single column vector

```
3.7321  
1.0000  
0.2679
```

```
Q = conv(P,P) % Convolve two vectors
```

Q = 1x7 single row vector

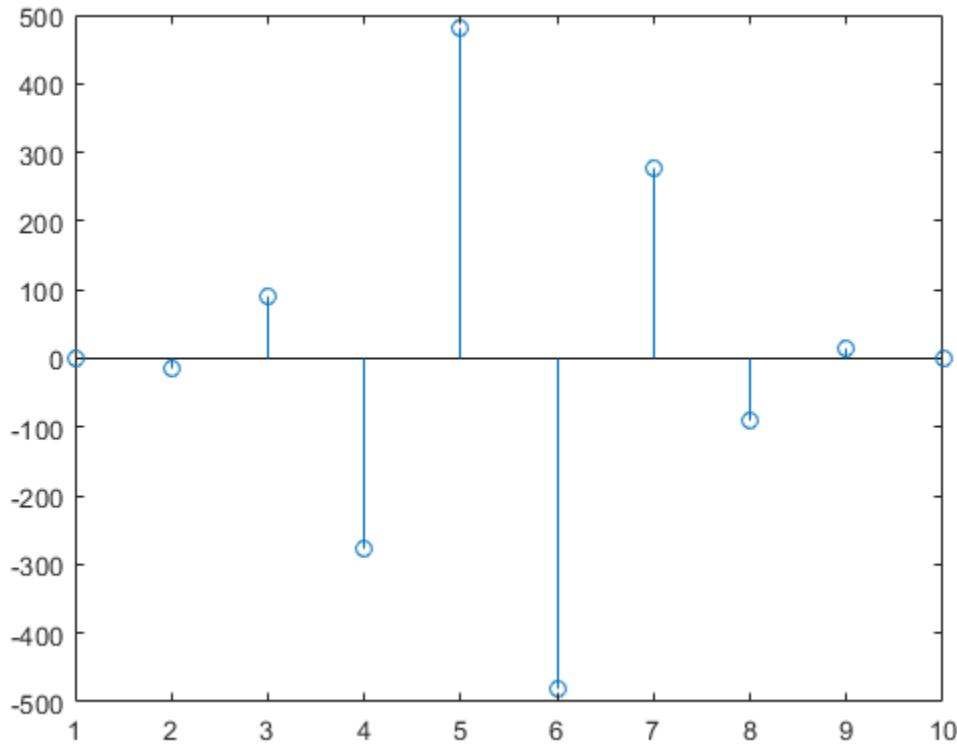
```
1 -10 35 -52 35 -10 1
```

```
R = conv(P,Q)
```

R = 1x10 single row vector

```
1 -15 90 -278 480 -480 278 -90 15 -1
```

```
stem(R); % Plot the result
```



### 用于处理单精度或双精度的一个程序

现在来看一个函数，该函数用于计算为使比率小于 single 或 double 数据类型的正确机器精度 (eps)，斐波那契数列需要的足够项数。

```
% How many terms needed to get single precision results?
fibodemo('single')
```

```
ans = 19
```

```
% How many terms needed to get double precision results?
fibodemo('double')
```

```
ans = 41
```

```
% Now let's look at the working code.
type fibodemo
```

```
function nterms = fibodemo(dtype)
%FIBODEMO Used by SINGLEMATH demo.
% Calculate number of terms in Fibonacci sequence.
```

```
% Copyright 1984-2014 The MathWorks, Inc.
```

```
fcurrent = ones(dtype);
fnext = fcurrent;
```

```
goldenMean = (ones(dtype)+sqrt(5))/2;
tol = eps(goldenMean);
nterms = 2;
while abs(fnext/fcurrent - goldenMean) >= tol
    nterms = nterms + 1;
    temp = fnext;
    fnext = fnext + fcurrent;
    fcurrent = temp;
end
```

请注意，我们初始化了几个变量，即 `fcurrent`、`fnext` 和 `goldenMean`，初始化所用的值取决于输入数据类型，容差 `tol` 也取决于该类型。与等效的双精度计算相比，单精度要求计算的项较少。

# 逻辑类

---

- “查找符合条件的数组元素” (第 5-2 页)
- “将逻辑数组约简为单个值” (第 5-6 页)

## 查找符合条件的数组元素

此示例说明如何通过对数组应用条件来筛选数组元素。例如，您可以检查矩阵中的偶数元素、查找多维数组中所有 0 值的位置，或者替换数据中的 NaN 值。您可以通过组合使用关系运算符和逻辑运算符来执行这些任务。关系运算符 ( $>$ 、 $<$ 、 $\geq$ 、 $\leq$ 、 $\equiv$ 、 $\sim\equiv$ ) 将不同的条件应用于数组，您可以使用逻辑运算符 **and**、**or** 和 **not** (分别用符号  $\&$ 、 $|$  和  $\sim$  表示) 将它们连接起来，从而应用多个条件。

### 应用单个条件

要应用单个条件，请首先创建一个  $5 \times 5$  矩阵，该矩阵包含介于 1 和 15 之间的随机整数。将随机数生成器重置为默认状态，以实现可再现性。

```
rng default
A = randi(15,5)
```

A = 5×5

13	2	3	3	10
14	5	15	7	1
2	9	15	14	13
14	15	8	12	15
10	15	13	15	11

使用小于号关系运算符  $<$  确定 A 中的哪些元素小于 9。将结果存储在 B 中。

```
B = A < 9
```

B = 5x5 logical array

0	1	1	1	0
0	1	0	1	1
1	0	0	0	0
0	0	1	0	0
0	0	0	0	0

结果为一个逻辑矩阵。B 中的每个值都表示为逻辑值 1 (true) 或逻辑值 0 (false) 的状态，以指示 A 的对应元素是否符合条件  $A < 9$ 。例如，A(1,1) 为 13，因此 B(1,1) 为逻辑值 0 (false)。但是，A(1,2) 为 2，因此 B(1,2) 为逻辑值 1 (true)。

虽然 B 包含有关 A 中哪些元素小于 9 的信息，但它不会指出这些元素的具体值是多少。您可以使用 B 创建 A 的索引，而不必逐元素比较这两个矩阵。

### A(B)

```
ans = 8×1
```

2
2
5
3
8
3
7
1

结果为一个由 A 中小于 9 的元素组成的列向量。由于 B 为逻辑矩阵，因此该运算称为**逻辑索引**。在本例中，用作索引的逻辑数组与另一个数组的大小相同，但这不是必需的。有关详细信息，请参阅“[数组索引](#)”。

某些问题需要符合条件的数组元素的位置信息，而非其实际值。在此示例中，您可以使用 **find** 函数查找 A 中小于 9 的所有元素。

```
I = find(A < 9)
```

```
I = 8×1
```

```
3
6
7
11
14
16
17
22
```

结果为一个由线性索引组成的列向量。每个索引描述 A 中一个小于 9 的元素的位置，因此实际上 A(I) 与 A(B) 返回的结果相同。差别在于 A(B) 使用逻辑索引，而 A(I) 使用线性索引。

## 应用多个条件

您可以使用逻辑 **and**、**or** 和 **not** 运算符将任意多个条件应用于一个数组；条件的数量并不局限于一个或两个。

首先，使用逻辑 **and** 运算符（由 **&** 表示）指定两个条件：元素必须**小于 9 且大于 2**。将这些条件指定为逻辑索引，以查看符合两个条件的元素。

```
A(A<9 & A>2)
```

```
ans = 5×1
```

```
5
3
8
3
7
```

结果为 A 中同时符合这两个条件的元素的列表。务必使用单独的语句指定每个条件，并用逻辑运算符连接起来。例如，您不能通过 A(2<A<9) 指定以上条件，因为其计算结果为 A(2<A | A<9)。

接下来，查找 A 中**小于 9 且为偶数**的元素。

```
A(A<9 & ~mod(A,2))
```

```
ans = 3×1
```

```
2
2
8
```

结果为 A 中小于 9 的所有偶数元素的列表。使用逻辑 NOT 运算符 ~ 将矩阵 `mod(A,2)` 转换为逻辑矩阵，并在可被 2 整除的元素位置放置逻辑值 1 (true)。

最后，查找 A 中小于 9、为偶数且不等于 2 的元素。

```
A(A<9 & ~mod(A,2) & A~=2)
```

```
ans = 8
```

结果 8 为偶数、小于 9 并且不等于 2。该元素是 A 中符合所有三个条件的唯一元素。

使用 `find` 函数获取等于 8 的元素（符合这些条件）的索引。

```
find(A<9 & ~mod(A,2) & A~=2)
```

```
ans = 14
```

结果指示  $A(14) = 8$ 。

### 替换符合条件的值

有时，同时更改多个现有数组元素的值会很有用。将逻辑索引与简单的赋值语句一起使用，可替换数组中符合条件的值。

将 A 中大于 10 的所有值替换为数值 10。

```
A(A>10) = 10
```

```
A = 5×5
```

```
10 2 3 3 10  
10 5 10 7 1  
2 9 10 10 10  
10 10 8 10 10  
10 10 10 10 10
```

然后，将 A 中不等于 10 的所有值替换为 NaN 值。

```
A(A~=10) = NaN
```

```
A = 5×5
```

```
10 NaN NaN NaN 10  
10 NaN 10 NaN NaN  
NaN NaN 10 10 10  
10 10 NaN 10 10  
10 10 10 10 10
```

最后，将 A 中的所有 NaN 值替换为 0，并应用逻辑 NOT 运算符 ~A。

```
A(isnan(A)) = 0;  
C = ~A
```

```
C = 5x5 logical array
```

```
0 1 1 1 0  
0 1 0 1 1
```

```
1 1 0 0 0  
0 0 1 0 0  
0 0 0 0 0
```

生成的矩阵用逻辑值 1 (true) 取代 NaN 值，用逻辑值 0 (false) 取代 10。逻辑 NOT 运算  $\sim A$  将数值数组转换为逻辑数组，因此  $A \& C$  返回逻辑值 0 (false) 的矩阵， $A \mid C$  返回逻辑值 1 (true) 的矩阵。

## 另请参阅

Logical Operators: Short Circuit | and | find | isnan | nan | not | or | xor

## 将逻辑数组约简为单个值

此示例说明如何使用 `any` 和 `all` 函数将整个数组约简为单个逻辑值。

`any` 和 `all` 函数分别是逻辑 `|` (OR) 和 `&` (AND) 运算符的自然扩展。但是，`any` 和 `all` 函数会比较一个数组的特定维度中的所有元素，而不是仅比较两个元素。就好像通过 `&` 或 `|` 运算符将所有元素连接起来，而使用 `any` 或 `all` 函数计算连接后的长逻辑表达式一样。因此，与核心逻辑运算符不同，`any` 和 `all` 函数可将所处理的数组维度的大小约简为 1。这样，便可以将许多逻辑值约简为单个逻辑条件。

首先，创建一个矩阵 `A`，该矩阵包含介于 1 和 25 之间的随机整数。将随机数生成器重置为默认状态，以实现可再现性。

```
rng default
A = randi(25,5)
```

`A = 5x5`

```
21   3   4   4   17
23   7   25  11   1
 4  14   24  23  22
23  24   13  20  24
16  25   21  24  17
```

接下来，使用 `mod` 函数以及逻辑 NOT 运算符 `~` 确定 `A` 中的哪些元素为偶数。

```
A = ~mod(A,2)
```

`A = 5x5 logical array`

```
0  0  1  1  0
0  0  0  0  0
1  1  1  0  1
0  1  0  1  1
1  0  0  1  0
```

在生成的矩阵中，当元素为偶数时，逻辑值为 1 (`true`)，当元素为奇数时，逻辑值为 0 (`false`)。

由于 `any` 和 `all` 函数可将所处理的维度的大小约简为 1，因此通常需要将其中一个函数应用两次，从而将二维矩阵约简为单个逻辑条件，例如 `any(any(A))`。但是，如果您使用表示法 `A(:)` 将 `A` 的所有元素视为单个列向量，则可以使用 `any(A(:))` 获取相同的逻辑信息，而不必嵌套函数调用。

确定 `A` 中是否有任何元素为偶数。

```
any(A(:))
```

```
ans = logical
1
```

您可以在对 `any` 或 `all` 的函数调用中执行逻辑比较和关系比较。这样便于快速测试数组的各种属性。

确定 `A` 中的所有元素是否都为奇数。

```
all(~A(:))
```

```
ans = logical  
0
```

确定 A 中是否有任何主对角线元素或上对角线元素为偶数。由于 `diag(A)` 和 `diag(A,1)` 返回的向量大小不同，因此在比较它们之前，首先需要将每条对角线约简为单个标量逻辑条件。您可以使用短路 OR 运算符 `||` 来执行比较，因为如果第一条对角线上的任何元素为偶数，则无论运算符右侧出现什么，整个表达式的计算结果都为 `true`。

```
any(diag(A)) || any(diag(A,1))
```

```
ans = logical  
1
```

## 另请参阅

Logical Operators: Short Circuit | `all` | `and` | `any` | `or` | `xor`



# 字符和字符串

---

- “字符串数组和字符数组中的文本” (第 6-2 页)
- “创建字符串数组” (第 6-5 页)
- “字符向量元胞数组” (第 6-12 页)
- “分析字符串数组的文本数据” (第 6-15 页)
- “测试空字符串和缺失值” (第 6-20 页)
- “格式化文本” (第 6-24 页)
- “比较文本” (第 6-31 页)
- “搜索和替换文本” (第 6-36 页)
- “从数值转换为字符数组” (第 6-41 页)
- “从字符数组转换为数值” (第 6-43 页)
- “十六进制和二进制值” (第 6-45 页)
- “有关字符串数组的常见问题解答” (第 6-48 页)
- “更新您的代码以接受字符串” (第 6-52 页)
- “函数摘要” (第 6-59 页)

## 字符串数组和字符数组中的文本

MATLAB® 中有两种表示文本的方式。从 R2016b 开始，您可以将文本存储在字符串数组中。在任何版本的 MATLAB 中，您都可以将文本存储在字符数组中。字符数组的一个典型用途是将文本片段存储为字符向量。MATLAB 用双引号显示字符串，用单引号显示字符向量。

### 用字符串数组表示文本

您可以使用 `string` 数据类型将任何  $1 \times n$  字符序列存储为字符串。从 R2017a 开始，您可以用双引号将文本括起来以创建字符串。

```
str = "Hello, world"
```

```
str =  
"Hello, world"
```

虽然文本 "Hello, world" 的长度为 12 个字符，但 `str` 本身是  $1 \times 1$  字符串或字符串标量。您可以使用字符串标量来指定文件名、图标签或任何其他文本信息片段。

要计算字符串中的字符数量，请使用 `strlength` 函数。

```
n = strlength(str)
```

```
n = 12
```

如果文本包含双引号，请在定义中使用两个双引号。

```
str = "They said, ""Welcome!"" and waved."
```

```
str =  
"They said, \"Welcome!\" and waved."
```

要将文本添加到字符串的末尾，请使用加号运算符 `+`。如果变量可以转换为字符串，则 `plus` 会转换并追加它。

```
fahrenheit = 71;  
celsius = (fahrenheit-32)/1.8;  
tempText = "temperature is " + celsius + "C"  
  
tempText =  
"temperature is 21.6667C"
```

从 R2019a 开始，您还可以使用 `append` 函数串联文本。

```
tempText2 = append("Today's ",tempText)  
  
tempText2 =  
"Today's temperature is 21.6667C"
```

`string` 函数可以转换不同类型的输入，如数值、日期时间、持续时间和分类值。例如，将 `pi` 的输出转换为字符串。

```
ps = string(pi)  
  
ps =  
"3.1416"
```

您可以将多个文本片段存储在字符串数组中。数组的每个元素都可以包含一个具有不同字符数的字符串，而无需填充。

```
str = ["Mercury","Gemini","Apollo";...
        "Skylab","Skylab B","ISS"]
```

```
str = 2x3 string array
    "Mercury"   "Gemini"   "Apollo"
    "Skylab"    "Skylab B"  "ISS"
```

**str** 是一个  $2 \times 3$  的字符串数组。您可以使用 **strlength** 函数计算字符串的长度。

```
N = strlength(str)
```

```
N = 2×3
```

```
7   6   6
6   8   3
```

从 R2018b 开始，MATLAB 和 MathWorks® 全线产品都支持字符串数组。接受字符数组（和字符向量元胞数组）作为输入的函数也接受字符串数组。

## 用字符向量表示文本

要使用 **char** 数据类型将  $1 \times n$  字符序列存储为字符向量，请用单引号将它引起来。

```
chr = 'Hello, world'
```

```
chr =
'Hello, world'
```

文本 'Hello, world' 的长度为 12 个字符，**chr** 将其存储为  $1 \times 12$  字符向量。

```
whos chr
```

Name	Size	Bytes	Class	Attributes
chr	$1 \times 12$	24	char	

如果文本包含单引号，请在定义中使用双重单引号。

```
chr = 'They said, "Welcome!" and waved.'
```

```
chr =
'They said, 'Welcome!' and waved.'
```

字符向量有两个主要用途：

- 指定单个文本片段，如文件名和图标签。
- 表示使用字符进行编码的数据。在这种情况下，您可能需要尽可能方便地访问单个字符。

例如，您可以将 DNA 序列存储为一个字符向量。

```
seq = 'GCTAGAATCC';
```

您可以通过索引来访问单个字符或字符子集，就像对数值数组进行索引一样。

```
seq(4:6)
```

```
ans =
'AGA'
```

用方括号串联字符向量，就像串联其他类型的数组一样。

```
seq2 = [seq 'ATTAGAAACC']  
  
seq2 =  
'GCTAGAATCCATTAGAAACC'
```

从 R2019a 开始，您还可以使用 **append** 串联文本。推荐使用 **append** 函数，因为它以一致的方式处理字符串数组、字符向量和字符向量元胞数组。

```
seq2 = append(seq,'ATTAGAAACC')  
  
seq2 =  
'GCTAGAATCCATTAGAAACC'
```

接受字符串数组作为输入的 MATLAB 函数也接受字符向量和字符向量元胞数组。

### 另请参阅

[append](#) | [cellstr](#) | [char](#) | [horzcat](#) | [plus](#) | [string](#) | [strlength](#)

### 相关示例

- “[创建字符串数组](#)”（第 6-5 页）
- “[分析字符串数组的文本数据](#)”（第 6-15 页）
- “[有关字符串数组的常见问题解答](#)”（第 6-48 页）
- “[更新您的代码以接受字符串](#)”（第 6-52 页）
- “[字符向量元胞数组](#)”（第 6-12 页）

## 创建字符串数组

R2016b 中引入了字符串数组。字符串数组可存储文本片段，并提供一组用于将文本按数据进行处理的函数。您可以对字符串数组进行索引、重构和进行串联，就像处理任何其他类型的数组一样。此外，还可以访问字符串中的字符，并使用 **plus** 运算符向字符串追加文本。要重新排列字符串数组中的字符串，请使用 **split**、**join** 和 **sort** 等函数。

### 根据变量创建字符串数组

MATLAB® 提供字符串数组来存储文本片段。字符串数组的每个元素都包含一个  $1 \times n$  字符序列。

从 R2017a 开始，您可以使用双引号创建字符串。

```
str = "Hello, world"
```

```
str =
"Hello, world"
```

作为备选方法，您可以使用 **string** 函数将字符向量转换为字符串。**chr** 为一个  $1 \times 17$  字符向量。**str** 为一个与该字符向量具有相同文本的  $1 \times 1$  字符串。

```
chr = 'Greetings, friend'
```

```
chr =
'Greetings, friend'
```

```
str = string(chr)
```

```
str =
"Greetings, friend"
```

使用 **[]** 运算符创建一个包含多个字符串的字符串数组。**str** 是一个  $2 \times 3$  字符串数组，其中包含六个字符串。

```
str = ["Mercury", "Gemini", "Apollo";
       "Skylab", "Skylab B", "ISS"]
```

```
str = 2x3 string array
      "Mercury"   "Gemini"    "Apollo"
      "Skylab"     "Skylab B"   "ISS"
```

通过 **strlength** 函数计算 **str** 中的每个字符串的长度。使用 **strlength** 而非 **length** 来确定字符串中的字符数量。

```
L = strlength(str)
```

```
L = 2×3
```

7	6	6
6	8	3

作为备选方法，您可以使用 **string** 函数将字符向量元胞数组转换为字符串数组。MATLAB 使用双引号显示字符串数组中的字符串，使用单引号显示元胞数组中的字符向量。

```
C = {'Mercury', 'Venus', 'Earth'}
```

```
C = 1x3 cell array
{'Mercury'}  {'Venus'}  {'Earth'}
```

```
str = string(C)
```

```
str = 1x3 string array
"Mercury"  "Venus"  "Earth"
```

除了字符向量，您还可以使用 **string** 函数将数值、日期时间、持续时间和分类值转换为字符串。

将数值数组转换为字符串数组。

```
X = [5 10 20 3.1416];
string(X)
```

```
ans = 1x4 string array
"5"  "10"  "20"  "3.1416"
```

将日期时间值转换为字符串。

```
d = datetime('now');
string(d)
```

```
ans =
"30-Jul-2019 17:12:01"
```

此外，您也可以使用 **readtable**、**textscan** 和 **fscanf** 函数将文件中的文本读入字符串数组。

### 创建空字符串和缺失字符串

字符串数组可以包含空值和缺失值。空字符串不包含字符。当您显示空字符串时，结果为一对其间不包含任何内容的双引号 ("")。缺失字符串相当于数值数组的 NaN。它指示字符串数组包含缺失值的位置。当您显示缺失字符串时，结果为 <missing>，且不带任何引号。

使用 **strings** 函数创建一个空字符串数组。在不带任何参数的情况下调用 **strings** 时，将会返回一个空字符串。请注意，**str** 的大小是  $1 \times 1$ ，而不是  $0 \times 0$ 。但是，**str** 不包含字符。

```
str = strings
```

```
str =
""
```

使用单引号创建一个空字符向量。请注意，**chr** 的大小是  $0 \times 0$ 。

```
chr = ''
```

```
chr =
```

```
0x0 empty char array
```

创建一个所有元素都是空字符串的字符串数组。您可以使用 **strings** 函数预分配一个字符串数组。

```
str = strings(2,3)
```

```
str = 2x3 string array
     ""    ""    ""
```

```
"" "" "
```

要创建缺失字符串，请使用 `string` 函数转换缺失值。缺失字符串显示为 `<missing>`。

```
str = string(missing)

str =
<missing>
```

您可以创建同时包含空字符串和缺失字符串的字符串数组。使用 `ismissing` 函数确定哪些元素为包含缺失值的字符串。请注意，空字符串不是缺失字符串。

```
str(1) = "";
str(2) = "Gemini";
str(3) = string(missing)

str = 1x3 string array
    "" "Gemini" <missing>
```

```
ismissing(str)

ans = 1x3 logical array

0 0 1
```

将一个缺失字符串与另一个字符串进行比较。结果始终为 `0 (false)`，即使您将一个缺失字符串与另一个缺失字符串进行比较也是如此。

```
str = string(missing);
str == "Gemini"

ans = logical
0

str == string(missing)

ans = logical
0
```

## 访问字符串数组的元素

字符串数组支持数组运算，例如进行索引和重构。使用数组索引访问 `str` 的第一行以及所有列。

```
str = ["Mercury","Gemini","Apollo";
       "Skylab","Skylab B","ISS"];
str(1,:)

ans = 1x3 string array
    "Mercury" "Gemini" "Apollo"
```

访问 `str` 的第二行中的第二个元素。

```
str(2,2)
```

```
ans =  
"Skylab B"
```

在 str 的边界之外分配一个新字符串。MATLAB 将扩展该数组并使用缺失值填充未分配的元素。

```
str(3,4) = "Mir"
```

```
str = 3x4 string array  
"Mercury" "Gemini" "Apollo" <missing>  
"Skylab" "Skylab B" "ISS" <missing>  
<missing> <missing> <missing> "Mir"
```

### 访问字符串中的字符

您可以使用花括号 {} 对字符串数组进行索引以直接访问字符。当您需要访问和修改字符串元素中的字符时，请使用花括号。通过花括号进行索引为可处理字符串数组或字符向量元胞数组的代码提供了兼容性。但是只要有可能，请尽量使用字符串函数来处理字符串中的字符。

使用花括号访问第二行中的第二个元素。chr 是一个字符向量，而不是字符串。

```
str = ["Mercury","Gemini","Apollo";  
       "Skylab","Skylab B","ISS"];  
chr = str{2,2}
```

```
chr =  
'Skylab B'
```

访问该字符向量并返回前三个字符。

```
str{2,2}{1:3}
```

```
ans =  
'Sky'
```

查找字符串中的空格字符，并将这些字符替换为短划线。使用 isspace 函数检查字符串中的单个字符。isspace 会返回一个逻辑向量，只要存在一个空格字符，该向量即包含一个 true 值。最后，显示修改后的字符串元素 str(2,2)。

```
TF = isspace(str{2,2})
```

```
TF = 1x8 logical array
```

```
0 0 0 0 0 0 1 0
```

```
str{2,2}(TF) = "_";  
str(2,2)
```

```
ans =  
"Skylab-B"
```

请注意，在该示例中，您也可以使用 replace 函数来替换空格，而无需借助于花括号索引。

```
replace(str(2,2)," ","_")
```

```
ans =  
"Skylab-B"
```

## 将字符串串联到字符串数组中

将字符串串联到字符串数组中，就像您串联任何其他类型的数组一样。

使用方括号 [] 串联两个字符串数组。

```
str1 = ["Mercury","Gemini","Apollo"];
str2 = ["Skylab","Skylab B","ISS"];
str = [str1 str2]

str = 1x6 string array
"Mercury"  "Gemini"  "Apollo"  "Skylab"  "Skylab B"  "ISS"
```

转置 str1 和 str2。将它们进行串联，然后将列标题垂直串联到字符串数组上。当您将字符向量串联到字符串数组中时，字符向量会自动转换为字符串。

```
str1 = str1';
str2 = str2';
str = [str1 str2];
str = [{"Mission:","Station:"} ; str]

str = 4x2 string array
"Mission:"  "Station:"
"Mercury"  "Skylab"
"Gemini"  "Skylab B"
"Apollo"  "ISS"
```

## 向字符串追加文本

要向字符串追加文本，请使用 plus 运算符 +。plus 运算符可向字符串追加文本，但不会更改字符串数组的大小。

在名字数组中追加一个姓氏。如果您将某个字符向量追加到字符串，则该字符向量将自动转换为字符串。

```
names = ["Mary";"John";"Elizabeth";"Paul";"Ann"];
names = names + ' Smith'

names = 5x1 string array
"Mary Smith"
"John Smith"
"Elizabeth Smith"
"Paul Smith"
"Ann Smith"
```

追加不同的姓氏。您可以将文本从一个字符串数组或字符向量元胞数组追加到另一个字符串数组。当您添加非标量数组时，这些数组的大小必须相同。

```
names = ["Mary";"John";"Elizabeth";"Paul";"Ann"];
lastnames = ["Jones";"Adams";"Young";"Burns";"Spencer"];
names = names + " " + lastnames

names = 5x1 string array
"Mary Jones"
"John Adams"
"Elizabeth Young"
"Paul Burns"
```

```
"Ann Spencer"
```

追加缺失字符串。当您使用加号运算符追加缺失字符串时，输出为一个缺失字符串。

```
str1 = "Jones";
str2 = string(missing);
str1 + str2

ans =
<missing>
```

### 拆分、联接和排序字符串数组

MATLAB 提供了一组丰富的函数来处理字符串数组。例如，您可以使用 `split`、`join` 和 `sort` 函数重新排列字符串数组 `names`，以使名字按姓氏的字母顺序排列。

按空格字符拆分 `names`。拆分会将 `names` 从  $5 \times 1$  字符串数组更改为  $5 \times 2$  数组。

```
names = ["Mary Jones"; "John Adams"; "Elizabeth Young"; "Paul Burns"; "Ann Spencer"];
names = split(names)

names = 5x2 string array
"Mary"      "Jones"
"John"      "Adams"
"Elizabeth" "Young"
"Paul"      "Burns"
"Ann"       "Spencer"
```

将 `names` 的列交换位置，使姓在第一列。在每个姓后面添加一个逗号。

```
names = [names(:,2) names(:,1)];
names(:,1) = names(:,1) + ','

names = 5x2 string array
"Jones,"    "Mary"
"Adams,"   "John"
"Young,"   "Elizabeth"
"Burns,"   "Paul"
"Spencer," "Ann"
```

将姓和名联接起来。`join` 函数在它联接的字符串之间放置一个空格字符。进行联接之后，`names` 为一个  $5 \times 1$  字符串数组。

```
names = join(names)

names = 5x1 string array
"Jones, Mary"
"Adams, John"
"Young, Elizabeth"
"Burns, Paul"
"Spencer, Ann"
```

对 `names` 的元素进行排序，使它们按字母顺序排列。

```
names = sort(names)
```

```
names = 5x1 string array
    "Adams, John"
    "Burns, Paul"
    "Jones, Mary"
    "Spencer, Ann"
    "Young, Elizabeth"
```

## 另请参阅

[ismissing](#) | [isspace](#) | [join](#) | [plus](#) | [sort](#) | [split](#) | [string](#) | [strings](#) | [strlength](#)

## 相关示例

- “分析字符串数组的文本数据”（第 6-15 页）
- “搜索和替换文本”（第 6-36 页）
- “比较文本”（第 6-31 页）
- “测试空字符串和缺失值”（第 6-20 页）
- “有关字符串数组的常见问题解答”（第 6-48 页）
- “更新您的代码以接受字符串”（第 6-52 页）

## 字符向量元胞数组

要将文本存储为字符向量，请将其用单引号引起。通常，字符向量包含您视为单个信息片段的文本，例如文件名或图标签。如果您有多个文本片段，例如文件名列表，则您可以将它们存储在元胞数组中。如果一个元胞数组的元素均为字符向量，则它是字符向量元胞数组。

### 注意

- 从 R2018b 开始，推荐的文本存储方式是使用字符串数组。如果您创建具有 **string** 数据类型的变量，请将它们存储在字符串数组而不是元胞数组中。有关详细信息，请参阅“字符串数组和字符数组中的文本”（第 6-2 页）和“更新您的代码以接受字符串”（第 6-52 页）。
- 虽然我们过去常用字符串元胞数组来描述此类元胞数组，但这种描述不再准确，因为此类元胞数组包含的是字符向量而不是字符串。

## 创建字符向量元胞数组

要创建字符向量元胞数组，请使用花括号 {}，就像创建任何元胞数组一样。例如，使用字符向量元胞数组来存储名称列表。

```
C = {'Li','Sanchez','Jones','Yang','Larson'}
```

```
C = 1x5 cell array
    {'Li'}  {'Sanchez'}  {'Jones'}  {'Yang'}  {'Larson'}
```

**C** 中的字符向量可以有不同长度，因为元胞数组不要求其内容具有相同的大小。要确定 **C** 中字符向量的长度，请使用 **strlength** 函数。

```
L = strlength(C)
```

```
L = 1×5
```

```
2    7    5    4    6
```

## 访问元胞数组中的字符向量

要访问元胞数组中的字符向量，请使用花括号 {} 对其进行索引。提取第一个元胞的内容，并将其存储为字符向量。

```
C = {'Li','Sanchez','Jones','Yang','Larson'};
chr = C{1}
```

```
chr =
'Li'
```

使用另一个字符向量为第一个元胞赋值。

```
C{1} = 'Yang'
```

```
C = 1x5 cell array
    {'Yang'}  {'Sanchez'}  {'Jones'}  {'Yang'}  {'Larson'}
```

要引用元胞的子集而不是其内容，请使用圆括号进行索引。

**C(1:3)**

```
ans = 1x3 cell array
    {'Yang'}  {'Sanchez'}  {'Jones'}
```

虽然您可以通过索引来访问元胞的内容，但是，大多数接受元胞数组作为输入的函数对整个元胞数组进行操作。例如，您可以使用 **strcmp** 函数将 **C** 的内容与字符串进行比较。如果匹配，**strcmp** 返回 1，否则返回 0。

**TF = strcmp(C,'Yang')**

**TF = 1x5 logical array**

```
1  0  0  1  0
```

您可以对 **TF** 求和，以求出匹配数。

**num = sum(TF)**

**num = 2**

使用 **TF** 作为逻辑索引以返回 **C** 中的匹配项。如果使用圆括号进行索引，则输出是只包含匹配项的元胞数组。

**M = C(TF)**

```
M = 1x2 cell array
    {'Yang'}  {'Yang'}
```

## 将元胞数组转换为字符串数组

从 R2018b 开始，MATLAB® 和 MathWorks® 全线产品都支持字符串数组。因此，建议您使用字符串数组而不是字符串元胞数组。（不过，接受字符串数组作为输入的 MATLAB 函数也接受字符向量和字符串元胞数组。）

您可以将字符串元胞数组转换为字符串数组。要转换字符串元胞数组，请使用 **string** 函数。

**C = {'Li','Sanchez','Jones','Yang','Larson'}**

```
C = 1x5 cell array
    {'Li'}  {'Sanchez'}  {'Jones'}  {'Yang'}  {'Larson'}
```

**str = string(C)**

```
str = 1x5 string array
    "Li"  "Sanchez"  "Jones"  "Yang"  "Larson"
```

事实上，**string** 函数可以转换任何元胞数组，只要数组中的内容都能被转换为字符串即可。

**C2 = {5, 10, 'some text', datetime('today')}**

```
C2=1×4 cell
{[5]} {[10]} {"some text"} {[30-Jul-2019]}
```

```
str2 = string(C2)
```

```
str2 = 1x4 string array
"5" "10" "some text" "30-Jul-2019"
```

### 另请参阅

[cellstr](#) | [char](#) | [iscellstr](#) | [strcmp](#) | [string](#)

### 详细信息

- “字符串数组和字符数组中的文本” (第 6-2 页)
- “访问元胞数组中的数据” (第 12-5 页)
- “创建字符串数组” (第 6-5 页)
- “更新您的代码以接受字符串” (第 6-52 页)
- “有关字符串数组的常见问题解答” (第 6-48 页)

## 分析字符串数组的文本数据

以下示例演示如何以字符串数组形式存储文件中的文本、按单词频率对其进行排序、绘制结果图，以及收集文件中找到的单词的基本统计信息。

### 将文本文件导入字符串数组

使用 `fileread` 函数读取莎士比亚的十四行诗中的文本。`fileread` 会以  $1 \times 100266$  字符向量的形式返回文本。

```
sonnets = fileread('sonnets.txt');
sonnets(1:35)

ans =
'THE SONNETS

by William Shakespeare'
```

使用 `string` 函数将文本转换为字符串。然后，使用 `splitlines` 函数按换行符对其进行拆分。`sonnets` 将变成一个  $2625 \times 1$  字符串数组，其中每个字符串都包含这些诗中的一行。显示 `sonnets` 的前五行。

```
sonnets = string(sonnets);
sonnets = splitlines(sonnets);
sonnets(1:5)

ans = 5x1 string array
"THE SONNETS"
""
"by William Shakespeare"
""
""
```

### 清理字符串数组

要计算 `sonnets` 中的单词的频率，请首先删除空字符串和标点符号对其进行清理。然后，将其重构为一个以元素形式包含单个单词的字符串数组。

从该字符串数组中删除不含字符 ("") 的字符串。将 `sonnets` 的每个元素都与 "" (空字符串) 进行比较。从 R2017a 开始，您可以使用双引号创建字符串（包括空字符串）。`TF` 是一个逻辑向量，如果 `sonnets` 包含一个不含字符的字符串，该向量相应位置即包含一个 `true` 值。使用 `TF` 对 `sonnets` 进行索引，然后删除所有不含字符的字符串。

```
TF = (sonnets == "");
sonnets(TF) = [];
sonnets(1:10)

ans = 10x1 string array
"THE SONNETS"
"by William Shakespeare"
" I"
" From fairest creatures we desire increase,"
" That thereby beauty's rose might never die,"
" But as the riper should by time decease,"
" His tender heir might bear his memory:"
" But thou, contracted to thine own bright eyes,"
```

```
" Feed'st thy light's flame with self-substantial fuel,"  
" Making a famine where abundance lies,"
```

将一些标点符号替换为空格字符。例如，替换句点、逗号和分号。保留撇号，因为它们可能是十四行诗中的某些单词的一部分，例如 light's。

```
p = [".","?",",",";","'"];  
sonnets = replace(sonnets,p," ");  
sonnets(1:10)
```

```
ans = 10x1 string array  
"THE SONNETS"  
"by William Shakespeare"  
" I"  
" From fairest creatures we desire increase "  
" That thereby beauty's rose might never die "  
" But as the riper should by time decease "  
" His tender heir might bear his memory "  
" But thou contracted to thine own bright eyes "  
" Feed'st thy light's flame with self-substantial fuel "  
" Making a famine where abundance lies "
```

去除 sonnets 的每个元素中的前导和尾随空格字符。

```
sonnets = strip(sonnets);  
sonnets(1:10)
```

```
ans = 10x1 string array  
"THE SONNETS"  
"by William Shakespeare"  
"I"  
"From fairest creatures we desire increase"  
"That thereby beauty's rose might never die"  
"But as the riper should by time decease"  
"His tender heir might bear his memory"  
"But thou contracted to thine own bright eyes"  
"Feed'st thy light's flame with self-substantial fuel"  
"Making a famine where abundance lies"
```

将 sonnets 拆分为以单个单词为元素的字符串数组。您可以使用 **split** 函数，根据空白字符或所指定的分隔符拆分字符串数组的元素。然而，**split** 要求字符串数组的每个元素都能拆分为相同数目的新字符串。**sonnets** 的元素包含不同数目的空格，因此不能拆分为相同数目的字符串。要对 **sonnets** 使用 **split** 函数，请编写一个 **for** 循环，以便每次对一个元素调用 **split**。

使用 **strings** 函数创建空字符串数组 **sonnetWords**。编写一个 **for** 循环，以使用 **split** 函数拆分 **sonnets** 的每个元素。将 **split** 的输出串联到 **sonnetWords** 中。**sonnetWords** 的每个元素都是 **sonnets** 中的单个单词。

```
sonnetWords = strings(0);  
for i = 1:length(sonnets)  
    sonnetWords = [sonnetWords ; split(sonnets(i))];  
end  
sonnetWords(1:10)
```

```
ans = 10x1 string array  
"THE"
```

```
"SONNETS"
"by"
"William"
"Shakespeare"
"I"
"From"
"fairest"
"creatures"
"we"
```

### 根据频率对单词进行排序

查找 `sonnetWords` 中的唯一单词。计算单词的数量并根据其频率进行排序。

要将只有大小写不同的单词计算为同一个单词，请将 `sonnetWords` 转换为小写。例如，`The` 和 `the` 计算为同一个单词。使用 `unique` 函数查找唯一的单词。然后，使用 `histcounts` 函数计算每个唯一单词出现的次数。

```
sonnetWords = lower(sonnetWords);
[words,~,idx] = unique(sonnetWords);
numOccurrences = histcounts(idx,numel(words));
```

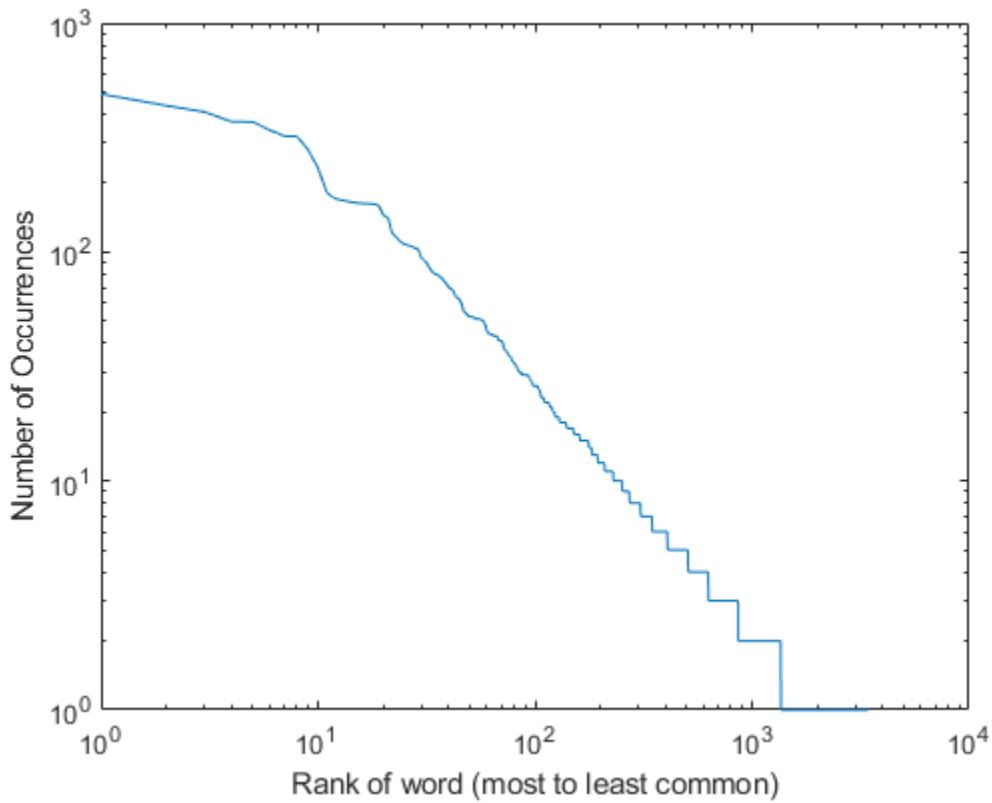
对 `sonnetWords` 中的单词按出现的次数（从最常见到最不常见）进行排序。

```
[rankOfOccurrences,rankIndex] = sort(numOccurrences,'descend');
wordsByFrequency = words(rankIndex);
```

### 绘制单词频率图

从最常见的单词到最不常见的单词绘制 Sonnets 中单词的出现情况图。Zipf 定律指出，大型主体文本中的单词出现情况分布遵循幂律分布。

```
loglog(rankOfOccurrences);
xlabel('Rank of word (most to least common)');
ylabel('Number of Occurrences');
```



显示 Sonnets 中最常见的十个单词。

**wordsByFrequency(1:10)**

```
ans = 10x1 string array
"and"
"the"
"to"
"my"
"of"
"I"
"in"
"that"
"thy"
"thou"
```

### 用表收集基本统计信息

计算 `sonnetWords` 中每个单词的总出现次数。计算出现次数占单词总数的百分比，并计算从最常见到最不常见的累积百分比。将单词和单词的基本统计信息写入表中。

```
numOccurrences = numOccurrences(rankIndex);
numOccurrences = numOccurrences';
numWords = length(sonnetWords);
T = table;
T.Words = wordsByFrequency;
T.NumOccurrences = numOccurrences;
```

```
T.PercentOfText = numOccurrences / numWords * 100.0;
T.CumulativePercentOfText = cumsum(numOccurrences) / numWords * 100.0;
```

显示最常见的十个单词的统计信息。

**T(1:10,:)**

ans=10×4 table

Words	NumOccurrences	PercentOfText	CumulativePercentOfText
"and"	490	2.7666	2.7666
"the"	436	2.4617	5.2284
"to"	409	2.3093	7.5377
"my"	371	2.0947	9.6324
"of"	370	2.0891	11.722
"I"	341	1.9254	13.647
"in"	321	1.8124	15.459
"that"	320	1.8068	17.266
"thy"	280	1.5809	18.847
"thou"	233	1.3156	20.163

十四行诗中的最常见单词 and 出现了 490 次。最常见的十个单词加起来占文本的 20.163%。

## 另请参阅

[histcounts](#) | [join](#) | [lower](#) | [replace](#) | [sort](#) | [split](#) | [splitlines](#) | [string](#) | [strip](#) | [table](#) | [unique](#)

## 相关示例

- “[创建字符串数组](#)” (第 6-5 页)
- “[搜索和替换文本](#)” (第 6-36 页)
- “[比较文本](#)” (第 6-31 页)
- “[测试空字符串和缺失值](#)” (第 6-20 页)

## 测试空字符串和缺失值

字符串数组可以同时包含空字符串和缺失值。空字符串不包含字符，并且显示为其间不包含任何内容的双引号 ("")。您可以使用 == 运算符来确定某个字符串是否为空字符串。空字符串是其他所有字符串的子字符串。因此，**contains** 等函数始终会在其他字符串中查找空字符串。字符串数组还可以包含缺失值。字符串数组中的缺失值显示为 <missing>。要查找字符串数组中的缺失值，请使用 **ismissing** 函数而不是 == 运算符。

### 测试空字符串

您可以使用 == 运算符来检测字符串数组中的空字符串。

从 R2017a 开始，您可以使用其间不包含任何内容的双引号来创建空字符串 ("")。请注意，**str** 的大小是  $1 \times 1$ ，而不是  $0 \times 0$ 。但是，**str** 不包含字符。

```
str = ""
str =
""
```

使用单引号创建一个空字符串向量。请注意，**chr** 的大小是  $0 \times 0$ 。字符数组 **chr** 实际上是一个空数组，而不仅仅是一个不包含字符的数组。

```
chr = ''
chr =
0x0 empty char array
```

使用 **strings** 函数创建一个空字符串数组。该数组的每个元素都是一个不包含任何字符的字符串。

```
str2 = strings(1,3)
str2 =
    ""    ""    ""
```

将 **str** 与一个空字符串进行比较以测试其是否为空字符串。

```
if (str == "")
    disp 'str has zero characters'
end

str has zero characters
```

请不要使用 **isempty** 函数来测试空字符串。不包含字符的字符串的大小仍然是  $1 \times 1$ 。但是，您可以使用 **isempty** 函数来测试字符串数组是否至少具有一个大小为零的维度。

使用 **strings** 函数创建一个空字符串数组。要成为空数组，至少一个维度的大小必须为零。

```
str = strings(0,3)
str =
0x3 empty string array
```

使用 **isempty** 函数测试 **str**。

```

isempty(str)

ans = logical
1

测试字符串数组中的空字符串。== 运算符会返回一个大小与字符串数组相同的逻辑数组。

str = ["Mercury","","Apollo"]

str = 1x3 string array
"Mercury"    ""    "Apollo"

str == ""

ans = 1x3 logical array

0 1 0

```

### 在其他字符串中查找空字符串

字符串始终包含空字符串作为子字符串。实际上，空字符串总是位于每个字符串的开头和结尾。此外，在字符串中的任意两个连续字符之间也总能找到空字符串。

创建一个字符串。然后，测试其是否包含空字符串。

```

str = "Hello, world";
TF = contains(str,"")

TF = logical
1

```

测试 str 是否以空字符串开头。

```

TF = startsWith(str,"")

TF = logical
1

```

计算 str 中的字符数量。然后，计算 str 中的空字符串数量。**count** 函数会计算位于 str 的开头和结尾以及每对字符之间的空字符串的数量。因此，如果 str 包含 N 个字符，则它同时包含 N+1 个空字符串。

```

str

str =
"Hello, world"

strlength(str)

ans = 12

count(str,"")

ans = 13

```

将一个子字符串替换为空字符串。当您使用空字符串调用 **replace** 时，它会删除该子字符串并将其替换为不包含字符的字符串。

```
replace(str,"world","")
```

```
ans =  
"Hello,"
```

使用 `insertAfter` 函数在空字符串后面插入一个子字符串。由于每对字符之间都存在空字符串，因此 `insertAfter` 会在每对字符之间插入子字符串。

```
insertAfter(str,"","-")
```

```
ans =  
"-H-e-l-l-o-,- -w-o-r-l-d-"
```

通常，用于替换、擦除、提取或插入子字符串的字符串函数允许您将空字符串指定为要修改的子字符串的开头和结尾。当您执行此操作时，这些函数将作用于字符串的开头和结尾以及每对字符之间。

### 测试缺失值

您可以使用 `ismissing` 函数来检测字符串数组中的缺失值。缺失字符串相当于数值数组的 `NaN`。它指示字符串数组包含缺失值的位置。缺失字符串显示为 `<missing>`。

要创建缺失字符串，请使用 `string` 函数转换缺失值。

```
str = string(missing)
```

```
str =  
<missing>
```

您可以创建同时包含空字符串和缺失字符串的字符串数组。使用 `ismissing` 函数确定哪些元素为包含缺失值的字符串。请注意，空字符串不是缺失字符串。

```
str(1) = "";  
str(2) = "Gemini";  
str(3) = string(missing)
```

```
str = 1x3 string array  
"" "Gemini" <missing>
```

```
ismissing(str)
```

```
ans = 1x3 logical array
```

```
0 0 1
```

将 `str` 与缺失字符串进行比较。比较结果始终为 `0 (false)`，即使您将一个缺失字符串与另一个缺失字符串进行比较也是如此。

```
str == string(missing)
```

```
ans = 1x3 logical array
```

```
0 0 0
```

要查找缺失字符串，请使用 **ismissing** 函数。不要使用 **==** 运算符。

## 另请参阅

**all | any | contains | endsWith | eq | erase | eraseBetween | extractAfter | extractBefore | extractBetween | insertAfter | insertBefore | ismissing | replace | replaceBetween | startsWith | string | strings | strlength**

## 相关示例

- “创建字符串数组” (第 6-5 页)
- “分析字符串数组的文本数据” (第 6-15 页)
- “搜索和替换文本” (第 6-36 页)
- “比较文本” (第 6-31 页)

## 格式化文本

要将数据转换为文本并控制其格式，您可以将格式化操作符与常见的转换函数（如 `num2str` 和 `sprintf`）结合使用。这些操作符可控制记数法、对齐方式、有效位数以及其他内容。它们与 C 编程语言中的 `printf` 函数使用的操作符类似。格式化文本的典型用途包括用于显示和输出文件的文本。

例如，`%f` 使用定点记数法将浮点值转换为文本。通过为该操作符添加信息来调整格式，例如使用 `%.2f` 表示小数点后两位数，或使用 `%12f` 表示输出中的 12 个字符，并根据需要用空格填充。

```
A = pi*ones(1,3);
txt = sprintf(['%f | %.2f | %12f', A])
```

```
txt =
'3.141593 | 3.14 |    3.141593'
```

您可以将操作符与普通文本以及格式设定符中的特殊字符组合使用。例如，`\n` 会插入一个换行符。

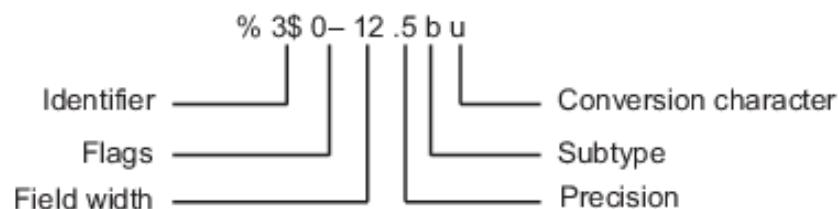
```
txt = sprintf('Displaying pi: \n %f \n %.2f \n %12f', A)
```

```
txt =
'Displaying pi:
3.141593
3.14
3.141593'
```

支持格式化操作符的函数为 `compose`、`num2str`、`sprintf`、`fprintf` 以及错误处理函数 `assert`、`error`、`warning` 和 `MException`。

## 格式化操作符的字段

格式化操作符可以包含六个字段，如图所示。从右至左，这些字段分别为转换字符、子类型、精度、字段宽度、标志以及数值标识符。（该操作符中不允许使用空格字符。在此显示空格字符只是为方便阅读。）除了前导的 `%` 字符之外，转换字符是唯一的必需字段。



### 转换字符

转换字符指定输出的表示法。它包含单个字符并显示在格式设定符的最后。

设定符	说明
<code>c</code>	单个字符。
<code>d</code>	十进制记数法（有符号）。
<code>e</code>	指数记数法（使用小写 e，如 <code>3.1415e+00</code> 中一样）。
<code>E</code>	指数记数法（使用大写 E，如 <code>3.1415E+00</code> 中一样）。

设定符	说明
f	定点记数法。
g	更紧凑的 %e 或 %f。 (将不输出无意义的零。)
G	与 %g 相同，但使用大写 E。
o	八进制记数法 (无符号)。
s	字符向量或字符串数组。
u	十进制记数法 (无符号)。
x	十六进制记数法 (无符号，使用小写字母 a-f)。
X	十六进制记数法 (无符号，使用大写字母 A-F)。

例如，使用不同的转换字符来格式化数值 46，以十进制、定点、指数和十六进制格式显示该数值。

```
A = 46*ones(1,4);
txt = sprintf("%d %f %e %X", A)

txt =
'46 46.000000 4.600000e+01 2E'
```

## 子类型

子类型字段为单个字母字符，该字符紧挨在转换字符之前。如果没有子类型字段，转换字符 %o、%x、%X 和 %u 会将输入数据按整数进行处理。要将输入数据按浮点值进行处理并将它们转换为八进制、十进制或十六进制表示形式，请使用以下子类型设定符之一。

- |   |   |
|---|---|
| b | 输入数据为双精度浮点值，而不是无符号整数。例如，要以十六进制格式输出双精度值，请使用类似 %bx 的格式。 |
| t | 输入数据为单精度浮点值，而不是无符号整数。                                 |

## 精度

格式化操作符中的精度字段是一个非负整数，紧跟在句点之后。例如，在操作符 %7.3f 中，精度为 3。对于 %g 操作符，精度指示要显示的有效位数。对于 %f、%e 和 %E 操作符，精度指示要显示在小数点右侧的位数。

使用精度字段按不同精度显示数值。

```
txt = sprintf("%g %.2g %f %.2f", pi*50*ones(1,4))

txt =
'157.08 1.6e+02 157.079633 157.08'
```

虽然您可以在格式化操作符中为输入文本指定精度（例如，在 %s 操作符中），但通常不会这么做。如果您将精度指定为 p，并且 p 小于输入文本中的字符数，则输出将仅包含前 p 个字符。

## 字段宽度

格式化操作符中的字段宽度是一个非负整数，用于在格式化输入值时指定输出中的位数或字符数。例如，在操作符 %7.3f 中，字段宽度为 7。

指定不同的字段宽度。要显示每个输出的宽度，请使用 | 字符。默认情况下，当字段宽度大于字符数时，输出文本会使用空格字符进行填充。

```
txt = sprintf('|%e|%15e|%f|%15f|', pi*50*ones(1,4))
```

```
txt =
'|1.570796e+02| 1.570796e+02|157.079633| 157.079633|'
```

用在文本输入时，字段宽度可确定是否要使用空格填充输出文本。如果字段宽度小于或等于输入文本中的字符数，则没有任何影响。

```
txt = sprintf("%30s", 'Pad left with spaces')
```

```
txt =
' Pad left with spaces'
```

## 标志

标志是可选项，用于控制输出文本的其他格式。下表介绍了可用作标志的字符。

字符	说明	示例
减号 (-)	在字段中左对齐转换后的参数。	%-5.2d
加号 (+)	对于数值，始终输出前导的符号字符 (+ 或 -)。 对于文本值，在字段中右对齐转换后的参数。	%+5.2d %+5s
空格	在值之前插入空格。	% 5.2f
零 (0)	用零而不是空格进行填充。	%05.2f
井号 (#)	修改选定的数值转换：  • 对于 %o、%x 或 %X，将输出 0、0x 或 0X 前缀。 • 对于 %f、%e 或 %E，即使精度 为零也将输出小数点。 • 对于 %g 或 %G，不删除尾随零 或小数点。	%#5.0f

左对齐或右对齐输出。默认行为是右对齐输出文本。

```
txt = sprintf('right-justify: %12.2f\nleft-justify: %-12.2f',...
    12.3, 12.3)
```

```
txt =
'right-justify:      12.30
left-justify: 12.30      '
```

显示正数的 + 符号。对于正数，默认行为是忽略前导的 + 符号。

```
txt = sprintf('no sign: %12.2f\nsign: %+12.2f',...
    12.3, 12.3)
```

```
txt =
'no sign:      12.30
sign:      +12.30'
```

用空格或零向左侧填充。默认行为是使用空格进行填充。

```
txt = sprintf('Pad with spaces: %12.2f\nPad with zeroes: %012.2f',...
    5.2, 5.2)
```

```
txt =
'Pad with spaces:      5.20
Pad with zeroes: 000000005.20'
```

**注意** 您可以在格式化操作符中指定多个标志。

## 值标识符

默认情况下，**sprintf** 等函数会按顺序将值从输入参数插入到输出文本中。要按非顺序处理输入参数，请通过在格式设定符中使用数值标识符来指定顺序。指定非连续的参数时，需要在 % 符号后紧接一个整数，并在整数后面添加 \$ 符号。

按顺序排序	按标识符排序
<code>sprintf('%s %s %s',...     '1st','2nd','3rd')</code>	<code>sprintf('%3\$s %2\$s %1\$s',...     '1st','2nd','3rd')</code>
<code>ans =</code>  <code>'1st 2nd 3rd'</code>	<code>ans =</code>  <code>'3rd 2nd 1st'</code>

## 特殊字符

特殊字符可以是输出文本的一部分。但是，由于它们不能以普通文本形式输入，因此需要使用特定的字符序列来表示。要将特殊字符插入到输出文本中，请使用下表中的任何字符序列。

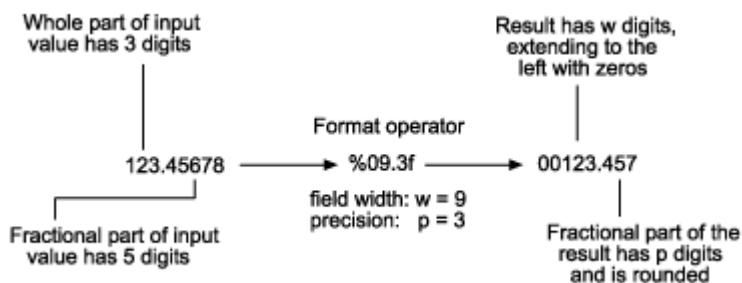
特殊字符	格式设定符中的表示形式
单引号	"
百分比字符	%%
反斜杠	\ \
警报	\a
退格符	\b
换页符	\f
换行符	\n
回车符	\r
水平制表符	\t
垂直制表符	\v
其 Unicode 数值可以通过十六进制数 N 表示的字符	\xN
示例： <code>sprintf('\x5A')</code> 返回 'Z'	
其 Unicode 数值可以通过八进制数 N 表示的字符	\N
示例： <code>sprintf('\132')</code> 返回 'Z'	

## 设置字段宽度和精度

格式化操作符遵循一组规则来按指定的字段宽度和精度格式化输出文本。您也可以在格式设定符之外为字段宽度和精度指定值，并将带编号的标识符与字段宽度和精度结合使用。

### 精度和字段宽度的格式化规则

下图演示了字段宽度和精度设置如何影响格式化函数的输出。在该图中，格式化操作符中 % 符号后面的零表示为输出文本添加前导零，而非空格字符。



- 如果未指定精度，则精度默认为 6。
- 如果精度  $p$  小于输入项的小数部分的位数，则在小数点后只会显示  $p$  位数。输出项中的小数值将被舍入。
- 如果精度  $p$  大于输入项的小数部分的位数  $f$ ，则在小数点后会显示  $p$  位数。输出项中的小数部分将使用  $p-f$  个零向右侧扩充。
- 如果未指定字段宽度，则字段宽度默认为  $p+1+n$ ，其中  $n$  为输入值的整数部分的位数。
- 如果字段宽度  $w$  大于  $p+1+n$ ，则输出值的整数部分将使用  $w-(p+1+n)$  个其他字符向左侧填充。其他字符为空格字符，除非格式化操作符包含 0 标志。在这种情况下，其他字符为零。

### 在格式设定符外部指定字段宽度和精度

您可以使用顺序参数列表中的值来指定字段宽度和精度。使用星号 (\*) 来代替格式化操作符的字段宽度或精度字段。

例如，格式化并显示三个数值。在每种情况中，都使用星号以指定字段宽度或精度取自跟在格式设定符后面的输入参数。

```
txt = sprintf("%*f %.*f %.*.*f",...
    15,123.45678, ...
    3,16.42837, ...
    6,4,pi)
```

```
txt =
' 123.456780 16.428 3.1416'
```

下表介绍了该示例中每个格式化操作符的作用。

格式化操作符	说明
%*f	将宽度指定为以下输入参数 15。
%.*f	将精度指定为以下输入参数 3。

格式化操作符	说明
%.*.f	将宽度和精度指定为以下输入参数 6 和 4。

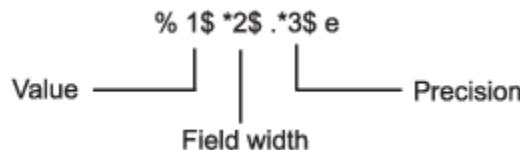
您可以混合使用两种形式。例如，从后跟的输入参数获取字段宽度，从格式设定符获取精度。

```
txt = sprintf("%*.2f", 5, 123.45678)
```

```
txt =
'123.46'
```

### 在宽度和精度字段中指定带编号的标识符

您也可以使用下图中所示的语法将字段宽度和精度指定为非顺序的参数列表中的值。在格式化操作符中，使用跟在编号标识符和 \$ 符号后面的星号指定字段宽度和精度。使用跟在格式设定符后面的输入参数指定字段宽度和精度的值。



例如，格式化并显示三个数值。在每种情况中，都使用编号标识符以指定字段宽度或精度取自跟在格式设定符后面的输入参数。

```
txt = sprintf("%1$*4$f %2$.*5$f %3$*6$.*7$f',...
    123.45678, 16.42837, pi, 15, 3, 6, 4)
```

```
txt =
' 123.456780 16.428 3.1416'
```

下表介绍了该示例中每个格式化操作符的作用。

格式化操作符	说明
%1\$*4\$f	1\$ 将第一个输入参数 123.45678 指定为值
	*4\$ 将第四个输入参数 15 指定为字段宽度
%2\$.*5\$f	2\$ 将第二个输入参数 16.42837 指定为值
	.*5\$ 将第五个输入参数 3 指定为精度
%3\$*6\$.*7\$f	3\$ 将第三个输入参数 pi 指定为值
	*6\$ 将第六个输入参数 6 指定为字段宽度
	.*7\$ 将第七个输入参数 4 指定为精度

### 使用标识符的限制

如果有任一格式化操作符包含标识符字段，则格式设定符中的所有操作符都必须包含标识符字段。如果您在同一函数调用中同时使用顺序和非顺序的次序，则输出将在首次在顺序标识符和非顺序标识符之间切换时被截断。

有效语法	无效语法
<pre>sprintf('%d %d %d %d',...     1,2,3,4) ans = '1 2 3 4'</pre>	<pre>sprintf('%d %3\$d %d %d',...     1,2,3,4) ans = '1 '</pre>

如果函数调用提供的输入参数的数量多于格式设定符中的格式化操作符数量，则这些操作符将被重复使用。但是，只有使用顺序排序的函数调用才会重复使用格式化操作符。当您使用编号标识符时，无法重复使用格式化操作符。

有效语法	无效语法
<pre>sprintf('%d',1,2,3,4) ans = '1234'</pre>	<pre>sprintf('%1\$d',1,2,3,4) ans = '1'</pre>

如果您在输入数据为向量或数组时使用编号标识符，则输出不包含格式化数据。

有效语法	无效语法
<pre>v = [1.4 2.7 3.1]; sprintf('%4f %.4f %.4f',v) ans = '1.4000 2.7000 3.1000'</pre>	<pre>v = [1.4 2.7 3.1]; sprintf('%3\$.4f %1\$.4f %2\$.4f',v) ans = 1×0 empty char array</pre>

### 另请参阅

[compose](#) | [fprintf](#) | [num2str](#) | [sprintf](#)

### 相关示例

- “从字符数组转换为数值”（第 6-43 页）
- “从数值转换为字符数组”（第 6-41 页）

# 比较文本

以不同的方式比较字符数组和字符串数组中的文本。R2016b 中引入了字符串数组。您可以使用关系运算符和 `strcmp` 函数来比较字符串数组和字符向量。您可以使用 `sort` 函数对字符串数组进行排序，就像对任何其他类型的数组进行排序一样。MATLAB® 还提供用于检查文本片段中的字符的函数。例如，您可以确定字符向量或字符串数组中的哪些字符为字母或空格字符。

## 比较字符串数组是否相等

您可以使用关系运算符 `==` 和 `~=` 来比较字符串数组是否相等。当您比较字符串数组时，输出为一个逻辑数组。如果关系为 `true`，则该逻辑数组包含 `1`；如果关系不为 `true`，则包含 `0`。

创建两个字符串标量。从 R2017a 开始，您可以使用双引号创建字符串。

```
str1 = "Hello";
str2 = "World";
str1,str2

str1 =
"Hello"

str2 =
"World"
```

比较 `str1` 和 `str2` 是否相等。

```
str1 == str2

ans = logical
0
```

将一个包含多个元素的字符串数组与一个字符串标量进行比较。

```
str1 = ["Mercury","Gemini","Apollo";...
        "Skylab","Skylab B","International Space Station"];
str2 = "Apollo";
str1 == str2

ans = 2x3 logical array
```

```
0 0 1
0 0 0
```

将一个字符串数组与一个字符向量进行比较。只要其中一个变量为字符串数组，您就可以进行比较。

```
chr = 'Gemini';
TF = (str1 == chr)

TF = 2x3 logical array
```

```
0 1 0
0 0 0
```

使用 `TF` 对 `str1` 进行索引以提取与 `Gemini` 匹配的字符串元素。您可以使用逻辑数组对某个数组进行索引。

```
str1(TF)
```

```
ans =  
"Gemini"
```

使用 `~=` 运算符进行不相等比较。对 `str1` 进行索引以提取与 'Gemini' 不匹配的元素。

```
TF = (str1 ~= chr)
```

```
TF = 2x3 logical array
```

```
1 0 1  
1 1 1
```

```
str1(TF)
```

```
ans = 5x1 string array  
"Mercury"  
"Skylab"  
"Skylab B"  
"Apollo"  
"International Space Station"
```

比较两个非标量字符串数组。当您比较两个非标量数组时，这两个数组的大小必须相同。

```
str2 = ["Mercury","Mars","Apollo";...  
        "Jupiter","Saturn","Neptune"];  
TF = (str1 == str2)
```

```
TF = 2x3 logical array
```

```
1 0 1  
0 0 0
```

对 `str1` 进行索引以提取匹配项。

```
str1(TF)
```

```
ans = 2x1 string array  
"Mercury"  
"Apollo"
```

### 使用其他关系运算符比较字符串数组

您也可以使用关系运算符 `>`、`>=`、`<` 和 `<=` 来比较字符串。以大写字母开头的字符串位于以小写字母开头的字符串前面。例如，字符串 "ABC" 小于 "abc"。数字和某些标点符号也位于字母前面。

```
"ABC" < "abc"
```

```
ans = logical  
1
```

使用 `>` 运算符将一个包含名字的字符串数组与另一个名字进行比较。名字 Sanchez、de Ponte 和 Nash 位于 Matthews 后面，因为 S、d 和 N 都大于 M。

```
str = ["Sanchez","Jones","de Ponte","Crosby","Nash"];
TF = (str > "Matthews")
```

TF = 1x5 logical array

1 0 1 0 1

**str(TF)**

```
ans = 1x3 string array
"Sanchez" "de Ponte" "Nash"
```

## 对字符串数组排序

您可以对字符串数组进行排序。MATLAB® 使用 UTF-16 字符编码方案以 Unicode® 方式存储字符。字符和字符串数组按 UTF-16 代码点顺序进行排序。对于同时也是 ASCII 字符的字符，此顺序意味着大写字母在小写字母之前。数字和某些标点符号也在字母之前。

对字符串数组 str 进行排序。

**sort(str)**

```
ans = 1x5 string array
"Crosby" "Jones" "Nash" "Sanchez" "de Ponte"
```

对一个 2×3 字符串数组进行排序。sort 函数会单独对每列中的元素进行排序。

**sort(str2)**

```
ans = 2x3 string array
"Jupiter" "Mars" "Apollo"
"Mercury" "Saturn" "Neptune"
```

要对每行中的元素进行排序，请沿第二个维度对 str2 进行排序。

**sort(str2,2)**

```
ans = 2x3 string array
"Apollo" "Mars" "Mercury"
"Jupiter" "Neptune" "Saturn"
```

## 比较字符向量

您可以对字符向量和字符向量元胞数组进行相互比较。使用 strcmp 函数比较两个字符向量，或者使用 strncmp 比较前 N 个字符。您也可以使用 strcmpi 和 strncmpi 进行不区分大小写的比较。

使用 strcmp 函数比较两个字符向量。chr1 和 chr2 不相同。

```
chr1 = 'hello';
chr2 = 'help';
TF = strcmp(chr1,chr2)
```

```
TF = logical
0
```

请注意，MATLAB `strcmp` 与 `strcmp` 的 C 版本不同。当两个字符数组相同而非不同时，`strcmp` 的 C 版本会返回 0。

使用 `strncmp` 函数比较前两个字符。由于两个字符串都以字符 `h` 开头，因此 `TF` 为 1。

**TF = strncmp(chr1,chr2,2)**

TF = logical  
1

比较两个字符向量元胞数组。strcmp 会返回一个与元胞数组大小相同的逻辑数组。

```
C1 = {'pizza'; 'chips'; 'candy'};  
C2 = {'pizza'; 'chocolate'; 'pretzels'};  
stremp(C1,C2)
```

ans = 3x1 logical array

100

### 检查字符串数组和字符数组中的字符

您可以使用 `isstrprop`、`isletter` 和 `isspace` 函数检查字符串数组或字符数组中的字符。

- `isstrprop` 可检查字符串数组或字符数组中的字符。
  - `isletter` 和 `isspace` 函数只能检查字符数组中的字符。

确定字符串向量中的哪些字符是空格字符。`isspace` 会返回一个与 `chr` 大小相同的逻辑向量。

```
chr = 'Four score and seven years ago';  
TF = isspace(chr)
```

**TF = 1x30 logical array**

0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0

**isstrprop** 函数可以查询字符的许多不同特性。 **isstrprop** 可以确定字符串或字符向量中的字符是字母、字母数字字符、十进制或十六进制数字还是标点字符。

确定字符串中的哪些字符是标点符号。`isstrprop` 会返回一个长度等于 `str` 中的字符数量的逻辑向量。

```
str = "A horse! A horse! My kingdom for a horse!"
```

str =

"A horse! A horse! My kingdom for a horse!"

```
isstrprop(str,"punct")
```

ans = 1x41 logical array

确定字符向量 `chr` 中的哪些字符是字母。

```
isstrprop(chr,"alpha")  
ans = 1x30 logical array  
1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1
```

## 另请参阅

[eq](#) | [ge](#) | [gt](#) | [isletter](#) | [isspace](#) | [isstrprop](#) | [le](#) | [lt](#) | [ne](#) | [sort](#) | [strcmp](#)

## 相关示例

- “字符串数组和字符数组中的文本” (第 6-2 页)
- “创建字符串数组” (第 6-5 页)
- “分析字符串数组的文本数据” (第 6-15 页)
- “搜索和替换文本” (第 6-36 页)
- “测试空字符串和缺失值” (第 6-20 页)

## 搜索和替换文本

您可以搜索字符数组和字符串数组中的文本，并将子字符串替换为新文本。R2016b 中引入了字符串数组以及可搜索和替换文本的新函数。使用 `contains` 等函数搜索子字符串。同样地，使用 `replace` 函数替换字符串中的文本，或使用 `extractBetween` 等函数提取文本。您可以将上述任意函数用于字符向量或字符串数组。为实现兼容性，您也可以将 `strfind` 和 `strrep` 等函数同时用于字符向量和字符串数组。

### 搜索文本

使用 `contains`、`startsWith` 和 `endsWith` 函数标识字符串数组、字符向量或字符向量元胞数组中的文本。

创建一个字符串。从 R2017a 开始，您可以使用双引号创建字符串。

```
str = "Rosemary Jones"
```

```
str =
"Rosemary Jones"
```

确定 `str` 是否包含子字符串 `mary`。如果 `contains` 函数在该字符串中的任意位置找到此子字符串，则会返回逻辑值 1。

```
TF = contains(str,"mary")
```

```
TF = logical
1
```

您也可以使用 `strfind` 函数查找匹配的文本。`strfind` 会返回每个匹配项开头字符的索引。在本例中，`strfind` 返回的是 5，因为 `mary` 中的 `m` 是 `str` 的第五个字符。

```
idx = strfind(str,"mary")
```

```
idx = 5
```

使用 `strfind` 查找多个匹配项。存在多个匹配项时，`strfind` 会以数组形式返回这些索引。

```
idx = strfind(str,"s")
```

```
idx = 1×2
```

```
3 14
```

创建一个包含许多名字的字符串数组。确定哪些名字包含子字符串 `Ann`。`contains` 函数会返回一个逻辑数组，只要 `str` 带有一个包含 `Ann` 的元素，该数组即包含一个 1。要创建一个仅包含匹配项的新字符串数组，请使用 `TF` 对 `str` 进行索引。

```
str = ["Rosemary Ann Jones","Peter Michael Smith","Ann Marie Young"]
```

```
str = 1x3 string array
"Rosemary Ann Jones" "Peter Michael Smith" "Ann Marie Young"
```

```
TF = contains(str,"Ann")
```

```
TF = 1x3 logical array
```

```
1 0 1
```

```
matches = str(TF)
matches = 1x2 string array
"Rosemary Ann Jones" "Ann Marie Young"
```

查找以 Ann 开头的字符串。

```
TF = startsWith(str,"Ann");
matches = str(TF)
```

```
matches =
"Ann Marie Young"
```

同样地，`endsWith` 函数可查找以指定的文本片段结尾的字符串。

您也可以使用 `contains`、`startsWith` 和 `endsWith` 函数来确定字符向量是否包含文本。

```
chr = 'John Paul Jones'
```

```
chr =
'John Paul Jones'
```

```
TF = contains(chr,'Paul')
```

```
TF = logical
1
```

```
TF = endsWith(chr,'Paul')
```

```
TF = logical
0
```

使用 `contains` 函数查找字符串数组的行中的文本。`census1905` 包含数行 1905 年的模拟普查数据。每一行都包含一个名字、出生年份以及当年用该名字起名的次数。

```
census1905 = ["Ann Mary","1905","230";
    "John","1905","5400";
    "Mary","1905","4600";
    "Maryjane","1905","304";
    "Paul","1905","1206"];
```

查找名字等于 Mary 的行。

```
TF = (census1905(:,1) == "Mary");
census1905(TF,:)
ans = 1x3 string array
"Mary" "1905" "4600"
```

使用 `contains` 函数查找名字属于 Mary 的变化形式的行。

```
TF = contains(census1905(:,1),"Mary");
census1905(TF,:)
```

```
ans = 3x3 string array
    "Ann Mary"  "1905"  "230"
    "Mary"      "1905"  "4600"
    "Maryjane"   "1905"  "304"
```

### 替换文本

您可以使用 `replace` 函数替换字符串数组、字符向量或字符向量元胞数组中的文本。

创建一个字符串。将子字符串 `mary` 替换为 `anne`。

```
str = "Rosemary Jones"

str =
"Rosemary Jones"

newStr = replace(str,"mary","anne")

newStr =
"Roseanne Jones"
```

您也可以使用 `strrep` 函数来替换文本。但是，建议使用 `replace` 函数。

```
newStr = strrep(str,"Jones","Day")

newStr =
"Rosemary Day"

创建一个包含许多名字的字符串数组。

str = ["Rosemary Ann Jones","Peter Michael Smith","Ann Marie Young"]
```

指定要替换的多个名字。

```
oldText = ["Ann","Michael"];
newText = ["Beth","John"];
newStr = replace(str,oldText,newText)

newStr = 1x3 string array
    "Rosemary Beth Jones"  "Peter John Smith"  "Beth Marie Young"
```

替换字符向量中的文本。您可以将 `replace` 和 `replaceBetween` 用于字符向量和字符串。

```
chr = 'Mercury, Gemini, Apollo'

chr =
'Mercury, Gemini, Apollo'

replace(chr,'Gemini','Mars')

ans =
'Mercury, Mars, Apollo'
```

替换文件名字符串数组中的文本。将文件名追加到网站地址。文件名中包含空格，但空格不能是 Web 地址的一部分。将空格字符 `"` 替换为 `%20`（这是 Web 地址的标准）。

```

str = ["Financial Report.docx";
       "Quarterly 2015 Details.docx";
       "Slides.pptx"]

str = 3x1 string array
    "Financial Report.docx"
    "Quarterly 2015 Details.docx"
    "Slides.pptx"

newStr = replace(str," ","%20")

newStr = 3x1 string array
    "Financial%20Report.docx"
    "Quarterly%202015%20Details.docx"
    "Slides.pptx"

filenames = "http://example.com/Documents/" + newStr

filenames = 3x1 string array
    "http://example.com/Documents/Financial%20Report.docx"
    "http://example.com/Documents/Quarterly%202015%20Details.docx"
    "http://example.com/Documents/Slides.pptx"

```

## 提取文本

使用 `extractAfter`、`extractBefore` 和 `extractBetween` 函数从字符串数组或字符向量中提取子字符串。使用上述函数可提取位于指定的文本片段之前、之后或中间的不同子字符串。

创建一个包含文件名的字符串数组。使用 `extractAfter` 函数提取位于 `C:\Temp\` 之后的名称部分。

```

str = ["C:\Temp\MyReport.docx";
       "C:\Temp\Data\Sample1.csv";
       "C:\Temp\Slides.pptx"]

str = 3x1 string array
    "C:\Temp\MyReport.docx"
    "C:\Temp\Data\Sample1.csv"
    "C:\Temp\Slides.pptx"

filenames = extractAfter(str,"C:\Temp\")

filenames = 3x1 string array
    "MyReport.docx"
    "Data\Sample1.csv"
    "Slides.pptx"

```

从在 XML 标记内对名称进行编码的字符串数组中提取客户名称。

```

str = ["<CustomerName>Elizabeth Day</CustomerName>";
       "<CustomerName>George Adams</CustomerName>";
       "<CustomerName>Sarah Young</CustomerName>"]

str = 3x1 string array
    "<CustomerName>Elizabeth Day</CustomerName>"

```

```
"<CustomerName>George Adams</CustomerName>"  
"<CustomerName>Sarah Young</CustomerName>"  
  
names = extractBetween(str,"<CustomerName>","</CustomerName>")  
  
names = 3x1 string array  
"Elizabeth Day"  
"George Adams"  
"Sarah Young"
```

### 另请参阅

[contains](#) | [endsWith](#) | [erase](#) | [eraseBetween](#) | [extractAfter](#) | [extractBefore](#) | [extractBetween](#)  
| [insertAfter](#) | [insertBefore](#) | [replace](#) | [replaceBetween](#) | [startsWith](#) | [strfind](#) | [strrep](#)

### 相关示例

- “字符串数组和字符数组中的文本” (第 6-2 页)
- “创建字符串数组” (第 6-5 页)
- “分析字符串数组的文本数据” (第 6-15 页)
- “比较文本” (第 6-31 页)
- “测试空字符串和缺失值” (第 6-20 页)
- “正则表达式” (第 2-48 页)

# 从数值转换为字符串数组

## 本节内容

- “函数摘要”（第 6-41 页）
- “将数字转换为字符代码”（第 6-41 页）
- “将数字表示为文本”（第 6-41 页）
- “转换为特定基数”（第 6-42 页）

## 函数摘要

此表中列出的函数提供了多种将数值数据转换为字符串数组的方法。

函数	说明	示例
<b>char</b>	将正整数转换为等效的字符。（截断任何小数部分。）	[72 105] → 'Hi'
<b>string</b>	将包含双精度值的数组转换为字符串数组。	[72 105] → "72" "105" [3.1416 2.178] → "3.1416" "2.178"
<b>int2str</b>	将正整数或负整数转换为字符类型。（对任何小数部分四舍五入。）	[72 105] → '72 105'
<b>num2str</b>	将数值类型转换为指定精度和格式的字符类型。	[72 105] → '72/105/'（格式设置为 %1d/）
<b>mat2str</b>	将数值类型转换为指定精度的字符类型，并返回 MATLAB 可以计算的字符向量。	[72 105] → '[72 105]'
<b>dec2hex</b>	将正整数转换为字符类型的十六进制基数。	[72 105] → '48 69'
<b>dec2bin</b>	将正整数转换为字符类型的二进制基数。	[72 105] → '1001000 1101001'
<b>dec2base</b>	将正整数转换为字符类型的数字，基数可从 2 到 36。	[72 105] → '110 151'（基数设置为 8）

## 将数字转换为字符代码

**char** 函数将整数转换为 Unicode 字符代码，并返回由对等字符构成的字符串数组：

```
x = [77 65 84 76 65 66];
char(x)
ans =
'MATLAB'
```

## 将数字表示为文本

**int2str**、**num2str** 和 **mat2str** 函数将数值表示为文本，其中每个字符表示输入值中一位单独的数字。**int2str** 和 **num2str** 函数常被用来为绘图添加标签。例如，下列线条使用 **num2str** 为绘图的 x 轴准备自动化标签：

```
function plotlabel(x, y)
plot(x, y)
```

```
chr1 = num2str(min(x));
chr2 = num2str(max(x));
out = ['Value of f from ' chr1 ' to ' chr2];
xlabel(out);
```

### 转换为特定基数

另一类转换函数将数值更改为字符数组，将十进制值用另一种基数形式表示，例如二进制或十六进制表示形式。这些函数包括 **dec2hex**、**dec2bin** 和 **dec2base**。

# 从字符数组转换为数值

## 本节内容

- “函数摘要”（第 6-43 页）
- “从字符代码转换”（第 6-43 页）
- “转换代表数值的文本”（第 6-44 页）
- “从特定基数转换”（第 6-44 页）

## 函数摘要

此表中列出的函数提供了多种将字符数组转换为数值数据的方法。

函数	说明	示例
<b>double, single</b>	将字符转换为表示该字符的整数代码。	'Hi' → 72 105
<b>uint8, uint16, uint32, uint64</b>		
<b>int8, int16, int32, int64</b>		
<b>str2num</b>	将字符类型转换为数值类型。	'72 105' → [72 105]
<b>str2double</b>	与 <b>str2num</b> 类似，但提供更佳的性能，用于处理字符串数组和字符向量元胞数组。	"72" "105" → [72 105] {'72' '105'} → [72 105]
<b>hex2num</b>	将数值类型转换为指定精度的字符类型，并返回 MATLAB 可以计算的字符数组。	'A' → '-1.4917e-154'
<b>hex2dec</b>	将数值类型的十六进制基数转换为正整数。	'A' → 10
<b>bin2dec</b>	将字符类型的二进制数字转换为十进制数字。	'1010' → 10
<b>base2dec</b>	将字符类型的 2 到 36 内任何进制的数字转换为十进制数字。	'12' → 10 (如果 base == 8)

## 从字符代码转换

字符数组和字符串数组将每个字符存储为一个 16 位数值。使用一个整数转换函数（例如 **uint8**）或 **double** 函数将字符转换为其数值，使用 **char** 可恢复其字符表示形式：

```
name = 'Thomas R. Lee';

name = double(name)
name =
    84 104 111 109 97 115 32 82 46 32 76 101 101

name = char(name)
name =

'Thomas R. Lee'
```

## 转换代表数值的文本

使用 `str2num` 将字符数组转换为由其表示的数值：

```
chr = '37.294e-1';  
val = str2num(chr)  
val =  
  
3.7294
```

`str2double` 函数会将字符串数组或字符向量元胞数组转换为其代表的双精度值：

```
c = {'37.294e-1'; '-58.375'; '13.796'};  
str = string({'3.14159', '2.718'});
```

```
d = str2double(c)
```

```
d =
```

```
3.7294  
-58.3750  
13.7960
```

```
x = str2double(str)
```

```
x =
```

```
3.1416 2.7180
```

```
whos  
Name      Size            Bytes  Class     Attributes  
c          3x1             380   cell  
d          3x1             24    double  
str        1x2             196   string  
x          1x2             16    double
```

## 从特定基数转换

要将非十进制数字的字符表示转换为该数字的值，请使用以下函数之一：`hex2num`、`hex2dec`、`bin2dec` 或 `base2dec`。

`hex2num` 和 `hex2dec` 函数都接受十六进制（以 16 为基数）输入，但 `hex2num` 返回它表示的 IEEE 双精度浮点数，而 `hex2dec` 则将输入转换为十进制整数。

# 十六进制和二进制值

您可以将数字表示为十六进制或二进制值。在某些情况下，这些表示方式更加方便。例如，您可以使用二进制值来表示硬件寄存器的位。在 MATLAB® 中，有两种方式来表示十六进制和二进制值：

- 作为字面值。从 R2019b 开始，您可以使用适当的前缀作为表示法，将十六进制和二进制值写成字面值。例如，字面值 `0x2A` 指定 42，MATLAB 会将其存储为数字而不是文本。
- 作为字符串或字符向量。例如，字符向量 `'2A'` 将数字 42 表示为十六进制值。当您使用文本表示十六进制或二进制值时，请用引号将它括起来。MATLAB 将这种表示存储为文本，而不是数字。

MATLAB 中的一些函数可用于将数字与其十六进制及二进制表示相互转换。

## 使用十六进制和二进制表示法表示整数

十六进制字面值以 `0x` 或 `0X` 前缀开头，而二进制字面值以 `0b` 或 `0B` 前缀开头。MATLAB 将采用这种表示法的数字存储为整数。例如，以下两个字面值都表示整数 42。

**A = 0x2A**

```
A = uint8
    42
```

**B = 0b101010**

```
B = uint8
    42
```

当您用此表示法表示数字时，不要用引号。使用 `0-9`、`A-F` 和 `a-f` 表示十六进制数字。使用 `0` 和 `1` 表示二进制数字。

默认情况下，MATLAB 将数字存储为能够容纳它的最小无符号整数类型。但是，您可以使用可选后缀来指定存储该值的整数类型。

- 要指定无符号 8 位、16 位、32 位和 64 位整数类型，请使用后缀 `u8`、`u16`、`u32` 和 `u64`。
- 要指定有符号 8 位、16 位、32 位和 64 位整数类型，请使用后缀 `s8`、`s16`、`s32` 和 `s64`。

例如，要存储为有符号 32 位整数的十六进制字面值的表示如下。

**A = 0x2As32**

```
A = int32
    42
```

指定有符号整数类型时，您也可以将负数写作字面值。用 2 的补码形式表示负数。例如，您可以使用后缀 `s8` 以字面值指定负数。

**A = 0xFFs8**

```
A = int8
    -1
```

由于 MATLAB 将这些字面值存储为数字，因此您可以在任何使用数值数组的上下文或函数中使用它们。

## 将十六进制和二进制值表示为文本

您还可以使用 `dec2hex` 和 `dec2bin` 函数将整数转换为字符向量，以十六进制或二进制值表示它们。将整数转换为十六进制。

```
hexStr = dec2hex(255)
```

```
hexStr =  
'FF'
```

将整数转换为二进制。

```
binStr = dec2bin(16)
```

```
binStr =  
'10000'
```

由于这些函数生成文本，因此当您需要表示数值的文本时，可以使用这些函数。例如，您可以将这些值追加到标题或图标签上，或将它们写入以十六进制或二进制表示形式存储数字的文件中。

### 将十六进制值构成的数组表示为文本

要将数值数组转换为文本，推荐的方法是使用 **compose** 函数。此函数返回与输入数值数组大小相同的字符串数组。要生成十六进制格式，请使用 %X 作为格式设定符。

```
A = [255 16 12 1024 137]
```

```
A = 1x5
```

```
255      16      12     1024     137
```

```
hexStr = compose("%X",A)
```

```
hexStr = 1x5 string array  
"FF" "10" "C" "400" "89"
```

**dec2hex** 和 **dec2bin** 函数还将数值数组转换为文本，以十六进制或二进制值表示它们。但是，这些函数返回字符数组，其中每行表示输入数值数组中的一个数字，必要时用零填充。

### 将二进制表示转换为十六进制表示

要将二进制值转换为十六进制值，请从二进制字面值开始，并将其转换为表示其十六进制值的文本。由于字面值会被解释为数字，您可以将其直接指定为 **dec2hex** 的输入参数。

```
D = 0b1111;  
hexStr = dec2hex(D)
```

```
hexStr =  
'F'
```

如果从十六进制字面值开始，则可以使用 **dec2bin** 将其转换为表示其二进制值的文本。

```
D = 0x8F;  
binStr = dec2bin(D)
```

```
binStr =  
'10001111'
```

### 使用二进制值的按位运算

二进制数的一个典型用途是表示位。例如，许多设备都有寄存器，用于访问代表内存中数据或设备状态的位集合。当使用这样的硬件时，您可以在 MATLAB 中使用数字来表示寄存器中的值。使用二进制值和按位运算来表示和访问特定位。

创建一个表示 8 位寄存器的数字。从二进制表示开始很方便，但该数字是以整数形式存储的。

```
register = 0b10010110
```

```
register = uint8
150
```

要获取或设置特定位的值，请使用位运算。例如，使用 **bitand** 和 **bitshift** 函数获得第五位的值。（将该位移至第一个位置，以使 MATLAB 返回 0 或 1。在此示例中，第五位是 1。）

```
b5 = bitand(register,0b10000);
b5 = bitshift(b5,-4)
```

```
b5 = uint8
1
```

要将第五位翻转为 0，请使用 **bitset** 函数。

```
register = bitset(register,5,0)
```

```
register = uint8
134
```

由于 **register** 是整数，请使用 **dec2bin** 函数以二进制格式显示所有位。**binStr** 是字符向量，表示不带 **0b** 前缀的二进制值。

```
binStr = dec2bin(register)
```

```
binStr =
'10000110'
```

## 另请参阅

**bin2dec | bitand | bitset | bitshift | dec2bin | dec2hex | hex2dec | sprintf | sscanf**

## 详细信息

- “从字符数组转换为数值”（第 6-43 页）
- “从数值转换为字符数组”（第 6-41 页）
- “格式化文本”（第 6-24 页）
- “按位运算”（第 2-36 页）
- “执行循环冗余校验”（第 2-41 页）

## 有关字符串数组的常见问题解答

MATLAB 在 R2016b 中引入了 `string` 数据类型。从 R2018b 开始，您可以使用字符串数组处理 MathWorks 产品中的文本。字符串数组可存储文本片段，并提供一组用于将文本按数据进行处理的函数。您可以对字符串数组进行索引、重构和进行串联，就像处理任何其他类型的数组一样。有关详细信息，请参阅“[创建字符串数组](#)”（第 6-5 页）。

字符串数组的行为大多数方面与字符向量和字符向量元胞数组类似。但是，字符串数组和字符数组之间存在一些关键差异，这些差异可能会导致意外的结果。对于所有这些差异，按推荐的方法来使用字符串，便可获得预期的结果。

### 为什么在命令形式中使用字符串会返回错误？

以命令形式使用 `cd`、`dir`、`copyfile` 或 `load` 等函数时，应避免使用双引号。在命令形式中，放在双引号中的参数可能会导致错误。要将参数指定为字符串，请使用函数形式。

对于命令语法，应该用空格而不是逗号来分隔输入，而且不需要将输入参数放在括号中。例如，您可以将 `cd` 函数与命令语法结合使用来更改文件夹。

```
cd C:\Temp
```

文本 `C:\Temp` 是一个字符向量。在命令形式中，所有参数都始终为字符向量。如果您有一个包含空格的参数（例如文件夹名称），请将其放在单引号中以作为一个输入参数来指定。

```
cd 'C:\Program Files'
```

但是，如果使用双引号指定参数，`cd` 将引发错误。

```
cd "C:\Program Files"
```

```
Error using cd
Too many input arguments.
```

根据您使用的函数和指定的参数，错误消息可能有所不同。例如，如果您将 `load` 函数与命令语法结合使用，并使用双引号来指定参数，则 `load` 将引发不同的错误。

```
load "myVariables.mat"
```

```
Error using load
Unable to read file "myVariables.mat": Invalid argument.
```

在命令形式中，双引号被视为字面文本的一部分，而不是字符串构造运算符。如果您以函数形式编写等效的 `cd "C:\Program Files"`，则它看起来会像是在用两个参数调用 `cd`。

```
cd("C:\Program','Files")
```

以字符串形式指定参数时，请使用函数语法。支持命令语法的所有函数也支持函数语法。例如，您可以将 `cd` 与函数语法结合使用，并以带双引号的字符串形式指定输入参数。

```
cd("C:\Program Files")
```

### 为什么元胞数组中的字符串返回错误？

如果有多个字符串，请将它们存储在字符串数组而不是元胞数组中。使用方括号而不是花括号创建字符串数组。在存储和操作文本方面，字符串数组比元胞数组更高效。

```
str = ["Venus","Earth","Mars"]
```

```
str = 1×3 string array
    "Venus"   "Earth"   "Mars"
```

请避免使用字符串元胞数组。如果您使用元胞数组，便意味着您放弃了使用字符串数组带来的性能优势。事实上，大多数函数都不接受将字符串元胞数组作为输入参数、选项或者作为名称-值对组的值。例如，如果指定字符串元胞数组作为输入参数，则 `contains` 函数将引发错误。

```
C = {"Venus","Earth","Mars"};
```

```
C = 1×3 cell array
    {"Venus"}   {"Earth"}   {"Mars"}
```

```
TF = contains(C,"Earth")
```

```
Error using contains
```

```
First argument must be a string array, character vector, or cell array of character vectors.
```

这种情况下，应以字符串数组形式指定参数。

```
str = ["Venus","Earth","Mars"];
TF = contains(str,"Earth");
```

在 R2016b 之前，术语“字符串元胞数组”是指其元素全部为字符向量的元胞数组。但是，为了与字符串数组区分开来，将这类元胞数组称为“字符向量元胞数组”更为准确。

元胞数组可以包含任何数据类型的变量，包括字符串。仍然可以创建其元素全部为字符串的元胞数组。而且，如果您已经在代码中指定字符向量元胞数组，则将单引号替换为双引号可能看起来就像一次简单的更新。但是，不建议您创建或使用字符串元胞数组。

## 为什么使用 `length()` 调用字符串会返回 1?

`length` 函数通常用于确定字符向量中的字符数。但是，要确定字符串中的字符数，请使用 `strlength` 函数，而不是 `length`。

使用单引号创建一个字符向量。要确定其长度，请使用 `length` 函数。由于 `C` 是向量，因此它的长度等于字符数。`C` 是一个  $1 \times 11$  向量。

```
C = 'Hello world';
L = length(C)
```

```
L = 11
```

使用双引号创建具有相同字符的字符串。虽然它存储 11 个字符，但 `str` 是一个  $1 \times 1$  字符串数组，或者说字符串标量。如果使用 `length` 调用字符串标量，则无论它存储多少个字符，输出参数都为 1。

```
str = "Hello World";
L = length(str)
```

```
L = 1
```

要确定字符串中的字符数，请使用在 R2016b 中引入的 `strlength` 函数。为了实现兼容性，`strlength` 仍然可用于字符向量。在这两种情况下，`strlength` 都返回字符数。

```
L = strlength(C)
```

```
L = 11
```

```
L = strlength(str)
```

```
L = 11
```

**strlength** 还可用于包含多个字符串的字符串数组以及字符向量元胞数组。

**length** 函数返回数组的最长维度的大小。对于字符串数组，**length** 返回数组最长维度的字符串的数量，而不返回字符串内的字符数。

### 为什么 **isempty("")** 返回 0？

一个字符串可以不包含任何字符。这样的字符串称为空字符串。您可以使用一对不包含任何内容的双引号来指定空字符串。

```
L = strlength("")
```

```
L = 0
```

但是，空字符串并不是空数组。空字符串是刚好没有任何字符的字符串标量。

```
sz = size("")
```

```
sz = 1×2
    1     1
```

如果对空字符串调用 **isempty**，将返回 0 (false)，因为它并不是空数组。

```
tf = isempty("")
```

```
tf = logical
    0
```

但是，如果对空字符数组调用 **isempty**，则返回 1 (true)。用一对空单引号 " 指定的字符数组是一个  $0 \times 0$  字符数组。

```
tf = isempty("")
```

```
tf = logical
    1
```

要测试一段文本是否不包含任何字符，最佳做法是使用 **strlength** 函数。无论输入是字符串标量还是字符向量，都可以使用相同的调用。

```
str = "";
if strlength(str) == 0
    disp('String has no text')
end
```

```
String has no text
```

```
chr = '';
if strlength(chr) == 0
    disp('Character vector has no text')
end
```

```
Character vector has no text
```

## 为什么使用方括号追加字符串返回多个字符串？

可以使用方括号在字符向量后面追加文本。但是，如果使用方括号向字符串数组添加文本，新文本将作为字符串数组的新元素进行串联。要为字符串追加文本，请使用 `plus` 运算符或 `strcat` 函数。

例如，如果串联两个字符串，则结果是一个  $1 \times 2$  字符串数组。

```
str = ["Hello" "World"]
```

```
str = 1×2 string array
    "Hello"    "World"
```

但是，如果串联两个字符向量，则结果是一个更长的字符向量。

```
str = ['Hello' 'World']
```

```
chr = 'HelloWorld'
```

要为字符串（或字符串数组的元素）追加文本，请使用 `plus` 运算符而不是方括号。

```
str = "Hello" + "World"
```

```
str = "HelloWorld"
```

另外，也可以使用 `strcat` 函数。无论输入参数是字符串还是字符向量，都可以使用 `strcat` 追加文本。

```
str = strcat("Hello","World")
```

```
str = "HelloWorld"
```

不管使用方括号、`plus` 还是 `strcat`，都可以指定任意数量的参数。在 `Hello` 和 `World` 之间追加一个空白字符。

```
str = "Hello" + " " + "World"
```

```
str = "Hello World"
```

## 另请参阅

`cd` | `contains` | `copyfile` | `dir` | `isempty` | `length` | `load` | `plus` | `size` | `sprintf` | `strcat` | `string` | `strlength`

## 相关示例

- “创建字符串数组”（第 6-5 页）
- “测试空字符串和缺失值”（第 6-20 页）
- “比较文本”（第 6-31 页）
- “更新您的代码以接受字符串”（第 6-52 页）

# 更新您的代码以接受字符串

在 R2016b 中，MATLAB 引入了字符串数组作为文本的数据类型。在未来的版本中，所有 MathWorks 产品都将兼容字符串数组。兼容意味着如果您可以将文本指定为字符向量或字符向量元胞数组，则还可以将其指定为字符串数组。现在，您可以在自己的代码中采用字符串数组作为文本数据类型。

如果您为其他 MATLAB 用户编写代码，则最好将您的 API 更新为接受字符串数组，同时保持与其他文本数据类型的向后兼容性。采用字符串可使您的代码与 MathWorks 产品保持一致。

如果您的代码中依赖项很少，或者您正在开发新代码，则请考虑使用字符串数组作为您的主要文本数据类型以提高性能。在这种情况下，最佳做法是适当编写或更新您的 API，使之接受字符向量、字符向量元胞数组或字符串数组类型的输入参数。

有关字符串数组和其他术语的定义，请参阅“字符和字符串数组的术语”（第 6-57 页）。

## 什么是字符串数组？

在 MATLAB 中，可以用两种方式存储文本数据。一种方式是使用字符数组，它是一个字符序列，就像数值数组是一个数字序列一样。从 R2016b 开始采用的另一种方式是将一个字符序列存储在一个字符串中。您可以将多个字符串存储在一个字符串数组中。有关详细信息，请参阅“字符和字符串”。

## 在旧 API 中采用字符串类型的建议方法

如果您的代码有很多依赖项，并且您必须保持向后兼容性，请按照下列方法更新函数和类来实现向后兼容的 API。

### 函数

- 接受字符串数组作为输入参数。
  - 如果某个输入参数可以是字符向量或字符向量元胞数组，则更新您的代码，使字符串数组也可以作为输入参数。例如，假设某函数具有一个输入参数，该输入参数可以指定为字符向量（使用单引号）。最佳做法是更新函数，使得该参数可以指定为字符向量或字符串标量（使用双引号）。
  - 在名称-值对组参数中接受字符串作为名称和值。
    - 在名称-值对组参数中，允许将名称指定为字符向量或字符串 - 即将名称用单引号或双引号括起来。如果某个值可以是字符向量或字符向量元胞数组，则更新您的代码，使之也可以是字符串数组。
- 不要接受字符串数组的元胞数组作为文本输入参数。
  - 字符串数组的元胞数组在每个元胞中都有一个字符串数组。例如，`{"hello","world"}` 是字符串数组的元胞数组。虽然您可以创建这样的元胞数组，但不推荐将其用于存储文本。字符串数组的元素具有相同的数据类型并以高效方式存储。如果将字符串存储在元胞数组中，则无法利用字符串数组带来的优势。

但是，如果您的代码接受异构元胞数组作为输入，则考虑接受包含字符串的元胞数组。您可以将这种元胞数组中的任何字符串转换为字符向量。

- 一般情况下，不要更改输出类型。
  - 如果您的函数返回字符向量或字符向量元胞数组，则即使该函数接受字符串数组作为输入，也不要更改输出类型。例如，`fileread` 函数接受指定为字符向量或字符串的输入文件名，但该函数以字符向量形式返回文件内容。通过保持输出类型不变，您可以保持向后兼容性。

- 当函数修改了输入文本时，会返回相同的数据类型。
  - 如果您的函数修改了输入文本并将修改后的文本作为输出参数返回，则输入和输出参数应该具有相同的数据类型。例如，`lower` 函数接受文本作为输入参数，将其转换为全部小写字母并返回。如果输入参数是字符向量，则 `lower` 返回字符向量。如果输入是字符串数组，则 `lower` 返回字符串数组。
- 考虑向导入函数添加 '`TextType`' 参数。
  - 如果您的函数从文件导入数据，并且至少有部分数据可以是文本，则考虑添加一个输入参数，该参数指定是将文本作为字符数组还是字符串数组返回。例如，`readtable` 函数提供 '`TextType`' 名称-值对组参数。此参数指定 `readtable` 返回的表是包含字符向量元胞数组形式的文本还是包含字符串数组形式的文本。

## 类

- 将方法视为函数。
  - 要采用字符串，请将方法视作函数。接受字符串数组作为输入参数，并且通常不要更改输出参数的数据类型，如前一节中所述。
- 不要更改属性的数据类型。
  - 如果某个属性是字符向量或字符向量元胞数组，则不要更改其类型。当您访问此类属性时，返回值仍然是字符向量或者字符向量元胞数组。  
作为替代方案，您可以添加一个作为字符串的新属性，并使其依赖旧属性来保持兼容性。
- 使用字符串数组设置属性。
  - 如果某个属性可以使用字符向量或字符向量元胞数组来设置，则更新您的类，使之同样可以使用字符串数组来设置。但是，不要更改该属性的数据类型。此时，应将输入字符串数组转换为该属性的数据类型，然后设置属性。
- 添加 `string` 方法。
  - 如果您的类已有 `char` 和/或 `cellstr` 方法，请添加一个 `string` 方法。如果类中有对象可以表示为字符向量或字符向量元胞数组，则也要使该对象能够表示为字符串数组。

## 如何在旧 API 中采用字符串数组

要在旧 API 中采用字符串，您需要接受字符串数组作为输入参数，然后将它们转换为字符向量或字符向量元胞数组。如果您在函数的开头执行此类转换，则无需更新函数的其余部分。

`convertStringsToChars` 函数提供了一种处理所有输入参数的方法，只转换其中的字符串数组参数。要使您的现有代码接受字符串数组作为输入，请在函数和方法的开头添加对 `convertStringsToChars` 的调用。

例如，如果您定义了接受三个输入参数的函数 `myFunc`，则使用 `convertStringsToChars` 处理所有三个输入。其余的代码保持不变。

```
function y = myFunc(a,b,c)
[a,b,c] = convertStringsToChars(a,b,c);
<line 1 of original code>
<line 2 of original code>
...

```

在此示例中，参数 `[a,b,c]` 覆盖了原有的输入参数。如果有任意输入参数不是字符串数组，则该参数保持不变。

如果 `myFunc` 接受可变数目的输入参数，则处理由 `varargin` 指定的所有参数。

```
function y = myFunc(varargin)
    varargin{1} = convertStringsToChars(varargin{1});
    ...

```

### 性能方面的考虑

在转换一个输入参数时，`convertStringsToChars` 函数更高效。如果您的函数对性能要求比较高，则您可以一次转换一个输入参数，同时保持代码的其余部分不变。

```
function y = myFunc(a,b,c)
    a = convertStringsToChars(a);
    b = convertStringsToChars(b);
    c = convertStringsToChars(c);
    ...

```

## 在新代码中采用字符串的建议方法

如果您的代码中依赖项很少，或者您正在开发全新代码，请考虑使用字符串数组作为主要文本数据类型。处理大量文本时，字符串数组能够提供良好的性能和高效的内存使用。与字符向量元胞数组不同，字符串数组具有同构数据类型。字符串数组更有利于轻松编写可维护的代码。要在使用字符串数组的同时保持与其他文本数据类型的向后兼容性，请遵循这些方法。

### 函数

- 接受任何文本数据类型作为输入参数。
  - 如果某个输入参数可以是字符串数组，则也允许它是字符向量或字符向量元胞数组。
  - 接受字符数组作为名称-值对组参数中的名称和值。
    - 在名称-值对组参数中，允许将名称指定为字符向量或字符串 - 即将名称用单引号或双引号括起来。如果某个值可以是字符串数组，则使之也可以是字符向量或字符向量元胞数组。
  - 不要接受字符串数组的元胞数组作为文本输入参数。
    - 字符串数组的元胞数组在每个元胞中都有一个字符串数组。虽然您可以创建这样的元胞数组，但不推荐将其用于存储文本。如果您的代码使用字符串作为主要文本数据类型，则将文本的多个片段存储在一个字符串数组中，而不是存储在字符串数组的元胞数组中。

但是，如果您的代码接受异构元胞数组作为输入，则考虑接受包含字符串的元胞数组。

- 一般情况下，返回字符串。
  - 如您的函数返回文本形式的输出参数，则将它们作为字符串数组返回。
  - 当函数修改了输入文本时，会返回相同的数据类型。
    - 如您的函数修改了输入文本并将修改后的文本作为输出参数返回，则输入和输出参数应该具有相同的数据类型。

### 类

- 将方法视为函数。

- 如前一节所述，接受字符向量和字符向量元胞数组作为输入参数。一般情况下，将字符串作为输出返回。
- 将属性指定为字符串数组。
  - 如果某个属性包含文本，则使用字符串数组设置该属性。当您访问该属性时，将它的值作为字符串数组返回。

## 如何在新代码中保持兼容性

当您编写新代码或修改代码以使用字符串数组作为主要文本数据类型时，请保持与其他文本数据类型的向后兼容性。您可以接受字符向量或字符向量元胞数组作为输入参数，然后立即将它们转换为字符串数组。如果您在函数的开头执行此转换，则您的其余代码只能使用字符串数组。

`convertCharsToStrings` 函数提供一种处理所有输入参数的方法，只转换其中的字符向量参数或字符向量元胞数组参数。要使您的新代码接受这些文本数据类型作为输入，请在函数和方法的开头添加对 `convertCharsToStrings` 的调用。

例如，如果您定义了接受三个输入参数的函数 `myFunc`，则使用 `convertCharsToStrings` 处理所有三个输入。

```
function y = myFunc(a,b,c)
    [a,b,c] = convertCharsToStrings(a,b,c);
    <line 1 of original code>
    <line 2 of original code>
    ...

```

在此示例中，参数 `[a,b,c]` 覆盖了原有的输入参数。如果有任意输入参数不是字符向量或字符向量元胞数组，则该参数保持不变。

如果 `myFunc` 接受可变数目的输入参数，则处理由 `varargin` 指定的所有参数。

```
function y = myFunc(varargin)
    varargin{:} = convertCharsToStrings(varargin{:});
    ...

```

### 性能方面的考虑

在转换一个输入参数时，`convertCharsToStrings` 函数更高效。如果您的函数对性能要求比较高，则您可以一次转换一个输入参数，同时保持代码的其余部分不变。

```
function y = myFunc(a,b,c)
    a = convertCharsToStrings(a);
    b = convertCharsToStrings(b);
    c = convertCharsToStrings(c);
    ...

```

## 如何手动转换输入参数

尽可能避免手动转换包含文本的输入参数，而改为使用 `convertStringsToChars` 或 `convertCharsToStrings` 函数。自行检查输入参数的数据类型并进行转换不仅繁琐，而且容易出错。

如果您必须转换输入参数，请使用下表中的函数。

转换	函数
将字符串标量转换为字符向量	<code>char</code>
将字符串数组转换为字符向量元胞数组	<code>cellstr</code>
将字符向量转换为字符串标量	<code>string</code>
将字符向量元胞数组转换为字符串数组	<code>string</code>

## 如何检查参数数据类型

要检查可能包含文本的输入参数的数据类型，请考虑使用下表中显示的模式。

要求的输入参数类型	旧检查	新检查
字符向量或字符串标量	<code>ischar(X)</code>	<code>ischar(X)    isStringScalar(X)</code> <code>validateattributes(X, {'char','string'},{'scalarmtext'})</code>
字符向量或字符串标量	<code>validateattributes(X,{'char'},{'row'})</code>	<code>validateattributes(X, {'char','string'},{'scalarmtext'})</code>
非空字符向量或字符串标量	<code>ischar(X) &amp;&amp; ~isempty(X)</code>	<code>(ischar(X)    isStringScalar(X)) &amp;&amp; strlength(X) ~= 0</code> <code>(ischar(X)    isStringScalar(X)) &amp;&amp; X ~= ''</code>
字符向量元胞数组或字符串数组	<code>iscellstr(X)</code>	<code>iscellstr(X)    isstring(X)</code>
任何文本数据类型	<code>ischar(X)    iscellstr(X)</code>	<code>ischar(X)    iscellstr(X)    isstring(X)</code>

### 检查空字符串

空字符串是指没有字符的字符串。MATLAB 将空字符串显示为一对中间没有任何内容的双引号 ("")。但是，一个空字符串仍然是一个  $1 \times 1$  字符串数组。它不是空数组。

推荐使用 `strlength` 函数检查字符串是否为空。

```
str = '';
tf = (strlength(str) ~= 0)
```

**注意** 不要使用 `isempty` 函数检查空字符串。一个空字符串没有字符，但仍然是一个  $1 \times 1$  字符串数组。

`strlength` 函数返回字符串数组中每个字符串的长度。如果字符串必须是字符串标量，且不为空，则两个条件都检查。

```
tf = (isStringScalar(str) && strlength(str) ~= 0)
```

如果 `str` 既可以是字符向量也可以是字符串标量，则您仍可使用 `strlength` 来确定它的长度。如果输入参数是空字符向量 ("")，则 `strlength` 返回 0。

```
tf = ((ischar(str) || isStringScalar(str)) && strlength(str) ~= 0)
```

## 检查空字符串数组

实际上，空字符串数组是空数组，即至少有一个维度的长度为 0 的数组。

要创建空字符串数组，推荐的方法是使用 `strings` 函数，并指定至少一个输入参数为 0。当输入为空字符串数组时，`isempty` 函数返回 1。

```
str = strings(0);
tf = isempty(str)
```

`strlength` 函数返回与输入字符串数组大小相同的数值数组。如果输入是一个空字符串数组，则 `strlength` 返回一个空数组。

```
str = strings(0);
L = strlength(str)
```

## 检查缺失字符串

字符串数组还可以包含缺失字符串。缺失字符串相当于数值数组的 `NaN`。它指示字符串数组包含缺失值的位置。缺失字符串显示为 `<missing>`（不带引号）。

您可以使用 `missing` 函数创建缺失字符串。要检查缺失字符串，推荐的方法是使用 `ismissing` 函数。

```
str = string(missing);
tf = ismissing(str)
```

---

**注意** 不要通过将字符串与缺失字符串进行比较来检查缺失字符串。

---

缺失字符串不等于自身，就像 `NaN` 不等于它自身一样。

```
str = string(missing);
f = (str == missing)
```

## 字符和字符串数组的术语

MathWorks 文档使用下列术语来描述字符和字符串数组。为保持一致，请在您自己的文档、错误消息和警告中使用这些术语。

- 字符向量 - 由字符构成的  $1 \times n$  数组，数据类型为 `char`。
- 字符数组 - 由字符构成的  $m \times n$  数组，数据类型为 `char`。
- 字符向量元胞数组 - 每个元胞包含一个字符向量的元胞数组。
- 字符串或字符串标量 -  $1 \times 1$  字符串数组。一个字符串标量可以包含  $1 \times n$  字符序列，但它本身是一个对象。区分术语“字符串标量”和“字符向量”是为了精确表示大小和数据类型。否则，您可以在说明中只使用术语“字符串”。
- 字符串向量 -  $1 \times n$  或  $n \times 1$  字符串数组。如果只允许使用一种大小，请在文档中加以说明。例如，使用“ $1 \times n$  字符串数组”来描述该大小的数组。
- 字符串数组 -  $m \times n$  字符串数组。
- 空字符串 - 没有字符的字符串标量。
- 空字符串数组 - 至少有一个维度的大小为 0 的字符串数组。
- 缺失字符串 - 作为缺失值的字符串标量（显示为 `<missing>`）。

### 另请参阅

`cellstr | char | convertCharsToStrings | convertContainedStringsToChars | convertStringsToChars | isStringScalar | iscellstr | ischar | isstring | string | strings | strlength | validateattributes`

### 详细信息

- “创建字符串数组” (第 6-5 页)
- “测试空字符串和缺失值” (第 6-20 页)
- “比较文本” (第 6-31 页)
- “搜索和替换文本” (第 6-36 页)
- “有关字符串数组的常见问题解答” (第 6-48 页)

## 函数摘要

MATLAB 提供以下函数用于处理字符数组：

- 用于创建字符数组的函数
- 用于修改字符数组的函数
- 用于读取和处理字符数组的函数
- 用于搜索或比较字符数组的函数
- 用于判断类或内容的函数
- 在数值数据和文本数据类型之间转换的函数
- 将字符向量元胞数组当作集合进行处理的函数

### 用于创建字符数组的函数

函数	说明
'chr'	创建在引号之间指定的字符向量。
<b>blanks</b>	创建一个由空值组成的字符向量。
<b>sprintf</b>	将格式化数据写为文本。
<b>strcat</b>	串联字符数组。
<b>char</b>	垂直串联字符数组。

### 用于修改字符数组的函数

函数	说明
<b>deblank</b>	删除尾部空白。
<b>lower</b>	将所有字母转换为小写。
<b>sort</b>	按升序或降序对元素进行排序。
<b>strjust</b>	对齐字符数组。
<b>strrep</b>	替换字符数组中的文本。
<b>trimstr</b>	删除前导和尾随空白。
<b>upper</b>	将所有字母转换为大写。

### 用于读取和处理字符数组的函数

函数	说明
<b>eval</b>	执行 MATLAB 表达式。
<b>sscanf</b>	按照一定格式控制读取字符数组。

### 用于搜索或比较字符数组的函数

函数	说明
<b>regexp</b>	匹配正则表达式（第 2-48 页）。
<b>strcmp</b>	比较字符数组。
<b>strcmpi</b>	比较字符数组，忽略大小写。
<b>strfind</b>	查找字符向量中的词汇。
<b>strncmp</b>	比较字符数组的前 N 个字符。
<b>strncmpi</b>	比较前 N 个字符并忽略大小写。
<b>strtok</b>	查找字符向量中的标文。
<b>textscan</b>	读取字符数组中的数据。

### 用于判断类或内容的函数

函数	说明
<b>iscellstr</b>	对于字符向量元胞数组，返回 <b>true</b> 。
<b>ischar</b>	对于字符数组，返回 <b>true</b> 。
<b>isletter</b>	对于字母，返回 <b>true</b> 。
<b>isstrprop</b>	确定字符串是否为指定的类别。
<b>isspace</b>	对于空白字符，返回 <b>true</b> 。

### 在数值数据和文本数据类型之间转换的函数

函数	说明
<b>char</b>	转换为字符数组。
<b>cellstr</b>	将字符数组转换为字符向量元胞数组。
<b>double</b>	将字符数组转换为数值代码。
<b>int2str</b>	将整数表示为文本。
<b>mat2str</b>	将矩阵转换为您可以对其运行 <b>eval</b> 的字符数组。
<b>num2str</b>	将数字表示为文本。
<b>str2num</b>	将字符向量转换为它代表的数字。
<b>str2double</b>	将字符向量转换为它代表的双精度值。

### 将字符向量元胞数组当作集合进行处理的函数

函数	说明
<b>intersect</b>	设置两个向量的交集。
<b>ismember</b>	检测集合的成员。
<b>setdiff</b>	返回两个向量的集合差。
<b>setxor</b>	设置两个向量的异或。
<b>union</b>	设置两个向量的并集。
<b>unique</b>	设置向量的唯一元素。

# 日期和时间

---

- “表示 MATLAB 中的日期和时间” (第 7-2 页)
- “指定时区” (第 7-5 页)
- “将日期和时间转换为儒略日期或 POSIX 时间” (第 7-7 页)
- “设置日期和时间显示格式” (第 7-10 页)
- “生成日期与时间的序列” (第 7-13 页)
- “在不同区域设置之间共享代码和数据” (第 7-19 页)
- “提取或分配日期时间数组的日期和时间分量” (第 7-21 页)
- “合并来自各自变量的日期和时间” (第 7-24 页)
- “日期和时间算术运算” (第 7-26 页)
- “比较日期和时间” (第 7-31 页)
- “绘制日期和持续时间图” (第 7-34 页)
- “支持日期和时间数组的核心函数” (第 7-39 页)
- “在日期时间数组、数值和文本之间转换” (第 7-40 页)
- “日期向量和字符串结转” (第 7-45 页)
- “转换日期向量返回意外输出” (第 7-46 页)

## 表示 MATLAB 中的日期和时间

存储日期和时间信息的主要方法是使用 **datetime** 数组，该数组支持算术运算、排序、比较、绘图和格式化显示方式。算术差异的结果在 **duration** 数组中返回；如果使用基于日历的函数，则在 **calendarDuration** 数组中返回。

例如，创建一个 MATLAB 日期时间数组表示以下两个日期：2014 年 6 月 28 日上午 6 点和 2014 年 6 月 28 日上午 7 点。为此日期时间指定年、月、日、时、分和秒分量的数值。

```
t = datetime(2014,6,28,6:7,0,0)  
t =  
28-Jun-2014 06:00:00 28-Jun-2014 07:00:00
```

通过为日期时间数组的属性分配新的值，更改日期或时间分量的值。例如，通过为 **Day** 属性分配新的值，更改每个日期时间值的日期。

```
t.Day = 27:28  
t =  
27-Jun-2014 06:00:00 28-Jun-2014 07:00:00
```

通过更改数组的 **Format** 属性，可更改其显示格式。下面的格式没有显示任何时间分量。但日期时间数组中的值不会发生任何变化。

```
t.Format = 'MMM dd, yyyy'  
t =  
Jun 27, 2014 Jun 28, 2014
```

如果用一个 **datetime** 数组减另一个，结果是以固定长度为单位的 **duration** 数组。

```
t2 = datetime(2014,6,29,6,30,45)  
t2 =  
29-Jun-2014 06:30:45  
  
d = t2 - t  
  
d =  
48:30:45 23:30:45
```

默认情况下，**duration** 数组按照“时:分:秒”的格式显示。通过更改数组的 **Format** 属性，可更改持续时间的显示格式。您可以使用单一单位显示持续时间的值，例如小时数。

```
d.Format = 'h'  
d =  
48.512 hrs 23.512 hrs
```

您可以使用 **seconds**、**minutes**、**hours**、**days** 或 **years** 函数创建单一单位的持续时间。例如，创建 2 天的持续时间，其中每天都是 24 小时整。

```
d = days(2)
```

```
d =
2 days
```

您也可以按可变长度的单一单位创建日历持续时间。例如，一个月可能为 28、29、30 或 31 天。指定 2 个月的日历持续时间。

**L = calmonths(2)**

```
L =
2mo
```

您还可以使用 **caldays**、**calweeks**、**calquarters** 和 **calyears** 函数按其他单位指定日历持续时间。

添加一定的日历月数和日历天数。天数与月数仍然是分开的，这是因为月中的天数不是固定的，只有您向某一特定的日期时间添加了日历持续时间后才能确定。

**L = calmonths(2) + caldays(35)**

```
L =
2mo 35d
```

将日历持续时间与日期时间值相加以计算新的日期。

**t2 = t + calmonths(2) + caldays(35)**

```
t2 =
```

```
Oct 01, 2014 Oct 02, 2014
```

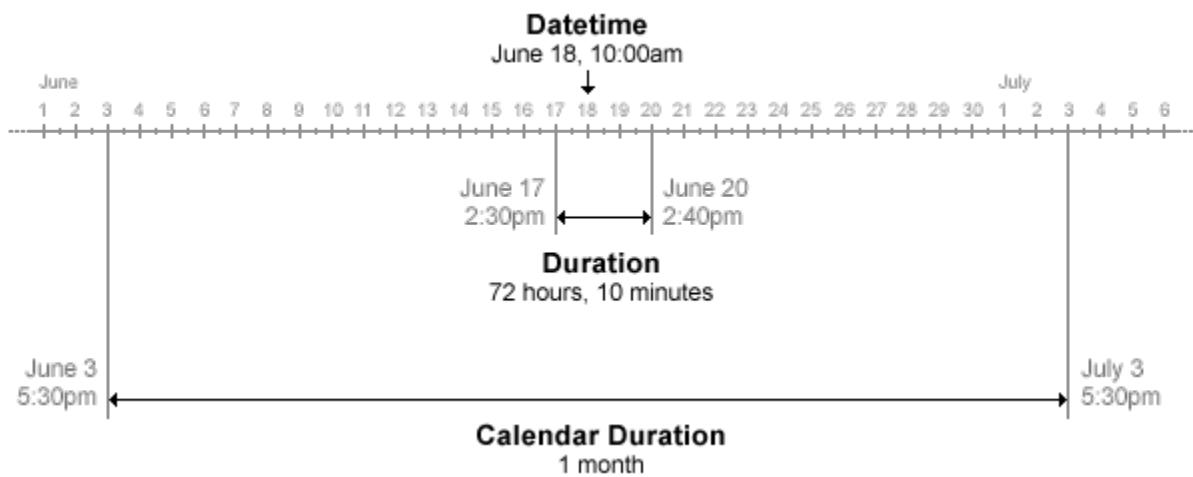
**t2** 也是一个 **datetime** 数组。

**whos t2**

Name	Size	Bytes	Class	Attributes
t2	1x2	161	datetime	

总的说来，表示日期和时间的方式有多种，MATLAB 为每种方法提供了一个数据类型：

- 使用 **datetime** 数据类型表示时间点。  
例如：2014 年 6 月 18 日（星期三）10:00:00
- 使用 **duration** 数据类型表示以固定长度单位计算的一段时间或持续时间。使用 **duration** 数据类型时，1 天始终等于 24 小时，1 年始终等于 365.2425 天。  
例如：72 小时 10 分钟
- 使用 **calendarDuration** 数据类型表示以可变长度单位计算的一段时间或持续时间。  
例如：1 个月（可能为 28、29、30 或 31 天）。  
**calendarDuration** 数据类型还考虑了夏令时更改和闰年，因此 1 天可能大于或小于 24 小时，1 年可能有 365 或 366 天。



**另请参阅**

[calendarDuration](#) | [datetime](#) | [duration](#)

## 指定时区

在 MATLAB 中，时区包含了与协调世界时 (UTC) 的时间偏移、夏令时偏移以及对这些值的一系列历史调整。时区设置存储在每个 `datetime` 数组的 `TimeZone` 属性中。当您创建一个日期时间时，默认情况下是未设置时区的。即日期时间的 `TimeZone` 属性为空 ('')。如果您不用处理来自多个时区的日期时间值，也无需考虑夏令时，则无需指定此属性。

在创建日期时间时，您可以使用 `'TimeZone'` 名称-值对组参数指定时区。时区值 `'local'` 指定系统时区。要显示每个日期时间的时区偏移，请在 `'Format'` 参数的值中包含 `'Z'` 等时区偏移设定符。

```
t = datetime(2014,3,8,9,6,0,0,'TimeZone','local',...
    'Format','d-MMM-y HH:mm:ss Z')
```

`t =`

8-Mar-2014 06:00:00 -0500 9-Mar-2014 06:00:00 -0400

根据日期时间是否在夏令时期间发生，将显示不同的时区偏移。

您可以修改现有日期时间的时区。例如，使用圆点表示法更改 `t` 的 `TimeZone` 属性。您可以将时区值指定为 IANA 时区数据库中的时区地区的名称。时区地区综合考虑了从该地理区域观察到的标准时间和夏令时与 UTC 之间的偏移所适用的当前规则和历史规则。

```
t.TimeZone = 'Asia/Shanghai'
```

`t =`

8-Mar-2014 19:00:00 +0800 9-Mar-2014 18:00:00 +0800

您也可以将时区值指定为 `+HH:mm` 或 `-HH:mm` 形式的字符向量，它表示该时区与 UTC 之间存在一个固定的偏移量，而不考虑夏令时。

```
t.TimeZone = '+08:00'
```

`t =`

8-Mar-2014 19:00:00 +0800 9-Mar-2014 18:00:00 +0800

对于包含时区信息的日期时间数组，在运算时会自动考虑时区差异。例如，创建一个不同时区的日期时间。

```
u = datetime(2014,3,9,6,0,0,'TimeZone','Europe/London',...
    'Format','d-MMM-y HH:mm:ss Z')
```

`u =`

9-Mar-2014 06:00:00 +0000

查看两个日期时间数组之间的时差。

```
dt = t - u
```

`dt =`

-19:00:00 04:00:00

当您执行涉及日期时间数组的运算时，这些数组要么必须都有所关联的时区，要么必须都没有设置时区。

## 另请参阅

[datetime](#) | [timezones](#)

## 相关示例

- “表示 MATLAB 中的日期和时间” (第 7-2 页)
- “将日期和时间转换为儒略日期或 POSIX 时间” (第 7-7 页)

## 将日期和时间转换为儒略日期或 POSIX 时间

您可以转换 `datetime` 数组，以采用专门的数值格式来表示时间点。通常，这些格式会将时间点表示为自指定的起点以来经过的秒数或天数。例如，儒略日期是指自儒略周期开始以来经过的整天和小数天数。POSIX® 时间是指自 UTC (协调世界时) 1970 年 1 月 1 日 00:00:00 以来经过的秒数。MATLAB® 提供 `juliandate` 和 `posixtime` 函数来将 `datetime` 数组转换为儒略日期和 POSIX 时间。

虽然 `datetime` 数组不必带有时区，但将“未设置时区的”`datetime` 值转换为儒略日期或 POSIX 时间可能会导致意外的结果。要确保获得预期的结果，请在转换之前指定时区。

### 在转换之前指定时区

您可以为 `datetime` 数组指定时区，但不一定需要执行该操作。实际上，`datetime` 函数默认情况下会创建一个“未设置时区的”`datetime` 数组。

为当前日期和时间创建 `datetime` 值。

```
d = datetime('now')
```

```
d = datetime
30-Jul-2019 17:13:35
```

`d` 是根据您的计算机上的本地时间构建的，没有与其关联的时区。在许多情况下，您可能会认为可将未设置时区的 `datetime` 数组中的时间视为本地时间。但是，`juliandate` 和 `posixtime` 函数将未设置时区的 `datetime` 数组中的时间视为 UTC 时间，而非本地时间。为避免出现任何混乱，建议您避免对未设置时区的 `datetime` 数组使用 `juliandate` 和 `posixtime`。例如，避免在代码中使用 `posixtime(datetime('now'))`。

如果 `datetime` 数组包含不表示 UTC 时间的值，请使用 `TimeZone` 名称-值对组参数来指定时区，以使 `juliandate` 和 `posixtime` 正确解释 `datetime` 值。

```
d = datetime('now','TimeZone','America/New_York')
```

```
d = datetime
30-Jul-2019 17:13:35
```

作为备选方法，您可以在创建该数组后指定 `TimeZone` 属性。

```
d.TimeZone = 'America/Los_Angeles'
```

```
d = datetime
30-Jul-2019 14:13:35
```

要查看时区的完整列表，请使用 `timezones` 函数。

### 将已设置时区和未设置时区的日期时间值转换为儒略日期

儒略日期是指前公历日历中自公元前 4714 年 11 月 24 日或前儒略日历中自公元前 4713 年 1 月 1 日中午起的天数（包括小数天数）。要将 `datetime` 数组转换为儒略日期，请使用 `juliandate` 函数。

创建一个 `datetime` 数组并指定其时区。

```
DZ = datetime('2016-07-29 10:05:24') + calmonths(1:3);
DZ.TimeZone = 'America/New_York'
```

```
DZ = 1x3 datetime array
29-Aug-2016 10:05:24 29-Sep-2016 10:05:24 29-Oct-2016 10:05:24
```

将 D 转换为等同的儒略日期。

```
format longG
```

```
JDZ = juliandate(DZ)
```

```
JDZ = 1x3
```

```
2457630.08708333 2457661.08708333 2457691.08708333
```

为 DZ 创建未设置时区的副本。将 D 转换为等同的儒略日期。由于 D 没有时区，因此 juliandate 将时间视为 UTC 时间。

```
D = DZ;
```

```
D.TimeZone = '';
```

```
JD = juliandate(D)
```

```
JD = 1x3
```

```
2457629.92041667 2457660.92041667 2457690.92041667
```

比较 JDZ 和 JD。其差等于 UTC 和 America/New\_York 时区之间的时区偏移量（以小数天数计）。

```
JDZ - JD
```

```
ans = 1x3
```

```
0.166666666511446 0.166666666511446 0.166666666511446
```

### 将已设置时区和未设置时区的日期时间值转换为 POSIX 时间

POSIX 时间是指自 UTC（协调世界时）1970 年 1 月 1 日 00:00:00 以来经过的秒数（包括小数秒），忽略闰秒。要将 datetime 数组转换为 POSIX 时间，请使用 posixtime 函数。

创建一个 datetime 数组并指定其时区。

```
DZ = datetime('2016-07-29 10:05:24') + calmonths(1:3);
```

```
DZ.TimeZone = 'America/New_York'
```

```
DZ = 1x3 datetime array
```

```
29-Aug-2016 10:05:24 29-Sep-2016 10:05:24 29-Oct-2016 10:05:24
```

将 D 转换为等同的 POSIX 时间。

```
PTZ = posixtime(DZ)
```

```
PTZ = 1x3
```

```
1472479524 1475157924 1477749924
```

为 DZ 创建未设置时区的副本。将 D 转换为等同的 POSIX 时间。由于 D 没有时区，因此 posixtime 将时间视为 UTC 时间。

```
D = DZ;  
D.TimeZone = ";  
PT = posixtime(D)
```

PT = 1×3

```
1472465124      1475143524      1477735524
```

比较 PTZ 和 PT。其差等于 UTC 和 America/New\_York 时区之间的时区偏移量（以秒计）。

PTZ - PT

ans = 1×3

```
14400      14400      14400
```

## 另请参阅

`datetime` | `juliandate` | `posixtime` | `timezones`

## 相关示例

- “表示 MATLAB 中的日期和时间”（第 7-2 页）
- “指定时区”（第 7-5 页）

## 设置日期和时间显示格式

### 本节内容

- “单个日期和持续时间数组的格式”（第 7-10 页）
- “`datetime` 显示格式”（第 7-10 页）
- “`duration` 显示格式”（第 7-11 页）
- “`calendarDuration` 显示格式”（第 7-11 页）
- “默认 `datetime` 格式”（第 7-12 页）

### 单个日期和持续时间数组的格式

`datetime`、`duration` 和 `calendarDuration` 数组有一个 `Format` 属性，可控制每个数组中各值的显示方式。当您创建一个日期时间数组时，除非您明确提供格式，否则将使用 MATLAB 的全球默认日期时间显示格式。可使用圆点表示法访问 `Format` 属性以查看或更改其值。例如，要将 `datetime` 数组 `t` 的显示格式设置为默认格式，请键入：

```
t.Format = 'default'
```

更改 `Format` 属性并不会更改数组中的值，它只会更改其显示方式。例如，以下可能是同一个 `datetime` 值的表示方式（后两种不显示任何时间分量）：

```
Thursday, August 23, 2012 12:35:00  
August 23, 2012  
23-Aug-2012
```

`datetime`、`duration` 和 `calendarDuration` 数据类型的 `Format` 属性可接受不同的格式作为输入。

### datetime 显示格式

您可以将 `Format` 属性设置为下述字符向量之一。

Format 的值	说明
'default'	使用默认的显示格式。
'defaultdate'	使用默认的日期显示格式，不显示时间分量。

如要更改默认格式，请参考“默认 `datetime` 格式”（第 7-12 页）。

您也可以使用字母 A-Z 和 a-z 来指定自定义日期格式。可以使用连字符、空格或冒号等非字母字符来分隔字段。下表列举了多种常见的显示格式和纽约市 2014 年 4 月 19 日（星期六）下午 9:41:06 的格式化输出示例。

Format 的值	示例
'yyyy-MM-dd'	2014-04-19
'dd/MM/yyyy'	19/04/2014
'dd.MM.yyyy'	19.04.2014
'yyyy 年 MM 月 dd 日'	2014 年 04 月 19 日
'MMMM d, yyyy'	April 19, 2014

Format 的值	示例
'eeee, MMMM d, yyyy h:mm a'	Saturday, April 19, 2014 9:41 PM
'MMMM d, yyyy HH:mm:ss Z'	April 19, 2014 21:41:06 -0400
'yyyy-MM-dd'T'HH:mmXXX'	2014-04-19T21:41-04:00

有关有效符号标识符的完整列表，请参阅日期时间数组的 **Format** 属性。

---

**注意** `datetime` 可接受的字母标识符与 `datestr`、`datenum` 和 `datevec` 函数所使用的字母标识符不同。

---

## duration 显示格式

要将持续时间显示为包含小数部分（例如 1.234 小时）的单个数字，请指定以下字符向量之一：

Format 的值	说明
'y'	精确定长年的数目。固定长度的一年等于 365.2425 天。
'd'	精确定长天的数目。固定长度的一天等于 24 小时。
'h'	小时数
'm'	分钟数
's'	秒数

要指定显示的小数位数，请使用 `format` 函数。

要以数字计时器的形式显示持续时间，请指定下列字符向量之一。

- 'dd:hh:mm:ss'
- 'hh:mm:ss'
- 'mm:ss'
- 'hh:mm'

您还可以通过附加最多 9 个 S 字符来显示最多 9 个秒小数位。例如，'hh:mm:ss.SSS' 以 3 位数显示持续时间的毫秒数。

更改 `Format` 属性并不会更改数组中的值，它只会更改其显示方式。

## calendarDuration 显示格式

将 `calendarDuration` 数组的 `Format` 属性指定为一个字符向量，其中可依序包含以下字符 `y`、`q`、`m`、`w`、`d` 和 `t`。格式必须包含 `m`、`d` 和 `t`。

下表描述了用这些字符表示的日期和时间分量。

字符	单位	必需？
y	年	否
q	季度（3 个月的倍数）	否

字符	单位	必需?
m	月	是
w	周	否
d	天	是
t	时间 (小时、分钟和秒)	是

要指定秒数显示的小数位数，请使用 **format** 函数。

如果时间分量的日期或时间值为零，将不显示该值。

更改 **Format** 属性并不会更改数组中的值，它只会更改其显示方式。

## 默认 **datetime** 格式

对于那些在创建时未显式设置显示格式的 **datetime** 组，您可以设置默认格式来控制其显示方式。当您将 **datetime** 数组的 **Format** 属性设置为 '**default**' 或 '**defaultdate**' 时，也将适用这些格式。当您更改默认设置时，设置为使用默认格式的 **datetime** 数组将自动按新设置显示。

对默认格式的更改将在各个 MATLAB 会话间保持一致。

要指定默认格式，请键入

```
datetime.setDefaultFormats('default',fmt)
```

其中 **fmt** 是由字母 A-Z 和 a-z 组成的字符向量，用于对上面所提到的 **datetime** 数组的 **Format** 属性进行描述。例如，

```
datetime.setDefaultFormats('default','yyyy-MM-dd hh:mm:ss')
```

将默认日期时间格式设置为包含 4 位数的年份、2 位数的月份、2 位数的天数以及小时、分钟和秒数值。

此外，您还可以为所创建的不含时间分量的日期时间指定一个默认格式。例如，

```
datetime.setDefaultFormats('defaultdate','yyyy-MM-dd')
```

将默认日期格式设置为包含 4 位数的年份、2 位数的月份以及 2 位数的天数。

如要将默认格式和默认的仅日期格式重置为出厂默认设置，请键入

```
datetime.setDefaultFormats('reset')
```

出厂默认格式取决于您的系统区域设置。

您也可以在**预设**对话框中设置默认格式。有关详细信息，请参阅“**设置命令行窗口预设项**”。

## 另请参阅

[calendarDuration](#) | [datetime](#) | [duration](#) | [format](#)

# 生成日期与时间的序列

## 本节内容

- “两个端点间已知步长的日期时间或持续时间值的序列”（第 7-13 页）
- “添加持续时间或日历持续时间以创建日期的序列”（第 7-15 页）
- “指定日期或持续时间序列的长度和端点”（第 7-16 页）
- “使用日历规则的日期时间值的序列”（第 7-16 页）

## 两个端点间已知步长的日期时间或持续时间值的序列

此示例说明如何使用冒号 (:) 运算符生成 `datetime` 或 `duration` 值的序列，该方法与创建规律间隔数值向量的方法相同。

### 使用默认步长

从 2013 年 11 月 1 日开始至 2013 年 11 月 5 日结束，创建日期时间值的序列。默认步长为一个日历天。

```
t1 = datetime(2013,11,1,8,0,0);
t2 = datetime(2013,11,5,8,0,0);
t = t1:t2
```

```
t = 1x5 datetime array
Columns 1 through 3
01-Nov-2013 08:00:00 02-Nov-2013 08:00:00 03-Nov-2013 08:00:00
Columns 4 through 5
04-Nov-2013 08:00:00 05-Nov-2013 08:00:00
```

### 指定步长

使用 `caldays` 函数指定步长为 2 个日历天。

```
t = t1:caldays(2):t2
t = 1x3 datetime array
01-Nov-2013 08:00:00 03-Nov-2013 08:00:00 05-Nov-2013 08:00:00
```

用天以外的其他单位指定步长。创建间隔为 18 小时的日期时间值序列。

```
t = t1:hours(18):t2
t = 1x6 datetime array
Columns 1 through 3
01-Nov-2013 08:00:00 02-Nov-2013 02:00:00 02-Nov-2013 20:00:00
Columns 4 through 6
03-Nov-2013 14:00:00 04-Nov-2013 08:00:00 05-Nov-2013 02:00:00
```

使用 `years`、`days`、`minutes` 和 `seconds` 函数，以其他固定长度的日期和时间单位创建日期时间和持续时间的序列。创建 0 到 3 分钟之间的 `duration` 值序列，增量为 30 秒。

```
d = 0:seconds(30):minutes(3)  
  
d = 1x7 duration array  
0 sec 30 sec 60 sec 90 sec 120 sec 150 sec 180 sec
```

### 比较固定长度持续时间和日历持续时间的步长

将时区赋给 `t1` 和 `t2`。在 `America/New_York` 时区中，`t1` 现在刚好发生在夏令时更改前。

```
t1.TimeZone = 'America/New_York';  
t2.TimeZone = 'America/New_York';
```

如果使用一个日历天的步长创建序列，则连续的 `datetime` 值之间的差并非始终为 24 小时。

```
t = t1:t2;  
dt = diff(t)  
  
dt = 1x4 duration array  
24:00:00 25:00:00 24:00:00 24:00:00
```

创建日期时间值的序列，以一个固定长度天间隔开，

```
t = t1:days(1):t2  
  
t = 1x5 datetime array  
Columns 1 through 3  
  
01-Nov-2013 08:00:00 02-Nov-2013 08:00:00 03-Nov-2013 07:00:00  
  
Columns 4 through 5  
  
04-Nov-2013 07:00:00 05-Nov-2013 07:00:00
```

验证连续的 `datetime` 值之间的差是否为 24 小时。

```
dt = diff(t)  
  
dt = 1x4 duration array  
24:00:00 24:00:00 24:00:00 24:00:00
```

### 整数步长

如果用整数来指定步长，则该整数将被解释为若干个 24 小时天数。

```
t = t1:1:t2  
  
t = 1x5 datetime array  
Columns 1 through 3  
  
01-Nov-2013 08:00:00 02-Nov-2013 08:00:00 03-Nov-2013 07:00:00  
  
Columns 4 through 5
```

```
04-Nov-2013 07:00:00 05-Nov-2013 07:00:00
```

## 添加持续时间或日历持续时间以创建日期的序列

此示例演示了如何向一个日期时间值添加一个持续时间或日历持续时间来创建日期时间值的序列。

创建一个日期时间标量，表示 2013 年 11 月 1 日上午 8:00。

```
t1 = datetime(2013,11,1,8,0,0);
```

将此日期时间值与固定长度的小时序列相加。

```
t = t1 + hours(0:2)
```

```
t = 1x3 datetime array
```

```
01-Nov-2013 08:00:00 01-Nov-2013 09:00:00 01-Nov-2013 10:00:00
```

将此日期时间值与日历月的序列相加。

```
t = t1 + calmonths(1:5)
```

```
t = 1x5 datetime array
```

```
Columns 1 through 3
```

```
01-Dec-2013 08:00:00 01-Jan-2014 08:00:00 01-Feb-2014 08:00:00
```

Columns 4 through 5

```
01-Mar-2014 08:00:00 01-Apr-2014 08:00:00
```

**t** 中的每个日期时间都出现在每个月的第一天。

验证 **t** 中的日期间隔是否为 1 个月。

```
dt = caldiff(t)
```

```
dt = 1x4 calendarDuration array
```

```
1mo 1mo 1mo 1mo
```

确定各日期间的天数。

```
dt = caldiff(t,'days')
```

```
dt = 1x4 calendarDuration array
```

```
31d 31d 28d 31d
```

将多个日历月与日期 2014 年 1 月 31 日相加以创建一个日期序列，使每个日期为每个月的最后一天。

```
t = datetime(2014,1,31) + calmonths(0:11)
```

```
t = 1x12 datetime array
```

```
Columns 1 through 5
```

```
31-Jan-2014 28-Feb-2014 31-Mar-2014 30-Apr-2014 31-May-2014
```

Columns 6 through 10

```
30-Jun-2014 31-Jul-2014 31-Aug-2014 30-Sep-2014 31-Oct-2014
```

Columns 11 through 12

```
30-Nov-2014 31-Dec-2014
```

## 指定日期或持续时间序列的长度和端点

此示例说明如何使用 `linspace` 函数在两个指定的端点之间创建等间距的日期时间或持续时间值。

在 2014 年 4 月 14 日与 2014 年 8 月 4 日之间创建五个等间距日期的序列。首先，定义端点。

```
A = datetime(2014,04,14);
B = datetime(2014,08,04);
```

`linspace` 的第三个输入参数指定了在两个端点间要生成的线性间隔点的个数。

```
C = linspace(A,B,5)
```

```
C = 1x5 datetime array
14-Apr-2014 12-May-2014 09-Jun-2014 07-Jul-2014 04-Aug-2014
```

在 1 和 5.5 小时之间创建六个等间距持续时间的序列。

```
A = duration(1,0,0);
B = duration(5,30,0);
C = linspace(A,B,6)
```

```
C = 1x6 duration array
01:00:00 01:54:00 02:48:00 03:42:00 04:36:00 05:30:00
```

## 使用日历规则的日期时间值的序列

此示例说明如何使用 `dateshift` 函数来生成日期和时间的序列，其中每个实例遵循日历单位或时间单位的有关规则。例如，每个日期时间值必定出现在一个月的开始、一个星期中特定的一天，或是一分钟要结束的时候。序列中生成的日期时间值不一定是等间距的。

### 一个星期中特定某天的日期

生成一个由后续三个连续星期一组成的日期序列。首先，定义今天的日期。

```
t1 = datetime('today','Format','dd-MMM-yyyy eee')
t1 = datetime
30-Jul-2019 Tue
```

`dateshift` 的第一个输入参数始终是您要从其生成序列的 `datetime` 数组。将 `'dayofweek'` 指定为第二个输入参数，表示输出序列中的日期时间值必须落在一个星期中的特定某天。您可以通过数值或名称来指定一周中的星期几。例如，可以将星期一指定为 2 或 `'Monday'`。

```
t = dateshift(t1,'dayofweek',2,1:3)

t = 1x3 datetime array
 05-Aug-2019 Mon  12-Aug-2019 Mon  19-Aug-2019 Mon
```

## 每月的开始日期

从 2014 年 4 月 1 日开始生成由每月开始日期组成的序列。将 'start' 指定为 `dateshift` 的第二个输入参数，表示输出序列中的所有日期时间值应当落在特定时间单位的开始。第三个输入参数定义了时间单位，在本例中为月。`dateshift` 的最后一个输入参数可以是整数值数组，用来指定 `t1` 的推移方式。在本例中，0 对应当前月的起始，4 对应从 `t1` 起第四个月的起始。

```
t1 = datetime(2014,04,01);
t = dateshift(t1,'start','month',0:4)

t = 1x5 datetime array
 01-Apr-2014  01-May-2014  01-Jun-2014  01-Jul-2014  01-Aug-2014
```

## 每月的末尾日期

从 2014 年 4 月 1 日开始生成每月末尾日期的序列。

```
t1 = datetime(2014,04,01);
t = dateshift(t1,'end','month',0:2)

t = 1x3 datetime array
 30-Apr-2014  31-May-2014  30-Jun-2014
```

确定各日期间的天数。

```
dt = caldiff(t,'days')

dt = 1x2 calendarDuration array
 31d  30d
```

日期不是等间距的。

## 其他日期和时间单位

您可以指定星期、天和小时等其他时间单位。

```
t1 = datetime('now')

t1 = datetime
 30-Jul-2019 17:06:58

t = dateshift(t1,'start','hour',0:4)

t = 1x5 datetime array
Columns 1 through 3

 30-Jul-2019 17:00:00  30-Jul-2019 18:00:00  30-Jul-2019 19:00:00

Columns 4 through 5
```

```
30-Jul-2019 20:00:00 30-Jul-2019 21:00:00
```

### 以前发生的日期和时间

生成从上一个小时开始的日期时间值的序列。**dateshift** 最后一个输入参数中的负整数对应早于 **t1** 的日期时间值。

```
t = dateshift(t1,'start','hour',-1:1)
```

```
t = 1x3 datetime array
```

```
30-Jul-2019 16:00:00 30-Jul-2019 17:00:00 30-Jul-2019 18:00:00
```

### 另请参阅

[dateshift](#) | [linspace](#)

# 在不同区域设置之间共享代码和数据

## 本节内容

- “编写独立于区域设置的日期和时间代码”（第 7-19 页）
- “用其他语言编写日期”（第 7-20 页）
- “读取以其他语言编写的日期”（第 7-20 页）

## 编写独立于区域设置的日期和时间代码

与其他区域设置中的 MATLAB® 用户共享用于处理日期和时间的代码时，请遵循以下最佳做法。这些做法可确保相同的代码生成相同的输出显示，并且包含日期和时间的输出文件能在不同国家/地区的系统或采用不同语言设置的系统中正常读取。

创建与语言无关的日期时间值。也就是说，创建使用月份数字而非月份名称的日期时间值，例如 01 而不是 January。避免使用星期几的名称。

例如，执行以下代码：

```
t = datetime('today','Format','yyyy-MM-dd')
t = datetime
2019-07-30
```

而非：

```
t = datetime('today','Format','eeee, dd-MMM-yyyy')
t = datetime
Tuesday, 30-Jul-2019
```

使用 24 小时制格式而不是 12 小时制格式来显示小时。指定日期时间值的显示格式时使用 'HH' 标识符。

例如，执行以下代码：

```
t = datetime('now','Format','HH:mm')
t = datetime
17:12
```

而非：

```
t = datetime('now','Format','hh:mm a')
t = datetime
05:12 PM
```

指定时区信息的显示格式时，请使用 Z 或 X 标识符而非小写字母 z，以避免创建可能在其他语言或地区中无法识别的时区名称。

将时区赋给 t。

```
t.TimeZone = 'America/New_York';
```

指定一个与语言无关的包含时区的显示格式。

```
t.Format = 'dd-MM-yyyy Z'
```

```
t = datetime  
30-07-2019 -0400
```

如果您共享文件而不共享代码，则不需要在使用 MATLAB 时编写与区域设置无关的代码。但是，当您写入文件时，请确保表示日期和时间的任何文本都与语言无关。这样，其他 MATLAB 用户便可以轻松读取文件，而不必指定用于解释日期和时间数据的区域设置。

## 用其他语言编写日期

当您使用 `char` 或 `cellstr` 函数时，请为表示日期和时间的文本指定合适的格式。例如，使用 `cellstr` 将两个日期时间值转换为一个字符串向量元胞数组。指定格式和区域设置，以将每个日期时间值的日期、月份和年份表示为文本。

```
t = [datetime('today');datetime('tomorrow')]
```

```
t = 2x1 datetime array  
30-Jul-2019  
31-Jul-2019
```

```
S = cellstr(t,'dd. MMMM yyyy','de_DE')
```

```
S = 2x1 cell array  
{'30. Juli 2019'}  
{'31. Juli 2019'}
```

`S` 是一个用德语表示日期的字符串向量元胞数组。您可以将 `S` 导出到文本文件中，以用于区域设置为 `de_DE` 的系统。

## 读取以其他语言编写的日期

可以读取包含其他语言（不同于 MATLAB 所用的语言）的日期和时间的文本文件，这取决于系统区域设置。将 `textscan` 或 `readtable` 函数与 `DateLocale` 名称-值对组参数结合使用，以指定函数在解释文件中的日期时所采用的区域设置。此外，若文件包含计算机默认编码无法识别的字符，可能还需要为其指定字符编码。

- 当使用 `textscan` 函数读取文本文件时，请在使用 `fopen` 打开文件时指定文件编码。编码是 `fopen` 的第四个输入参数。
- 当使用 `readtable` 函数读取文本文件时，请使用 `FileEncoding` 名称-值对参数指定与该文件关联的字符编码。

## 另请参阅

`cellstr` | `char` | `datetime` | `readtable` | `textscan`

## 提取或分配日期时间数组的日期和时间分量

此示例演示了从现有日期时间数组中提取日期和时间分量的两种方法：访问数组属性或调用函数。然后，示例演示了如何通过修改数组属性来修改日期和时间分量。

### 访问属性以检索日期和时间分量

创建一个 `datetime` 数组。

```
t = datetime('now') + calyears(0:2) + calmonths(0:2) + hours(20:20:60)
```

```
t = 1x3 datetime array
31-Jul-2019 13:07:05 01-Sep-2020 09:07:05 03-Oct-2021 05:07:05
```

获取数组中每个日期时间的年份值。使用圆点表示法访问 `t` 的 `Year` 属性。

```
t_years = t.Year
```

```
t_years = 1x3
```

```
2019      2020      2021
```

输出 `t_years` 是一个数值数组。

通过访问 `Month` 属性获取 `t` 中每个日期时间的月份值。

```
t_months = t.Month
```

```
t_months = 1x3
```

```
7      9      10
```

您可以通过分别访问 `Hour`、`Minute` 和 `Second` 属性，检索 `t` 中每个日期时间的日、时、分和秒分量。

### 使用函数检索日期和时间分量

使用 `month` 函数获取 `t` 中每个日期时间的月份数字。使用函数是检索 `t` 的特定日期或时间分量的替代方法。

```
m = month(t)
```

```
m = 1x3
```

```
7      9      10
```

使用 `month` 函数而非 `Month` 属性获取 `t` 中每个日期时间的完整月份名称。

```
m = month(t,'name')
```

```
m = 1x3 cell array
{July}  {September}  {October}
```

您可以通过分别使用 `year`、`quarter`、`week`、`hour`、`minute` 和 `second` 函数，检索 `t` 中每个日期时间的年、季、周、日、时、分和秒分量。

获取 **t** 中每个日期时间属于一年中第几周这一数字。

```
w = week(t)
```

```
w = 1×3
```

```
31 36 41
```

### 获取多个日期和时间分量

使用 **ymd** 函数获取 **t** 的年份、月份和日期值，并作为三个单独的数值数组。

```
[y,m,d] = ymd(t)
```

```
y = 1×3
```

```
2019 2020 2021
```

```
m = 1×3
```

```
7 9 10
```

```
d = 1×3
```

```
31 1 3
```

使用 **hms** 函数获取 **t** 的小时、分钟和秒值，并作为三个单独的数值数组。

```
[h,m,s] = hms(t)
```

```
h = 1×3
```

```
13 9 5
```

```
m = 1×3
```

```
7 7 7
```

```
s = 1×3
```

```
5.0242 5.0242 5.0242
```

### 修改日期和时间分量

通过修改某个现有 **datetime** 数组的属性，将新值赋给该数组中的分量。使用圆点表示法访问特定属性。

将 **t** 中所有日期时间值的年份数字更改为 2014。使用圆点表示法修改 **Year** 属性。

```
t.Year = 2014
```

```
t = 1x3 datetime array  
31-Jul-2014 13:07:05 01-Sep-2014 09:07:05 03-Oct-2014 05:07:05
```

将 `t` 中三个日期时间值的月份分别更改为 1 月、2 月和 3 月。您必须将新的值指定为数值数组。

```
t.Month = [1,2,3]
```

```
t = 1x3 datetime array  
31-Jan-2014 13:07:05 01-Feb-2014 09:07:05 03-Mar-2014 05:07:05
```

通过为 `TimeZone` 属性赋值来设置 `t` 的时区。

```
t.TimeZone = 'Europe/Berlin';
```

更改 `t` 的显示格式，只显示日期而不显示时间信息。

```
t.Format = 'dd-MMM-yyyy'
```

```
t = 1x3 datetime array  
31-Jan-2014 01-Feb-2014 03-Mar-2014
```

如果您为日期时间分量分配的值超出常规范围，则 MATLAB® 会对分量进行归一化。一个月中日期数字的常规范围是 1 到 31。指定一个超出该范围的日期值。

```
t.Day = [-1 1 32]
```

```
t = 1x3 datetime array  
30-Dec-2013 01-Feb-2014 01-Apr-2014
```

月份和年份数字进行了调整，从而使所有值保持在每个日期分量的常规范围之内。在本例中，2014 年 1 月 -1 日转换为 2013 年 12 月 30 日。

## 另请参阅

`datetime` | `hms` | `week` | `ymd`

## 合并来自各自变量的日期和时间

本示例展示如何从文本文件读取日期和时间数据。然后再展示如何将存储在各自变量中的日期和时间信息合并到单个日期时间变量中。

创建名为 `schedule.txt` 且包含以下数据的空格分隔文本文件（要创建该文件，请使用任意文本编辑器，然后复制并粘贴）：

```
Date Name Time
10.03.2015 Joe 14:31
10.03.2015 Bob 15:33
11.03.2015 Bob 11:29
12.03.2015 Kim 12:09
12.03.2015 Joe 13:05
```

使用 `readtable` 函数读取该文件。使用 `%D` 转换设定符，以日期时间值的格式读取第一列和第三列数据。

```
T = readtable('schedule.txt','Format','%{dd.MM.uuuu}D %s %{HH:mm}D','Delimiter',' ')
```

```
T =
    Date      Name      Time
    _____
10.03.2015  'Joe'    14:31
10.03.2015  'Bob'    15:33
11.03.2015  'Bob'    11:29
12.03.2015  'Kim'    12:09
12.03.2015  'Joe'    13:05
```

`readtable` 将返回包含三个变量的表。

更改 `T.Date` 和 `T.Time` 变量的显示格式，以查看日期和时间信息。由于文件第一列（“Date”）中的数据不含时间信息，因此 `T.Date` 中生成的日期时间值的时间部分默认设为午夜零点。由于文件第三列（“Time”）中的数据没有关联的日期，因此 `T.Time` 中生成的日期时间值的日期部分默认设为当前日期。

```
T.Date.Format = 'dd.MM.uuuu HH:mm';
T.Time.Format = 'dd.MM.uuuu HH:mm';
T
```

```
T =
    Date      Name      Time
    _____
10.03.2015 00:00  'Joe'    12.12.2014 14:31
10.03.2015 00:00  'Bob'    12.12.2014 15:33
11.03.2015 00:00  'Bob'    12.12.2014 11:29
12.03.2015 00:00  'Kim'    12.12.2014 12:09
12.03.2015 00:00  'Joe'    12.12.2014 13:05
```

通过将 `T.Date` 与 `T.Time` 中的时间值相加，合并来自两个不同表变量的日期和时间信息。使用 `timeofday` 函数从 `T.Time` 中提取时间信息。

```
myDatetime = T.Date + timeofday(T.Time)
```

```
myDatetime =
10.03.2015 14:31
10.03.2015 15:33
11.03.2015 11:29
```

12.03.2015 12:09  
12.03.2015 13:05

**另请参阅**

`readtable | timeofday`

## 日期和时间算术运算

此示例演示了如何进行日期和时间值的加减运算，以此计算未来和过去的日期以及以精确单位或日历单位计量的流逝的持续时间。日期和时间数组可以进行加、减、乘、除运算，这些运算符的使用方法与其他 MATLAB® 数据类型相同。但是，日期和时间存在一些特定行为。

### 为日期时间数组加上或减去一定的持续时间

创建一个日期时间标量。默认情况下，日期时间数组并未与时区关联。

```
t1 = datetime('now')
```

```
t1 = datetime
30-Jul-2019 17:14:28
```

为其加上时数序列以求得未来的时间点。

```
t2 = t1 + hours(1:3)
```

```
t2 = 1x3 datetime array
30-Jul-2019 18:14:28 30-Jul-2019 19:14:28 30-Jul-2019 20:14:28
```

验证 `t2` 中每一对日期时间值的差是否为 1 小时。

```
dt = diff(t2)
```

```
dt = 1x2 duration array
01:00:00 01:00:00
```

`diff` 返回由小时数、分钟数和秒数构成的精确持续时间。

从日期时间值减去分钟数序列以查找过去的时间点。

```
t2 = t1 - minutes(20:10:40)
```

```
t2 = 1x3 datetime array
30-Jul-2019 16:54:28 30-Jul-2019 16:44:28 30-Jul-2019 16:34:28
```

将数值数组与 `datetime` 数组相加。MATLAB® 将数值数组中的每个值视作 24 小时一天的精确天数。

```
t2 = t1 + [1:3]
```

```
t2 = 1x3 datetime array
31-Jul-2019 17:14:28 01-Aug-2019 17:14:28 02-Aug-2019 17:14:28
```

### 包含时区的日期时间值加法

如要处理不同时区的日期时间值，或是要考虑夏令时更改，请使用关联了时区的日期时间数组。创建一个 `datetime` 标量，表示纽约时间 2014 年 3 月 8 日。

```
t1 = datetime(2014,3,8,0,0,0,'TimeZone','America/New_York')
```

```
t1 = datetime
08-Mar-2014 00:00:00
```

通过与天数序列（每一天固定 24 小时）相加查找未来时间点。

```
t2 = t1 + days(0:2)
```

```
t2 = 1x3 datetime array
08-Mar-2014 00:00:00 09-Mar-2014 00:00:00 10-Mar-2014 01:00:00
```

由于夏令时变化发生在 2014 年 3 月 9 日，因此 t2 中的第三个日期时间不是在午夜。

验证 t2 中每一对日期时间值的差是否为 24 小时。

```
dt = diff(t2)
```

```
dt = 1x2 duration array
24:00:00 24:00:00
```

通过将日期时间值与 **years**、**hours**、**minutes** 和 **seconds** 等函数的输出结果相加，您还可以分别为其加上以年份、小时、分钟和秒等其他单位表示的固定长度持续时间。

如要考虑夏令时更改，则您应当用日历持续时间代替持续时间。将日期时间值与日历持续时间相加减时，日历持续时间会考虑夏令时变化。

将多个日历天与 t1 相加。

```
t3 = t1 + caldays(0:2)
```

```
t3 = 1x3 datetime array
08-Mar-2014 00:00:00 09-Mar-2014 00:00:00 10-Mar-2014 00:00:00
```

查看 t3 中每一对日期时间值之间的差并非始终为 24 小时，这是因为夏令时变化发生在 3 月 9 日。

```
dt = diff(t3)
```

```
dt = 1x2 duration array
24:00:00 23:00:00
```

### 将日历持续时间与日期时间数组相加

将多个日历月与 2014 年 1 月 31 日相加。

```
t1 = datetime(2014,1,31)
```

```
t1 = datetime
31-Jan-2014
```

```
t2 = t1 + calmonths(1:4)
```

```
t2 = 1x4 datetime array
28-Feb-2014 31-Mar-2014 30-Apr-2014 31-May-2014
```

`t2` 中的每个日期时间都出现在每个月的最后一天。

使用 `caldiff` 函数计算 `t2` 中每一对日期时间值在日历天数上的差值。

```
dt = caldiff(t2,'days')
```

```
dt = 1x3 calendarDuration array  
31d 30d 31d
```

`dt` 中连续的日期时间值对之间相差的天数并非始终相同，这是因为不同的月份由不同的天数组成。

将多个日历年与 2014 年 1 月 31 日相加。

```
t2 = t1 + calyears(0:4)
```

```
t2 = 1x5 datetime array  
31-Jan-2014 31-Jan-2015 31-Jan-2016 31-Jan-2017 31-Jan-2018
```

使用 `caldiff` 函数计算 `t2` 中每一对日期时间值在日历天数上的差值。

```
dt = caldiff(t2,'days')
```

```
dt = 1x4 calendarDuration array  
365d 365d 366d 365d
```

`dt` 中连续的日期时间值对之间相差天数并非始终相同，这是因为 2016 年是闰年，有 366 天。

您可以使用 `calquarters`、`calweeks` 和 `caldays` 函数创建日历季度、日历周或日历天数组，然后将其与日期时间数组相加或相减。

日历持续时间相加时不适用加法交换律。当您将一个以上的 `calendarDuration` 数组与日期时间值相加时，MATLAB® 会按其在命令中出现的顺序相加。

在 2014 年 1 月 31 日基础上先加上 3 个日历月，再加上 30 个日历天。

```
t2 = datetime(2014,1,31) + calmonths(3) + caldays(30)
```

```
t2 = datetime  
30-May-2014
```

对同一日期，先加上 30 个日历天，然后再加上 3 个日历月。所得到的结果并不相同，这是因为当您将日历持续时间与日期时间值相加时，加上的天数取决于原始日期。

```
t2 = datetime(2014,1,31) + caldays(30) + calmonths(3)
```

```
t2 = datetime  
02-Jun-2014
```

### 日历持续时间算术运算

创建两个日历持续时间，然后对其求和。

```
d1 = calyears(1) + calmonths(2) + caldays(20)
```

```

d1 = calendarDuration
    1y 2mo 20d

d2 = calmonths(11) + caldays(23)
d2 = calendarDuration
    11mo 23d

d = d1 + d2
d = calendarDuration
    2y 1mo 43d

```

当您求两个或更多日历持续时间之和时，大于 12 的月数将转进到年数。但是，较大的天数不会转进到月数，因为不同的月份由不同的天数组成。

用因子 2 乘以 **d** 以增加其大小。日历持续时间的值必须是整数，因此只能用整数值与其相乘。

**2\*d**

```

ans = calendarDuration
    4y 2mo 86d

```

### 以精确单位计算流逝时间

将一个 **datetime** 数组与另一个相减，按精确的小时、分钟和秒数计算流逝的时间。

求出日期时间值序列与前一天起始时间之间的精确时长。

```

t2 = datetime('now') + caldays(1:3)
t2 = 1x3 datetime array
    31-Jul-2019 17:14:32  01-Aug-2019 17:14:32  02-Aug-2019 17:14:32

```

```
t1 = datetime('yesterday')
```

```
t1 = datetime
    29-Jul-2019
```

```
dt = t2 - t1
```

```
dt = 1x3 duration array
    65:14:32  89:14:32  113:14:32
```

**whos dt**

Name	Size	Bytes	Class	Attributes
dt	1x3	40	duration	

**dt** 以“时:分:秒”的格式存储持续时间。

通过更改 **dt** 的 **Format** 属性，以天为单位查看流逝的持续时间。

```
dt.Format = 'd'
```

```
dt = 1x3 duration array  
2.7184 days 3.7184 days 4.7184 days
```

用因子 1.2 乘以 dt 以扩大持续时间的值。由于持续时间具有精确长度，您可以将其与小数值相乘和相除。

```
dt2 = 1.2*dt
```

```
dt2 = 1x3 duration array  
3.2621 days 4.4621 days 5.6621 days
```

### 以日历单位计算流逝时间

使用 **between** 函数求两个日期之间流逝的日历年、月和天数。

```
t1 = datetime('today')
```

```
t1 = datetime  
30-Jul-2019
```

```
t2 = t1 + calmonths(0:2) + caldays(4)
```

```
t2 = 1x3 datetime array  
03-Aug-2019 03-Sep-2019 04-Oct-2019
```

```
dt = between(t1,t2)
```

```
dt = 1x3 calendarDuration array  
4d 1mo 4d 2mo 4d
```

### 另请参阅

[between](#) | [caldiff](#) | [diff](#)

# 比较日期和时间

此示例说明如何比较 `datetime` 和 `duration` 数组。可以使用 `>` 和 `<` 等关系运算符对两个 `datetime` 数组或两个 `duration` 数组中的值执行逐元素比较。

## 比较日期时间数组

比较两个 `datetime` 数组。这些数组的大小必须相同，或其中一项可以为标量。

```
A = datetime(2013,07,26) + calyears(0:2:6)
```

```
A = 1x4 datetime array
26-Jul-2013 26-Jul-2015 26-Jul-2017 26-Jul-2019
```

```
B = datetime(2014,06,01)
```

```
B = datetime
01-Jun-2014
```

`A < B`

```
ans = 1x4 logical array
```

```
1 0 0 0
```

当 `A` 中的日期时间发生在 `B` 中的日期时间之前时，`<` 运算符返回逻辑值 `1` (`true`)。

将 `datetime` 数组与表示日期的文本进行比较。

```
A >= 'September 26, 2014'
```

```
ans = 1x4 logical array
```

```
0 1 1 1
```

`datetime` 数组的比较考虑了每个数组的时区信息。

比较洛杉矶 2014 年 9 月 1 日下午 4:00 与纽约同一天的下午 5:00。

```
A = datetime(2014,09,01,16,0,0,'TimeZone','America/Los_Angeles',...
'Format','dd-MMM-yyyy HH:mm:ss Z')
```

```
A = datetime
01-Sep-2014 16:00:00 -0700
```

```
B = datetime(2014,09,01,17,0,0,'TimeZone','America/New_York',...
'Format','dd-MMM-yyyy HH:mm:ss Z')
```

```
B = datetime
01-Sep-2014 17:00:00 -0400
```

`A < B`

```
ans = logical  
0
```

洛杉矶下午 4:00 发生在纽约同一天的下午 5:00 点之后。

### 比较持续时间

比较两个 **duration** 数组。

```
A = duration([2,30,30;3,15,0])
```

```
A = 2x1 duration array  
02:30:30  
03:15:00
```

```
B = duration([2,40,0;2,50,0])
```

```
B = 2x1 duration array  
02:40:00  
02:50:00
```

```
A >= B
```

```
ans = 2x1 logical array
```

```
0  
1
```

比较持续时间数组与数值数组。数值数组中的元素被视为若干个固定长度（24 小时）的天。

```
A < [1; 1/24]
```

```
ans = 2x1 logical array
```

```
1  
0
```

### 确定日期和时间是否包含在区间内。

使用 **isbetween** 函数确定 **datetime** 数组中的值是否位于闭区间内。

定义区间的端点。

```
tlower = datetime(2014,08,01)
```

```
tlower = datetime  
01-Aug-2014
```

```
tupper = datetime(2014,09,01)
```

```
tupper = datetime  
01-Sep-2014
```

创建一个 `datetime` 数组并确定其值是否位于 `t1` 和 `t2` 所界定的区间内。

```
A = datetime(2014,08,21) + calweeks(0:2)
```

```
A = 1x3 datetime array  
21-Aug-2014 28-Aug-2014 04-Sep-2014
```

```
tf = isbetween(A,tlower,tupper)
```

```
tf = 1x3 logical array
```

```
1 1 0
```

## 另请参阅

`isbetween`

## 详细信息

- “使用关系运算符进行数组比较”（第 2-27 页）

## 绘制日期和持续时间图

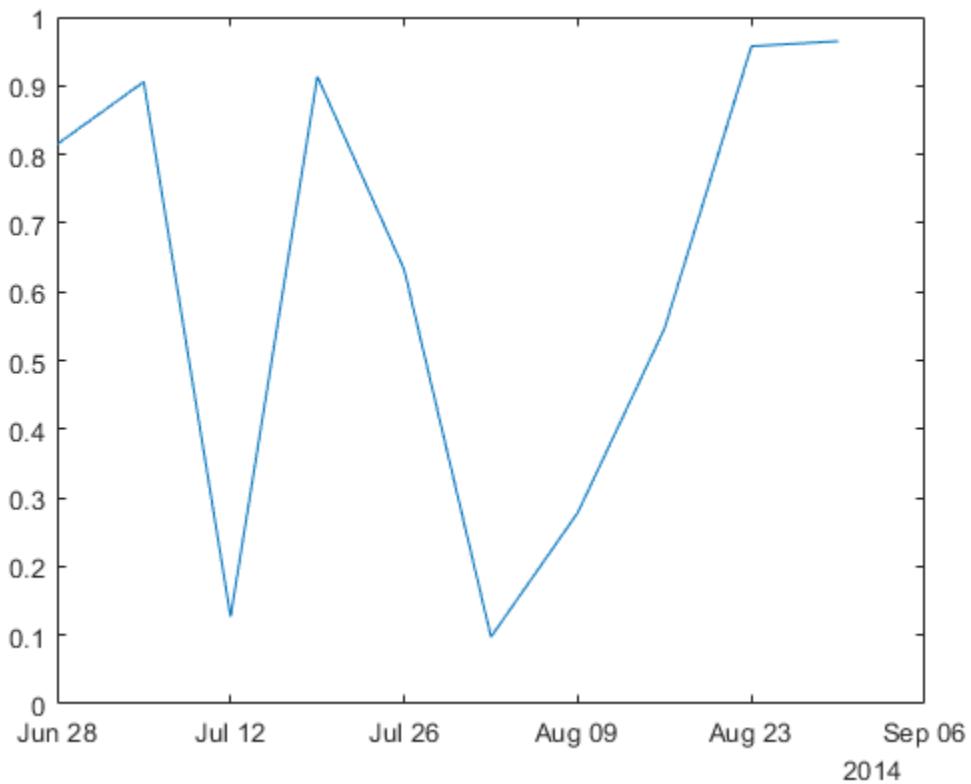
您可以使用各种图形函数创建日期时间和持续时间值的绘图。此外，也可以自定义坐标区，例如更改刻度标签的格式或更改坐标轴范围。

### 绘制日期线图

以 x 轴为日期时间值来创建线图。然后，更改刻度标签的格式以及 x 坐标轴范围。

创建 t 作为日期序列，创建 y 作为随机数据。使用 **plot** 函数绘制向量图。

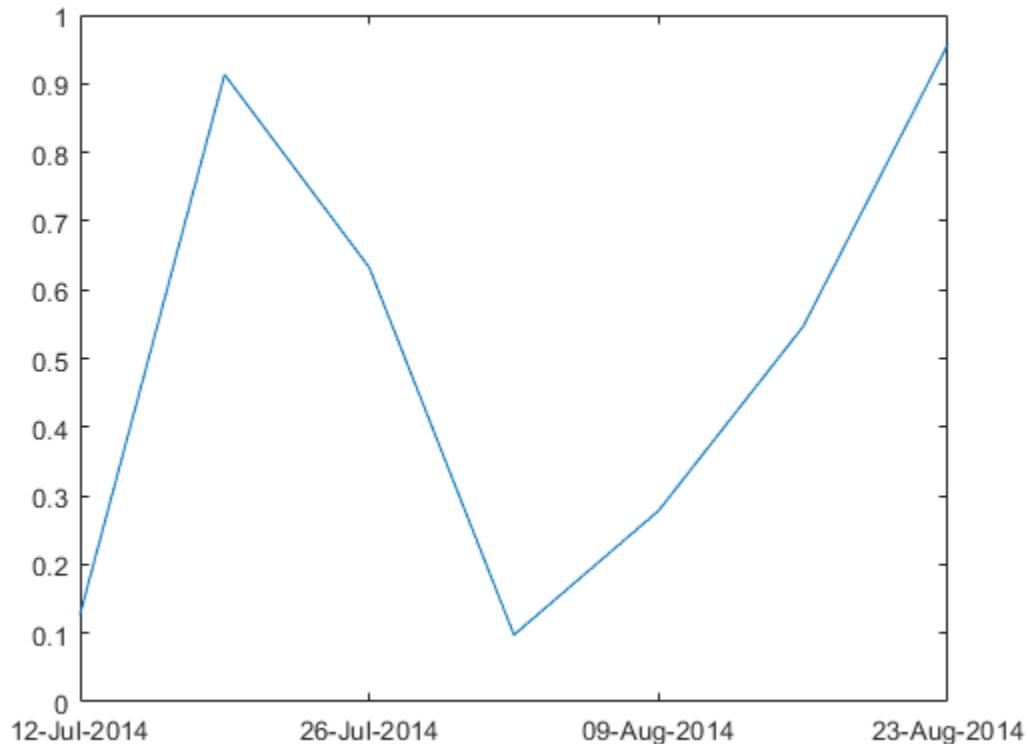
```
t = datetime(2014,6,28) + calweeks(0:9);
y = rand(1,10);
plot(t,y);
```



默认情况下，**plot** 会根据数据范围选择刻度线位置。当您放大和缩小绘图时，刻度标签会根据坐标轴范围自动调整。

更改 x 坐标轴范围。此外，更改沿 x 轴的刻度标签的格式。有关格式设置选项的列表，请参阅 **xtickformat** 函数。

```
xlim(datetime(2014,[7 8],[12 23]))
xtickformat('dd-MMM-yyyy')
```

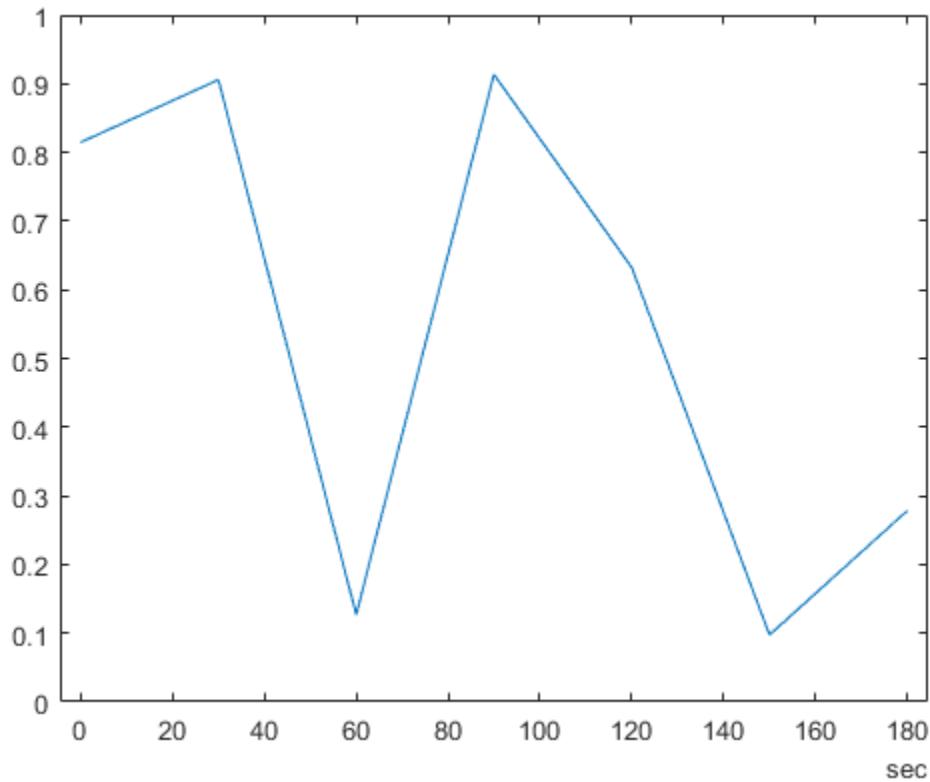


## 绘制持续时间线图

以 x 轴为持续时间值来创建线图。然后，更改刻度标签的格式以及 x 坐标轴范围。

创建 t 作为 0 到 3 分钟之间的七个线性分隔的持续时间值。创建 y 作为随机数据向量。绘制数据图。

```
t = 0:seconds(30):minutes(3);  
y = rand(1,7);  
plot(t,y);
```



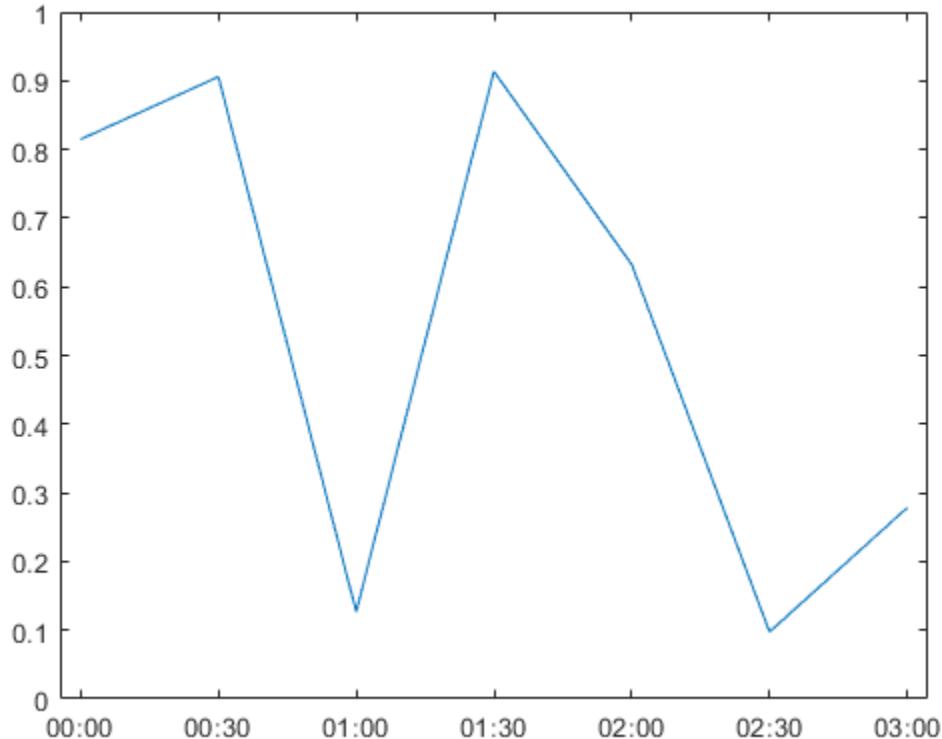
查看 x 坐标轴范围。由于持续时间刻度标签是用单一单位（分钟）表示的，因此这些范围按该单位进行存储。

```
xl = xlim
```

```
xl = 1x2 duration array  
-4.5 sec  184.5 sec
```

更改持续时间刻度标签的格式，以便以包含多个时间单位的数字计时器形式显示。有关格式设置选项的列表，请参阅 `xtickformat` 函数。

```
xtickformat('mm:ss')
```



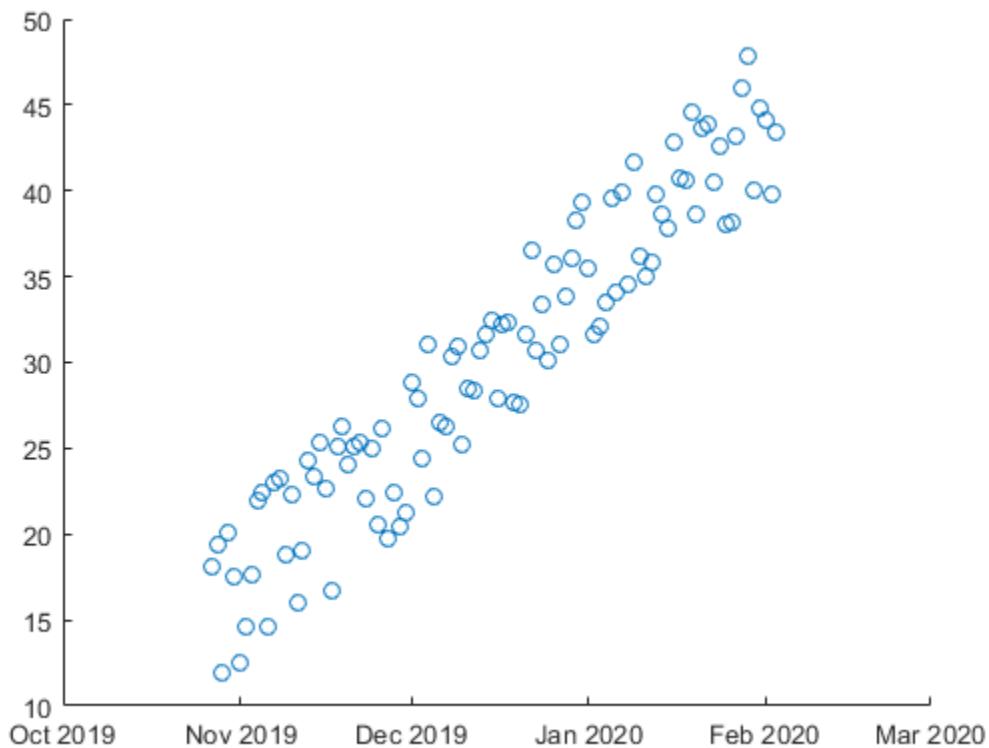
再次查看 x 坐标轴范围。由于持续时间刻度标签现在是用多个单位表示的，因此这些范围按 24 小时一天的单位进行存储。

```
xl = xlim
xl = 1x2 duration array
-00:04  03:04
```

## 用日期和持续时间绘制散点图

使用 `scatter` 或 `scatter3` 函数以日期时间或持续时间值为输入值创建散点图。例如，创建 x 轴为日期值的散点图。

```
t = datetime('today') + caldays(1:100);
y = linspace(10,40,100) + 10*rand(1,100);
scatter(t,y)
```



## 支持日期和持续时间的绘图

您可以使用日期时间或持续时间值创建其他类型的绘图。以下图形函数支持日期时间和持续时间值。

<b>bar</b>	<b>barh</b>
<b>plot</b>	<b>plot3</b>
<b>semilogx</b> (x 值必须为数值)	<b>semilogy</b> (y 值必须为数值)
<b>stem</b>	<b>stairs</b>
<b>scatter</b>	<b>scatter3</b>
<b>area</b>	<b>mesh</b>
<b>surf</b>	<b>surface</b>
<b>fill</b>	<b>fill3</b>
<b>line</b>	<b>text</b>
<b>histogram</b>	

## 另请参阅

[datetime](#) | [plot](#) | [xtickformat](#)

## 支持日期和时间数组的核心函数

MATLAB 中的许多函数在处理日期和时间数组时与处理其他数组并无不同。

下表列出了除处理常规数组外还可处理 `datetime`、`duration` 和 `calendarDuration` 数组的常见 MATLAB 函数。

<code>size</code>	<code>isequal</code>	<code>intersect</code>	<code>plus</code>	<code>plot</code>
<code>length</code>	<code>isequaln</code>	<code>ismember</code>	<code>minus</code>	<code>plot3</code>
<code>ndims</code>		<code>setdiff</code>	<code>uminus</code>	<code>scatter</code>
<code>numel</code>	<code>eq</code>	<code>setxor</code>	<code>times</code>	<code>scatter3</code>
<code>isrow</code>	<code>ne</code>	<code>unique</code>	<code>rdivide</code>	<code>bar</code>
<code>iscolumn</code>	<code>lt</code>	<code>union</code>	<code>ldivide</code>	<code>barh</code>
<code>cat</code>	<code>le</code>	<code>abs</code>	<code>mtimes</code>	<code>histogram</code>
<code>horzcat</code>	<code>ge</code>	<code>floor</code>	<code>mrdivide</code>	<code>stem</code>
<code>vertcat</code>	<code>gt</code>	<code>ceil</code>	<code>mldivide</code>	<code>stairs</code>
<code>permute</code>	<code>sort</code>	<code>round</code>	<code>diff</code>	<code>area</code>
<code>reshape</code>	<code>sortrows</code>	<code>min</code>	<code>sum</code>	<code>mesh</code>
<code>transpose</code>	<code>issorted</code>	<code>max</code>	<code>char</code>	<code>surf</code>
<code>ctranspose</code>		<code>mean</code>	<code>string</code>	<code>surface</code>
<code>linspace</code>		<code>median</code>	<code>cellstr</code>	
		<code>mode</code>		
				<code>semilogx</code>
				<code>semilogy</code>
				<code>fill</code>
				<code>fill3</code>
				<code>line</code>
				<code>text</code>

## 在日期时间数组、数值和文本之间转换

### 本节内容

- “概述”（第 7-40 页）
- “在日期时间和字符向量之间转换”（第 7-40 页）
- “在日期时间和字符串数组之间转换”（第 7-41 页）
- “在日期时间和日期向量之间转换”（第 7-42 页）
- “将日期序列值转换为日期时间”（第 7-43 页）
- “将日期时间数组转换为数值”（第 7-43 页）

### 概述

**datetime** 是表示时间点的最佳数据类型。**datetime** 值具有灵活的显示格式，最高可达纳秒级精度，并能将时区、夏令时和闰秒因素计算在内。但如果需要处理在 MATLAB R2014a 或更早版本中编写的代码，或者与使用此类版本的其他人共享代码，则可能需要处理存储为以下三种格式之一的日期和时间：

- 日期字符串（第 7-40 页） - 一个字符向量。

**Example:** Thursday, August 23, 2012 9:45:44.946 AM

- 日期向量（第 7-42 页） - 一个 1x6 数值向量，包含年、月、日、时、分和秒。

**Example:** [2012 8 23 9 45 44.946]

- 日期序列值（第 7-43 页） - 等于 ISO 日历（指定使用公历日历）中自 0000 年 1 月 0 日以来的天数的单个数值。日期序列值适合用作某些不接受 **datetime** 或 **duration** 数据类型的 MATLAB 函数的输入。

**Example:** 7.3510e+005

日期字符串、向量和数字可以作为值数组存储。可将多个日期字符串存储在字符向量元胞数组中，多个日期向量存储在  $m \times 6$  矩阵中，多个日期序列值存储在矩阵中。

您可以使用 **datetime** 函数将上述任意格式转换为 **datetime** 数组。如果您现有的 MATLAB 代码只能接受日期序列值或日期向量，请分别使用 **datanum** 或 **datevec** 函数将 **datetime** 数组转换为期望的日期格式。要将 **datetime** 数组转换为字符向量，请使用 **char** 或 **cellstr** 函数。

从 R2016b 开始，您也可以使用 **string** 函数将 **datetime** 数组转换为字符串数组。

### 在日期时间和字符向量之间转换

日期字符串可以是一个包含与特定日期和/或时间有关的字段的字符向量。可以通过多种方式以文本格式表示日期和时间。例如，以下所有内容均为表示 2010 年 8 月 23 日下午 04:35:42 的字符向量：

```
'23-Aug-2010 04:35:06 PM'
'Wednesday, August 23'
'08/23/10 16:35'
'Aug 23 16:35:42.946'
```

日期字符串包含分隔字段的字符，例如下面使用的连字符、空格和冒号：

```
d = '23-Aug-2010 16:35:42'
```

使用 **datetime** 函数将一个或多个日期字符串转换为 **datetime** 数组。为了获得最佳性能，请将输入日期字符串的格式指定为 **datetime** 的输入参数。

---

**注意** **datetime** 用于描述日期和时间格式的设定符与 **datestr**、**datevec** 和 **datenum** 函数接受的设定符有所不同。

有关日期和时间格式设定符的完整列表，请参阅 **datetime** 数据类型的 **Format** 属性。

---

```
t = datetime(d,'InputFormat','dd-MMM-yyyy HH:mm:ss')
```

```
t =
```

```
datetime
```

```
23-Aug-2010 16:35:42
```

虽然日期字符串 **d** 和 **datetime** 标量 **t** 类似，但它们并不相等。查看每个变量的大小和数据类型。

```
whos d t
```

Name	Size	Bytes	Class	Attributes
d	1x20	40	char	
t	1x1	17	datetime	

使用 **char** 或 **cellstr** 将 **datetime** 数组转换为字符向量。例如，将当前日期和时间转换为时间戳以附加到文件名。

```
t = datetime('now','Format','yyyy-MM-dd''T''HHmmss')
```

```
t =
```

```
datetime
```

```
2017-01-03T151105
```

```
S = char(t);
filename = [myTest_,S]
```

```
filename =
```

```
'myTest_2017-01-03T151105'
```

## 在日期时间和字符串数组之间转换

从 R2016b 开始，您可以使用 **string** 函数来创建字符串数组。如果字符串数组包含日期字符串，则您可以使用 **datetime** 函数将字符串数组转换为 **datetime** 数组。同样地，您可以使用 **string** 函数将 **datetime** 数组转换为字符串数组。

转换字符串数组。MATLAB 用双引号来显示字符串。为了获得最佳性能，请将输入日期字符串的格式指定为 **datetime** 的输入参数。

```
str = string({'24-Oct-2016 11:58:17';
'19-Nov-2016 09:36:29';
'12-Dec-2016 10:09:06'})
```

```

str =
3×1 string array
"24-Oct-2016 11:58:17"
"19-Nov-2016 09:36:29"
"12-Dec-2016 10:09:06"

t = datetime(str,'InputFormat','dd-MMM-yyyy HH:mm:ss')

t =
3×1 datetime array
24-Oct-2016 11:58:17
19-Nov-2016 09:36:29
12-Dec-2016 10:09:06

```

将 **datetime** 值转换为字符串。

```

t = datetime('25-Dec-2016 06:12:34');
str = string(t)

str =
"25-Dec-2016 06:12:34"

```

## 在日期时间和日期向量之间转换

日期向量是一个包含双精度数的 1x6 向量。日期向量的元素为整数值，但秒元素例外，它可以为小数。时间值采用 24 小时表示法表示。没有 AM 或 PM 设置。

日期向量按以下顺序排列：

**year month day hour minute second**

以下日期向量表示 2012 年 10 月 24 日上午 10:45:07：

**[2012 10 24 10 45 07]**

使用 **datetime** 函数将一个或多个日期向量转换为 **datetime** 数组：

```
t = datetime([2012 10 24 10 45 07])
```

t =

**datetime**

24-Oct-2012 10:45:07

使用 **year**、**month** 和 **day** 等函数（而不使用 **datevec**）来提取日期时间值的分量：

```
y = year(t)
```

y =

2012

或者，访问相应的属性，例如使用 **t.Year** 访问年值：

```
y = t.Year
```

```
y =
```

```
2012
```

## 将日期序列值转换为日期时间

日期序列值将日历日期表示为自固定的基准日期后经过的天数。在 MATLAB 中，日期序列值 1 表示 0000 年 1 月 1 日。

时间序列可以表示小数形式的天数（从午夜开始）；例如，6 p.m. 等于 0.75 序列天。因此，MATLAB 中的字符串 '31-Oct-2003, 6:00 PM' 是日期数字 731885.75。

使用 **datetime** 函数将一个或多个日期序列值转换为 **datetime** 数组。请指定要转换的日期数字的类型：

```
t = datetime(731885.75,'ConvertFrom','datenum')
```

```
t =
```

```
datetime
```

```
31-Oct-2003 18:00:00
```

## 将日期时间数组转换为数值

部分 MATLAB 函数接受数值数据类型，但不接受日期时间值作为输入参数。如要将这些函数应用于日期和时间数据，请先将日期时间值转换为有意义的数值。然后，再调用函数。例如，**log** 函数接受 **double** 输入，但不接受 **datetime** 输入。假设您有一个 **datetime** 数组，其中的日期跨越了一项研究或试验的整个过程。

```
t = datetime(2014,6,18) + calmonths(1:4)
```

```
t =
```

```
1×4 datetime array
```

```
18-Jul-2014 18-Aug-2014 18-Sep-2014 18-Oct-2014
```

减去原点值。例如，起点值可能是某个试验的开始日期。

```
dt = t - datetime(2014,7,1)
```

```
dt =
```

```
1×4 duration array
```

```
408:00:00 1152:00:00 1896:00:00 2616:00:00
```

**dt** 是一个 **duration** 数组。分别使用 **years**、**days**、**hours**、**minutes** 或 **seconds** 函数，将 **dt** 转换为以年、日、时、分或秒为单位的值的 **double** 数组。

```
x = hours(dt)
```

```
x =
```

```
408      1152      1896      2616
```

将 **double** 数组作为输入传递给 **log** 函数。

```
y = log(x)
```

```
y =
```

```
6.0113 7.0493 7.5475 7.8694
```

## 另请参阅

[cellstr](#) | [char](#) | [datenum](#) | [datetime](#) | [datevec](#) | [duration](#) | [string](#)

## 详细信息

- “表示 MATLAB 中的日期和时间” (第 7-2 页)
- “提取或分配日期时间数组的日期和时间分量” (第 7-21 页)
- 前公历日历

## 日期向量和字符串结转

如果某元素不在常规范围内，则 MATLAB 会同时调整该日期向量元素及其前面的元素。例如，如果分钟元素为 70，则 MATLAB 会将小时元素调整 1 并将分钟元素设置为 10。如果分钟元素为 -15，则 MATLAB 会将小时元素减少 1 并将分钟元素设置为 45。月份值是一个例外。MATLAB 将小于 1 的月份值设置为 1。

在以下示例中，月份元素的值为 22。MATLAB 将年份值增加到 2010 并将月份设置为十月 (October)。

```
datestr([2009 22 03 00 00 00])
```

```
ans =  
03-Oct-2010
```

结转值也适用于表示日期和时间的文本中的时间和日期值。例如，2010 年 10 月 3 日和 2010 年 9 月 33 日被解释为相同日期，与同一日期序列值对应。

```
datenum('03-Oct-2010')
```

```
ans =  
734414
```

```
datenum('33-Sep-2010')
```

```
ans =  
734414
```

以下示例接受输入月份（07 或 July），找出其前一个月的最后一天（6 月 30 日），并将该日期减去字段设定符中的天数（5 天），最后生成返回日期 2010 年 6 月 25 日。

```
datestr([2010 07 -05 00 00 00])
```

```
ans =  
25-Jun-2010
```

## 转换日期向量返回意外输出

由于日期向量是一个  $1 \times 6$  的数值向量，因此 **datestr** 可能会将您的输入日期向量解释为日期序列值向量（反之亦然），并返回异常输出。

以一个包含年份 3000 的日期向量为例。此年份超出了 **datestr** 解释为日期向量元素的年份的范围。因此，输入被解释为一个  $1 \times 6$  的日期序列值向量：

```
datestr([3000 11 05 10 32 56])
```

```
ans =
```

```
18-Mar-0008
11-Jan-0000
05-Jan-0000
10-Jan-0000
01-Feb-0000
25-Feb-0000
```

此处，**datestr** 将 3000 解释为一个日期序列值，并将其转换为日期字符串 '18-Mar-0008'。另外，**datestr** 还将接下来的五个元素均转换为日期字符串。

在将此类日期向量转换为字符向量时，请首先使用 **datenum** 将其转换为日期序列值。然后，再使用 **datestr** 将日期数字转换为字符向量：

```
dn = datenum([3000 11 05 10 32 56]);
ds = datestr(dn)
```

```
ds =
```

```
05-Nov-3000 10:32:56
```

在将日期转换为字符向量时，**datestr** 会使用启发式规则将输入解释为日期向量或日期序列值。以一个  $m \times 6$  矩阵为例。**datestr** 在以下情况下将该矩阵解释为  $m$  个日期向量：

- 前五列包含整数。
- 每一行之和的绝对值位于 1500–2500 范围内。

对于任何一行，如果上述任意一个条件为 `false`，则 **datestr** 会将  $m \times 6$  矩阵解释为  $m \times 6$  日期序列值。

通常，1700–2300 范围内带有年份的日期将被解释为日期向量。但是，**datestr** 可能会将月份值、日期值、小时值、分钟值或秒值超出正常范围的行解释为日期序列值。例如，**datestr** 能够正确解释年份 2014 的以下日期向量：

```
datestr([2014 06 21 10 51 00])
```

```
ans =
```

```
21-Jun-2014 10:51:00
```

但是，给定一个超出通常范围 (1–31) 的日期值，**datestr** 则会为该向量中的每个元素返回一个日期值：

```
datestr([2014 06 2110 10 51 00])
```

```
ans =
```

```
06-Jul-0005  
06-Jan-0000  
10-Oct-0005  
10-Jan-0000  
20-Feb-0000  
00-Jan-0000
```

如果您具有一个 `datestr` 可能错误地解释为日期序列值的日期向量矩阵，请首先使用 `datenum` 将该矩阵转换为日期序列值。然后，再使用 `datestr` 转换日期值。

如果您具有一个 `datestr` 可能解释为日期向量的日期序列值矩阵，请首先将该矩阵转换为列向量。然后，再使用 `datestr` 转换列向量。



# 分类数组

---

- “创建分类数组” (第 8-2 页)
- “将表变量中的文本转换为分类数组” (第 8-6 页)
- “对分类数据绘图” (第 8-10 页)
- “比较分类数组元素” (第 8-16 页)
- “合并分类数组” (第 8-19 页)
- “使用乘法合并分类数组” (第 8-22 页)
- “使用分类数组访问数据” (第 8-24 页)
- “使用受保护的分类数组” (第 8-30 页)
- “使用分类数组的好处” (第 8-34 页)
- “有序分类数组” (第 8-36 页)
- “支持分类数组的核心函数” (第 8-39 页)

## 创建分类数组

此示例说明如何创建分类数组。**categorical** 是一个数据类型，用来存储值来自一组有限离散类别的数据。这些分类可以采用自然排序，但并不要求一定如此。分类数组可用来有效地存储并方便地处理数据，同时还为数值赋予有意义的名称。分类数组通常用在表中以定义由行构成的组。

默认情况下，分类数组包含的是未采用数学排序的类别。例如，离散的宠物类别集合 `{'dog' 'cat' 'bird'}` 未采用有意义的数学排序，因此 MATLAB® 使用字母排序 `{'bird' 'cat' 'dog'}`。有序分类数组包含的类别具有有意义的数学排序。例如，离散的大小类别集合 `{'small', 'medium', 'large'}` 采用数学排序 `small < medium < large`。

当您基于字符向量元胞数组或字符串数组创建分类数组时，前导空格和尾随空格将被删除。例如，如果您将文本 `{' cat' 'dog' }` 指定为类别，则在将它们转换为类别时，它们将变成 `{'cat' 'dog'}`。

### 基于字符向量元胞数组创建分类数组

您可以使用 **categorical** 函数基于数值数组、逻辑数组、字符串数组、字符向量元胞数组或现有的分类数组来创建分类数组。

创建一个包含新英格兰地区的各州名的  $1 \times 11$  字符向量元胞数组。

```
state = {'MA','ME','CT','VT','ME','NH','VT','MA','NH','CT','RI'};
```

将元胞数组 `state` 转换为未采用数学排序的分类数组。

```
state = categorical(state)
```

```
state = 1x11 categorical array
 Columns 1 through 9
```

```
MA    ME    CT    VT    ME    NH    VT    MA    NH
```

```
Columns 10 through 11
```

```
CT    RI
```

```
class(state)
```

```
ans =
'categorical'
```

列出变量 `state` 中的离散类别。

```
categories(state)
```

```
ans = 6x1 cell array
{'CT'}
{'MA'}
{'ME'}
{'NH'}
{'RI'}
{'VT'}
```

这些类别按字母顺序列出。

## 基于字符向量元胞数组创建有序分类数组

创建一个大小包含八个对象的  $1 \times 8$  字符向量元胞数组。

```
AllSizes = {'medium','large','small','small','medium',...
    'large','medium','small'};
```

元胞数组 AllSizes 具有三个相异值：'large'、'medium' 和 'small'。使用字符向量元胞数组时，没有方便的方式来指示  $\text{small} < \text{medium} < \text{large}$ 。

将元胞数组 AllSizes 转换为有序分类数组。使用 valueset 指定用于定义类别的值 small、medium 和 large。对于有序分类数组，所指定的第一个类别是最小的，最后一个类别是最大的。

```
valueset = {'small','medium','large'};
sizeOrd = categorical(AllSizes,valueset,'Ordinal',true)
```

```
sizeOrd = 1x8 categorical array
Columns 1 through 6
```

```
    medium    large    small    small    medium    large
```

```
Columns 7 through 8
```

```
    medium    small
```

```
class(sizeOrd)
```

```
ans =
'categorical'
```

分类数组 sizeOrd 中值的顺序保持不变。

列出分类变量 sizeOrd 中的离散类别。

```
categories(sizeOrd)
```

```
ans = 3x1 cell array
{'small'}
{'medium'}
{'large'}
```

这些类别按指定的顺序列出以匹配数学排序  $\text{small} < \text{medium} < \text{large}$ 。

## 基于分 bin 数值数据创建有序分类数组

创建由 0 到 50 之间的 100 个随机数构成的向量。

```
x = rand(100,1)*50;
```

使用 **discretize** 函数，通过对 x 的值进行 bin 划分，创建一个分类数组。将 0 到 15 之间的所有值归入第一个 bin，15 到 35 之间的所有值归入第二个 bin，35 到 50 之间的所有值归入第三个 bin。每个 bin 包括左端点，但不包括右端点。

```
catnames = {'small','medium','large'};
binnedData = discretize(x,[0 15 35 50], 'categorical',catnames);
```

`binnedData` 是一个包含三个类别的  $100 \times 1$  有序分类数组，其中 `small < medium < large`。

使用 `summary` 函数输出每个类别中的元素数量。

```
summary(binnedData)
```

small	30
medium	35
large	35

### 基于字符串数组创建分类数组

从 R2016b 开始，您可以使用 `string` 函数创建字符串数组，并将字符串数组转换为分类数组。

创建一个包含行星名称的字符串数组。

```
str = string({'Earth','Jupiter','Neptune','Jupiter','Mars','Earth'})  
  
str = 1x6 string array  
"Earth" "Jupiter" "Neptune" "Jupiter" "Mars" "Earth"
```

将 `str` 转换为分类数组。

```
planets = categorical(str)  
  
planets = 1x6 categorical array  
Earth Jupiter Neptune Jupiter Mars Earth
```

将缺失的元素添加到 `str`，并将其转换为分类数组。在 `str` 包含缺失值的位置，`planets` 包含未定义值。

```
str(8) = 'Mars'  
  
str = 1x8 string array  
Columns 1 through 6  
  
"Earth" "Jupiter" "Neptune" "Jupiter" "Mars" "Earth"  
  
Columns 7 through 8  
  
<missing> "Mars"
```

```
planets = categorical(str)  
  
planets = 1x8 categorical array  
Columns 1 through 6  
  
Earth Jupiter Neptune Jupiter Mars Earth  
  
Columns 7 through 8  
  
<undefined> Mars
```

### 另请参阅

`categorical` | `categories` | `discretize` | `summary`

## 相关示例

- “将表变量中的文本转换为分类数组” (第 8-6 页)
- “使用分类数组访问数据” (第 8-24 页)
- “比较分类数组元素” (第 8-16 页)

## 详细信息

- “使用分类数组的好处” (第 8-34 页)
- “有序分类数组” (第 8-36 页)

## 将表变量中的文本转换为分类数组

以下示例演示了如何将表中的变量从字符向量元胞数组转换为分类数组。

### 加载样本数据并创建表

加载从 100 位患者收集的样本数据。

```
load patients
```

```
whos
```

Name	Size	Bytes	Class	Attributes
Age	100x1	800	double	
Diastolic	100x1	800	double	
Gender	100x1	12212	cell	
Height	100x1	800	double	
LastName	100x1	12416	cell	
Location	100x1	15008	cell	
SelfAssessedHealthStatus	100x1	12340	cell	
Smoker	100x1	100	logical	
Systolic	100x1	800	double	
Weight	100x1	800	double	

将患者的 Age、Gender、Height、Weight、SelfAssessedHealthStatus 和 Location 数据存储在表中。将变量 LastName 中的唯一标识符用作行名称。

```
T = table(Age,Gender,Height,Weight, ...
          SelfAssessedHealthStatus,Location, ...
          'RowNames',LastName);
```

### 将表变量从字符向量元胞数组转换为分类数组

字符向量元胞数组 Gender 和 Location 包含离散的唯一值集合。

将 Gender 和 Location 转换为分类数组。

```
T.Gender = categorical(T.Gender);
T.Location = categorical(T.Location);
```

变量 SelfAssessedHealthStatus 包含四个唯一值：Excellent、Fair、Good 和 Poor。

将 SelfAssessedHealthStatus 转换为一个有序分类数组，这样这些类别采用数学排序 Poor < Fair < Good < Excellent。

```
T.SelfAssessedHealthStatus = categorical(T.SelfAssessedHealthStatus, ...
                                         {'Poor','Fair','Good','Excellent'},'Ordinal',true);
```

### 输出摘要

使用 summary 汇总表来查看每个变量的数据类型、说明、单位和其他描述性统计量。

```
format compact
```

```
summary(T)
```

Variables:

Age: 100x1 double

```

Values:
    Min      25
    Median   39
    Max      50
Gender: 100x1 categorical
Values:
    Female    53
    Male     47
Height: 100x1 double
Values:
    Min      60
    Median   67
    Max      72
Weight: 100x1 double
Values:
    Min     111
    Median 142.5
    Max     202
SelfAssessedHealthStatus: 100x1 ordinal categorical
Values:
    Poor        11
    Fair        15
    Good       40
    Excellent  34
Location: 100x1 categorical
Values:
    County General Hospital    39
    St. Mary s Medical Center   24
    VA Hospital                 37

```

表变量 **Gender**、**SelfAssessedHealthStatus** 和 **Location** 为分类数组。摘要包含每个类别中的元素数的统计。例如，该摘要指示 100 位患者中的 53 位为女性，47 位为男性。

### 基于类别选择数据

创建一个子表 **T1**，其包含在 **County General Hospital** 就医的所有女性患者的年龄、身高和体重。您可以轻松地基于分类数组 **Gender** 和 **Location** 中的值创建一个逻辑向量。

```
rows = T.Location=='County General Hospital' & T.Gender=='Female';
```

**rows** 是一个  $100 \times 1$  的逻辑向量，对于表中满足指定条件（性别为女性，位置为 **County General Hospital**）的行，该逻辑向量会在其对应位置包含逻辑值 **true** (1)。

定义变量的子集。

```
vars = {'Age','Height','Weight'};
```

使用圆括号创建子表 **T1**。

```
T1 = T(rows,vars)
```

```
T1=19×3 table
    Age    Height    Weight
    _____
Brown    49        64        119
Taylor   31        66        132
Anderson 45        68        128
Lee      44        66        146
```

Walker	28	65	123
Young	25	63	114
Campbell	37	65	135
Evans	39	62	121
Morris	43	64	135
Rivera	29	63	130
Richardson	30	67	141
Cox	28	66	111
Torres	45	70	137
Peterson	32	60	136
Ramirez	48	64	137
Bennett	35	64	131
:			

A 是一个  $19 \times 3$  的表。

由于有序分类数组对其类别采用数学排序，因此可以使用关系运算（例如大于和小于）对它们执行按元素比较。

创建一个子表 T2，其包含健康状况评估为较差或一般的所有患者的性别、年龄、身高和体重。

首先，定义要包括在表 T2 中的行子集。

```
rows = T.SelfAssessedHealthStatus <= 'Fair';
```

然后，定义要包括在表 T2 中的变量子集。

```
vars = {'Gender','Age','Height','Weight'};
```

使用圆括号创建子表 T2。

```
T2 = T(rows,vars)
```

T2=26×4 table

	Gender	Age	Height	Weight
Johnson	Male	43	69	163
Jones	Female	40	67	133
Thomas	Female	42	66	137
Jackson	Male	25	71	174
Garcia	Female	27	69	131
Rodriguez	Female	39	64	117
Lewis	Female	41	62	137
Lee	Female	44	66	146
Hall	Male	25	70	189
Hernandez	Male	36	68	166
Lopez	Female	40	66	137
Gonzalez	Female	35	66	118
Mitchell	Male	39	71	164
Campbell	Female	37	65	135
Parker	Male	30	68	182
Stewart	Male	49	68	170
:				

T2 是一个  $26 \times 4$  的表。

## 另请参阅

### 相关示例

- “创建和使用表” (第 9-2 页)
- “创建分类数组” (第 8-2 页)
- “访问表中的数据” (第 9-33 页)
- “使用分类数组访问数据” (第 8-24 页)

### 详细信息

- “使用分类数组的好处” (第 8-34 页)
- “有序分类数组” (第 8-36 页)

## 对分类数据绘图

此示例演示了如何对分类数组中的数据绘图。

### 加载样本数据

加载从 100 位患者收集的样本数据。

```
load patients
```

```
whos
```

Name	Size	Bytes	Class	Attributes
Age	100x1	800	double	
Diastolic	100x1	800	double	
Gender	100x1	12212	cell	
Height	100x1	800	double	
LastName	100x1	12416	cell	
Location	100x1	15008	cell	
SelfAssessedHealthStatus	100x1	12340	cell	
Smoker	100x1	100	logical	
Systolic	100x1	800	double	
Weight	100x1	800	double	

### 基于字符向量元胞数组创建分类数组

工作区变量 **Location** 是一个字符向量元胞数组，它包含患者就医的三个唯一医疗机构。

为了更方便地访问和比较数据，请将 **Location** 转换为一个分类数组。

```
Location = categorical(Location);
```

汇总分类数组。

```
summary(Location)
```

County General Hospital	39
St. Mary's Medical Center	24
VA Hospital	37

39 位患者在 County General Hospital 就医，24 位患者在 St. Mary's Medical Center 就医，37 位患者在 VA Hospital 就医。

工作区变量 **SelfAssessedHealthStatus** 包含四个唯一值 **Excellent**、**Fair**、**Good** 和 **Poor**。

将 **SelfAssessedHealthStatus** 转换为一个有序分类数组，这样这些类别采用数学排序 **Poor < Fair < Good < Excellent**。

```
SelfAssessedHealthStatus = categorical(SelfAssessedHealthStatus,...  
{'Poor' 'Fair' 'Good' 'Excellent'},'Ordinal',true);
```

汇总分类数组 **SelfAssessedHealthStatus**。

```
summary(SelfAssessedHealthStatus)
```

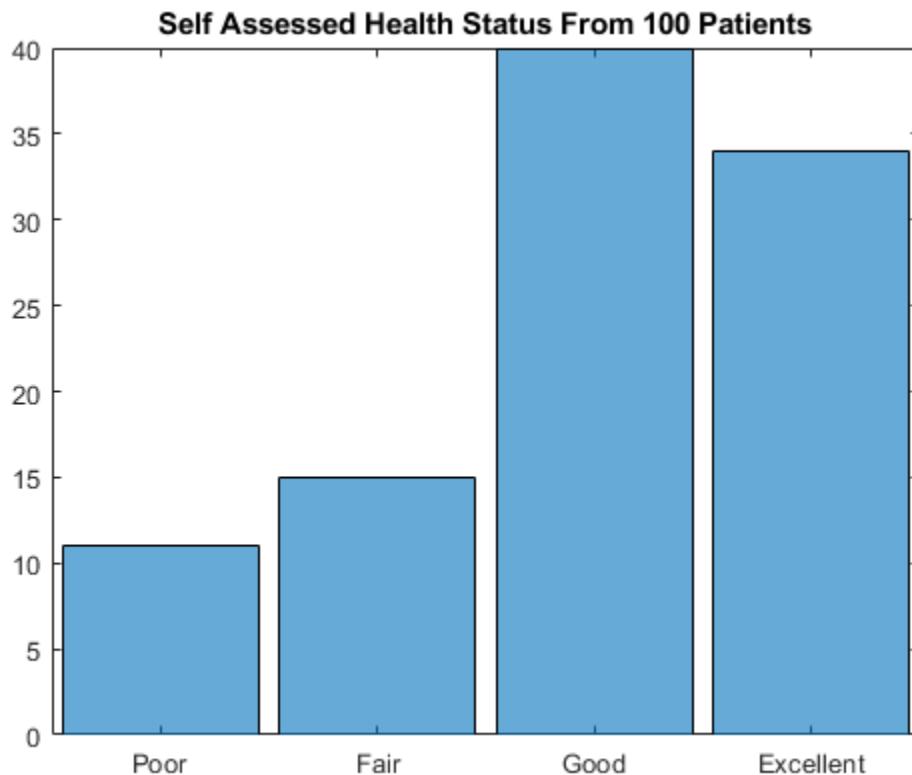
Poor	11
Fair	15

Good	40
Excellent	34

### 绘制直方图

直接基于分类数组创建一个直方条形图。

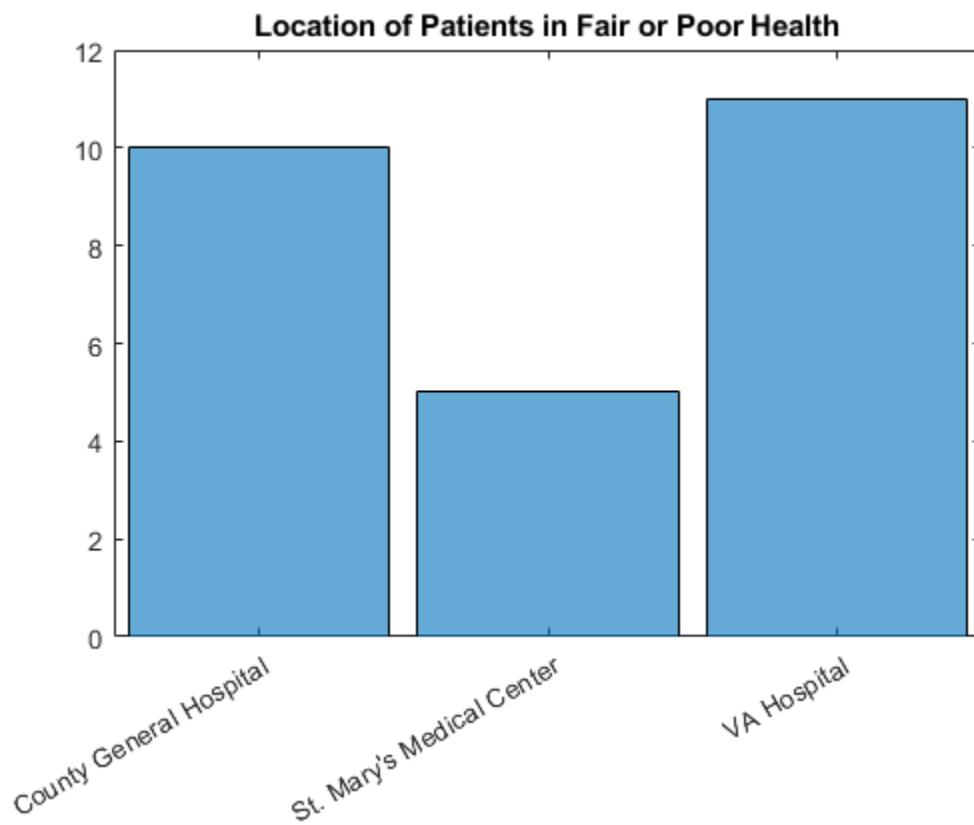
```
figure
histogram(SelfAssessedHealthStatus)
title('Self Assessed Health Status From 100 Patients')
```



函数 `histogram` 接受分类数组 `SelfAssessedHealthStatus`, 并对四个类别中的每一个类别绘制类别计数。

仅为健康状况评估为 `Fair` 或 `Poor` 的患者绘制医院位置直方图。

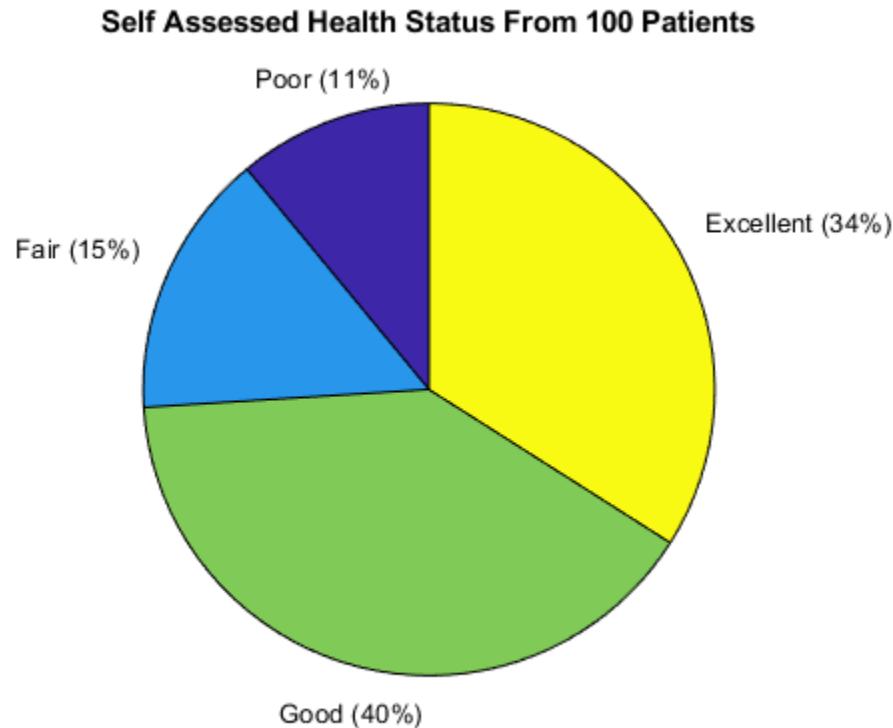
```
figure
histogram(Location(SelfAssessedHealthStatus<='Fair'))
title('Location of Patients in Fair or Poor Health')
```



### 创建饼图

从分类数组直接创建饼图。

```
figure  
pie(SelfAssessedHealthStatus);  
title('Self Assessed Health Status From 100 Patients')
```

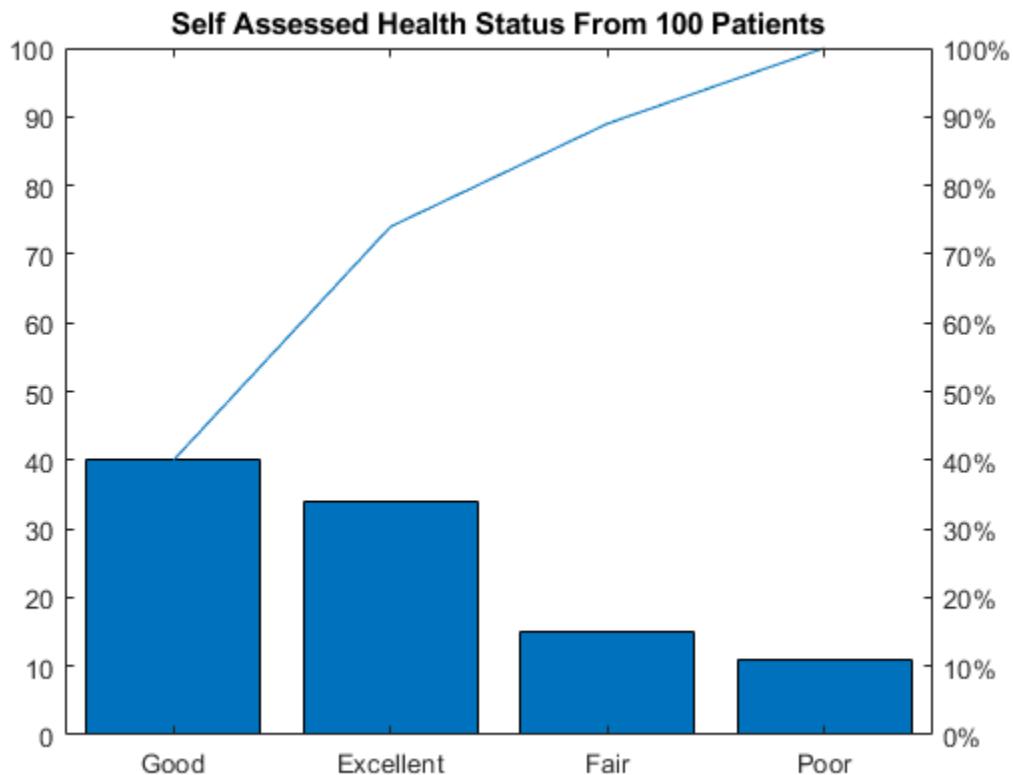


函数 `pie` 接受分类数组 `SelfAssessedHealthStatus`, 并绘制了一个包含四个类别的饼图。

### 创建帕累托图

根据 `SelfAssessedHealthStatus` 的四个类别各自的类别计数创建帕累托图。

```
figure  
A = countcats(SelfAssessedHealthStatus);  
C = categories(SelfAssessedHealthStatus);  
pareto(A,C);  
title('Self Assessed Health Status From 100 Patients!')
```



`pareto` 的第一个输入参数必须是向量。如果分类数组为矩阵或多维数组，则在调用 `countcats` 和 `pareto` 之前将其重构为向量。

### 创建散点图

将字符向量元胞数组转换为分类数组。

```
Gender = categorical(Gender);
```

汇总分类数组 `Gender`。

```
summary(Gender)
```

```
Female    53
Male     47
```

`Gender` 是一个  $100 \times 1$  的分类数组，包含两个类别 `Female` 和 `Male`。

使用分类数组 `Gender` 分别访问每种性别的 `Weight` 和 `Height`。

```
X1 = Weight(Gender=='Female');
Y1 = Height(Gender=='Female');
```

```
X2 = Weight(Gender=='Male');
Y2 = Height(Gender=='Male');
```

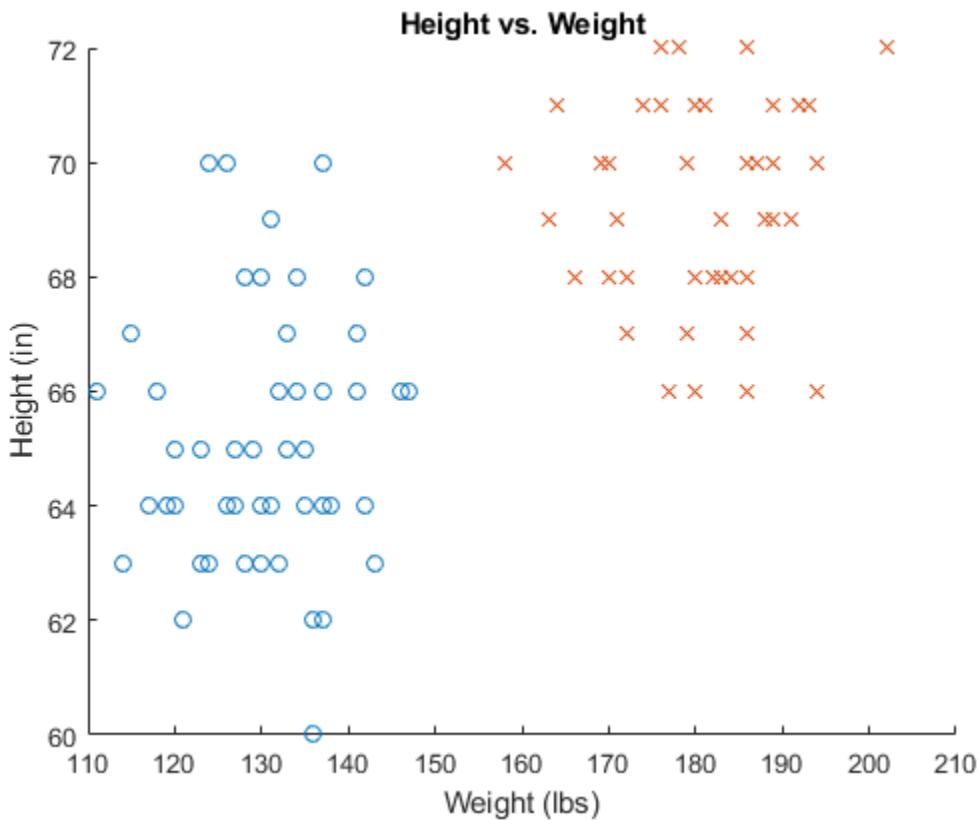
`X1` 和 `Y1` 是  $53 \times 1$  的数值数组，包含女性患者的数据。

X2 和 Y2 是  $47 \times 1$  的数值数组，包含男性患者的数据。

创建一个身高与体重的散点图。使用一个圈指示女性患者的数据，使用一个叉指示男性患者的数据。

```
figure
h1 = scatter(X1,Y1,'o');
hold on
h2 = scatter(X2,Y2,'x');

title('Height vs. Weight')
xlabel('Weight (lbs)')
ylabel('Height (in)')
```



## 另请参阅

[bar](#) | [categorical](#) | [countcats](#) | [histogram](#) | [pie](#) | [rose](#) | [scatter](#) | [summary](#)

## 相关示例

- “使用分类数组访问数据”（第 8-24 页）

## 比较分类数组元素

此示例演示了如何对分类数组执行关系运算。

### 基于字符向量元胞数组创建分类数组

创建一个  $2 \times 4$  的字符向量元胞数组。

```
C = {'blue' 'red' 'green' 'blue';...
      'blue' 'green' 'green' 'blue'};

colors = categorical(C)

colors = 2x4 categorical array
    blue    red    green    blue
    blue    green   green    blue
```

`colors` 是一个  $2 \times 4$  的分类数组。

列出分类数组的类别。

```
categories(colors)
```

```
ans = 3x1 cell array
{'blue'}
{'green'}
{'red'}
```

### 确定元素是否相等

使用关系运算符 `eq` (`==`) 比较 `colors` 的第一行和第二行。

```
colors(1,:) == colors(2,:)
```

```
ans = 1x4 logical array
```

```
1 0 1 1
```

两行数据间仅第二列的值有所不同。

### 比较整个数组与字符向量

将整个分类数组 `colors` 与字符向量 `'blue'` 进行比较以查找所有 `blue` 值的位置。

```
colors == 'blue'
```

```
ans = 2x4 logical array
```

```
1 0 0 1
1 0 0 1
```

`colors` 中有四个蓝色条目，数组的每个角一个。

## 转换为有序分类数组

对 `colors` 中的类别添加数学排序。指定表示色谱排序的类别顺序 `red < green < blue`。

```
colors = categorical(colors,{'red','green','blue'},'Ordinal',true)
```

```
colors = 2x4 categorical array
    blue    red    green    blue
    blue    green   green    blue
```

分类数组中的元素仍然相同。

列出 `colors` 中的离散类别。

```
categories(colors)
```

```
ans = 3x1 cell array
    {'red'}
    {'green'}
    {'blue'}
```

## 按顺序比较元素

确定 `colors` 的第一列中的元素是否大于第二列中的元素。

```
colors(:,1) > colors(:,2)
```

```
ans = 2x1 logical array
```

```
1
1
```

第一列中的两个值 `blue` 都大于第二列中的对应值 `red` 和 `green`。

查找 `colors` 小于 '`blue`' 的所有元素。

```
colors < 'blue'
```

```
ans = 2x4 logical array
```

```
0 1 1 0
0 1 1 0
```

函数 `lt (<)` 指示所有 `green` 和 `red` 值为 1 的位置。

## 另请参阅

[categorical](#) | [categories](#)

## 相关示例

- “使用分类数组访问数据”（第 8-24 页）

### 详细信息

- “关系运算”
- “使用分类数组的好处”（第 8-34 页）
- “有序分类数组”（第 8-36 页）

# 合并分类数组

此示例演示了如何合并两个分类数组。

## 创建分类数组

创建分类数组 A，其中包含教室 A 中的 25 个学生的首选午餐饮料。

```
A = gallery('integerdata',3,[25,1],1);
A = categorical(A,1:3,['milk' 'water' 'juice']);
```

A 是一个  $25 \times 1$  的分类数组，包含三个不同的类别：milk、water 和 juice。

汇总分类数组 A。

```
summary(A)
```

milk	8
water	8
juice	9

教室 A 中的八个学生喜欢牛奶，八个学生喜欢水，九个学生喜欢果汁。

创建另外一个分类数组 B，其中包含教室 B 中的 28 个学生的首选饮料。

```
B = gallery('integerdata',3,[28,1],3);
B = categorical(B,1:3,['milk' 'water' 'juice']);
```

B 是一个  $28 \times 1$  的分类数组，包含与 A 相同的类别。

汇总分类数组 B。

```
summary(B)
```

milk	12
water	10
juice	6

教室 B 中的十二个学生喜欢牛奶，十个学生喜欢水，六个学生喜欢果汁。

## 串联分类数组

将教室 A 和 B 中的数据串联为一个单独的分类数组 Group1。

```
Group1 = [A;B];
```

汇总分类数组 Group1

```
summary(Group1)
```

milk	20
water	18
juice	15

Group1 是一个包含以下三个类别的  $53 \times 1$  分类数组：milk、water 和 juice。

## 创建具有不同类别的分类数组

创建一个分类数组 Group2，其中包含为其提供了额外的苏打饮料选择的 50 个学生的数据。

```
Group2 = gallery('integerdata',4,[50,1],2);
Group2 = categorical(Group2,1:4,{juice' 'milk' 'soda' 'water'});
```

汇总分类数组 Group2。

```
summary(Group2)
```

juice	18
milk	10
soda	13
water	9

Group2 是一个包含以下四个类别的  $50 \times 1$  分类数组： juice、 milk、 soda 和 water。

### 串联具有不同类别的数组

串联 Group1 和 Group2 中的数据。

```
students = [Group1;Group2];
```

汇总生成的分类数组 students。

```
summary(students)
```

milk	30
water	27
juice	33
soda	13

串联操作会将第二个输入所独有的类别 soda 追加到第一个输入的类别列表的末尾： milk、 water、 juice、 soda。

使用 reordercats 更改分类数组 students 中的类别顺序。

```
students = reordercats(students,{juice','milk','water','soda'});
```

```
categories(students)
```

```
ans = 4x1 cell array
{'juice'}
{'milk'}
{'water'}
{'soda'}
```

### 分类数组的并集

使用函数 union 可查找 Group1 和 Group2 中的唯一响应。

```
C = union(Group1,Group2)
```

```
C = 4x1 categorical array
milk
water
juice
soda
```

union 返回 Group1 和 Group2 的合并值并且没有重复项。在本例中， C 等效于串联的类别 students。

此示例中的所有分类数组都不是有序的。要合并有序分类数组，它们必须具有相同的类别集合，包括其顺序。

## 另请参阅

`cat` | `categorical` | `categories` | `horzcat` | `summary` | `union` | `vertcat`

## 相关示例

- “创建分类数组” (第 8-2 页)
- “使用乘法合并分类数组” (第 8-22 页)
- “将表变量中的文本转换为分类数组” (第 8-6 页)
- “使用分类数组访问数据” (第 8-24 页)

## 详细信息

- “有序分类数组” (第 8-36 页)

## 使用乘法合并分类数组

此示例说明如何使用 `times` 函数来合并分类数组，包括有序分类数组和包含未定义元素的数组。当对两个分类数组调用 `times` 时，将会输出一个包含新类别的分类数组。新类别集是从输入数组的类别创建的所有有序对集合，即笛卡尔积。`times` 使用输入数组对应元素的有序对构成输出数组的每个元素。输出数组的大小与输入数组相同。

### 合并两个分类数组

使用 `times` 合并两个分类数组。各输入数组必须具有相同的元素数量，但可以包含不同的类别数量。

```
A = categorical({'blue','red','green'});
B = categorical({'+','-','+'});
C = A.*B

C = 1x3 categorical array
    blue +    red -    green +
```

### 类别的笛卡尔积

显示 `C` 的类别。这些类别是可从 `A` 和 `B` 的类别创建的所有有序对，也称为笛卡尔积。

`categories(C)`

```
ans = 6x1 cell array
    {'blue +'}
    {'blue -'}
    {'green +'}
    {'green -'}
    {'red +'}
    {'red -'}
```

因此，`A.*B` 并不等于 `B.*A`。

`D = B.*A`

```
D = 1x3 categorical array
    + blue    - red    + green
```

`categories(D)`

```
ans = 6x1 cell array
    {'+ blue'}
    {'+ green'}
    {'+ red'}
    {'- blue'}
    {'- green'}
    {'- red'}
```

### 使用未定义元素的乘法

合并两个分类数组。如果 `A` 或 `B` 包含未定义元素，`C` 的相应元素也将是 `undefined`。

```
A = categorical({'blue','red','green','black'});
B = categorical({'+','-','+','-'});
```

```
A = removecats(A,{'black'});
C = A.*B

C = 1x4 categorical array
    blue +    red -    green +    <undefined>
```

### 有序分类数组的笛卡尔积

合并两个有序分类数组。仅当 A 和 B 都是有序的，C 才是有序分类数组。C 的类别排序遵循输入分类数组的排序。

```
A = categorical({'blue','red','green'},{'green','red','blue'},'Ordinal',true);
B = categorical({'+', '-', '+'}, 'Ordinal', true);
C = A.*B;
categories(C)

ans = 6x1 cell array
    {'green +'}
    {'green -'}
    {'red +'}
    {'red -'}
    {'blue +'}
    {'blue -'}
```

### 另请参阅

[categorical](#) | [categories](#) | [summary](#) | [times](#)

### 相关示例

- “创建分类数组”（第 8-2 页）
- “合并分类数组”（第 8-19 页）
- “使用分类数组访问数据”（第 8-24 页）

### 详细信息

- “有序分类数组”（第 8-36 页）

## 使用分类数组访问数据

### 本节内容

- “按类别选择数据”（第 8-24 页）
- “使用分类数组访问数据的常见方法”（第 8-24 页）

### 按类别选择数据

基于值选择数据通常很有用。此类数据选择涉及基于一个变量中的值创建一个逻辑向量，然后使用该逻辑向量选择另一个变量中的值的子集。您可以创建一个逻辑向量，以便通过查找位于特定范围中的数值数组中的值来选择数据。此外，您可以通过创建特定的离散值来创建逻辑向量。使用分类数组时，您可以轻松地：

- **选择特定类别的元素。**对于分类数组，使用逻辑 == 或 ~= 选择属于或不属于特定类别的数据。要选择一组特定类别的数据，请使用 `ismember` 函数。  
对于有序分类数组，可以使用不等运算符 >、>=、< 或 <= 计算特定类别之上或之下的类别中的数据。
- **删除特定类别的数据。**使用逻辑运算符包括或排除特定类别的数据。
- **查找不属于所定义类别的元素。**分类数组指示哪些元素不属于 `<undefined>` 定义的类别。使用 `isundefined` 函数查找被测元素中的未定义值。

### 使用分类数组访问数据的常见方法

此示例演示了如何使用分类数组建立索引和进行搜索。您可以按相似方式使用表中存储的分类数组来访问数据。

#### 加载样本数据

加载从 100 位患者收集的样本数据。

```
load patients
whos
```

Name	Size	Bytes	Class	Attributes
Age	100x1	800	double	
Diastolic	100x1	800	double	
Gender	100x1	12212	cell	
Height	100x1	800	double	
LastName	100x1	12416	cell	
Location	100x1	15008	cell	
SelfAssessedHealthStatus	100x1	12340	cell	
Smoker	100x1	100	logical	
Systolic	100x1	800	double	
Weight	100x1	800	double	

#### 基于字符向量元胞数组创建分类数组

`Gender` 和 `Location` 包含各类别中的数据。每个元胞数组都包含从一小组唯一值中获取的字符向量（分别表示两个性别和三个位置）。将 `Gender` 和 `Location` 转换为分类数组。

```
Gender = categorical(Gender);
Location = categorical(Location);
```

## 搜索单个类别的成员

对于分类数组，可以使用逻辑运算符 `==` 和 `~=` 来查找属于或不属于特定类别的数据。

确定是否存在到 'Rampart General Hospital' 位置就医的任何患者。

```
any(Location=='Rampart General Hospital')
```

```
ans = logical  
0
```

不存在到 Rampart General Hospital 就医的患者。

## 搜索一组类别的成员

您可以使用 `ismember` 查找一组特定类别中的数据。为到 County General Hospital 或 VA Hospital 就医的患者创建一个逻辑向量。

```
VA_CountyGenIndex = ...  
ismember(Location,['County General Hospital','VA Hospital']);
```

`VA_CountyGenIndex` 是一个  $100 \times 1$  的逻辑数组，对于分类数组 `Location` 中属于 County General Hospital 或 VA Hospital 类别的成员的每个元素，该数组都会在其对应位置包含逻辑值 `true` (1)。输出 `VA_CountyGenIndex` 包含 76 个非零元素。

使用逻辑向量 `VA_CountyGenIndex` 选择在 County General Hospital 或 VA Hospital 就医的患者的 `LastName`。

```
VA_CountyGenPatients = LastName(VA_CountyGenIndex);
```

`VA_CountyGenPatients` 是一个  $76 \times 1$  的字符向量元胞数组。

## 选择特定类别中的元素以绘图

使用 `summary` 函数可输出一份包含每一类别的类别名称及元素数的摘要。

```
summary(Location)
```

County General Hospital	39
St. Mary's Medical Center	24
VA Hospital	37

`Location` 是一个包含三个类别的  $100 \times 1$  分类数组。County General Hospital 出现在 39 个元素中，St. Mary's Medical Center 出现在 24 个元素中，VA Hospital 出现在 37 个元素中。

使用 `summary` 函数输出 `Gender` 的摘要。

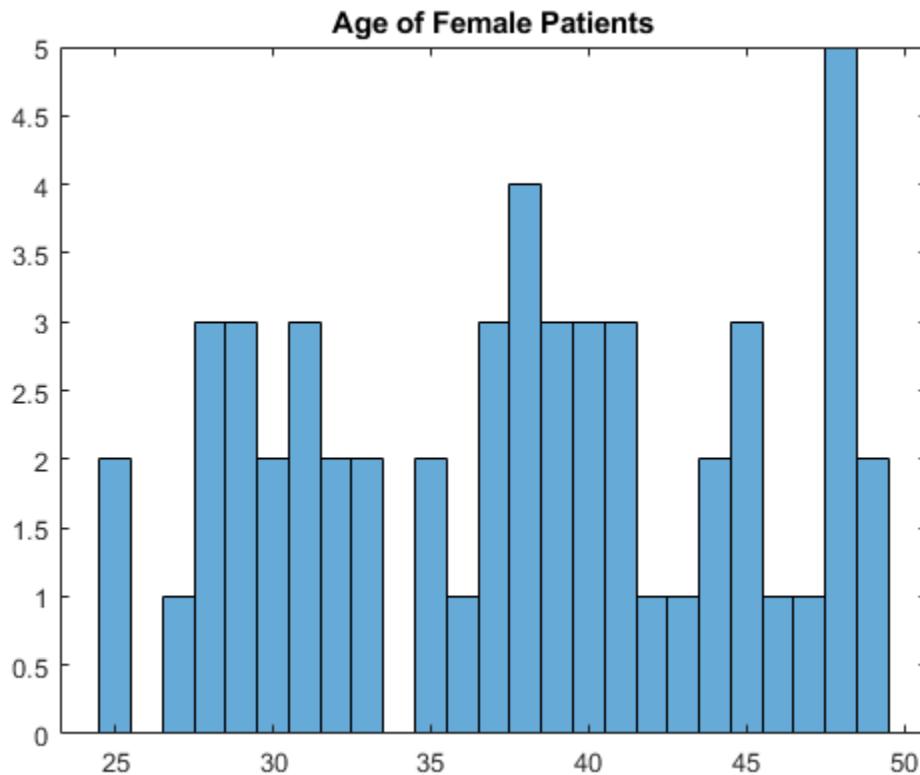
```
summary(Gender)
```

Female	53
Male	47

`Gender` 是一个包含两个类别的  $100 \times 1$  分类数组。Female 出现在 53 个元素中，Male 出现在 47 个元素中。

使用逻辑运算符 `==` 仅访问女性患者的年龄。然后利用这些数据绘制一个直方图。

```
figure()
histogram(Age(Gender=='Female'))
title('Age of Female Patients')
```



histogram(Age(Gender=='Female')) 对 53 个女性患者的年龄数据绘图。

#### 删除特定类别中的数据

您可以使用逻辑运算符包括或排除特定类别中的数据。删除工作区变量 **Age** 和 **Location** 中到 VA Hospital 就医的所有患者。

```
Age = Age(Location ~= 'VA Hospital');
Location = Location(Location ~= 'VA Hospital');
```

现在，**Age** 是一个  $63 \times 1$  的数值数组，**Location** 是一个  $63 \times 1$  的分类数组。

列出 **Location** 的类别以及每个类别中的元素数。

```
summary(Location)
```

County General Hospital	39
St. Mary's Medical Center	24
VA Hospital	0

到 VA Hospital 就医的患者已从 **Location** 中删除，但 VA Hospital 仍是一个类别。

使用 **removecats** 函数从 **Location** 的类别中删除 VA Hospital。

```
Location = removecats(Location,'VA Hospital');
```

验证 VA Hospital 类别是否已删除。

**categories(Location)**

```
ans = 2x1 cell array
{'County General Hospital' }
{'St. Mary's Medical Center'}
```

Location 是一个包含两个类别的  $63 \times 1$  分类数组。

### 删除元素

您可以按索引删除元素。例如，您可以通过使用 **Location(2:end)** 选择其余元素来删除 Location 的第一个元素。不过，更简单的元素删除方式是使用 **[]**。

```
Location(1) = [];
summary(Location)
```

County General Hospital	38
St. Mary's Medical Center	24

Location 是一个包含两个类别的  $62 \times 1$  分类数组。删除第一个元素对同一类别中的其他元素没有任何影响，并且不会删除该类别本身。

### 检查未定义的数据

从 Location 中删除类别 County General Hospital。

```
Location = removecats(Location,'County General Hospital');
```

显示分类数组 Location 的前八个元素。

**Location(1:8)**

```
ans = 8x1 categorical array
St. Mary's Medical Center
<undefined>
St. Mary's Medical Center
St. Mary's Medical Center
<undefined>
<undefined>
St. Mary's Medical Center
St. Mary's Medical Center
```

删除 County General Hospital 类别后，先前属于该类别的元素不再属于为 Location 定义的任何类别。分类数组将这些元素表示为 undefined。

使用 **isundefined** 函数可查找被测数据中不属于任何类别的值。

```
undefinedIndex = isundefined(Location);
```

undefinedIndex 是一个  $62 \times 1$  的分类数组，其中与 Location 中所有未定义元素相对应的位置均包含逻辑值 true (1)。

## 设置未定义的元素

使用 `summary` 函数输出 `Location` 中未定义的元素数量。

### `summary(Location)`

```
St. Mary's Medical Center    24
<undefined>                 38
```

`Location` 的第一个元素属于 `St. Mary's Medical Center` 类别。将第一个元素设置为 `undefined`，使其不再属于任何类别。

```
Location(1) = '<undefined>';
summary(Location)
```

```
St. Mary's Medical Center    23
<undefined>                 39
```

您可以将选定元素设置为 `undefined` 而不删除类别或更改其他元素的类别。将元素设置为 `undefined` 以指示包含未知值的元素。

## 预分配包含未定义元素的分类数组

您可以使用未定义的元素预分配分类数组的大小，以提高性能。创建一个分类数组，其中包含仅具有已知位置的元素。

```
definedIndex = ~isundefined(Location);
newLocation = Location(definedIndex);
summary(newLocation)
```

```
St. Mary's Medical Center    23
```

扩展 `newLocation` 的大小，使其成为一个  $200 \times 1$  的分类数组。将上一新元素设置为 `undefined`。所有其他新元素也会设置为 `undefined`。23 个原始元素将保留它们已有的值。

```
newLocation(200) = '<undefined>';
summary(newLocation)
```

```
St. Mary's Medical Center    23
<undefined>                 177
```

`newLocation` 可为您计划以后存储到数组中的值留出空间。

## 另请参阅

`any` | `categorical` | `categories` | `histogram` | `isundefined` | `removecats` | `summary`

## 相关示例

- “创建分类数组”（第 8-2 页）
- “将表变量中的文本转换为分类数组”（第 8-6 页）
- “对分类数据绘图”（第 8-10 页）
- “比较分类数组元素”（第 8-16 页）
- “使用受保护的分类数组”（第 8-30 页）

## 详细信息

- “使用分类数组的好处” (第 8-34 页)
- “有序分类数组” (第 8-36 页)

## 使用受保护的分类数组

此示例演示了如何使用包含受保护类别的分类数组。

当使用 **categorical** 函数创建分类数组时，您可以选择指定是否保护类别。有序分类数组始终具有受保护的类别，但您也可以使用 '**Protected**',**true** 名称-值对组参数创建受保护的非有序分类数组。

当指定的值不在数组的类别列表中时，该数组将自动更新，以便其类别列表中包括该新值。类似地，您可以合并具有不同类别的（非有序）分类数组。生成的类别中将同时包括这两个数组中的类别。

在向受保护的分类数组指定新值时，这些值必须属于现有类别之一。类似地，您只能合并具有相同类别的受保护数组。

- 如果要合并具有受保护类别的两个非有序分类数组，则它们必须具有相同的类别，顺序无关紧要。生成的分类数组使用第一个数组中的类别顺序。
- 如果要合并始终具有受保护类别的两个有序分类数组，则它们必须具有相同的类别，顺序也必须相同。

要向该数组添加新类别，必须使用 **addcats** 函数。

### 创建有序分类数组

创建一个大小包含 10 个对象的分类数组。对值 'S'、'M' 和 'L' 应用名称 **small**、**medium** 和 **large**。

```
A = categorical({'M';'L';'S';'S';'M';'L';'M';'L';'M';'S'},...
    {'S','M','L'},{'small','medium','large'},'Ordinal',true)
```

```
A = 10x1 categorical array
    medium
    large
    small
    small
    medium
    large
    medium
    large
    medium
    small
```

**A** 是一个  $10 \times 1$  的分类数组。

显示 **A** 的类别。

```
categories(A)
```

```
ans = 3x1 cell array
    {'small'}
    {'medium'}
    {'large'}
```

### 验证类别是否受保护

当创建有序分类数组时，这些类别始终受保护。

使用 **isprotected** 函数验证 **A** 的类别是否受保护。

```
tf = isprotected(A)
```

```
tf = logical  
1
```

A 的类别受保护。

### 指定新类别值

如果尝试指定一个不属于任何现有类别的新类别值，则 MATLAB® 将返回错误。例如，您无法像表达式 `A(2) = 'xlarge'` 中一样将值 'xlarge' 赋给分类数组，因为 `xlarge` 不是 A 的类别。MATLAB® 会返回以下错误：

```
Error using categorical/subsasgn (line 68)
```

```
Cannot add a new category 'xlarge' to this categorical array  
because its categories are protected. Use ADDCATS to  
add the new category.
```

要为 `xlarge` 添加新类别，请使用 `addcats` 函数。因为 A 是有序的，所以必须指定新类别的顺序。

```
A = addcats(A,'xlarge','After','large');
```

现在，使用 'xlarge' 进行赋值，因为它属于现有类别。

```
A(2) = 'xlarge'
```

```
A = 10x1 categorical array  
medium  
xlarge  
small  
small  
medium  
large  
medium  
large  
medium  
small
```

A 现在是一个包含四个类别的  $10 \times 1$  分类数组，其中 `small < medium < large < xlarge`。

### 合并两个有序分类数组

创建另外一个有序分类数组 B，其大小包含五个项目。

```
B = categorical([2;1;1;2;2],1:2,{'xsmall','small'},'Ordinal',true)
```

```
B = 5x1 categorical array  
small  
xsmall  
xsmall  
small  
small
```

**B** 是一个包含两个类别的  $5 \times 1$  分类数组，其中 **xsmall < small**。

要合并两个始终具有受保护类别的有序分类数组，它们必须具有相同的类别，而且这些类别的顺序也必须相同。

将类别 'xsmall' 添加到 A 的类别 'small' 之前。

```
A = addcats(A,'xsmall','Before','small');
```

```
categories(A)
```

```
ans = 5x1 cell array
{'xsmall'}
{'small' }
{'medium'}
{'large' }
{'xlarge'}
```

将类别 {'medium','large','xlarge'} 添加到 B 的类别 'small' 之后。

```
B = addcats(B,['medium','large','xlarge'],'After','small');
```

```
categories(B)
```

```
ans = 5x1 cell array
{'xsmall'}
{'small' }
{'medium'}
{'large' }
{'xlarge'}
```

A 和 B 的类别现在相同，包括其顺序也相同。

垂直串联 A 和 B。

```
C = [A;B]
```

```
C = 15x1 categorical array
medium
xlarge
small
small
medium
large
medium
large
medium
small
small
xsmall
xsmall
small
small
```

B 中的值将追加到 A 中的值之后。

列出 C 的类别。

**categories(C)**

```
ans = 5x1 cell array
{'xsmall'}
{'small' }
{'medium'}
{'large' }
{'xlarge'}
```

C 是一个包含五个类别的  $16 \times 1$  有序分类数组，其中 `xsmall < small < medium < large < xlarge`。

## 另请参阅

`addcats` | `categorical` | `categories` | `isordinal` | `isprotected` | `summary`

## 相关示例

- “创建分类数组” (第 8-2 页)
- “将表变量中的文本转换为分类数组” (第 8-6 页)
- “使用分类数组访问数据” (第 8-24 页)
- “合并分类数组” (第 8-19 页)
- “使用乘法合并分类数组” (第 8-22 页)

## 详细信息

- “有序分类数组” (第 8-36 页)

## 使用分类数组的好处

### 本节内容

- “分类数据的自然表示形式”（第 8-34 页）
- “字符向量的数学排序”（第 8-34 页）
- “降低内存要求”（第 8-34 页）

### 分类数据的自然表示形式

`categorical` 是用于存储具有以下特征的数据的数据类型：此类数据值来自离散分类有限集合。使用分类数组的一种常见替代方法是使用字符数组或字符向量元胞数组。要比较字符数组和字符向量元胞数组中的值，必须使用略微复杂的 `strcmp`。使用分类数组时，您可以像比较数值数组一样，使用逻辑运算符 `eq` (`==`) 比较元素。使用分类数组的另一种常见替代方法是将使用整数的分类数据存储在数值数组中。使用数值数组会丢失类别名称中的所有有用描述信息，还容易让人认为这些整数值只具有其常规的数值意义，而分类数据则不同。

### 字符向量的数学排序

分类数组是一种方便且节省内存的容器，适用于有限离散类别集合中的非数值数据。当类别按照有意义的数学方法排序时，分类数组会特别有用，例如对于一个由离散类别集合 `{'small','medium','large'}` 中的条目（其中 `small < medium < large`）组成的数组，便是如此。

字符数组或字符向量元胞数组不可采用字母顺序之外的排序方式。因此，不可以进行不相等比较，例如大于和小于。使用分类数组时，您可以使用关系运算来测试相等性，并执行具有有意义的数学排序的逐元素比较。

### 降低内存要求

以下示例演示了如何比较以字符向量元胞数组或分类数组形式存储数据所需的内存。分类数组具有定义为字符向量的类别，在字符向量元胞数组或 `char` 数组中存储和处理这些类别需要较大的代价。分类数组仅存储每个类别名称的一个副本，通常会降低存储该数组所需的内存量。

创建一个字符向量元胞数组示例。

```
state = [repmat({'MA'},25,1);repmat({'NY'},25,1);
         repmat({'CA'},50,1);...
         repmat({'MA'},25,1);repmat({'NY'},25,1)];
```

显示有关变量 `state` 的信息。

```
whos state
```

Name	Size	Bytes	Class	Attributes
state	150x1	17400	cell	

变量 `state` 是一个字符向量元胞数组，需要 17,400 字节的内存。

将 `state` 转换为分类数组。

```
state = categorical(state);
```

显示变量 `state` 中的离散类别。

```
categories(state)
```

```
ans = 3x1 cell array
{'CA'}
{'MA'}
{'NY'}
```

`state` 包含 150 个元素，但仅有三个不同类别。

显示有关变量 `state` 的信息。

```
whos state
```

Name	Size	Bytes	Class	Attributes
state	150x1	500	categorical	

使用分类数组极大降低了存储该变量所需的内存。

## 另请参阅

[categorical | categories](#)

## 相关示例

- “创建分类数组”（第 8-2 页）
- “将表变量中的文本转换为分类数组”（第 8-6 页）
- “比较分类数组元素”（第 8-16 页）
- “使用分类数组访问数据”（第 8-24 页）

## 详细信息

- “有序分类数组”（第 8-36 页）

## 有序分类数组

### 本节内容

- “类别的顺序”（第 8-36 页）
- “如何创建有序分类数组”（第 8-36 页）
- “使用有序分类数组”（第 8-37 页）

### 类别的顺序

`categorical` 是用于存储具有以下特征的数据的数据类型：其值来自离散分类有限集合，可以采用自然顺序。您可以指定并重新排列所有分类数组中的类别的顺序。但是，只有有序分类数组的类别可被视为具有数学排序。当需要使用 `min`、`max` 函数或需要进行关系运算（例如大于和小于）时，可以考虑使用有序分类数组。

宠物类别的离散集合 `{'dog' 'cat' 'bird'}` 具有无意义的数学排序。您可以使用任意类别顺序，关联的数据的意义不会改变。例如，`pets = categorical({'bird','cat','dog','dog','cat'})` 创建一个分类数组，相应类别按字母顺序列出：`{'bird' 'cat' 'dog'}`。您可以选择将类别的顺序指定或更改为 `{'dog' 'cat' 'bird'}`，数据的意义不会改变。

有序分类数组包含的类别具有有意义的数学排序。例如，离散的大小类别集合 `{'small', 'medium', 'large'}` 采用数学排序 `small < medium < large`。列出的第一个类别是最小的类别，最后一个类别是最大的类别。有序分类数组中类别的顺序会影响对有序分类数组进行关系比较的结果。

### 如何创建有序分类数组

此示例说明如何使用带 `'Ordinal'`,`true` 名称-值对组参数的 `categorical` 函数创建有序分类数组。

#### 基于字符向量元胞数组创建有序分类数组

基于字符向量元胞数组 `A` 创建一个有序分类数组 `sizes`。使用 `valueset`（指定为一个由唯一值构成的向量）定义 `sizes` 的类别。

```
A = {'medium' 'large';'small' 'medium'; 'large' 'small'};
valueset = {'small', 'medium', 'large'};

sizes = categorical(A,valueset,'Ordinal',true)

sizes = 3x2 categorical array
    medium    large
    small     medium
    large     small
```

`sizes` 是一个包含三个类别的  $3 \times 2$  有序分类数组，其中 `small < medium < large`。`valueset` 中的值的顺序即为 `sizes` 中的类别顺序。

#### 基于整数创建有序分类数组

基于一个整数数组创建一个等效的分类数组。使用值 1、2 和 3 分别定义类别 `small`、`medium` 和 `large`。

```
A2 = [2 3; 1 2; 3 1];
valueset = 1:3;
```

```

catnames = {'small','medium','large'};

sizes2 = categorical(A2,valueset,catnames,'Ordinal',true)

sizes2 = 3x2 categorical array
    medium    large
    small    medium
    large    small

```

比较 sizes 和 sizes2

```

isequal(sizes,sizes2)

ans = logical
    1

```

sizes 和 sizes2 是等效的分类数组，具有相同的类别顺序。

### 将非有序分类数组转换为有序分类数组

基于字符串向量元胞数组 A 创建一个非有序分类数组。

```

sizes3 = categorical(A)

sizes3 = 3x2 categorical array
    medium    large
    small    medium
    large    small

```

确定分类数组是否为有序。

```

isordinal(sizes3)

ans = logical
    0

```

sizes3 是一个包含三个类别的非有序分类数组：{'large','medium','small'}。 sizes3 的类别是 A 的唯一值且经过排序。您必须使用输入参数 valueset 指定一个不同的类别顺序。

将 sizes3 转换为有序分类数组，其中 small < medium < large。

```

sizes3 = categorical(sizes3',{'small','medium','large'},'Ordinal',true);

sizes3 现在是一个与 sizes 和 sizes2 等效的 3×2 有序分类数组。

```

## 使用有序分类数组

要合并或比较两个有序分类数组，这两个输入数组的类别集合必须相同，其顺序也必须相同。此外，有序分类数组始终受保护。因此，在向有序分类数组指定新值时，这些值必须属于现有类别之一。有关详细信息，请参阅“使用受保护的分类数组”（第 8-30 页）。

### 另请参阅

[categorical](#) | [categories](#) | [isequal](#) | [isordinal](#)

### 相关示例

- “创建分类数组” (第 8-2 页)
- “将表变量中的文本转换为分类数组” (第 8-6 页)
- “比较分类数组元素” (第 8-16 页)
- “使用分类数组访问数据” (第 8-24 页)

### 详细信息

- “使用分类数组的好处” (第 8-34 页)

## 支持分类数组的核心函数

MATLAB 中的许多函数在处理分类数组时与处理其他数组并无不同。但有一些函数在处理分类数组时，可能会有不一样的行为表现。如果多个输入参数是有序分类数组，函数通常要求这些数组具有相同的类别集合，顺序也必须相同。此外，一些函数（例如 `max` 和 `gt`）会要求输入分类数组必须是有序分类数组。

下表列出了除了处理分类数组还可处理其他数组的常见 MATLAB 函数。

<code>size</code>	<code>isequal</code>	<code>intersect</code>	<code>plot</code>	<code>double</code>
<code>length</code>	<code>isequaln</code>	<code>ismember</code>	<code>plot3</code>	<code>single</code>
<code>ndims</code>		<code>setdiff</code>	<code>scatter</code>	<code>int8</code>
<code>numel</code>	<code>eq</code>	<code>setxor</code>	<code>scatter3</code>	<code>int16</code>
<code>isrow</code>	<code>ne</code>	<code>unique</code>	<code>bar</code>	<code>int32</code>
<code>iscolumn</code>	<code>lt</code>	<code>union</code>	<code>barh</code>	<code>int64</code>
<code>cat</code>	<code>le</code>	<code>times</code>	<code>histogram</code>	<code>uint8</code>
<code>horzcat</code>	<code>ge</code>			<code>uint16</code>
<code>vertcat</code>	<code>gt</code>	<code>sort</code>	<code>pie</code>	<code>uint32</code>
	<code>min</code>	<code>sortrows</code>	<code>rose</code>	<code>uint64</code>
	<code>max</code>	<code>issorted</code>	<code>stem</code>	<code>char</code>
	<code>median</code>	<code>permute</code>	<code>stairs</code>	<code>string</code>
	<code>mode</code>	<code>reshape</code>	<code>area</code>	<code>cellstr</code>
		<code>transpose</code>	<code>mesh</code>	
		<code>ctranspose</code>	<code>surf</code>	
			<code>surface</code>	
			<code>semilogx</code>	
			<code>semilogy</code>	
			<code>fill</code>	
			<code>fill3</code>	
			<code>line</code>	
			<code>text</code>	



# 表

---

- “创建和使用表” (第 9-2 页)
- “添加和删除表行” (第 9-11 页)
- “添加、删除和重新排列表变量” (第 9-14 页)
- “清除表中的杂乱数据和缺失数据” (第 9-20 页)
- “修改单位、说明和表变量名称” (第 9-25 页)
- “向表和时间表中添加自定义属性” (第 9-28 页)
- “访问表中的数据” (第 9-33 页)
- “对表执行计算” (第 9-45 页)
- “将数据拆分为不同组并计算统计量” (第 9-48 页)
- “拆分表数据变量并应用函数” (第 9-51 页)
- “使用表的好处” (第 9-55 页)
- “对变量分组以拆分数据” (第 9-60 页)
- “R2016b 中对 DimensionNames 属性的更改” (第 9-62 页)

## 创建和使用表

以下示例说明如何根据工作区变量创建表，使用表数据，并将表写入文件以供日后使用。**table** 是一个数据类型，可将异构数据和元数据属性（例如变量名称、行名称、说明和变量单位）收集到一个容器中。

表适用于列向数据或表格数据，这些数据通常以列形式存储于文本文件或电子表格中。表中的每个变量可以具有不同的数据类型，但必须具有相同的行数。不过，表中的变量并不限于列向量。例如，表变量可以包含具有多列的矩阵，只要它的行数与其他表变量相同即可。表的典型用途是存储试验数据，使用行表示不同的观测值，使用列表示不同的测量变量。

表是用于收集和整理相关数据变量以及查看和汇总数据的便捷容器。例如，您可以提取变量以执行计算并方便地将相应结果添加为新的表变量。完成计算时，将该表写入文件以保存结果。

### 创建并查看表

根据工作区变量创建一个表并查看它。或者，使用 **导入工具** 或 **readtable** 功能根据电子表格或文本文件创建表。使用这些函数从文件导入数据时，每个列都会变为表变量。

将 100 位患者的样本数据从 **patients** MAT 文件加载到工作区变量中。

```
load patients
whos
```

Name	Size	Bytes	Class	Attributes
Age	100x1	800	double	
Diastolic	100x1	800	double	
Gender	100x1	12212	cell	
Height	100x1	800	double	
LastName	100x1	12416	cell	
Location	100x1	15008	cell	
SelfAssessedHealthStatus	100x1	12340	cell	
Smoker	100x1	100	logical	
Systolic	100x1	800	double	
Weight	100x1	800	double	

使用包含患者数据的列向变量填充表。您可以按名称访问和分配表变量。当您从工作区变量分配表变量时，您可以为表变量指定不同名称。

创建一个表并使用 **Gender**、**Smoker**、**Height** 和 **Weight** 工作区变量填充它。显示前五行。

```
T = table(Gender,Smoker,Height,Weight);
T(1:5,:)
```

	Gender	Smoker	Height	Weight
{'Male' }	true	71	176	
{'Male' }	false	69	163	
{'Female'}	false	64	131	
{'Female'}	false	67	133	
{'Female'}	false	64	119	

作为备选方法，可以使用 **readtable** 函数从逗号分隔文件中读取患者数据。**readtable** 会读取文件中的所有列。

通过读取文件 `patients.dat` 中的所有列来创建一个表。

```
T2 = readtable('patients.dat');
T2(1:5,:)
```

```
ans=5×10 table
  LastName    Gender    Age      Location     Height    Weight    Smoker    Systolic    Diastolic    SelfA
  _____    _____    ____    _____    _____    _____    _____    _____    _____    _____    _____
  {'Smith'}   {'Male'}   38    {'County General Hospital'}    71      176      1       124      93      {'Excellent'}
  {'Johnson'}  {'Male'}   43    {'VA Hospital'}           69      163      0       109      77      {'Fair'}
  {'Williams'} {'Female'} 38    {'St. Mary's Medical Center'}  64      131      0       125      83      {'Good'}
  {'Jones'}    {'Female'} 40    {'VA Hospital'}           67      133      0       117      75      {'Fair'}
  {'Brown'}   {'Female'} 49    {'County General Hospital'}  64      119      0       122      80      {'Good'}
```

您可以使用圆点表示法 `T.varname` 分配多个列向表变量，其中 `T` 是表，`varname` 是所需的变量名称。创建随机的数字标识符。然后，将其赋给一个表变量，并将该表变量命名为 `ID`。您分配到表中的所有变量必须具有相同的行数。显示 `T` 的前五行。

```
T.ID = randi(1e4,100,1);
T(1:5,:)
```

```
ans=5×5 table
  Gender    Smoker    Height    Weight    ID
  _____    _____    _____    _____    _____
  {'Male'}  true      71      176      8148
  {'Male'}  false     69      163      9058
  {'Female'} false    64      131      1270
  {'Female'} false    67      133      9134
  {'Female'} false    64      119      6324
```

您分配到表中的所有变量必须具有相同的行数。

使用 `summary` 函数创建汇总表来查看每个变量的数据类型、说明、单位和其他描述性统计量。

```
summary(T)
```

Variables:

Gender: 100x1 cell array of character vectors

Smoker: 100x1 logical

Values:

True	34
False	66

Height: 100x1 double

Values:

Min	60
Median	67
Max	72

Weight: 100x1 double

Values:

Min	111
Median	142.5
Max	202

ID: 100x1 double

Values:

Min	120
Median	5485.5
Max	9706

返回表的大小。

**size(T)**

ans = 1×2

100 5

T 包含 100 行和 5 个变量。

创建一个新的包含 T 中前五行的较小表并显示该表。您可以在括号中使用数值索引指定行和变量。此方法类似于通过索引数值数组来创建子数组的情况。Tnew 是一个 5×5 的表。

**Tnew = T(1:5,:)**

Tnew=5×5 table

Gender	Smoker	Height	Weight	ID
{'Male'}	true	71	176	8148
{'Male'}	false	69	163	9058
{'Female'}	false	64	131	1270
{'Female'}	false	67	133	9134
{'Female'}	false	64	119	6324

创建一个包含 Tnew 中所有行以及第二至最后行中的变量的较小表。使用 end 关键字指示表的最后一个变量或最后一行。Tnew 是一个 5×4 的表。

**Tnew = Tnew(:,2:end)**

Tnew=5×4 table

Smoker	Height	Weight	ID
true	71	176	8148
false	69	163	9058
false	64	131	1270
false	67	133	9134
false	64	119	6324

## 按行和变量名称访问数据

使用行和变量名称而非数值索引向 T 中添加行名称并对表进行索引。通过将 LastName 工作区变量赋给 T 的 RowNames 属性来添加行名称。

```
T.Properties.RowNames = LastName;
```

显示 T 的前五行以及行名称。

```
T(1:5,:)
```

```
ans=5×5 table
    Gender    Smoker    Height    Weight    ID
    _____    _____    _____    _____    ____
Smith    {'Male'}    true      71       176     8148
Johnson  {'Male'}    false     69       163     9058
Williams  {'Female'}  false     64       131     1270
Jones    {'Female'}  false     67       133     9134
Brown    {'Female'}  false     64       119     6324
```

返回 T 的大小。此大小不会更改，因为在计算表大小时不会包括行和变量名称。

```
size(T)
```

```
ans = 1×2
```

```
100    5
```

选择姓氏为 'Smith' 和 'Johnson' 的患者的所有数据。在本例中，使用行名称比使用数值索引更为简单。Tnew 是一个  $2 \times 5$  的表。

```
Tnew = T({'Smith','Johnson'},:)
```

```
Tnew=2×5 table
    Gender    Smoker    Height    Weight    ID
    _____    _____    _____    _____    ____
Smith    {'Male'}    true      71       176     8148
Johnson  {'Male'}    false     69       163     9058
```

通过变量名称索引来选择名为 'Johnson' 的患者的身高和体重。Tnew 是一个  $1 \times 2$  的表。

```
Tnew = T('Johnson',{'Height','Weight'})
```

```
Tnew=1×2 table
    Height    Weight
    _____    _____
Johnson   69       163
```

您可以使用圆点语法访问表变量（如 T.Height 中所示），也可以按命名索引访问表变量（如 T(:, 'Height') 中所示）。

### 计算结果并将其添加为表变量

您可以访问表变量的内容，然后使用 MATLAB® 函数对它们执行计算。根据现有表变量中的数据计算体质指数 (BMI)，并将其添加为新变量。绘制 BMI 与患者状态为烟民或非烟民的关系图。向表中添加血压读数，并绘制血压与 BMI 的关系图。

使用表变量 **Weight** 和 **Height** 计算 BMI。您可以提取 **Weight** 和 **Height** 以进行计算，同时在包含其余患者数据的表中方便地保留 **Weight**、**Height** 和 **BMI**。显示 T 的前五行。

```
T.BMI = (T.Weight*0.453592)./(T.Height*0.0254).^2;
```

```
T(1:5,:)
```

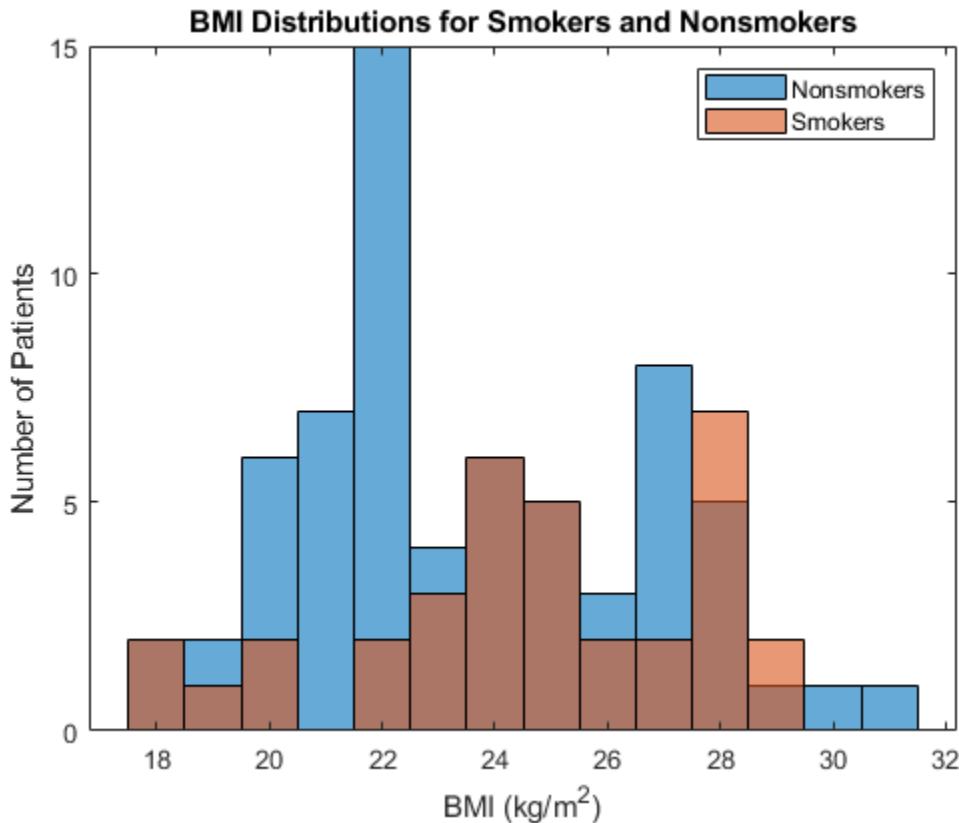
```
ans=5×6 table
    Gender    Smoker    Height    Weight    ID    BMI
    _____    ____    _____    _____    __    __
    Smith     {'Male'}   true      71      176    8148  24.547
    Johnson   {'Male'}   false     69      163    9058  24.071
    Williams  {'Female'} false     64      131    1270  22.486
    Jones     {'Female'} false     67      133    9134  20.831
    Brown     {'Female'} false     64      119    6324  20.426
```

填充 BMI 的变量单位和变量说明属性。您可以将元数据添加到任何表变量中，以进一步描述该变量中包含的数据。

```
T.Properties.VariableUnits{'BMI'} = 'kg/m^2';
T.Properties.VariableDescriptions{'BMI'} = 'Body Mass Index';
```

创建一个直方图以探索这组患者的吸烟与体质指数之间是否存在关系。您可以使用 Smoker 表变量中的逻辑值来索引 BMI，这是因为每行都包含同一患者的 BMI 和 Smoker 值。

```
tf = (T.Smoker == false);
h1 = histogram(T.BMI(tf),'BinMethod','integers');
hold on
tf = (T.Smoker == true);
h2 = histogram(T.BMI(tf),'BinMethod','integers');
xlabel('BMI (kg/m^2)');
ylabel('Number of Patients');
legend('Nonsmokers','Smokers');
title('BMI Distributions for Smokers and Nonsmokers');
hold off
```

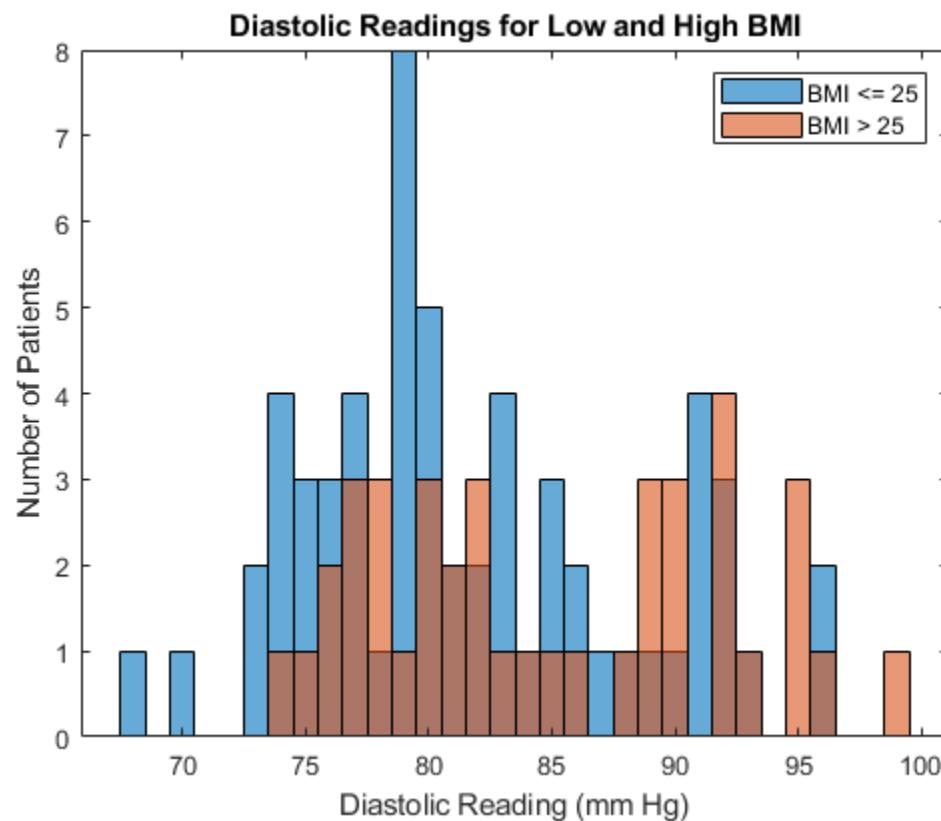


从工作区变量 Systolic 和 Diastolic 中添加患者的血压读数。每一行都包含同一患者的 Systolic、Diastolic 和 BMI 值。

T.Systolic = Systolic;  
T.Diastolic = Diastolic;

创建一个直方图来显示 Diastolic 和 BMI 的较高值之间是否存在关系。

```
tf = (T.BMI <= 25);
h1 = histogram(T.Diastolic(tf),'BinMethod','integers');
hold on
tf = (T.BMI > 25);
h2 = histogram(T.Diastolic(tf),'BinMethod','integers');
xlabel('Diastolic Reading (mm Hg)');
ylabel('Number of Patients');
legend('BMI <= 25','BMI > 25');
title('Diastolic Readings for Low and High BMI');
hold off
```



### 对表变量和输出行进行重新排序

要准备表进行输出，请按名称重新排列该表行的顺序，并按位置或名称重新排列表变量。显示表的最终排列。

按行名称对表进行排序，以使患者按字母顺序列出。

```
T = sortrows(T,'RowNames');
```

```
T(1:5,:)
```

```
ans=5×8 table
    Gender    Smoker    Height    Weight    ID    BMI    Systolic    Diastolic
    _____    _____    _____    _____    __    ___    _____    _____
    Adams    {'Female'}    false    66    137    8235    22.112    127    83
    Alexander    {'Male'}    true    69    171    1300    25.252    128    99
    Allen    {'Female'}    false    63    143    7432    25.331    113    80
    Anderson    {'Female'}    false    68    128    1577    19.462    114    77
    Bailey    {'Female'}    false    68    130    2239    19.766    113    81
```

创建一个 **BloodPressure** 变量，将血压读数保存在一个  $100 \times 2$  的表变量中。

```
T.BloodPressure = [T.Systolic T.Diastolic];
```

从表中删除 **Systolic** 和 **Diastolic**，因为它们是冗余的。

```
T.Systolic = [];
T.Diastolic = [];
```

**T(1:5,:)**

```
ans=5×7 table
    Gender    Smoker    Height    Weight    ID    BMI    BloodPressure
    _____    _____    _____    _____    ____    ____    _____
    Adams     {'Female'}  false     66      137    8235   22.112   127    83
    Alexander {'Male'}   true     69      171    1300   25.252   128    99
    Allen     {'Female'}  false     63      143    7432   25.331   113    80
    Anderson  {'Female'}  false     68      128    1577   19.462   114    77
    Bailey    {'Female'}  false     68      130    2239   19.766   113    81
```

要将 ID 作为第一列, 请按位置重新排列表变量。

**T = T(:,[5 1:4 6 7]);**

**T(1:5,:)**

```
ans=5×7 table
    ID    Gender    Smoker    Height    Weight    BMI    BloodPressure
    ____    _____    _____    _____    _____    ____    _____
    Adams  8235   {'Female'}  false     66      137    22.112   127    83
    Alexander 1300   {'Male'}   true     69      171    25.252   128    99
    Allen   7432   {'Female'}  false     63      143    25.331   113    80
    Anderson 1577   {'Female'}  false     68      128    19.462   114    77
    Bailey   2239   {'Female'}  false     68      130    19.766   113    81
```

您还可以按名称对表变量进行重新排序。要重新排列表变量以使 **Gender** 成为最后一个变量, 请执行以下操作:

- 1 在该表的 **VariableNames** 属性中查找 'Gender'。
- 2 将 'Gender' 移到变量名称元胞数组末尾。
- 3 使用名称元胞数组重新排列表变量的顺序。

```
varnames = T.Properties.VariableNames;
others = ~strcmp('Gender',varnames);
varnames = [varnames(others) 'Gender'];
T = T(:,varnames);
```

显示经过重新排序的表的前五行。

**T(1:5,:)**

```
ans=5×7 table
    ID    Smoker    Height    Weight    BMI    BloodPressure    Gender
    ____    _____    _____    _____    ____    _____    _____
    Adams  8235   false     66      137    22.112   127    83    {'Female'}
    Alexander 1300   true     69      171    25.252   128    99    {'Male'}
    Allen   7432   false     63      143    25.331   113    80    {'Female'}
    Anderson 1577   false     68      128    19.462   114    77    {'Female'}
```

```
Bailey    2239  false    68    130    19.766   113    81    {'Female'}
```

### 将表写入文件

您可以将整个表写入文件，或创建一个子表来将原始表的选定部分写入一个单独的文件。

使用 `writetable` 函数将 `T` 写入文件。

```
writetable(T,'allPatientsBMI.txt');
```

您可以使用 `readtable` 函数将 `allPatientsBMI.txt` 中的数据读入新表中。

创建一个子表并将其写入一个单独的文件中。删除包含烟民患者数据的行。然后删除 `Smoker` 变量。`nonsmokers` 仅包含非烟民患者的数据。

```
nonsmokers = T;
toDelete = (nonsmokers.Smoker == true);
nonsmokers(toDelete,:) = [];
nonsmokers.Smoker = [];
```

将 `nonsmokers` 写入文件。

```
writetable(nonsmokers,'nonsmokersBMI.txt');
```

### 另请参阅

`array2table` | `cell2table` | `readtable` | `sortrows` | `struct2table` | `summary` | `table` | `writetable` |  
导入工具

### 相关示例

- “清除表中的杂乱数据和缺失数据”（第 9-20 页）
- “修改单位、说明和表变量名称”（第 9-25 页）
- “访问表中的数据”（第 9-33 页）

### 详细信息

- “使用表的好处”（第 9-55 页）

# 添加和删除表行

此示例演示了如何在表中添加和删除行。您也可以使用变量编辑器来编辑表。

## 加载样本数据

加载样本患者数据并创建一个表 T。

```
load patients
T = table(LastName,Gender,Age,Height,Weight,Smoker,Systolic,Diastolic);
size(T)

ans = 1×2

100    8
```

表 T 包含 100 行和八个变量（列）。

## 通过串联添加行

将多位患者的数据从逗号分隔的文件 morePatients.csv 读入到表 T2 中。然后，将 T2 中的行追加到表 T 的末尾。

```
T2 = readtable('morePatients.csv');
Tnew = [T;T2];
size(Tnew)

ans = 1×2

104    8
```

表 Tnew 包含 104 行。要垂直串联两个表，这两个表必须具有相同的行数和相同的变量名称。如果变量名称不同，您可以将一个表中的行直接指定给另一个表，以此方式来插入新行。例如，`T(end+1:end+4,:)` = T2。

## 从元胞数组添加行

要追加元胞数组中存储的新行，请将元胞数组垂直串联到表的末尾。当元胞数组具有正确的列数并且其元胞的内容可以串联到对应的表变量上时，您可以直接从元胞数组进行串联。

```
cellPatients = {'Edwards','Male',42,70,158,0,116,83;
                'Falk','Female',28,62,125,1,120,71};
Tnew = [Tnew;cellPatients];
size(Tnew)

ans = 1×2

106    8
```

您也可以使用 `cell2table` 函数将元胞数组转换为表。

## 从结构体添加行

您也可以追加存储于结构体中的行。将该结构体转换为表，然后串联表。

```

structPatients(1,1).LastName = 'George';
structPatients(1,1).Gender = 'Male';
structPatients(1,1).Age = 45;
structPatients(1,1).Height = 76;
structPatients(1,1).Weight = 182;
structPatients(1,1).Smoker = 1;
structPatients(1,1).Systolic = 132;
structPatients(1,1).Diastolic = 85;

structPatients(2,1).LastName = 'Hadley';
structPatients(2,1).Gender = 'Female';
structPatients(2,1).Age = 29;
structPatients(2,1).Height = 58;
structPatients(2,1).Weight = 120;
structPatients(2,1).Smoker = 0;
structPatients(2,1).Systolic = 112;
structPatients(2,1).Diastolic = 70;

Tnew = [Tnew;struct2table(structPatients)];
size(Tnew)

ans = 1×2

```

108 8

### 忽略重复的行

要忽略表中的任何重复行，请使用 **unique** 函数。

```

Tnew = unique(Tnew);
size(Tnew)

```

ans = 1×2

106 8

**unique** 删除了两个重复行。

### 按行号删除行

删除表中的第 18、20 和 21 行。

```

Tnew([18,20,21],:) = [];
size(Tnew)

```

ans = 1×2

103 8

此表现在包含 103 位患者的信息。

### 按行名称删除行

首先，将标识符变量 **LastName** 指定为行名称。然后，从 **Tnew** 中删除变量 **LastName**。最后，使用行名称为这些行建立索引并将这些行删除。

```
Tnew.Properties.RowNames = Tnew.LastName;
Tnew.LastName = [];
Tnew('Smith', :) = [];
size(Tnew)

ans = 1×2

    102      7
```

现在，表比之前少了一行和一个变量。

### 搜索要删除的行

您也可以搜索表中的观测值。例如，删除年龄在 30 岁以下的所有患者对应的行。

```
toDelete = Tnew.Age < 30;
Tnew(toDelete,:) = [];
size(Tnew)

ans = 1×2
```

85 7

现在，表比之前少了 17 行。

## 另请参阅

[array2table](#) | [cell2table](#) | [readtable](#) | [struct2table](#) | [table](#)

## 相关示例

- “添加、删除和重新排列表变量”（第 9-14 页）
- “清除表中的杂乱数据和缺失数据”（第 9-20 页）

## 添加、删除和重新排列表变量

此示例说明如何在表中添加、删除和重新排列列向变量。您可以使用 `addvars`、`movevars` 和 `removevars` 函数添加、移动和删除表变量。作为备选方法，您还可以使用圆点语法或对表进行索引来修改表变量。使用 `splitvars` 和 `mergevars` 函数可以拆分多列变量和将多个变量合并为一个变量。最后，可以使用 `rows2vars` 函数调整表的方向，使表的行成为输出表的变量。

您也可以使用变量编辑器修改表变量。

### 加载样本数据并创建表

从 `patients` MAT 文件加载样本数据数组。显示加载到工作区中的变量的名称和大小。

```
load patients
whos -file patients
```

Name	Size	Bytes	Class	Attributes
Age	100x1	800	double	
Diastolic	100x1	800	double	
Gender	100x1	12212	cell	
Height	100x1	800	double	
LastName	100x1	12416	cell	
Location	100x1	15008	cell	
SelfAssessedHealthStatus	100x1	12340	cell	
Smoker	100x1	100	logical	
Systolic	100x1	800	double	
Weight	100x1	800	double	

创建两个表。用通过患者问卷收集来的信息创建一个表 `T`，再用测量患者得来的数据创建另一个表 `T2`。每个表有 100 行。

```
T = table(Age,Gender,Smoker);
T2 = table(Height,Weight,Systolic,Diastolic);
```

显示每个表的前五行。

```
head(T,5)
```

```
ans=5×3 table
    Age    Gender    Smoker
    ____    _____    _____

    38    {'Male'}    true
    43    {'Male'}    false
    38    {'Female'}  false
    40    {'Female'}  false
    49    {'Female'}  false
```

```
head(T2,5)
```

```
ans=5×4 table
    Height    Weight    Systolic    Diastolic
    _____    _____    _____    _____
    71        176       124        93
    69        163       109        77
```

```
64    131    125    83
67    133    117    75
64    119    122    80
```

### 串联添加来自另一个表的变量

通过将表 T 与 T2 水平串联来向前者添加变量。

```
T = [T T2];
```

显示 T 的前五行。

```
head(T,5)
```

```
ans=5×7 table
Age    Gender    Smoker    Height    Weight    Systolic    Diastolic
_____
38    {'Male' }  true      71      176      124      93
43    {'Male' }  false     69      163      109      77
38    {'Female'} false     64      131      125      83
40    {'Female'} false     67      133      117      75
49    {'Female'} false     64      119      122      80
```

表 T 现在包含 7 个变量和 100 行。

如果您要以水平方式串联的表具有行名称，则 `horzcat` 将通过匹配行名称来串联表。因此，这些表必须使用相同的行名称，但列顺序无关紧要。

### 将变量从工作区添加到表中

将来自工作区变量 LastName 的患者姓名添加到 T 中的第一个表变量之前。您可以使用新位置附近的变量的名称在表中指定任何位置。使用引号来引用表变量的名称。但是，如果输入参数是工作区变量，则不要对其使用引号。

```
T = addvars(T,'LastName','Before','Age');
head(T,5)
```

```
ans=5×8 table
LastName    Age    Gender    Smoker    Height    Weight    Systolic    Diastolic
_____
{'Smith' }  38    {'Male' }  true      71      176      124      93
{'Johnson'} 43    {'Male' }  false     69      163      109      77
{'Williams'} 38    {'Female'} false     64      131      125      83
{'Jones' }   40    {'Female'} false     67      133      117      75
{'Brown' }   49    {'Female'} false     64      119      122      80
```

您还可以使用数字指定表中的位置。例如，使用数字指定位置时，上述命令的等效语法为 `T = addvars(T,'LastName','Before',1)`。

### 使用圆点语法添加变量

添加新的表变量的另一种方法是使用圆点语法。当您使用圆点语法时，新变量始终添加为最后一个表变量。无论变量是什么数据类型，只要行数与表相同，就可以添加到表。

通过水平串联两个变量 **Systolic** 和 **Diastolic** 来创建一个针对血压的新变量。将它添加到 **T**。

```
T.BloodPressure = [Systolic Diastolic];
head(T,5)
```

	Last Name	Age	Gender	Smoker	Height	Weight	Systolic	Diastolic	Blood Pressure
{'Smith'}	38	{'Male'}	true	71	176	124	93	124	93
{'Johnson'}	43	{'Male'}	false	69	163	109	77	109	77
{'Williams'}	38	{'Female'}	false	64	131	125	83	125	83
{'Jones'}	40	{'Female'}	false	67	133	117	75	117	75
{'Brown'}	49	{'Female'}	false	64	119	122	80	122	80

**T** 现在包含 9 个变量和 100 行。一个表变量可以有多个列。因此，虽然 **BloodPressure** 有两列，但它是表变量。

将一个新变量 **BMI** 添加到表 **T** 中，该表包含每位患者的体重指数。**BMI** 是一个关于身高和体重的函数。当您计算 **BMI** 时，您可以引用 **T** 中的 **Weight** 和 **Height** 变量。

```
T.BMI = (T.Weight*0.453592)./(T.Height*0.0254).^2;
```

计算 **BMI** 时使用的运算符 **.** 和 **.^** 分别表示逐元素除法和乘方。

显示表 **T** 的前五行。

```
head(T,5)
```

	Last Name	Age	Gender	Smoker	Height	Weight	Systolic	Diastolic	Blood Pressure	BMI
{'Smith'}	38	{'Male'}	true	71	176	124	93	124	93	24.547
{'Johnson'}	43	{'Male'}	false	69	163	109	77	109	77	24.071
{'Williams'}	38	{'Female'}	false	64	131	125	83	125	83	22.486
{'Jones'}	40	{'Female'}	false	67	133	117	75	117	75	20.831
{'Brown'}	49	{'Female'}	false	64	119	122	80	122	80	20.426

### 在表中移动变量

使用 **movevars** 函数移动表变量 **BMI**，使其位于变量 **Weight** 之后。当按名称指定表变量时，请使用引号。

```
T = movevars(T,'BMI','After','Weight');
head(T,5)
```

	Last Name	Age	Gender	Smoker	Height	Weight	BMI	Systolic	Diastolic	Blood Pressure
{'Smith'}	38	{'Male'}	true	71	176	24.547	124	93	124	93
{'Johnson'}	43	{'Male'}	false	69	163	24.071	109	77	109	77
{'Williams'}	38	{'Female'}	false	64	131	22.486	125	83	125	83
{'Jones'}	40	{'Female'}	false	67	133	20.831	117	75	117	75
{'Brown'}	49	{'Female'}	false	64	119	20.426	122	80	122	80

您还可以使用数字指定表中的位置。例如，使用数字指定位置时，上述命令的等效语法为 `T = movevars(T,'BMI','After',6)`。按名称引用变量通常更方便。

### 使用索引移动表变量

除上述方法外，您还可以通过索引来移动表变量。您可以对表进行索引，使用的语法与对矩阵进行索引相同。

将 `BloodPressure` 移动到 `BMI` 之后。

```
T = T(:,[1:7 10 8 9]);
head(T,5)
```

	Last Name	Age	Gender	Smoker	Height	Weight	BMI	Blood Pressure	Systolic	Diastolic
{'Smith'}	38	{'Male'}	true	71	176	24.547	124	93	124	93
{'Johnson'}	43	{'Male'}	false	69	163	24.071	109	77	109	77
{'Williams'}	38	{'Female'}	false	64	131	22.486	125	83	125	83
{'Jones'}	40	{'Female'}	false	67	133	20.831	117	75	117	75
{'Brown'}	49	{'Female'}	false	64	119	20.426	122	80	122	80

在包含多个变量的表中，使用 `movevars` 函数通常更方便。

### 删除变量

要删除表变量，请使用 `removevars` 函数。删除 `Systolic` 和 `Diastolic` 表变量。

```
T = removevars(T,['Systolic','Diastolic']);
head(T,5)
```

	Last Name	Age	Gender	Smoker	Height	Weight	BMI	Blood Pressure
{'Smith'}	38	{'Male'}	true	71	176	24.547	124	93
{'Johnson'}	43	{'Male'}	false	69	163	24.071	109	77
{'Williams'}	38	{'Female'}	false	64	131	22.486	125	83
{'Jones'}	40	{'Female'}	false	67	133	20.831	117	75
{'Brown'}	49	{'Female'}	false	64	119	20.426	122	80

### 使用圆点语法删除变量

您也可以使用圆点语法和空矩阵 [] 删除变量。从表中删除 `Age` 变量。

```
T.Age = [];
head(T,5)
```

	Last Name	Gender	Smoker	Height	Weight	BMI	Blood Pressure
{'Smith'}	{'Male'}	true	71	176	24.547	124	93
{'Johnson'}	{'Male'}	false	69	163	24.071	109	77
{'Williams'}	{'Female'}	false	64	131	22.486	125	83
{'Jones'}	{'Female'}	false	67	133	20.831	117	75

```
{'Brown' }  {'Female'}  false    64     119    20.426   122    80
```

### 使用索引删除变量

您还可以使用索引和空矩阵 [] 删除变量。从表中删除 **Gender** 变量。

```
T(:, 'Gender') = [];
head(T, 5)
```

```
ans=5×6 table
 LastName  Smoker  Height  Weight  BMI  BloodPressure
 _____  _____  _____  _____  _____  _____
 {'Smith' }  true    71      176    24.547  124    93
 {'Johnson' }  false   69      163    24.071  109    77
 {'Williams' }  false   64      131    22.486  125    83
 {'Jones' }  false   67      133    20.831  117    75
 {'Brown' }  false   64      119    20.426  122    80
```

### 拆分和合并表变量

要将多列表变量拆分为多个单列变量，请使用 **splitvars** 函数。将变量 **BloodPressure** 拆分为两个变量。

```
T = splitvars(T, 'BloodPressure', 'NewVariableNames', {'Systolic', 'Diastolic'});
head(T, 5)
```

```
ans=5×7 table
 LastName  Smoker  Height  Weight  BMI  Systolic  Diastolic
 _____  _____  _____  _____  _____  _____  _____
 {'Smith' }  true    71      176    24.547  124    93
 {'Johnson' }  false   69      163    24.071  109    77
 {'Williams' }  false   64      131    22.486  125    83
 {'Jones' }  false   67      133    20.831  117    75
 {'Brown' }  false   64      119    20.426  122    80
```

同样，您可以使用 **mergevars** 函数将多个相关的表变量组合在一个变量中。将 **Systolic** 和 **Diastolic** 重新组合为一个变量，并将其命名为 **BP**。

```
T = mergevars(T, {'Systolic', 'Diastolic'}, 'NewVariableName', 'BP');
head(T, 5)
```

```
ans=5×6 table
 LastName  Smoker  Height  Weight  BMI      BP
 _____  _____  _____  _____  _____  _____
 {'Smith' }  true    71      176    24.547  124    93
 {'Johnson' }  false   69      163    24.071  109    77
 {'Williams' }  false   64      131    22.486  125    83
 {'Jones' }  false   67      133    20.831  117    75
 {'Brown' }  false   64      119    20.426  122    80
```

## 调整行的方向使其变为变量

您可以使用 `rows2vars` 函数调整表或时间表中行的方向，使它们成为输出表的变量。但是，如果该表具有多列变量，则必须先拆分它们，才能调用 `rows2vars`。

对 `T` 的行调整方向。指定 `T` 中患者的姓名是输出表中表变量的名称。`T3` 的第一个变量包含 `T` 的变量的名称。`T3` 的其余每个变量都包含来自 `T` 对应行的数据。

```
T = splitvars(T,'BP','NewVariableNames',{'Systolic','Diastolic'});
T3 = rows2vars(T,'VariableNamesSource','LastName');
T3(:,1:5)
```

```
ans=6×5 table
OriginalVariableNames Smith Johnson Williams Jones
_____
{'Smoker' }    1     0     0     0
{'Height' }   71    69    64    67
{'Weight' }  176   163   131   133
{'BMI' }    24.547 24.071 22.486 20.831
{'Systolic' } 124    109    125    117
{'Diastolic'} 93     77     83     75
```

您可以对 `T3` 使用圆点语法，以数组形式访问患者数据。但是，如果输入表的行值不能串联，则输出表的变量是元胞数组。

`T3.Smith`

```
ans = 6×1
1.0000
71.0000
176.0000
24.5467
124.0000
93.0000
```

## 另请参阅

`addvars` | `inner2outer` | `mergevars` | `movevars` | `removevars` | `rows2vars` | `splitvars` | `table`

## 相关示例

- “添加和删除表行”（第 9-11 页）
- “清除表中的杂乱数据和缺失数据”（第 9-20 页）
- “修改单位、说明和表变量名称”（第 9-25 页）

## 清除表中的杂乱数据和缺失数据

此示例演示了如何查找、清除和删除具有缺失数据的行。

### 加载样本数据

从一个逗号分隔的文本文件 `messy.csv` 加载样本数据。该文件包含许多不同的缺失数据指示符：

- 空字符串 ('')
- 句点 (.)
- NA
- NaN
- -99

要指定将视为空值的字符向量，请将 `'TreatAsEmpty'` 名称-值对组参数与 `readtable` 函数结合使用。  
(您可以使用 `disp` 函数来显示全部 21 行，即使以实时脚本方式运行此示例时也可以如此操作。)

```
T = readtable('messy.csv','TreatAsEmpty',{'!','NA'});
disp(T)
```

	A	B	C	D	E
{'afe1'}	3	{'yes' }	3	3	
{'egh3'}	NaN	{'no' }	7	7	
{'wth4'}	3	{'yes' }	3	3	
{'atn2'}	23	{'no' }	23	23	
{'arg1'}	5	{'yes' }	5	5	
{'jre3'}	34.6	{'yes' }	34.6	34.6	
{'wen9'}	234	{'yes' }	234	234	
{'ple2'}	2	{'no' }	2	2	
{'dbo8'}	5	{'no' }	5	5	
{'oi4'}	5	{'yes' }	5	5	
{'wnk3'}	245	{'yes' }	245	245	
{'abk6'}	563	{0x0 char}	563	563	
{'pnj5'}	463	{'no' }	463	463	
{'wnn3'}	6	{'no' }	6	6	
{'oks9'}	23	{'yes' }	23	23	
{'wba3'}	NaN	{'yes' }	NaN	14	
{'pkn4'}	2	{'no' }	2	2	
{'adw3'}	22	{'no' }	22	22	
{'poj2'}	-99	{'yes' }	-99	-99	
{'bas8'}	23	{'no' }	23	23	
{'gry5'}	NaN	{'yes' }	NaN	21	

`T` 是一个包含 21 行和 5 个变量的表。`'TreatAsEmpty'` 仅适用于文件中的数值列，无法处理指定为文本形式的数值（例如 `'-99'`）。

### 汇总表

使用 `summary` 函数创建汇总表来查看每个变量的数据类型、说明、单位和其他描述性统计量。

```
summary(T)
```

Variables:

A: 21x1 cell array of character vectors

B: 21x1 double

Values:

Min	-99
Median	14
Max	563
NumMissing	3

C: 21x1 cell array of character vectors

D: 21x1 double

Values:

Min	-99
Median	7
Max	563
NumMissing	2

E: 21x1 double

Values:

Min	-99
Median	14
Max	563

当从文件中导入数据时，默认情况是让 `readtable` 以字符串向量元胞数组形式读取包含非数值元素的任何变量。

### 查找具有缺失值的行

显示表 T 中至少具有一个缺失值的行子集。

```
TF = ismissing(T,{'!','NA','NaN'});  
rowsWithMissing = T(any(TF,2),:);  
disp(rowsWithMissing)
```

A	B	C	D	E
{'egh3'}	NaN	{'no'}	7	7
{'abk6'}	563	{0x0 char}	563	563
{'wba3'}	NaN	{'yes'}	NaN	14
{'poj2'}	-99	{'yes'}	-99	-99
{'gry5'}	NaN	{'yes'}	NaN	21

`readtable` 已将数值变量 B、D 和 E 中的 '!' 和 'NA' 替换为 NaN。

### 替换缺失值指示符

清除相应数据，以将代码 -99 所指示的缺失值替换为标准的 MATLAB® 数值缺失值指示符 NaN。

```
T = standardizeMissing(T,-99);  
disp(T)
```

	A	B	C	D	E
{afe1'}	3	{'yes' }	3	3	
{egh3'}	NaN	{'no' }	7	7	
{wth4'}	3	{'yes' }	3	3	
{atn2'}	23	{'no' }	23	23	
{arg1'}	5	{'yes' }	5	5	
{jre3'}	34.6	{'yes' }	34.6	34.6	
{wen9'}	234	{'yes' }	234	234	
{ple2'}	2	{'no' }	2	2	
{dbo8'}	5	{'no' }	5	5	
{oi4'}	5	{'yes' }	5	5	
{wnk3'}	245	{'yes' }	245	245	
{abk6'}	563	{0x0 char}	563	563	
{pnj5'}	463	{'no' }	463	463	
{wnn3'}	6	{'no' }	6	6	
{oks9'}	23	{'yes' }	23	23	
{wba3'}	NaN	{'yes' }	NaN	14	
{pkn4'}	2	{'no' }	2	2	
{adw3'}	22	{'no' }	22	22	
{poj2'}	NaN	{'yes' }	NaN	NaN	
{bas8'}	23	{'no' }	23	23	
{gry5'}	NaN	{'yes' }	NaN	21	

**standardizeMissing** 将三个 -99 替换为 NaN。

创建一个新表 T2，然后将缺失值替换为该表的前一行中的值。fillmissing 提供了许多方法来填充缺失值。

```
T2 = fillmissing(T,'previous');
disp(T2)
```

	A	B	C	D	E
{afe1'}	3	{'yes'}	3	3	
{egh3'}	3	{'no'}	7	7	
{wth4'}	3	{'yes'}	3	3	
{atn2'}	23	{'no'}	23	23	
{arg1'}	5	{'yes'}	5	5	
{jre3'}	34.6	{'yes'}	34.6	34.6	
{wen9'}	234	{'yes'}	234	234	
{ple2'}	2	{'no'}	2	2	
{dbo8'}	5	{'no'}	5	5	
{oi4'}	5	{'yes'}	5	5	
{wnk3'}	245	{'yes'}	245	245	
{abk6'}	563	{'yes'}	563	563	
{pnj5'}	463	{'no'}	463	463	
{wnn3'}	6	{'no'}	6	6	
{oks9'}	23	{'yes'}	23	23	
{wba3'}	23	{'yes'}	23	14	
{pkn4'}	2	{'no'}	2	2	
{adw3'}	22	{'no'}	22	22	
{poj2'}	22	{'yes'}	22	22	
{bas8'}	23	{'no'}	23	23	
{gry5'}	23	{'yes'}	23	21	

## 删除具有缺失值的行

创建一个新表 T3，该表仅包含 T 中不带缺失值的行。T3 只有 16 行。

```
T3 = rmmissing(T);
disp(T3)
```

A	B	C	D	E
{'afe1'}	3	{'yes'}	3	3
{'wth4'}	3	{'yes'}	3	3
{'atn2'}	23	{'no'}	23	23
{'arg1'}	5	{'yes'}	5	5
{'jre3'}	34.6	{'yes'}	34.6	34.6
{'wen9'}	234	{'yes'}	234	234
{'ple2'}	2	{'no'}	2	2
{'dbo8'}	5	{'no'}	5	5
{'oi4'}	5	{'yes'}	5	5
{'wnk3'}	245	{'yes'}	245	245
{'pnj5'}	463	{'no'}	463	463
{'wnn3'}	6	{'no'}	6	6
{'oks9'}	23	{'yes'}	23	23
{'pkn4'}	2	{'no'}	2	2
{'adw3'}	22	{'no'}	22	22
{'bas8'}	23	{'no'}	23	23

T3 包含 16 行和 5 个变量。

## 组织数据

先根据 C 以降序对 T3 的行进行排序，然后根据 A 以升序排序。

```
T3 = sortrows(T2,['C','A'],{'descend','ascend'});
disp(T3)
```

A	B	C	D	E
{'abk6'}	563	{'yes'}	563	563
{'afe1'}	3	{'yes'}	3	3
{'arg1'}	5	{'yes'}	5	5
{'gry5'}	23	{'yes'}	23	21
{'jre3'}	34.6	{'yes'}	34.6	34.6
{'oi4'}	5	{'yes'}	5	5
{'oks9'}	23	{'yes'}	23	23
{'poj2'}	22	{'yes'}	22	22
{'wba3'}	23	{'yes'}	23	14
{'wen9'}	234	{'yes'}	234	234
{'wnk3'}	245	{'yes'}	245	245
{'wth4'}	3	{'yes'}	3	3
{'adw3'}	22	{'no'}	22	22
{'atn2'}	23	{'no'}	23	23
{'bas8'}	23	{'no'}	23	23
{'dbo8'}	5	{'no'}	5	5
{'egh3'}	3	{'no'}	7	7
{'pkn4'}	2	{'no'}	2	2
{'ple2'}	2	{'no'}	2	2

```
{'pnj5'} 463 {'no'} 463 463
{'wnn3'} 6 {'no'} 6 6
```

在 C 中，各行首先按 'yes' 分组，然后再按 'no' 分组。然后在 A 中，各行以字母顺序排列。

对表进行重新排序，以使 A 和 C 彼此相邻。

```
T3 = T3(:,{'A','C','B','D','E'});
disp(T3)
```

A	C	B	D	E
{'abk6'}	{'yes'}	563	563	563
{'afe1'}	{'yes'}	3	3	3
{'arg1'}	{'yes'}	5	5	5
{'gry5'}	{'yes'}	23	23	21
{'jre3'}	{'yes'}	34.6	34.6	34.6
{'oii4'}	{'yes'}	5	5	5
{'oks9'}	{'yes'}	23	23	23
{'poj2'}	{'yes'}	22	22	22
{'wba3'}	{'yes'}	23	23	14
{'wen9'}	{'yes'}	234	234	234
{'wnk3'}	{'yes'}	245	245	245
{'wth4'}	{'yes'}	3	3	3
{'adw3'}	{'no'}	22	22	22
{'atn2'}	{'no'}	23	23	23
{'bas8'}	{'no'}	23	23	23
{'dbo8'}	{'no'}	5	5	5
{'egh3'}	{'no'}	3	7	7
{'pkn4'}	{'no'}	2	2	2
{'ple2'}	{'no'}	2	2	2
{'pnj5'}	{'no'}	463	463	463
{'wnn3'}	{'no'}	6	6	6

## 另请参阅

[fillmissing](#) | [ismissing](#) | [readtable](#) | [rmmissing](#) | [sortrows](#) | [standardizeMissing](#) | [summary](#)

## 相关示例

- “添加和删除表行”（第 9-11 页）
- “添加、删除和重新排列表变量”（第 9-14 页）
- “修改单位、说明和表变量名称”（第 9-25 页）
- “访问表中的数据”（第 9-33 页）
- “MATLAB 中的缺失数据”

# 修改单位、说明和表变量名称

此示例演示了如何访问和修改变量单位、说明和名称这几种表属性。您也可以使用变量编辑器来编辑这些属性值。

## 加载样本数据

加载样本患者数据并创建一个表。

```
load patients
BloodPressure = [Systolic Diastolic];
T = table(Gender,Age,Height,Weight,Smoker,BloodPressure);
```

显示表 T 的前五行。

```
T(1:5,:)
```

	Gender	Age	Height	Weight	Smoker	BloodPressure
{'Male'}	38	71	176	true	124	93
{'Male'}	43	69	163	false	109	77
{'Female'}	38	64	131	false	125	83
{'Female'}	40	67	133	false	117	75
{'Female'}	49	64	119	false	122	80

T 包含 100 行和 6 个变量。

## 添加变量单位

通过修改表属性 **VariableUnits** 来为表中的每个变量指定单位。使用字符向量元胞数组指定变量单位。

```
T.Properties.VariableUnits = {"'Yrs' 'In' 'Lbs' " "};
```

元胞数组中的空字符串向量指示对应的变量没有单位。

## 为单个变量添加变量说明

为变量 **BloodPressure** 添加变量说明。将单个字符向量赋给元胞数组中包含 **BloodPressure** 说明的元素。

```
T.Properties.VariableDescriptions{'BloodPressure'} = 'Systolic/Diastolic';
```

您可以使用变量名称 '**BloodPressure**' 或变量的数值索引 6 对包含变量说明的字符向量元胞数组进行索引。

## 汇总表

使用 **summary** 汇总表来查看每个变量的数据类型、说明、单位和其他描述性统计量。

```
summary(T)
```

Variables:

Gender: 100x1 cell array of character vectors

Age: 100x1 double

Properties:

Units: Yrs

Values:

Min	25
Median	39
Max	50

Height: 100x1 double

Properties:

Units: In

Values:

Min	60
Median	67
Max	72

Weight: 100x1 double

Properties:

Units: Lbs

Values:

Min	111
Median	142.5
Max	202

Smoker: 100x1 logical

Values:

True	34
False	66

BloodPressure: 100x2 double

Properties:

Description: Systolic/Diastolic

Values:

BloodPressure_1	BloodPressure_2
-----------------	-----------------

---

Min	109	68
Median	122	81.5
Max	138	99

BloodPressure 变量具有说明, Age、Height、Weight 和 BloodPressure 变量具有单位。

### 更改变量名称

将第一个变量的变量名称从 Gender 更改为 Sex.

```
T.Properties.VariableNames{'Gender'} = 'Sex';
```

显示表 T 的前五行。

**T(1:5,:)**

```
ans=5×6 table
    Sex      Age     Height    Weight    Smoker    BloodPressure
    _____    ____    _____    _____    _____    _____
{'Male' }    38      71      176      true      124      93
{'Male' }    43      69      163     false      109      77
{'Female"}   38      64      131     false      125      83
{'Female"}   40      67      133     false      117      75
{'Female"}   49      64      119     false      122      80
```

除了变量单位、说明和名称几个属性之外，还有表示行名称和维度名称、表说明以及用户数据的表属性。

## 另请参阅

[array2table](#) | [cell2table](#) | [readtable](#) | [struct2table](#) | [summary](#) | [table](#)

## 相关示例

- “添加、删除和重新排列表变量”（第 9-14 页）
- “访问表中的数据”（第 9-33 页）

## 向表和时间表中添加自定义属性

此示例说明如何向表和时间表中添加自定义属性、设置和访问属性值以及删除属性。

所有表和时间表都有一些包含有关这些表或其变量的元数据的属性。您可以通过 `T.Properties` 对象访问这些属性，其中 `T` 是表或时间表的名称。例如，`T.Properties.VariableNames` 返回一个元胞数组，其中包含 `T` 中变量的名称。

通过 `T.Properties` 访问的属性是 `table` 和 `timetable` 数据类型定义的一部分。不能添加或删除这些预定义的属性。但是，从 R2018b 开始，您可以通过修改表或时间表的 `T.Properties.CustomProperties` 对象来添加和删除您自己的自定义属性。

### 添加属性

将电力中断数据读取到表中。使用包含日期和时间的第一个变量 `OutageTime` 对表进行排序。然后显示前三行。

```
T = readtable('outages.csv');
T = sortrows(T,'OutageTime');
head(T,3)

ans=3×6 table
  Region      OutageTime    Loss   Customers  RestorationTime    Cause
  _____    _____       _____    _____    _____       _____
  {'SouthWest'}  2002-02-01 12:18  458.98  1.8202e+06  2002-02-07 16:50  {'winter storm'}
  {'MidWest' }  2002-03-05 17:53  96.563   2.8666e+05  2002-03-10 14:41  {'wind'     }
  {'MidWest' }  2002-03-16 06:18  186.44   2.1275e+05  2002-03-18 23:23  {'severe storm'}
```

显示其属性。这些是所有表都有的通用属性。请注意，还有一个 `CustomProperties` 对象，但默认情况下它并没有任何属性。

### `T.Properties`

```
ans =
TableProperties with properties:

  Description: "
  UserData: []
  DimensionNames: {'Row' 'Variables'}
  VariableNames: {1x6 cell}
  VariableDescriptions: {}
  VariableUnits: {}
  VariableContinuity: []
  RowNames: {}
  CustomProperties: No custom properties are set.
  Use addprop and rmprop to modify CustomProperties.
```

要添加自定义属性，请使用 `addprop` 函数。指定属性的名称。对于每个属性，还要指定它是包含整个表的元数据（类似于 `Description` 属性），还是包含其变量的元数据（类似于 `VariableNames` 属性）。如果属性包含变量元数据，则其值必须是一个长度等于变量数的向量。

添加一些自定义属性，其中包含输出文件名、文件类型以及表示要绘制哪些变量的指示符。最佳做法是将输入表指定为 `addprop` 的输出参数，使自定义属性成为同一个表的一部分。使用 '`table`' 选项指定输出文件名和文件类型为表元数据。使用 '`variable`' 选项指定绘图指示符为变量元数据。

```

T = addprop(T,{'OutputFileName','OutputFileType','ToPlot'}, ...
    {'table','table','variable'});
T.Properties

ans =
TableProperties with properties:

    Description: "
    UserData: []
    DimensionNames: {'Row' 'Variables'}
    VariableNames: {1x6 cell}
    VariableDescriptions: {}
    VariableUnits: {}
    VariableContinuity: []
    RowNames: {}

Custom Properties (access using t.Properties.CustomProperties.<name>):
    OutputFileName: []
    OutputFileType: []
    ToPlot: []

```

## 设置和访问自定义属性值

使用 **addprop** 添加自定义属性时，默认情况下它们的值为空数组。您可以使用圆点语法设置和访问自定义属性的值。

设置输出文件名和类型。这些属性包含表的元数据。然后为 **ToPlot** 属性指定一个逻辑数组。此属性包含变量的元数据。在以下示例中，对于要包含在图中的每个变量，**ToPlot** 属性值的元素为 **true**，对于要排除的每个变量，则为 **false**。

```

T.Properties.CustomProperties.OutputFileName = 'outageResults';
T.Properties.CustomProperties.OutputFileType = '.mat';
T.Properties.CustomProperties.ToPlot = [false false true true true false];
T.Properties

ans =
TableProperties with properties:

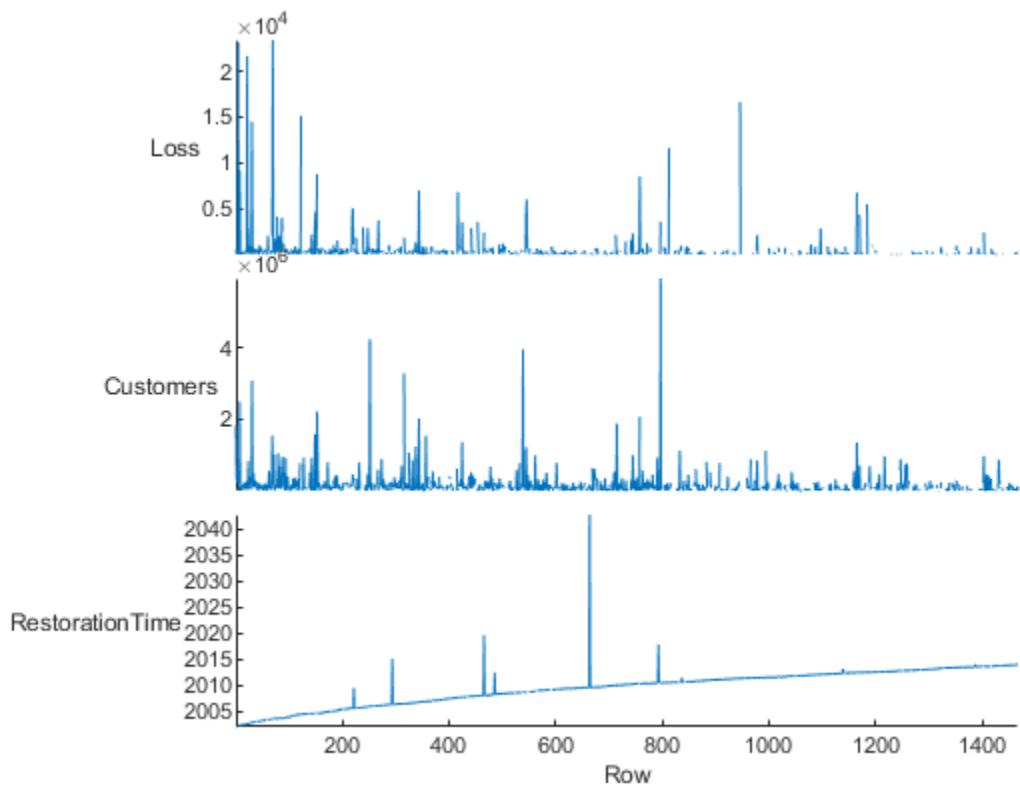
    Description: "
    UserData: []
    DimensionNames: {'Row' 'Variables'}
    VariableNames: {1x6 cell}
    VariableDescriptions: {}
    VariableUnits: {}
    VariableContinuity: []
    RowNames: {}

Custom Properties (access using t.Properties.CustomProperties.<name>):
    OutputFileName: 'outageResults'
    OutputFileType: '.mat'
    ToPlot: [0 0 1 1 1 0]

```

使用 **stackedplot** 函数将 T 中的变量绘制为一个堆叠图。要仅绘制 **Loss**、**Customers** 和 **RestorationTime** 值，请使用 **ToPlot** 自定义属性作为第二个输入参数。

```
stackedplot(T,T.Properties.CustomProperties.ToPlot);
```



移动或删除表变量时，预定义属性和自定义属性都会重新排序，以使其值所对应的变量保持不变。在此示例中，`ToPlot` 自定义属性的值与标记要绘制的变量保持对齐，就像 `VariableNames` 预定义属性的值保持对齐一样。

删除 `Customers` 变量并显示属性。

```
T.Customers = [];
T.Properties
```

```
ans =
TableProperties with properties:

    Description: "
    UserData: []
    DimensionNames: {'Row' 'Variables'}
    VariableNames: {1x5 cell}
    VariableDescriptions: {}
    VariableUnits: {}
    VariableContinuity: []
    RowNames: {}

Custom Properties (access using t.Properties.CustomProperties.<name>):
    OutputFileName: 'outageResults'
    OutputFileType: '.mat'
    ToPlot: [0 0 1 1 0]
```

将表转换为时间表，使用电力中断时间作为行时间。使用 `movevars` 函数将 `Region` 移到表的末尾，将 `RestorationTime` 移到第一个变量之前。请注意，属性已相应地进行了重新排序。`RestorationTime` 和 `Loss` 变量仍带有表示要在绘图中包含它们的指示符。

```
T = table2timetable(T);
T = movevars(T,'Region','After','Cause');
T = movevars(T,'RestorationTime','Before',1);
T.Properties

ans =
TimetableProperties with properties:

    Description: ''
    UserData: []
    DimensionNames: {'OutageTime' 'Variables'}
    VariableNames: {'RestorationTime' 'Loss' 'Cause' 'Region'}
    VariableDescriptions: {}
    VariableUnits: {}
    VariableContinuity: []
        RowTimes: [1468x1 datetime]
        StartTime: 2002-02-01 12:18
        SampleRate: NaN
        TimeStep: NaN

    Custom Properties (access using t.Properties.CustomProperties.<name>):
        OutputFileName: 'outageResults'
        OutputFileType: '.mat'
        ToPlot: [1 1 0 0]
```

## 删除属性

您可以使用 `rmprop` 函数删除表的任何或所有自定义属性。但是，不能使用它删除 `T.Properties` 中的预定义属性，因为这些属性是 `table` 数据类型定义的一部分。

删除 `OutputFileName` 和 `OutputFileType` 自定义属性。显示其余的表属性。

```
T = rmprop(T,['OutputFileName','OutputFileType']);
T.Properties

ans =
TimetableProperties with properties:

    Description: ''
    UserData: []
    DimensionNames: {'OutageTime' 'Variables'}
    VariableNames: {'RestorationTime' 'Loss' 'Cause' 'Region'}
    VariableDescriptions: {}
    VariableUnits: {}
    VariableContinuity: []
        RowTimes: [1468x1 datetime]
        StartTime: 2002-02-01 12:18
        SampleRate: NaN
        TimeStep: NaN

    Custom Properties (access using t.Properties.CustomProperties.<name>):
```

ToPlot: [1 1 0 0]

### 另请参阅

`addprop` | `head` | `movevars` | `readtable` | `rmprop` | `sortrows` | `stackedplot` | `table` |  
`table2timetable`

### 相关示例

- “修改单位、说明和表变量名称” (第 9-25 页)
- “访问表中的数据” (第 9-33 页)
- “添加、删除和重新排列表变量” (第 9-14 页)

# 访问表中的数据

## 本节内容

- “表索引语法一览”（第 9-33 页）
- “包含指定的行和变量的表”（第 9-35 页）
- “使用圆点表示法和逻辑值提取数据”（第 9-38 页）
- “圆点表示法支持任意变量名称或表达式”（第 9-41 页）
- “从指定的行和变量中提取数据”（第 9-43 页）

表是以变量形式存储列向数据的容器。表变量可以具有不同的数据类型和大小，只要所有变量具有相同的行数即可。表变量有名称，就像结构体的字段有名称一样。表的行可以有名称，但行名称不是必需的。要访问表数据，请使用行和变量的名称或数值索引对其进行索引。

对表进行索引的常见原因包括：

- 对行和变量重新排序或删除它们。
- 添加数组作为新的行或变量。
- 提取数据数组以用作函数的输入参数。

## 表索引语法一览

根据您使用的索引类型，您可以访问子表或表中提取的数组。使用以下方式进行索引：

- 圆括号 () 返回包含指定行和变量的表。
- 圆点表示法以数组形式返回变量的内容。
- 花括号 {} 返回由指定行和变量的内容串联而成的数组。

您可以通过名称、数值索引或数据类型指定行和变量。从 R2019b 开始，变量名称和行名称可以包含任何字符，包括空格和非 ASCII 字符。此外，它们可以由任何字符（而不仅仅是字母）开头。变量名称和行名称不必是有效的 MATLAB 标识符（由 `isvarname` 函数判别）。

输出的类型	语法	行	变量	示例
表，包含指定的行和变量	<code>T(rows,vars)</code>	<p>指定为：</p> <ul style="list-style-type: none"> <li>• 行号（在 1 和 m 之间）</li> <li>• 名称，如果 T 有行名称</li> <li>• 时间，如果 T 是时间表</li> <li>• 冒号 (:), 表示所有行</li> </ul>	<p>指定为：</p> <ul style="list-style-type: none"> <li>• 变量编号（在 1 和 n 之间）</li> <li>• 名称</li> <li>• 冒号 (:), 表示所有变量</li> </ul>	<ul style="list-style-type: none"> <li>• <code>T(1:5,[1 4 5])</code> 表，包含 T 中第一个、第四个和第五个变量的前五行</li> <li>• <code>T(:,{'A','B'})</code> 表，包含 T 中名为 'A' 和 'B' 的变量的所有行</li> </ul>

输出的类型	语法	行	变量	示例
<b>表，包含具有指定数据类型的变量</b>	<code>S = vartype(type); T(rows,S)</code>	<p>指定为：</p> <ul style="list-style-type: none"> <li>行号 (在 1 和 m 之间)</li> <li>名称，如果 T 有行名称</li> <li>时间，如果 T 是时间表</li> <li>冒号 (:), 表示所有行</li> </ul>	指定为数据类型，如 'numeric'、'categorical' 或 'datetime'	<ul style="list-style-type: none"> <li><code>S = vartype('numeric');</code> <code>T(1:5,S)</code> 表，包含 T 的前五行中的数值变量</li> </ul>
<b>数组，包含从一个变量中提取的数据</b>	<code>T.var</code> <code>T.(expression)</code>	未指定	指定为：	<ul style="list-style-type: none"> <li>变量名称 (不带引号)</li> <li>圆括号内的表达式，返回变量名称或变量编号</li> </ul>
<b>数组，包含从一个变量的指定行中提取的数据</b>	<code>T.var(rows)</code> <code>T.(expression)(rows)</code>	指定为数组的数值索引或逻辑索引	指定为：	<ul style="list-style-type: none"> <li>变量名称 (不带引号)</li> <li>圆括号内的表达式，返回变量名称或变量编号</li> </ul>

输出的类型	语法	行	变量	示例
<b>数组，串联来自指定的行和变量的数据</b>	<code>T{rows,vars}</code>	<p>指定为：</p> <ul style="list-style-type: none"> <li>行号（在 1 和 m 之间）</li> <li>名称，如果 T 有行名称</li> <li>时间，如果 T 是时间表</li> <li>冒号（：），表示所有行</li> </ul>	<p>指定为：</p> <ul style="list-style-type: none"> <li>变量编号（在 1 和 n 之间）</li> <li>名称</li> <li>冒号（：），表示所有变量</li> </ul>	<ul style="list-style-type: none"> <li><code>T{1:5,[1 4 5]}</code> 由 T 中第一个、第四个和第五个变量的前五行串联而成的数组</li> <li><code>T{:,{'A','B'}}</code> 由 T 中名为 'A' 和 'B' 的变量的所有行串联而成的数组</li> </ul>
<b>数组，串联指定行和变量中具有指定数据类型的数据</b>	<code>S = vartype(type); T{rows,S}</code>	<p>指定为：</p> <ul style="list-style-type: none"> <li>行号（在 1 和 m 之间）</li> <li>名称，如果 T 有行名称</li> <li>时间，如果 T 是时间表</li> <li>冒号（：），表示所有行</li> </ul>	指定为数据类型，如 'numeric'、'categorical' 或 'datetime'	<ul style="list-style-type: none"> <li><code>S = vartype('numeric');</code> <code>T{1:5,S}</code> 由 T 的前五行中的数值变量串联而成的数组</li> </ul>
<b>数组，串联变量的所有行的数据</b>	<code>T.Variables</code>	未指定	未指定	<ul style="list-style-type: none"> <li><code>T.Variables</code> 与 <code>T{:,}</code> 返回的数组相同</li> </ul>

## 包含指定的行和变量的表

将 100 位患者的样本数据从 `patients` MAT 文件加载到工作区变量中。

```
load patients
whos
```

Name	Size	Bytes	Class	Attributes
Age	100x1	800	double	
Diastolic	100x1	800	double	
Gender	100x1	12212	cell	
Height	100x1	800	double	
LastName	100x1	12416	cell	
Location	100x1	15008	cell	
SelfAssessedHealthStatus	100x1	12340	cell	
Smoker	100x1	100	logical	
Systolic	100x1	800	double	
Weight	100x1	800	double	

创建一个表并使用 `Age`、`Gender`、`Height`、`Weight` 和 `Smoker` 工作区变量填充它。将 `LastName` 中的唯一标识符用作行名称。T 是一个 100×5 表。（当您指定行名称时，它们不被视为表变量。）

```
T = table(Age,Gender,Height,Weight,Smoker,...  
'RowNames',LastName)
```

T=100×5 table

	Age	Gender	Height	Weight	Smoker
Smith	38	{'Male'}	71	176	true
Johnson	43	{'Male'}	69	163	false
Williams	38	{'Female'}	64	131	false
Jones	40	{'Female'}	67	133	false
Brown	49	{'Female'}	64	119	false
Davis	46	{'Female'}	68	142	false
Miller	33	{'Female'}	64	142	true
Wilson	40	{'Male'}	68	180	false
Moore	28	{'Male'}	68	183	false
Taylor	31	{'Female'}	66	132	false
Anderson	45	{'Female'}	68	128	false
Thomas	42	{'Female'}	66	137	false
Jackson	25	{'Male'}	71	174	false
White	39	{'Male'}	72	202	true
Harris	36	{'Female'}	65	129	false
Martin	48	{'Male'}	71	181	true
:					

### 使用数值索引建立索引

创建一个包含表 T 中所有变量前五行的子表。要指定所需的行和变量，请使用圆括号内数值索引。这种类型的索引类似于对数值数组进行索引。

```
T1 = T(1:5,:)
```

T1=5×5 table

	Age	Gender	Height	Weight	Smoker
Smith	38	{'Male'}	71	176	true
Johnson	43	{'Male'}	69	163	false
Williams	38	{'Female'}	64	131	false
Jones	40	{'Female'}	67	133	false
Brown	49	{'Female'}	64	119	false

T1 是一个 5×5 的表。

除了数值索引，您还可以在圆括号中使用行或变量名称。（在本例中，使用行索引和冒号比使用行或变量名称更紧凑。）

### 使用名称建立索引

选择姓氏为 'Williams' 和 'Brown' 的患者的所有数据。由于 T 中的行名称是患者的姓氏，请使用行名称对 T 进行索引。

```
T2 = T({'Williams','Brown'},:)
```

T2=2×5 table

	Age	Gender	Height	Weight	Smoker
Williams					
Brown					

Williams	38	{'Female'}	64	131	false
Brown	49	{'Female'}	64	119	false

**T2** 是一个  $2 \times 5$  的表。

您也可以按名称选择变量。创建一个表，其中只包含 **T** 中 **Height** 和 **Weight** 变量的前五行。显示该表。

```
T3 = T(1:5,{'Height','Weight'})
```

**T3=5×2 table**

Height	Weight	
Smith	71	176
Johnson	69	163
Williams	64	131
Jones	67	133
Brown	64	119

Smith	71	176
Johnson	69	163
Williams	64	131
Jones	67	133
Brown	64	119

表变量名称不必是有效的 MATLAB 标识符。它们可以包括空格和非 ASCII 字符，并且可以由任何字符开头。

向 **T** 添加一个带空格和短横线的变量名称。然后使用变量名称对 **T** 进行索引。

```
T = addvars(T,SelfAssessedHealthStatus,'NewVariableNames','Self-Assessed Health Status');
T(1:5,{'Age','Smoker','Self-Assessed Health Status'})
```

**ans=5×3 table**

Age	Smoker	Self-Assessed Health Status	
Smith	38	true	{'Excellent'}
Johnson	43	false	{'Fair'}
Williams	38	false	{'Good'}
Jones	40	false	{'Fair'}
Brown	49	false	{'Good'}

Smith	38	true	{'Excellent'}
Johnson	43	false	{'Fair'}
Williams	38	false	{'Good'}
Jones	40	false	{'Fair'}
Brown	49	false	{'Good'}

## 指定数据类型下标

除使用名称或编号指定变量外，您还可以创建一个数据类型下标来匹配具有相同数据类型的所有变量。

首先，创建一个数据类型下标来匹配数值表变量。

```
S = vartype('numeric')
```

**S =**

table vartype subscript:

Select table variables matching the type 'numeric'

See Access Data in a Table.

创建一个表，其中只包含 **T** 的前五行中的数值变量。

```
T4 = T(1:5,S)
```

```
T4=5×3 table
```

	Age	Height	Weight
Smith	38	71	176
Johnson	43	69	163
Williams	38	64	131
Jones	40	67	133
Brown	49	64	119

## 使用圆点表示法和逻辑值提取数据

根据 **patients** MAT 文件创建一个表。然后使用圆点表示法从表变量中提取数据。您还可以基于满足条件的表变量值生成逻辑索引，用它们进行索引。

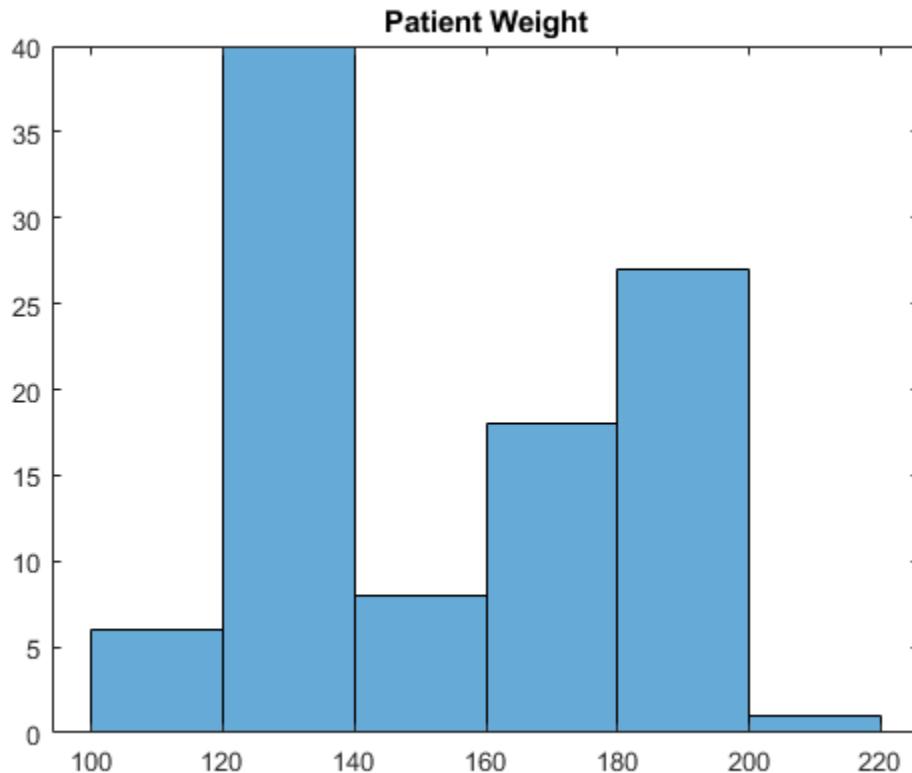
```
load patients
```

```
T = table(Age,Gender,Height,Weight,Smoker,...  
'RowNames',LastName);
```

### 从变量中提取数据

要从一个变量中提取数据，请使用圆点表示法。从变量 **Weight** 中提取数值。然后使用这些值绘制一个直方图。

```
histogram(T.Weight)  
title('Patient Weight')
```



T.Weight 是一个包含 100 行的双精度列向量。

#### 使用逻辑索引选择行

您可以使用逻辑索引数组对数组或表进行索引。通常，您使用逻辑表达式来确定表变量中的哪些值满足条件。表达式的结果是一个逻辑索引数组。

例如，创建一个逻辑数组，用于匹配年龄小于 40 的患者。

```
rows = T.Age < 40
```

```
rows = 100x1 logical array
```

```
1  
0  
1  
0  
0  
0  
1  
0  
1  
1  
:
```

要提取年龄小于 40 的患者的身高数据，请使用 rows 对 Height 变量进行索引。有 56 名患者的年龄小于 40。

**T.Height(rows)**

ans = 56×1

```
71
64
64
68
66
71
72
65
69
69
:
:
```

您可以用逻辑索引对表进行索引。显示 T 中年龄小于 40 的患者的行。

**T(rows,:)**

ans=56×5 table

	Age	Gender	Height	Weight	Smoker
Smith	38	{'Male'}	71	176	true
Williams	38	{'Female'}	64	131	false
Miller	33	{'Female'}	64	142	true
Moore	28	{'Male'}	68	183	false
Taylor	31	{'Female'}	66	132	false
Jackson	25	{'Male'}	71	174	false
White	39	{'Male'}	72	202	true
Harris	36	{'Female'}	65	129	false
Thompson	32	{'Male'}	69	191	true
Garcia	27	{'Female'}	69	131	true
Martinez	37	{'Male'}	70	179	false
Rodriguez	39	{'Female'}	64	117	false
Walker	28	{'Female'}	65	123	true
Hall	25	{'Male'}	70	189	false
Allen	39	{'Female'}	63	143	false
Young	25	{'Female'}	63	114	false
:					

您可以用一个逻辑表达式匹配多个条件。显示年龄小于 40 的烟民患者的行。

**rows = (T.Smoker==true & T.Age<40);**

**T(rows,:)**

ans=18×5 table

	Age	Gender	Height	Weight	Smoker
Smith	38	{'Male'}	71	176	true
Miller	33	{'Female'}	64	142	true
White	39	{'Male'}	72	202	true
Thompson	32	{'Male'}	69	191	true
Garcia	27	{'Female'}	69	131	true
Walker	28	{'Female'}	65	123	true

```

King      30  {'Male' }   67   186   true
Nelson    33  {'Male' }   66   180   true
Mitchell   39  {'Male' }   71   164   true
Turner     37  {'Male' }   70   194   true
Sanders    33  {'Female'}  67   115   true
Price      31  {'Male' }   72   178   true
Jenkins    28  {'Male' }   69   189   true
Long       39  {'Male' }   68   182   true
Patterson   37  {'Female'}  65   120   true
Flores     31  {'Female'}  66   141   true
:

```

## 圆点表示法支持任意变量名称或表达式

使用圆点表示法进行索引时，有两种方式可以指定变量。

- 使用名称，不带引号。例如，`T.Date` 指定名为 'Date' 的变量。
- 使用表达式，表达式置于点之后，用圆括号括起。例如，`T.('Start Date')` 指定名为 'Start Date' 的变量。

当表变量名称碰巧也是有效的 MATLAB® 标识符时，使用第一种语法。（有效的标识符以字母开头，仅包括字母、数字和下划线。）

指定下列各项时使用第二种语法：

- 指示变量在表中位置的编号。
- 不是有效 MATLAB 标识符的变量名称。
- 一个函数，其输出是表中变量的名称，或是添加到表中的变量。函数的输出必须为字符向量或字符串标量。

例如，根据 `patients` MAT 文件创建一个表。然后使用圆点表示法来访问表变量的内容。

```
load patients
```

```
T = table(Age,Gender,Height,Weight,Smoker, ...
'RowNames',LastName);
```

要按在表中的位置指定变量，请使用编号。`Age` 是 `T` 中的第一个变量，因此使用编号 1 来指定其位置。

```
T.(1)
```

```
ans = 100×1
```

```

38
43
38
40
49
46
33
40
28
31
:
```

要按名称指定变量，可以用引号将其括起来。由于 'Age' 是有效标识符，您可以使用 T.Age 或 T.(Age) 指定它。

**T.(Age)**

ans = 100×1

```
38
43
38
40
49
46
33
40
28
31
:
```

您指定的表变量名称不必是有效的 MATLAB 标识符。变量名称可以包括空格和非 ASCII 字符，并且可以由任何字符开头。但是，当您使用圆点表示法来访问具有此类名称的表变量时，您必须使用圆括号来指定它。

向 T 添加一个带空格和连字符的变量名称。

```
T = addvars(T,'SelfAssessedHealthStatus','NewVariableNames','Self-Assessed Health Status');
T(1:5,:);
```

ans=5×6 table

	Age	Gender	Height	Weight	Smoker	Self-Assessed Health Status
Smith	38	{'Male'}	71	176	true	{'Excellent'}
Johnson	43	{'Male'}	69	163	false	{'Fair'}
Williams	38	{'Female'}	64	131	false	{'Good'}
Jones	40	{'Female'}	67	133	false	{'Fair'}
Brown	49	{'Female'}	64	119	false	{'Good'}

使用圆点表示法访问新的表变量。显示前五个元素。

```
C = T.(‘Self-Assessed Health Status’);
C(1:5)
```

ans = 5x1 cell array

```
{'Excellent'}
{'Fair'}
{'Good'}
{'Fair'}
{'Good'}
```

您也可以将函数的输出用作变量名称。删除 T.(‘Self-Assessed Health Status’) 变量。然后将其替换为名称包含今天日期的变量。

```
T.(‘Self-Assessed Health Status’) = [];
T.(string(datetime(‘today’)) + ‘ Self Report’) = SelfAssessedHealthStatus;
T(1:5,:)
```

```
ans=5×6 table
Age    Gender    Height   Weight   Smoker   30-Jul-2019 Self Report
_____
Smith   38    {'Male'}    71     176    true     {'Excellent'}
Johnson 43    {'Male'}    69     163    false    {'Fair'}
Williams 38    {'Female'}  64     131    false    {'Good'}
Jones    40    {'Female'}  67     133    false    {'Fair'}
Brown    49    {'Female'}  64     119    false    {'Good'}
```

## 从指定的行和变量中提取数据

使用花括号进行索引会从表中提取数据，并生成数组，而不是子表。但是，除上述差异外，您可以像使用圆括号进行索引时一样，使用编号、名称和数据类型下标来指定行和变量。要从表中提取值，请使用花括号。如果从多个表变量中提取值，则这些变量的数据类型必须支持它们彼此串联。

### 指定行和变量

基于 `patients` 文件中的数值和逻辑数组创建一个表。

```
load patients
```

```
T = table(Age,Height,Weight,Smoker, ...
'RowNames',LastName);
```

从 `T` 的多个变量中提取数据。与圆点表示法不同，使用花括号进行索引可以从多个表变量中提取值，并将它们串联成一个数组。

提取前 5 位患者的身高和体重。使用数值索引选择前五行，使用变量名称选择变量 `Height` 和 `Weight`。

```
A = T{1:5,['Height','Weight']}
```

```
A = 5×2
```

```
71 176
69 163
64 131
67 133
64 119
```

`A` 是一个  $5 \times 2$  数值数组，而不是表。

如果您指定一个变量名称，则花括号索引生成的数组与使用圆点表示法得到的数组相同。但是，在使用花括号索引时，必须同时指定行和变量。例如，语法 `T.Height` 和 `T{:, 'Height'}` 返回相同的数组。

### 从所有行和变量中提取数据

如果所有表变量的数据类型都支持它们彼此串联，则可以使用 `T.Variables` 语法将所有表数据放入一个数组中。此语法等效于 `T{:, :}`，其中冒号表示所有行和所有变量。

```
A2 = T.Variables
```

```
A2 = 100×4
```

```
38  71  176  1
43  69  163  0
38  64  131  0
40  67  133  0
49  64  119  0
46  68  142  0
33  64  142  1
40  68  180  0
28  68  183  0
31  66  132  0
:
```

### 另请参阅

**addvars | histogram | table | vartype**

### 相关示例

- “使用表的好处” (第 9-55 页)
- “创建和使用表” (第 9-2 页)
- “修改单位、说明和表变量名称” (第 9-25 页)
- “对表执行计算” (第 9-45 页)
- “查找符合条件的数组元素” (第 5-2 页)

# 对表执行计算

此示例说明如何对表执行计算。

函数 `rowfun` 和 `varfun` 都可将指定的函数应用于表，而其他许多函数则需要使用数值或同构数组作为输入参数。您可以使用点索引提取单个变量中的数据，或使用花括号提取一个或多个变量中的数据。提取的数据会是一个数组，您可以将该数组用作为其他函数的输入。

## 将样本数据读入到表中

使用 `readtable` 函数将逗号分隔的文本文件 `testScores.csv` 中的数据读入到表中。`testScores.csv` 包含多个学生的测试分数。使用该文本文件第一列中的学生姓名作为表中的行名称。

```
T = readtable('testScores.csv','ReadRowNames',true)
```

```
T=10×4 table
    Gender    Test1    Test2    Test3
    _____  _____  _____  _____
    HOWARD    {'male'}    90    87    93
    WARD      {'male'}    87    85    83
    TORRES    {'male'}    86    85    88
    PETERSON  {'female'}  75    80    72
    GRAY       {'female'}  89    86    87
    RAMIREZ   {'female'}  96    92    98
    JAMES      {'male'}   78    75    77
    WATSON    {'female'}  91    94    92
    BROOKS    {'female'}  86    83    85
    KELLY      {'male'}   79    76    82
```

`T` 是一个包含 10 行和四个变量的表。

## 汇总表

使用 `summary` 函数汇总表来查看每个变量的数据类型、说明、单位和其他描述性统计量。

```
summary(T)
```

Variables:

Gender: 10x1 cell array of character vectors

Test1: 10x1 double

Values:

Min	75
Median	86.5
Max	96

Test2: 10x1 double

Values:

Min	75
Median	85

```
Max      94
```

Test3: 10x1 double

Values:

Min	72
Median	86
Max	98

汇总包含每个测试的最低分、中值分和最高分。

### 计算每行的平均值

使用花括号 {} 提取第二、第三和第四个变量中的数据，计算每一行的平均值，然后将该平均值存储于一个新变量 TestAvg 中。

```
T.TestAvg = mean(T{:,2:end},2)
```

T=10×5 table

	Gender	Test1	Test2	Test3	TestAvg
HOWARD	{'male'}	90	87	93	90
WARD	{'male'}	87	85	83	85
TORRES	{'male'}	86	85	88	86.333
PETERSON	{'female'}	75	80	72	75.667
GRAY	{'female'}	89	86	87	87.333
RAMIREZ	{'female'}	96	92	98	95.333
JAMES	{'male'}	78	75	77	76.667
WATSON	{'female'}	91	94	92	92.333
BROOKS	{'female'}	86	83	85	84.667
KELLY	{'male'}	79	76	82	79

您也可以使用变量名称 T{:,{'Test1','Test2','Test3'}} 或变量索引 T{:,2:4} 选择数据子集。

### 使用分组变量计算统计量

按学生的性别计算 TestAvg 的均值和最大值。

```
varfun(@mean,T,'InputVariables','TestAvg',...
    'GroupingVariables','Gender')
```

ans=2×3 table

	Gender	GroupCount	mean_TestAvg
{'female'}	5	87.067	
{'male'}	5	83.4	

### 替换数据值

每个测试的最高分为 100。使用花括号提取表中的数据，并将测试得分转换为 25 分制得分。

```
T{:,2:end} = T{:,2:end}*25/100
```

T=10×5 table

	Gender	Test1	Test2	Test3	TestAvg
HOWARD	{'male'}	90	87	93	90
WARD	{'male'}	87	85	83	85
TORRES	{'male'}	86	85	88	86.333
PETERSON	{'female'}	75	80	72	75.667
GRAY	{'female'}	89	86	87	87.333
RAMIREZ	{'female'}	96	92	98	95.333
JAMES	{'male'}	78	75	77	76.667
WATSON	{'female'}	91	94	92	92.333
BROOKS	{'female'}	86	83	85	84.667
KELLY	{'male'}	79	76	82	79

HOWARD	{'male' }	22.5	21.75	23.25	22.5
WARD	{'male' }	21.75	21.25	20.75	21.25
TORRES	{'male' }	21.5	21.25	22	21.583
PETERSON	{'female'}	18.75	20	18	18.917
GRAY	{'female'}	22.25	21.5	21.75	21.833
RAMIREZ	{'female'}	24	23	24.5	23.833
JAMES	{'male' }	19.5	18.75	19.25	19.167
WATSON	{'female'}	22.75	23.5	23	23.083
BROOKS	{'female'}	21.5	20.75	21.25	21.167
KELLY	{'male' }	19.75	19	20.5	19.75

## 更改变量名称

将变量名称从 TestAvg 更改为 Final.

**T.Properties.VariableNames{end} = 'Final'**

T=10×5 table

	Gender	Test1	Test2	Test3	Final
HOWARD	{'male' }	22.5	21.75	23.25	22.5
WARD	{'male' }	21.75	21.25	20.75	21.25
TORRES	{'male' }	21.5	21.25	22	21.583
PETERSON	{'female'}	18.75	20	18	18.917
GRAY	{'female'}	22.25	21.5	21.75	21.833
RAMIREZ	{'female'}	24	23	24.5	23.833
JAMES	{'male' }	19.5	18.75	19.25	19.167
WATSON	{'female'}	22.75	23.5	23	23.083
BROOKS	{'female'}	21.5	20.75	21.25	21.167
KELLY	{'male' }	19.75	19	20.5	19.75

## 另请参阅

[findgroups](#) | [rowfun](#) | [splitapply](#) | [summary](#) | [table](#) | [varfun](#)

## 相关示例

- “访问表中的数据”（第 9-33 页）
- “拆分表数据变量并应用函数”（第 9-51 页）

## 将数据拆分为不同组并计算统计量

此示例说明如何将 `patients.mat` 数据文件中的数据拆分为多个组。然后，它会显示如何为各组患者计算平均体重和体重指数以及血压读数差异。它还显示如何在表中汇总结果。

### 加载患者数据

加载从 100 位患者收集的样本数据。

```
load patients
```

将 `Gender` 和 `SelfAssessedHealthStatus` 转换为分类数组。

```
Gender = categorical(Gender);
SelfAssessedHealthStatus = categorical(SelfAssessedHealthStatus);
whos
```

Name	Size	Bytes	Class	Attributes
Age	100x1	800	double	
Diastolic	100x1	800	double	
Gender	100x1	346	categorical	
Height	100x1	800	double	
LastName	100x1	12416	cell	
Location	100x1	15008	cell	
SelfAssessedHealthStatus	100x1	592	categorical	
Smoker	100x1	100	logical	
Systolic	100x1	800	double	
Weight	100x1	800	double	

### 计算平均体重

使用 `Smoker` 变量将患者划分为非吸烟者和吸烟者。计算每个组的平均体重。

```
[G,smoker] = findgroups(Smoker);
meanWeight = splitapply(@mean,Weight,G)
```

```
meanWeight = 2×1
```

```
149.9091
161.9412
```

`findgroups` 函数会返回 `G`（从 `Smoker` 创建的组数目向量）。`splitapply` 函数会使用 `G` 将 `Weight` 分为两个组。`splitapply` 会将 `mean` 函数应用于每个组并将平均体重串联到向量中。

`findgroups` 会返回组标识符向量作为第二个输出参数。组标识符是逻辑值，因为 `Smoker` 包含逻辑值。第一组中的患者是非吸烟者，第二组中的患者是吸烟者。

```
smoker
```

```
smoker = 2x1 logical array
```

```
0
1
```

按吸烟者的性别和状态划分患者体重，并计算平均体重。

```
G = findgroups(Gender,Smoker);
meanWeight = splitapply(@mean,Weight,G)

meanWeight = 4×1

130.3250
130.9231
180.0385
181.1429
```

跨 Gender 和 Smoker 的唯一组合可确定四组患者：女性非吸烟者、女性吸烟者、男性非吸烟者和男性吸烟者。将这四个组及其平均体重汇总在一个表中。

```
[G,gender,smoker] = findgroups(Gender,Smoker);
T = table(gender,smoker,meanWeight)
```

```
T=4×3 table
    gender    smoker    meanWeight
    _____
    Female    false      130.32
    Female    true       130.92
    Male     false      180.04
    Male     true       181.14
```

T.gender 包含分类值，而 T.smoker 包含逻辑值。这些表变量的数据类型分别与 Gender 和 Smoker 的数据类型一致。

为四组患者计算体重指数 (BMI)。定义一个函数，该函数采用 Height 和 Weight 作为其两个输入参数，并计算 BMI。

```
meanBMIfcn = @(h,w)mean((w ./ (h.^2)) * 703);
BMI = splitapply(meanBMIfcn,Height,Weight,G)

BMI = 4×1

21.6721
21.6686
26.5775
26.4584
```

### 根据各自的报告对患者分组

计算将其健康状态报告为 Poor 或 Fair 的患者百分比。首先，使用 `splitapply` 统计每个组中的患者数：女性非吸烟者、女性吸烟者、男性非吸烟者和男性吸烟者。然后，使用 S 和 G 上的逻辑索引，仅统计其健康状况报告为 Poor 或 Fair 的那些患者。根据这两组计数，计算每个组的百分比。

```
[G,gender,smoker] = findgroups(Gender,Smoker);
S = SelfAssessedHealthStatus;
I = ismember(S,{'Poor','Fair'});
numPatients = splitapply(@numel,S,G);
numPF = splitapply(@numel,S(I),G(I));
numPF./numPatients

ans = 4×1
```

0.2500  
0.3846  
0.3077  
0.1429

比较健康状况报告为 Poor 或 Fair 的那些患者在 Diastolic 读数中的标准差, 以及健康状况报告为 Good 或 Excellent 的那些患者的相应标准差。

```
stdDiastolicPF = splitapply(@std,Diastolic(I),G(I));
stdDiastolicGE = splitapply(@std,Diastolic(~I),G(~I));
```

在表中收集结果。在这些患者中, 健康状况报告为 Poor 或 Fair 的女性非吸烟者的血压读数差别最大。

```
T = table(gender,smoker,numPatients,numPF,stdDiastolicPF,stdDiastolicGE,BMI)
```

T=4×7 table

gender	smoker	numPatients	numPF	stdDiastolicPF	stdDiastolicGE	BMI
Female	false	40	10	6.8872	3.9012	21.672
Female	true	13	5	5.4129	5.0409	21.669
Male	false	26	8	4.2678	4.8159	26.578
Male	true	21	3	5.6862	5.258	26.458

## 另请参阅

[findgroups](#) | [splitapply](#)

## 相关示例

- “拆分表数据变量并应用函数” (第 9-51 页)

## 详细信息

- “对变量分组以拆分数据” (第 9-60 页)

# 拆分表数据变量并应用函数

此示例演示如何将表中的电力中断数据按电力中断的区域和原因拆分为不同的组。然后，它会显示如何应用函数以计算每个组的统计量并将结果收集到一个表中。

## 加载电力中断数据

示例文件 `outages.csv` 包含表示美国电力中断的数据。该文件包含六个列：`Region`、`OutageTime`、`Loss`、`Customers`、`RestorationTime` 和 `Cause`。将 `outages.csv` 读入表中。

```
T = readtable('outages.csv');
```

将 `Region` 和 `Cause` 转换为分类数组，将 `OutageTime` 和 `RestorationTime` 转换为 `datetime` 数组。显示前五行。

```
T.Region = categorical(T.Region);
T.Cause = categorical(T.Cause);
T.OutageTime = datetime(T.OutageTime);
T.RestorationTime = datetime(T.RestorationTime);
T(1:5,:)
```

```
ans=5×6 table
  Region      OutageTime      Loss    Customers    RestorationTime      Cause
  _____      _____       _____      _____      _____       _____
  SouthWest  2002-02-01 12:18  458.98  1.8202e+06  2002-02-07 16:50  winter storm
  SouthEast   2003-01-23 00:49  530.14  2.1204e+05      NaT      winter storm
  SouthEast   2003-02-07 21:15  289.4   1.4294e+05  2003-02-17 08:14  winter storm
  West        2004-04-06 05:44  434.81  3.4037e+05  2004-04-06 06:10  equipment fault
  MidWest     2002-03-16 06:18  186.44  2.1275e+05  2002-03-18 23:23  severe storm
```

## 计算最大电力损失

确定每个地区因电力中断而造成的大电力损失。`findgroups` 函数会返回 `G`（从 `T.Region` 创建的组数目向量）。`splitapply` 函数会使用 `G` 将 `T.Loss` 分为五个组，对应五个区域。`splitapply` 会将 `max` 函数应用于每个组并将最大电力损失串联到向量中。

```
G = findgroups(T.Region);
maxLoss = splitapply(@max,T.Loss,G)
```

```
maxLoss = 5×1
104 ×
```

```
2.3141
2.3418
0.8767
0.2796
1.6659
```

计算因不同原因导致的电力中断而造成的大电力损失。要指定 `Cause` 为分组变量，请使用表索引。创建一个包含最大电力损失及其原因的表。

```
T1 = T(:, 'Cause');
[G, powerLosses] = findgroups(T1);
powerLosses.maxLoss = splitapply(@max,T.Loss,G)
```

```
powerLosses=10×2 table
  Cause      maxLoss
  _____
  attack      582.63
  earthquake   258.18
  energy emergency 11638
  equipment fault 16659
  fire         872.96
  severe storm 8767.3
  thunder storm 23418
  unknown       23141
  wind          2796
  winter storm 2883.7
```

**powerLosses** 是一个表，因为 **T1** 是一个表。您可以将最大损失附加为另一个表变量。

计算每个地区因不同原因而造成的大电力损失。要指定 **Region** 和 **Cause** 为分组变量，请使用表索引。创建一个包含最大电力损失的表，并显示前 15 行。

```
T1 = T(:,{'Region','Cause'});
[G,powerLosses] = findgroups(T1);
powerLosses.maxLoss = splitapply(@max,T.Loss,G);
powerLosses(1:15,:)
```

```
ans=15×3 table
  Region      Cause      maxLoss
  _____
  MidWest    attack      0
  MidWest    energy emergency 2378.7
  MidWest    equipment fault 903.28
  MidWest    severe storm 6808.7
  MidWest    thunder storm 15128
  MidWest    unknown       23141
  MidWest    wind          2053.8
  MidWest    winter storm 669.25
  NorthEast  attack      405.62
  NorthEast  earthquake   0
  NorthEast  energy emergency 11638
  NorthEast  equipment fault 794.36
  NorthEast  fire         872.96
  NorthEast  severe storm 6002.4
  NorthEast  thunder storm 23418
```

### 计算受影响客户的数量

确定不同原因和地区的电力中断对客户的影响。因为 **T.Loss** 包含 **Nan** 值，所以将 **sum** 打包在匿名函数中以使用 '**omitnan**' 输入参数。

```
osumFcn = @(x)(sum(x,'omitnan'));
powerLosses.totalCustomers = splitapply(osumFcn,T.Customers,G);
powerLosses(1:15,:)
```

```
ans=15×4 table
  Region      Cause      maxLoss  totalCustomers
  _____
```

MidWest	attack	0	0
MidWest	energy emergency	2378.7	6.3363e+05
MidWest	equipment fault	903.28	1.7822e+05
MidWest	severe storm	6808.7	1.3511e+07
MidWest	thunder storm	15128	4.2563e+06
MidWest	unknown	23141	3.9505e+06
MidWest	wind	2053.8	1.8796e+06
MidWest	winter storm	669.25	4.8887e+06
NorthEast	attack	405.62	2181.8
NorthEast	earthquake	0	0
NorthEast	energy emergency	11638	1.4391e+05
NorthEast	equipment fault	794.36	3.9961e+05
NorthEast	fire	872.96	6.1292e+05
NorthEast	severe storm	6002.4	2.7905e+07
NorthEast	thunder storm	23418	2.1885e+07

### 计算电力中断的平均持续时间

确定美国的所有电力中断的平均持续时间（以小时为单位）。将电力中断平均持续时间添加到 powerLosses。因为 T.RestorationTime 具有 NaT 值，所以在计算平均持续时间时请忽略生成的 NaN 值。

```
D = T.RestorationTime - T.OutageTime;
H = hours(D);
omeanFcn = @(x)(mean(x,'omitnan'));
powerLosses.meanOutage = splitapply(omeanFcn,H,G);
powerLosses(1:15,:)
```

ans=15×5 table

Region	Cause	maxLoss	totalCustomers	meanOutage
MidWest	attack	0	335.02	
MidWest	energy emergency	2378.7	6.3363e+05	5339.3
MidWest	equipment fault	903.28	1.7822e+05	17.863
MidWest	severe storm	6808.7	1.3511e+07	78.906
MidWest	thunder storm	15128	4.2563e+06	51.245
MidWest	unknown	23141	3.9505e+06	30.892
MidWest	wind	2053.8	1.8796e+06	73.761
MidWest	winter storm	669.25	4.8887e+06	127.58
NorthEast	attack	405.62	2181.8	5.5117
NorthEast	earthquake	0	0	0
NorthEast	energy emergency	11638	1.4391e+05	77.345
NorthEast	equipment fault	794.36	3.9961e+05	87.204
NorthEast	fire	872.96	6.1292e+05	4.0267
NorthEast	severe storm	6002.4	2.7905e+07	2163.5
NorthEast	thunder storm	23418	2.1885e+07	46.098

### 另请参阅

[findgroups](#) | [rowfun](#) | [splitapply](#) | [varfun](#)

## **相关示例**

- “访问表中的数据” (第 9-33 页)
- “对表执行计算” (第 9-45 页)
- “将数据拆分为不同组并计算统计量” (第 9-48 页)

## **详细信息**

- “对变量分组以拆分数据” (第 9-60 页)

# 使用表的好处

## 方便将混合类型的数据存储于单个容器中

您可以使用 `table` 数据类型来将混合类型的数据和元数据属性（例如变量名称、行名称、说明和变量单位）收集到单个容器中。表适用于列向数据或表格数据，这些数据通常以列形式存储于文本文件或电子表格中。例如，您可以使用表存储试验数据，使用行表示不同的观测对象，使用列表示不同的测量变量。

表由若干行向变量和若干列向变量组成。表中的每个变量可以具有不同的数据类型和不同的大小，但每个变量必须具有相同的行数。

例如，加载样本患者数据。

```
load patients
```

然后，将工作区变量 `Systolic` 和 `Diastolic` 合并为单个 `BloodPressure` 变量，并将工作区变量 `Gender` 从字符串元胞数组转换为分类数组。

```
BloodPressure = [Systolic Diastolic];
Gender = categorical(Gender);
```

```
whos('Gender','Age','Smoker','BloodPressure')
```

Name	Size	Bytes	Class	Attributes
Age	100x1	800	double	
BloodPressure	100x2	1600	double	
Gender	100x1	346	categorical	
Smoker	100x1	100	logical	

变量 `Age`、`BloodPressure`、`Gender` 和 `Smoker` 具有不同的数据类型，是要存储于表中的候选对象，因为它们都具有相同的行数 100。

现在，基于这些变量创建一个表并显示前五行。

```
T = table(Gender,Age,Smoker,BloodPressure);
T(1:5,:)
```

```
ans=5×4 table
  Gender    Age    Smoker    BloodPressure
  _____
  Male      38    true     124    93
  Male      43    false    109    77
  Female    38    false    125    83
  Female    40    false    117    75
  Female    49    false    122    80
```

该表以表格形式显示，并且变量名称位于顶部。

表中的每个变量都是一个单独的数据类型。如果向表中添加一个新行，MATLAB® 将强制使新数据与对应的表变量的数据类型保持一致。例如，如果您尝试添加新患者的信息，其中第一列包含患者的年龄而不是性别（如表达式 `T(end+1,:) = {37,'Female'},true,[130 84]` 中一样），则会出现错误：

```
Invalid RHS for assignment to a categorical array.
```

出现错误的原因是 MATLAB® 不能将数值数据 37 赋给分类数组 **Gender**。

为了便于比较表和结构体，假设一个结构体数组 **StructArray**，它等效于表 T。

```
StructArray = table2struct(T)
```

```
StructArray=100×4 struct
    Gender
    Age
    Smoker
    BloodPressure
```

结构体数组使用命名字段组织记录。每个字段的值都具有不同的数据类型或大小。现在，显示 **StructArray** 的第一个元素的命名字段。

```
StructArray(1)
```

```
ans = struct with fields:
    Gender: Male
        Age: 38
    Smoker: 1
    BloodPressure: [124 93]
```

结构体数组中的字段与表中的变量相似。但是，与表不同的是，结构体数组不能在字段中强制实施同质化要求。例如，**S.Gender** 的一些值可能为分类数组元素 **Male** 或 **Female**，另一些值可能为字符串向量 '**Male**' 或 '**Female**'，其他一些值可能为整数 0 或 1。

现在假设在一个标量结构体中存储的相同数据，四个字段，每一个包含表中的一个变量。

```
ScalarStruct = struct(
    'Gender',{Gender},...
    'Age',Age,...
    'Smoker',Smoker,...
    'BloodPressure',BloodPressure)

ScalarStruct = struct with fields:
    Gender: [100x1 categorical]
        Age: [100x1 double]
    Smoker: [100x1 logical]
    BloodPressure: [100x2 double]
```

与表不同，您不能强制使这些数据保持矩形结构。例如，字段 **ScalarStruct.Age** 的长度可能不同于其他字段。

使用表可以保持矩形结构体（类似于结构体数组）并强制使变量保持同质（类似于标量结构体中的字段）。尽管元胞数组没有命名字段，但它们与结构体数组和标量结构体一样具有许多相同的不足之处。如果每个变量中的矩形数据是同质的，请考虑使用表。然后便可使用数值或命名索引，还可以使用表属性存储元数据。

### 使用数值索引或命名索引访问数据

您可以使用圆括号、花括号或点索引对表进行索引。使用括号可以选择表中的一个数据子集并保留表容器。使用花括号和点索引可以从表中提取数据。在每个表索引方法中，可以通过名称或数值索引指定要访问的行或变量。

考虑上述样本表。表 T 中的每一行表示一位不同的患者。工作区变量 LastName 包含这 100 行数据的唯一标识符。通过将 RowNames 属性设置为 LastName 来向表中添加行名称，并显示更新后的表的前五行。

```
T.Properties.RowNames = LastName;
T(1:5,:)
```

```
ans=5×4 table
    Gender    Age    Smoker    BloodPressure
    _____
    Smith     Male    38    true      124    93
    Johnson   Male    43    false     109    77
    Williams  Female  38    false     125    83
    Jones     Female  40    false     117    75
    Brown     Female  49    false     122    80
```

除了为表添加标签，还可以使用行和变量名称访问表中的数据。例如，使用命名索引显示患者 Williams 和 Brown 的年龄和血压。

```
T({'Williams','Brown'},{'Age','BloodPressure'})
```

```
ans=2×2 table
    Age    BloodPressure
    _____
    Williams  38    125    83
    Brown     49    122    80
```

现在，使用数组索引返回一个等效的子表。返回第二和第四个变量中的第三和第四行。

```
T(3:2:5,2:2:4)
```

```
ans=2×2 table
    Age    BloodPressure
    _____
    Williams  38    125    83
    Brown     49    122    80
```

使用元胞数组或结构体时，不能如此灵活地使用命名索引或数值索引。

- 使用元胞数组时，必须使用 strcmp 查找所需的命名数据，然后才可以对数组进行索引。
- 使用标量结构体或结构体数组时，不能按编号引用字段。此外，使用标量结构体时，不能很轻松地选择变量子集或观测值子集。使用结构体数组时，可以选择观测值子集，但不能选择变量子集。
- 使用表时，可以根据命名索引或数值索引访问数据。此外，可以轻松地选择变量子集和行子集。

有关表索引的详细信息，请参阅“访问表中的数据”（第 9-33 页）。

### 使用表属性存储元数据

除了存储数据以外，表还具有可用来存储元数据的属性，例如变量名称、行名称、说明和变量单位。您可以使用 T.Properties.PropName 访问属性，其中 T 为表名称，PropName 为其中一个表属性。

例如，为 Age 添加表说明、变量说明和变量单位。

```
T.Properties.Description = 'Simulated Patient Data';
```

```
T.Properties.VariableDescriptions = ...
```

```
{'Male or Female' ...
```

```
"..."
```

```
'true or false' ...
```

```
'Systolic/Diastolic'};
```

```
T.Properties.VariableUnits{'Age'} = 'Yrs';
```

**VariableDescriptions** 的元胞数组中的各个空字符向量指示对应的变量没有说明。有关详细信息，请参阅 **table** 的“属性”部分。

要打印表摘要，请使用 **summary** 函数。

```
summary(T)
```

Description: Simulated Patient Data

Variables:

Gender: 100x1 categorical

Properties:

Description: Male or Female

Values:

Female	53
Male	47

Age: 100x1 double

Properties:

Units: Yrs

Values:

Min	25
Median	39
Max	50

Smoker: 100x1 logical

Properties:

Description: true or false

Values:

True	34
False	66

BloodPressure: 100x2 double

Properties:

Description: Systolic/Diastolic

Values:

BloodPressure_1	BloodPressure_2
-----------------	-----------------

Min	109	68
-----	-----	----

Median	122	81.5
Max	138	99

结构体和元胞数组没有用于存储元数据的属性。

## 另请参阅

[summary](#) | [table](#)

## 相关示例

- “[创建和使用表](#)” (第 9-2 页)
- “[修改单位、说明和表变量名称](#)” (第 9-25 页)
- “[访问表中的数据](#)” (第 9-33 页)

## 对变量分组以拆分数据

您可以使用分组变量将数据变量拆分为不同的组。通常，选择分组变量是拆分-应用-合并工作流的第一步。您可以将数据拆分为不同的组，对每个组应用一个函数，然后合并结果。您还可以在分组变量中表示缺失值，以便忽略数据变量中的对应值。

### 分组变量

分组变量是用于进行分组或分类的变量或观测值（即其他变量中的数据值）。分组变量可以是以下任意一种数据类型：

- 数值、逻辑值、分类、**datetime** 或 **duration** 向量
- 字符向量元胞数组
- 表，包含此列表中任何数据类型的表变量

数据变量是包含观测值的变量。分组变量必须具有与数据变量中的各个值对应的值。当分组变量中的对应值相同时，数据值属于同一个组。

下表显示了数据变量、分组变量以及您在使用分组变量拆分数据变量时可创建的组的示例。

数据变量	分组变量	数据组
[5 10 15 20 25 30]	[0 0 0 1 1]	[5 10 15 20] [25 30]
[10 20 30 40 50 60]	[1 3 3 1 2 1]	[10 40 60] [50] [20 30]
[64 72 67 69 64 68]	{'F','M','F','M','F','F'}	[64 67 64 68] [72 69]

当您使用字符向量元胞数组或分类数组作为分组变量时，可以为数据组赋予有意义的名称。如果要对变量进行分组，分类数组不失为一种高效且灵活的选择。

### 组定义

通常，组数和分组变量中的唯一值个数一样多。（分类数组还可包含数据中未表示的类别。）组和组的顺序取决于分组变量的数据类型。

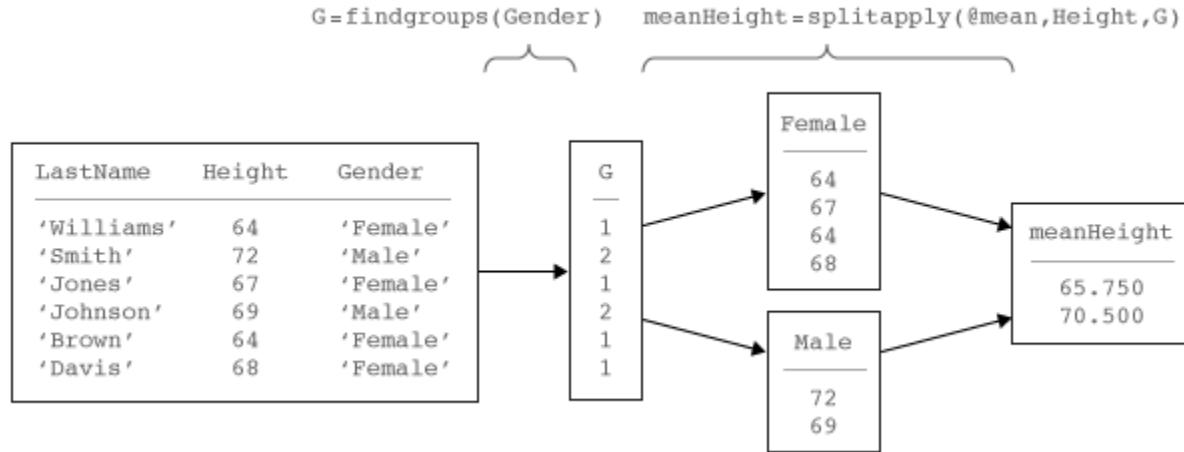
- 对于数值、逻辑值、**datetime** 或 **duration** 向量或字符向量元胞数组，这些组对应于按升序排序的唯一值。
- 对于分类数组，这些组对应于相应数组中观测到的唯一值，并按 **categories** 函数返回的顺序进行排序。

**findgroups** 函数可接受多个分组变量，例如 **G = findgroups(A1,A2)**。您还可以在表中包含多个分组变量，例如 **T = table(A1,A2); G = findgroups(T)**。**findgroups** 函数会按分组变量的相应元素中的唯一值组合定义组。**findgroups** 会首先按第一个分组变量的顺序决定顺序，然后按第二个分组变量的顺序，以此类推。例如，如果 **A1 = {'a','a','b','b'}** 和 **A2 = [0 1 0 0]**，则分组变量中的唯一值为 'a' 0、'a' 1 和 'b' 0（分别定义三个组）。

### 拆分-应用-合并工作流

选择分组变量并将数据变量拆分为多个组后，可以对这些组应用函数并合并结果。此工作流称为拆分-应用-合并工作流。您可以结合使用 **findgroups** 和 **splitapply** 函数来分析此工作流中的数据组。下图显示了使用分组变量 **Gender** 和数据变量 **Height** 计算不同性别人群的平均身高的简单示例。

**findgroups** 函数会返回根据分组变量中的唯一值定义组的组编号向量。**splitapply** 会在应用函数之前有效使用组编号将数据分为不同组。



## 缺少组值

分组变量可能会缺失值。下表显示了每种数据类型缺失的值指示符。如果分组变量有缺失值，则 **findgroups** 会指定 NaN 作为组成员，并且 **splitapply** 会忽略数据变量中的相应值。

分组变量数据类型	缺失值指示符
数值	NaN
逻辑值	(不可缺少)
分类	<undefined>
datetime	NaT
duration	NaN
字符串向量元胞数组	"
字符串	<missing>

## 另请参阅

[findgroups](#) | [rowfun](#) | [splitapply](#) | [varfun](#)

## 相关示例

- “访问表中的数据”（第 9-33 页）
- “拆分表数据变量并应用函数”（第 9-51 页）
- “将数据拆分为不同组并计算统计量”（第 9-48 页）

## R2016b 中对 DimensionNames 属性的更改

**table** 数据类型适用于将列向异构数据收集到单个容器中。表还包含元数据属性，例如变量名称、行名称、维度名称、说明和变量单位。从 R2016b 开始，您可以使用维度名称通过圆点下标来访问表数据和元数据。要支持该功能，维度名称必须满足与变量名称相同的要求。为实现向后兼容，表会在需要时通过自动修改维度名称来强制执行这些限制。

创建一个包含行名称和变量名称的表。

```
Number = [8; 21; 13; 20; 11];
Name = {'Van Buren'; 'Arthur'; 'Fillmore'; 'Garfield'; 'Polk'};
Party = categorical({'Democratic'; 'Republican'; 'Whig'; 'Republican'; 'Republican'});
T = table(Number, Party, 'RowNames', Name)
```

T =

	Number	Party
Van Buren	8	Democratic
Arthur	21	Republican
Fillmore	13	Whig
Garfield	20	Republican
Polk	11	Republican

显示其属性，包括维度名称。维度名称的默认值为 'Row' 和 'Variables'。

### T.Properties

ans =

struct with fields:

```
Description: ''
UserData: []
DimensionNames: {'Row' 'Variables'}
VariableNames: {'Number' 'Party'}
VariableDescriptions: {}
VariableUnits: {}
RowNames: {5×1 cell}
```

从 R2016b 开始，您可以为维度名称分配新名称，并使用它们访问表数据。维度名称必须是有效的 MATLAB 标识符，并且不能是保留名称 'Properties'、'RowNames' 或 'VariableNames' 之一。

为第一个维度名称分配一个新名称，并使用它访问表的行名称。

```
T.Properties.DimensionNames{1} = 'Name';
T.Name
```

ans =

```
5×1 cell array

'Van Buren'
'Arthur'
'Fillmore'
'Garfield'
'Polk'
```

新建一个称为 Name 的表变量。当您创建该变量时，表会修改其第一个维度名称以防止冲突。更新后的维度名称将变为 Name\_1。

```
T{:,'Name'} = {'Martin';'Chester';'Millard';'James';'James'}
```

Warning: DimensionNames property was modified to avoid conflicting dimension and variable names: 'Name'. See Compatibility Considerations for Using Tables for more details. This will become an error in a future release.

```
T =
```

	Number	Party	Name
Van Buren	8	Democratic	'Martin'
Arthur	21	Republican	'Chester'
Fillmore	13	Whig	'Millard'
Garfield	20	Republican	'James'
Polk	11	Republican	'James'

### T.Properties.DimensionNames

```
ans =
```

1×2 cell array

```
'Name_1'  'Data'
```

同样地，如果您分配的维度名称不是有效的 MATLAB 标识符，该名称也会被修改。

```
T.Properties.DimensionNames{1} = 'Last Name';
```

### T.Properties.DimensionNames

Warning: DimensionNames property was modified to make the name 'Last Name' a valid MATLAB identifier. See Compatibility Considerations for Using Tables for more details. This will become an error in a future release.

```
ans =
```

1×2 cell array

```
'LastName'  'Data'
```

在 R2016b 中，当维度名称不是有效的标识符或者与变量名称或保留名称冲突时，表将会发出警告，您可以继续处理用早期版本创建的代码和表。如果您遇到这些警告，建议您对代码进行更新以避免出现这些警告。



# 时间表

---

- “创建时间表” (第 10-2 页)
- “对时间表中的数据进行重采样和聚合” (第 10-5 页)
- “合并时间表并同步其数据” (第 10-8 页)
- “使用不同的方法对时间表变量重设时间并进行同步” (第 10-14 页)
- “按行时间和变量类型选择时间表数据” (第 10-18 页)
- “清理包含缺失、重复或不均匀时间的时间表” (第 10-23 页)
- “在表和时间表运算中使用行标签” (第 10-31 页)
- “洛马普列塔地震分析” (第 10-36 页)
- “使用 timetable 预处理和探索时间戳数据” (第 10-46 页)

## 创建时间表

以下示例演示如何创建时间表、合并时间表以及将多个时间表中的数据调整到一个公共时间向量中。公共时间向量可以包含其中一个时间表或两个时间表中的时间，也可以是一个您指定的全新时间向量。以下示例演示如何计算和显示不同时间表中包含的天气测量值的日均值。

时间表是一种表类型，用于将时间与每一行进行关联。时间表可以存储具有不同数据类型和大小的列向数据变量，只要每个变量的行数相同。此外，时间表还提供了特定于时间的函数，以合并数据、为数据建立下标以及调整数据。

### 从文件导入时间表

将空气质量数据和天气测量值加载到两个不同的时间表中。测量值的日期范围从 2015 年 11 月 15 日到 2015 年 11 月 19 日。空气质量数据来自建筑物内部的传感器，而天气测量值来自外部传感器。

使用 `readtable` 函数读取表中的空气质量数据。然后，使用 `table2timetable` 函数将其从表转换为时间表。`readtable` 函数仅返回表，而不是时间表。

```
indoors = readtable('indoors.csv');
indoors = table2timetable(indoors);
```

您也可以使用 `array2timetable` 函数根据  $M \times N$  数组来创建时间表，或者使用 `timetable` 函数根据工作区变量来创建时间表。

显示 `indoors` 的前五行。此时间表的每一行都带有一个标记该行数据的时间。

```
indoors(1:5,:)

ans=5×3 timetable
    Time        Humidity    AirQuality
    _____    _____    _____
    2015-11-15 00:00:24    36      80
    2015-11-15 01:13:35    36      80
    2015-11-15 02:26:47    37      79
    2015-11-15 03:39:59    37      82
    2015-11-15 04:53:11    36      80
```

加载包含天气测量值的时间表。显示 `outdoors` 的前五行。

```
load outdoors
outdoors(1:5,:)

ans=5×4 timetable
    Time        Humidity    TemperatureF    PressureHg
    _____    _____    _____    _____
    2015-11-15 00:00:24    49      51.3      29.61
    2015-11-15 01:30:24    48.9     51.5      29.61
    2015-11-15 03:00:24    48.9     51.5      29.61
    2015-11-15 04:30:24    48.8     51.5      29.61
    2015-11-15 06:00:24    48.7     51.5      29.6
```

## 同步时间表

时间表 `indoors` 和 `outdoors` 包含不同时间在建筑物内部和外部获取的不同测量值。使用 `synchronize` 函数将所有数据都合并到一个时间表中。

```
tt = synchronize(indoors,outdoors);
tt(1:5,:)
```

Time	Humidity_indoors	AirQuality	Humidity_outdoors	TemperatureF	PressureHg
2015-11-15 00:00:24	36	80	49	51.3	29.61
2015-11-15 01:13:35	36	80	NaN	NaN	NaN
2015-11-15 01:30:24	NaN	NaN	48.9	51.5	29.61
2015-11-15 02:26:47	37	79	NaN	NaN	NaN
2015-11-15 03:00:24	NaN	NaN	48.9	51.5	29.61

输出时间表 `tt` 包含两个时间表的所有时间。`synchronize` 会在 `tt` 中未放置数据值的位置放置一个缺失数据指示符。如果两个输入时间表包含一个同名变量（如 `Humidity`），`synchronize` 将会重命名这两个变量并将二者都添加到输出时间表。

再次同步这些时间表，这次将会通过线性插值来填充缺失的数据值。

```
ttLinear = synchronize(indoors,outdoors,'union','linear');
ttLinear(1:5,:)
```

Time	Humidity_indoors	AirQuality	Humidity_outdoors	TemperatureF	PressureHg
2015-11-15 00:00:24	36	80	49	51.3	29.61
2015-11-15 01:13:35	36	80	48.919	51.463	29.61
2015-11-15 01:30:24	36.23	79.77	48.9	51.5	29.61
2015-11-15 02:26:47	37	79	48.9	51.5	29.61
2015-11-15 03:00:24	37	80.378	48.9	51.5	29.61

## 调整一个时间表中的数据

您也可以将单个时间表中的数据调整到新的时间向量中。使用 `retime` 函数以 6 小时为时间间隔计算 `ttLinear` 中的变量的均值。如果在您调整数据后有任何行包含 `NaN` 值，请使用 `rmmissing` 函数将这些值删除。

```
tv = [datetime(2015,11,15):hours(6):datetime(2015,11,18)];
ttHourly = retime(ttLinear, tv, 'mean');
ttHourly = rmmissing(ttHourly);
```

## 绘制时间表数据图

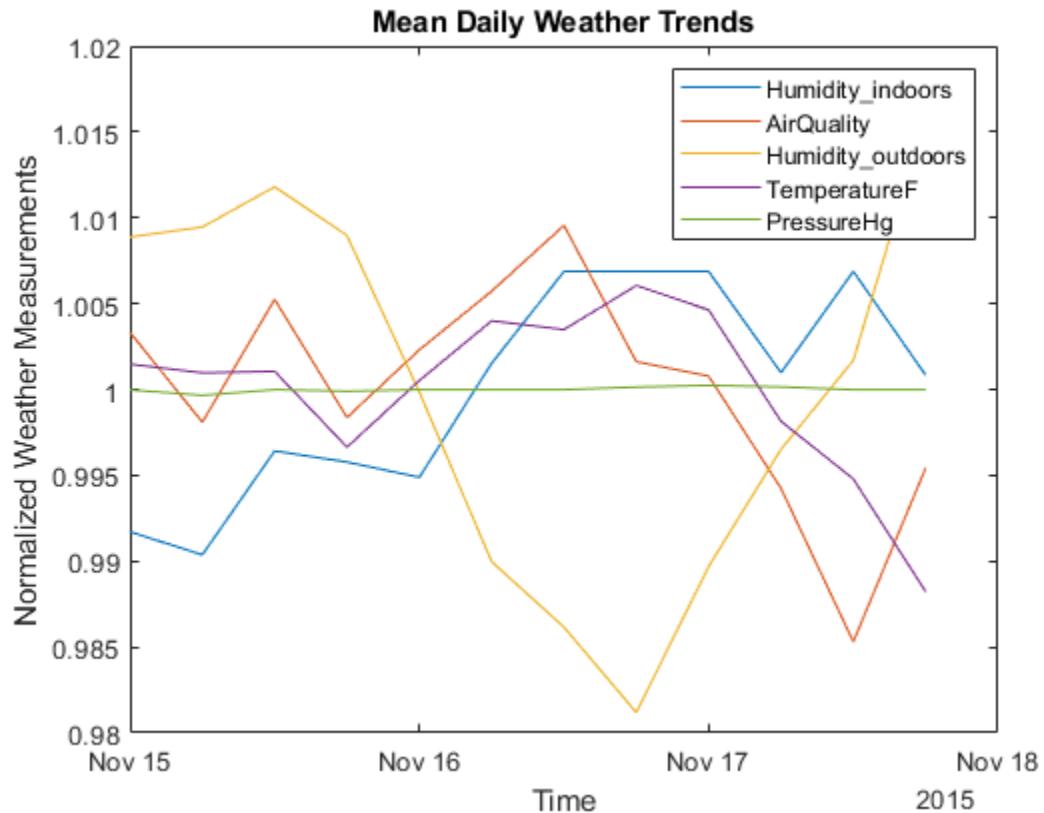
根据时间表中每个变量的均值对 `ttHourly` 中的数据进行归一化。绘制这些测量值的日均值图。您可以使用时间表的 `Variables` 属性来访问这些变量。`ttHourly.Variables` 会返回与 `ttHourly{:, :}` 相同的变量。

```
ttMeanVars = ttHourly.Variables./mean(ttHourly.Variables);
plot(ttHourly.Time,ttMeanVars);
legend(ttHourly.Properties.VariableNames,'Interpreter','none');
```

```

xlabel('Time');
ylabel('Normalized Weather Measurements');
title('Mean Daily Weather Trends');

```



## 另请参阅

[retime](#) | [rmmissing](#) | [synchronize](#) | [table2timetable](#) | [timerange](#) | [timetable](#)

## 相关示例

- “对时间表中的数据进行重采样和聚合”（第 10-5 页）
- “合并时间表并同步其数据”（第 10-8 页）
- “使用不同的方法对时间表变量重设时间并进行同步”（第 10-14 页）
- “按行时间和变量类型选择时间表数据”（第 10-18 页）
- “清理包含缺失、重复或不均匀时间的时间表”（第 10-23 页）

## 对时间表中的数据进行重采样和聚合

以下示例演示如何对时间表中的数据进行重采样和聚合。时间表是一种表类型，用于将时间与每一行进行关联。时间表可以存储具有不同数据类型和大小的列向数据变量，前提是每个变量的行数相同。通过 `retime` 函数，您可以对时间表数据进行重采样，或将时间表数据聚合到指定的时间 bin 中。

### 导入时间表

加载包含从 2015 年 11 月 15 日到 2015 年 11 月 19 日获取的天气测量值的时间表。该时间表包含在这段时间获取的湿度、温度和压力读数。

```
load outdoors
outdoors(1:5,:)

ans=5×4 timetable
Time      Humidity  TemperatureF  PressureHg
_____
2015-11-15 00:00:24    49       51.3       29.61
2015-11-15 01:30:24   48.9      51.5       29.61
2015-11-15 03:00:24   48.9      51.5       29.61
2015-11-15 04:30:24   48.8      51.5       29.61
2015-11-15 06:00:24   48.7      51.5       29.6
```

确定该时间表是否为规则时间表。在规则时间表中，所有连续的行时间之间的差分均相同。`outdoors` 不是规则时间表。

```
TF = isregular(outdoors)
```

```
TF = logical
0
```

求时间步的差分。差分在半分钟到一个半小时之间变化。

```
dt = unique(diff(outdoors.Time))
```

```
dt = 3x1 duration array
00:00:24
01:29:36
01:30:00
```

### 通过插值对时间表重采样

使用 `retime` 函数调整时间表中的数据。指定一个按小时计的时间向量。将时间表数据插入到新的行时间中。

```
TT = retime(outdoors,'hourly','spline');
TT(1:5,:)
```

```
ans=5×4 timetable
Time      Humidity  TemperatureF  PressureHg
_____
2015-11-15 00:00:00    49.001     51.298      29.61
2015-11-15 01:00:00    48.909     51.467      29.61
```

```
2015-11-15 02:00:00  48.902    51.51    29.61
2015-11-15 03:00:00  48.9       51.5     29.61
2015-11-15 04:00:00  48.844    51.498   29.611
```

### 通过最近邻值对时间表进行重采样

为 **TT** 指定一个按小时计的时间向量。对于 **TT** 中的每一行，复制 **outdoors** 中行时间最近的对应行中的值。

```
TT = retime(outdoors,'hourly','nearest');
TT(1:5,:)
```

```
ans=5×4 timetable
Time      Humidity  TemperatureF  PressureHg
_____
2015-11-15 00:00:00  49       51.3       29.61
2015-11-15 01:00:00  48.9     51.5       29.61
2015-11-15 02:00:00  48.9     51.5       29.61
2015-11-15 03:00:00  48.9     51.5       29.61
2015-11-15 04:00:00  48.8     51.5       29.61
```

### 聚合时间表数据并计算日均值

**retime** 函数提供了聚合方法，例如 **mean**。计算 **outdoors** 中的数据的日均值。

```
TT = retime(outdoors,'daily','mean');
TT
```

```
TT=4×4 timetable
Time      Humidity  TemperatureF  PressureHg
_____
2015-11-15 00:00:00  48.931    51.394    29.607
2015-11-16 00:00:00  47.924    51.571    29.611
2015-11-17 00:00:00  48.45     51.238    29.613
2015-11-18 00:00:00  49.5      50.8      29.61
```

### 将时间表数据调整为规则时间

计算以 6 小时为时间间隔的均值。使用 '**regular**' 输入参数和 '**TimeStep**' 名称-值对组参数指定一个规则时间步。

```
TT = retime(outdoors,'regular','mean','TimeStep',hours(6));
TT(1:5,:)
```

```
ans=5×4 timetable
Time      Humidity  TemperatureF  PressureHg
_____
2015-11-15 00:00:00  48.9       51.45     29.61
2015-11-15 06:00:00  48.9       51.45     29.6
2015-11-15 12:00:00  49.025    51.45     29.61
2015-11-15 18:00:00  48.9       51.225   29.607
2015-11-16 00:00:00  48.5       51.4      29.61
```

您也可以指定一个以六小时为时间间隔的时间向量。为该时间向量指定格式，以便在显示时间表时显示日期和时间。

```
tv = datetime(2015,11,15):hours(6):datetime(2015,11,18);
tv.Format = 'dd-MMM-yyyy HH:mm:ss';
TT = retime(outdoors, tv, 'mean');
TT(1:5,:)

ans=5×4 timetable
Time      Humidity  TemperatureF  PressureHg
_____
15-Nov-2015 00:00:00    48.9      51.45      29.61
15-Nov-2015 06:00:00    48.9      51.45      29.6
15-Nov-2015 12:00:00    49.025    51.45      29.61
15-Nov-2015 18:00:00    48.9      51.225    29.607
16-Nov-2015 00:00:00    48.5      51.4      29.61
```

## 另请参阅

[retime](#) | [synchronize](#) | [table2timetable](#) | [timetable](#)

## 相关示例

- “[创建时间表](#)”（第 10-2 页）
- “[合并时间表并同步其数据](#)”（第 10-8 页）
- “[使用不同的方法对时间表变量重设时间并进行同步](#)”（第 10-14 页）
- “[按行时间和变量类型选择时间表数据](#)”（第 10-18 页）
- “[清理包含缺失、重复或不均匀时间的时间表](#)”（第 10-23 页）

## 合并时间表并同步其数据

您可以通过各种方式来合并时间表并同步其数据。您可以垂直或水平串联时间表，但前提是时间表中包含相同的行时间或时间表变量。使用 `synchronize` 函数可合并具有不同行时间和时间表变量的时间表。

`synchronize` 会创建一个包含所有输入时间表中的全部变量的时间表。然后，它会将输入时间表中的数据同步到输出时间表的行时间。`synchronize` 可以使用缺失数据指示符、从最近邻行复制的值或插入的值来填充输出时间表的缺失元素。`synchronize` 还可以聚合所指定的时间 bin 上的时间表数据。

### 垂直串联时间表

从 `openPricesSmall` 加载时间表并将它们垂直串联。这些时间表为 `opWeek1` 和 `opWeek2`。它们包含一些股票在 2016 年 1 月的第一周和第二周期间的开盘价。

```
load openPricesSmall
```

显示这两个时间表。

`opWeek1`

opWeek1=5×3 timetable		
Time	AAPL	FB
08-Jan-2016 09:00:00	98.55	99.88
07-Jan-2016 09:00:00	98.68	100.5
06-Jan-2016 09:00:00	100.56	101.13
05-Jan-2016 09:00:00	105.75	102.89
04-Jan-2016 09:00:00	102.61	101.95

`opWeek2`

opWeek2=5×3 timetable		
Time	AAPL	FB
14-Jan-2016 09:00:00	97.96	95.85
13-Jan-2016 09:00:00	100.32	100.58
12-Jan-2016 09:00:00	100.55	99
11-Jan-2016 09:00:00	98.97	97.91
08-Jan-2016 09:00:00	98.55	99.88

串联这些时间表。当时间表具有相同的变量时，您可以垂直串联它们。行时间用于为各行添加标签，未包含在时间表变量中。请注意，时间表的行时间可以不按顺序，并且不必具有固定间隔。例如，`op` 不包含时值周末的日期。时间表还可以包含重复的时间。`op` 包含两行 `08-Jan-2016 09:00:00` 数据。

```
op = [opWeek2;opWeek1]
```

op=10×3 timetable		
Time	AAPL	FB
14-Jan-2016 09:00:00	97.96	95.85
13-Jan-2016 09:00:00	100.32	100.58
12-Jan-2016 09:00:00	100.55	99
11-Jan-2016 09:00:00	98.97	97.91
08-Jan-2016 09:00:00	98.55	99.88
08-Jan-2016 09:00:00	98.55	99.88

08-Jan-2016 09:00:00	98.55	99.88
08-Jan-2016 09:00:00	98.55	99.88
07-Jan-2016 09:00:00	98.68	100.5
06-Jan-2016 09:00:00	100.56	101.13
05-Jan-2016 09:00:00	105.75	102.89
04-Jan-2016 09:00:00	102.61	101.95

## 水平串联时间表

您也可以水平串联时间表。这些时间表必须具有相同的行时间和不同的变量。

显示时间表 `opOtherStocks`。该时间表与 `opWeek1` 具有相同的行时间，但变量表示不同的股票。

### `opOtherStocks`

`opOtherStocks=5×3 timetable`

Time	MSFT	TWTR
------	------	------

08-Jan-2016 09:00:00	52.37	20.51
07-Jan-2016 09:00:00	52.7	21
06-Jan-2016 09:00:00	54.32	21.62
05-Jan-2016 09:00:00	54.93	22.79
04-Jan-2016 09:00:00	54.32	22.64

串联 `opWeek1` 和 `opOtherStocks`。输出时间表包含一组行时间以及两个时间表中的变量。

`op = [opWeek1 opOtherStocks]`

`op=5×5 timetable`

Time	AAPL	FB	MSFT	TWTR
------	------	----	------	------

08-Jan-2016 09:00:00	98.55	99.88	52.37	20.51
07-Jan-2016 09:00:00	98.68	100.5	52.7	21
06-Jan-2016 09:00:00	100.56	101.13	54.32	21.62
05-Jan-2016 09:00:00	105.75	102.89	54.93	22.79
04-Jan-2016 09:00:00	102.61	101.95	54.32	22.64

## 同步时间表并指示缺失数据

从两个不同的时间表加载空气质量数据和天气测量值，并将这两个时间表进行同步。测量值的日期范围从 2015 年 11 月 15 日到 2015 年 11 月 19 日。空气质量数据来自建筑物内部的传感器，而天气测量值来自外部传感器。

```
load indoors
load outdoors
```

显示每个时间表的前五行。它们包含在不同时间获取的具有不同数量的测量值。

`indoors(1:5,:)`

`ans=5×3 timetable`

Time	Humidity	AirQuality
------	----------	------------

```
2015-11-15 00:00:24    36     80
2015-11-15 01:13:35    36     80
2015-11-15 02:26:47    37     79
2015-11-15 03:39:59    37     82
2015-11-15 04:53:11    36     80
```

**outdoors(1:5,:)****ans=5×4 timetable**

Time	Humidity	TemperatureF	PressureHg
2015-11-15 00:00:24	49	51.3	29.61
2015-11-15 01:30:24	48.9	51.5	29.61
2015-11-15 03:00:24	48.9	51.5	29.61
2015-11-15 04:30:24	48.8	51.5	29.61
2015-11-15 06:00:24	48.7	51.5	29.6

同步这些时间表。输出时间表 **tt** 包含两个时间表的所有时间。**synchronize** 会在 **tt** 中未放置数据值的位置放置一个缺失数据指示符。如果两个输入时间表包含一个同名变量（如 **Humidity**），**synchronize** 将会重命名这两个变量并将二者都添加到输出时间表。

```
tt = synchronize(indoors,outdoors);
tt(1:5,:)
```

**ans=5×6 timetable**

Time	Humidity_indoors	AirQuality	Humidity_outdoors	TemperatureF	PressureHg
2015-11-15 00:00:24	36	80	49	51.3	29.61
2015-11-15 01:13:35	36	80	NaN	NaN	NaN
2015-11-15 01:30:24	NaN	NaN	48.9	51.5	29.61
2015-11-15 02:26:47	37	79	NaN	NaN	NaN
2015-11-15 03:00:24	NaN	NaN	48.9	51.5	29.61

**同步并插入数据值**

同步时间表并通过线性插值填充缺失的时间表元素。要根据包含两个时间表中的所有时间的时间向量进行同步，请为输出时间指定 '**union**'。

```
ttLinear = synchronize(indoors,outdoors,'union','linear');
ttLinear(1:5,:)
```

**ans=5×6 timetable**

Time	Humidity_indoors	AirQuality	Humidity_outdoors	TemperatureF	PressureHg
2015-11-15 00:00:24	36	80	49	51.3	29.61
2015-11-15 01:13:35	36	80	48.919	51.463	29.61
2015-11-15 01:30:24	36.23	79.77	48.9	51.5	29.61
2015-11-15 02:26:47	37	79	48.9	51.5	29.61
2015-11-15 03:00:24	37	80.378	48.9	51.5	29.61

## 同步到规则时间

将这些时间表同步到一个按小时计的时间向量。输入时间表具有不规则行时间。输出时间表具有规则行时间，时间步为一个小时。

```
ttHourly = synchronize(indoors,outdoors,'hourly','linear');
ttHourly(1:5,:)
```

Time	Humidity_indoors	AirQuality	Humidity_outdoors	TemperatureF	PressureHg
2015-11-15 00:00:00	36	80	49	51.299	29.61
2015-11-15 01:00:00	36	80	48.934	51.432	29.61
2015-11-15 02:00:00	36.634	79.366	48.9	51.5	29.61
2015-11-15 03:00:00	37	80.361	48.9	51.5	29.61
2015-11-15 04:00:00	36.727	81.453	48.834	51.5	29.61

将时间表同步到 30 分钟的时间步。使用 'regular' 输入参数和 'TimeStep' 名称-值对组参数指定一个规则时间步。

```
ttHalfHour = synchronize(indoors,outdoors,'regular','linear','TimeStep',minutes(30));
ttHalfHour(1:5,:)
```

Time	Humidity_indoors	AirQuality	Humidity_outdoors	TemperatureF	PressureHg
2015-11-15 00:00:00	36	80	49	51.299	29.61
2015-11-15 00:30:00	36	80	48.967	51.366	29.61
2015-11-15 01:00:00	36	80	48.934	51.432	29.61
2015-11-15 01:30:00	36.224	79.776	48.9	51.499	29.61
2015-11-15 02:00:00	36.634	79.366	48.9	51.5	29.61

您也可以将这些时间表同步到一个时间间隔指定为半小时的时间向量。

```
tv = [datetime(2015,11,15):minutes(30):datetime(2015,11,18)];
tv.Format = indoors.Time.Format;
ttHalfHour = synchronize(indoors,outdoors,tv,'linear');
ttHalfHour(1:5,:)
```

Time	Humidity_indoors	AirQuality	Humidity_outdoors	TemperatureF	PressureHg
2015-11-15 00:00:00	36	80	49	51.299	29.61
2015-11-15 00:30:00	36	80	48.967	51.366	29.61
2015-11-15 01:00:00	36	80	48.934	51.432	29.61
2015-11-15 01:30:00	36.224	79.776	48.9	51.499	29.61
2015-11-15 02:00:00	36.634	79.366	48.9	51.5	29.61

## 同步并聚合数据值

同步这些时间表并计算输出时间表中的所有变量的日均值。

```
ttDaily = synchronize(indoors,outdoors,'daily','mean');
ttDaily
```

Time	Humidity_indoors	AirQuality	Humidity_outdoors	TemperatureF	PressureHg
2015-11-15 00:00:00	36.5	80.05	48.931	51.394	29.607
2015-11-16 00:00:00	36.85	80.35	47.924	51.571	29.611
2015-11-17 00:00:00	36.85	79.45	48.45	51.238	29.613
2015-11-18 00:00:00	NaN	NaN	49.5	50.8	29.61

将这些时间表同步到以 6 小时为时间间隔，并计算每个间隔的均值。

```
tt6Hours = synchronize(indoors,outdoors,'regular','mean','TimeStep',hours(6));
tt6Hours(1:5,:)
```

Time	Humidity_indoors	AirQuality	Humidity_outdoors	TemperatureF	PressureHg
2015-11-15 00:00:00	36.4	80.2	48.9	51.45	29.61
2015-11-15 06:00:00	36.4	79.8	48.9	51.45	29.6
2015-11-15 12:00:00	36.6	80.4	49.025	51.45	29.61
2015-11-15 18:00:00	36.6	79.8	48.9	51.225	29.607
2015-11-16 00:00:00	36.6	80.2	48.5	51.4	29.61

也可以指定一个以六小时为时间间隔的时间向量。

```
tv = [datetime(2015,11,15):hours(6):datetime(2015,11,18)];
tv.Format = indoors.Time.Format;
tt6Hours = synchronize(indoors,outdoors,tv,'mean');
tt6Hours(1:5,:)
```

Time	Humidity_indoors	AirQuality	Humidity_outdoors	TemperatureF	PressureHg
2015-11-15 00:00:00	36.4	80.2	48.9	51.45	29.61
2015-11-15 06:00:00	36.4	79.8	48.9	51.45	29.6
2015-11-15 12:00:00	36.6	80.4	49.025	51.45	29.61
2015-11-15 18:00:00	36.6	79.8	48.9	51.225	29.607
2015-11-16 00:00:00	36.6	80.2	48.5	51.4	29.61

## 另请参阅

[retime](#) | [synchronize](#) | [table2timetable](#) | [timetable](#)

## 相关示例

- “[创建时间表](#)”（第 10-2 页）
- “[对时间表中的数据进行重采样和聚合](#)”（第 10-5 页）
- “[使用不同的方法对时间表变量重设时间并进行同步](#)”（第 10-14 页）

- “按行时间和变量类型选择时间表数据”（第 10-18 页）
- “清理包含缺失、重复或不均匀时间的时间表”（第 10-23 页）

## 使用不同的方法对时间表变量重设时间并进行同步

此示例说明如何通过对不同的变量使用不同的方法来填充时间表变量的空缺。您可以使用时间表的 **VariableContinuity** 属性，指定各个时间表变量包含的是连续数据还是离散数据。当您使用 **retime** 函数对时间表重采样时，**retime** 会插入值、使用前面的值填充或使用缺失数据指示符填充，具体取决于 **VariableContinuity** 属性中的值。同样，**synchronize** 函数也会根据输入时间表的 **VariableContinuity** 属性插入或填充值。

### 创建时间表

创建一个包含 2017 年 5 月中若干天的模拟天气测量值的时间表。时间表变量 **Tmax** 和 **Tmin** 包含每日最高温度和最低温度的读数，**PrecipTotal** 包含该日的总降水量。**WXEvent** 是一个分类数组，用于记录某个给定日期是否发生了特定类型的天气事件，例如雷电或冰雹。该时间表包含从 2017 年 5 月 4 日到 5 月 10 日的模拟数据，但缺失 5 月 6 日和 5 月 7 日这两天的数据。

```
Date = [datetime(2017,5,4) datetime(2017,5,5) datetime(2017,5,8:10)]';
Tmax = [60 62 56 59 60]';
Tmin = [44 45 40 42 45]';
PrecipTotal = [0.2 0 0 0.15 0]';
WXEvent = [2 0 0 1 0]';
WXEvent = categorical(WXEvent,[0 1 2 3 4],{'None','Thunder','Hail','Fog','Tornado'});
Station1 = timetable(Date,Tmax,Tmin,PrecipTotal,WXEvent)
```

Station1=5×5 timetable

Date	Tmax	Tmin	PrecipTotal	WXEvent
04-May-2017	60	44	0.2	Hail
05-May-2017	62	45	0	None
08-May-2017	56	40	0	None
09-May-2017	59	42	0.15	Thunder
10-May-2017	60	45	0	None

### 对连续和离散的时间表变量重采样

要为缺失的这两天填充数据，可以使用 **retime** 函数。如果您在未指定方法的情况下调用 **retime**，则 **retime** 会使用缺失数据指示符来填充空缺。例如，**retime** 使用 **NaN** 值填充数值变量的空缺，使用 **undefined** 元素填充分类变量的空缺。

```
Station1Daily = retime(Station1,'daily')
```

Station1Daily=7×5 timetable

Date	Tmax	Tmin	PrecipTotal	WXEvent
04-May-2017	60	44	0.2	Hail
05-May-2017	62	45	0	None
06-May-2017	NaN	NaN	NaN	<undefined>
07-May-2017	NaN	NaN	NaN	<undefined>
08-May-2017	56	40	0	None
09-May-2017	59	42	0.15	Thunder
10-May-2017	60	45	0	None

如果您在调用 **retime** 时指定了方法，该函数将使用同一方法来填充每个变量的空缺。要将不同的方法应用于不同的变量，您可以调用 **retime** 多次，每次都对时间表进行索引以访问不同的变量子集。

但是，您也可以通过指定时间表的 **VariableContinuity** 属性来应用不同的方法。您可以指定各个变量是包含连续数据还是离散数据。然后，**retime** 函数将不同的方法应用于各个时间表变量，具体取决于对应的 **VariableContinuity** 值。

如果您指定 **VariableContinuity**，则 **retime** 函数会使用以下方法填充输出时间表变量：

- '**unset**' - 使用该类型的缺失数据指示符（例如对于数值变量使用 **Nan**）来填充值。
- '**continuous**' - 使用线性插值方法填充值。
- '**step**' - 使用上一个值填充值。
- '**event**' - 使用该类型的缺失数据指示符来填充值。

指定 **Station1** 中的温度数据为连续数据 (**continuous**)、**PrecipTotal** 为时间步数据 (**step**)，**WXEvent** 为事件数据 (**event**)。

```
Station1.Properties.VariableContinuity = {'continuous','continuous','step','event'};
Station1.Properties
```

```
ans =
TimetableProperties with properties:

    Description: "
        UserData: []
    DimensionNames: {'Date' 'Variables'}
    VariableNames: {'Tmax' 'Tmin' 'PrecipTotal' 'WXEvent'}
    VariableDescriptions: {}
    VariableUnits: {}
    VariableContinuity: [continuous continuous step event]
        RowTimes: [5x1 datetime]
        StartTime: 04-May-2017
        SampleRate: NaN
        TimeStep: NaN
    CustomProperties: No custom properties are set.
    Use addprop and rmprop to modify CustomProperties.
```

对 **Station1** 中的数据重采样。基于赋给 **VariableContinuity** 的值，**retime** 函数会插入温度数据、在 **PrecipTotal** 中填充前一天的值，并使用 **undefined** 元素填充 **WXEvent**。

```
Station1Daily = retime(Station1,'daily')
```

Date	Tmax	Tmin	PrecipTotal	WXEvent
04-May-2017	60	44	0.2	Hail
05-May-2017	62	45	0	None
06-May-2017	60	43.333	0	<undefined>
07-May-2017	58	41.667	0	<undefined>
08-May-2017	56	40	0	None
09-May-2017	59	42	0.15	Thunder
10-May-2017	60	45	0	None

如果您指定了方法，则 **retime** 会将该方法应用于所有变量，并覆盖 **VariableContinuity** 中的值。

```
Station1Missing = retime(Station1,'daily','fillwithmissing')
```

Station1Missing=7×5 timetable					
Date	Tmax	Tmin	PrecipTotal	WXEvent	
04-May-2017	60	44	0.2	Hail	
05-May-2017	62	45	0	None	
06-May-2017	NaN	NaN	NaN	<undefined>	
07-May-2017	NaN	NaN	NaN	<undefined>	
08-May-2017	56	40	0	None	
09-May-2017	59	42	0.15	Thunder	
10-May-2017	60	45	0	None	

### 同步包含连续和离散数据的时间表

`synchronize` 函数还可以使用不同的方法填充输出时间表变量，具体取决于在每个输入时间表的 `VariableContinuity` 属性中指定的值。

创建第二个时间表，其中包含第二个气象站中的压力读数（以毫巴为单位）。该时间表包含从 2017 年 5 月 4 日到 5 月 8 日的模拟读数。

```
Date = datetime(2017,5,4:8);
Pressure = [995 1003 1013 1018 1006];
Station2 = timetable(Date,Pressure)
```

Station2=5×2 timetable	
Date	Pressure
04-May-2017	995
05-May-2017	1003
06-May-2017	1013
07-May-2017	1018
08-May-2017	1006

使用 `synchronize` 函数同步这两个气象站中的数据。`synchronize` 会根据 `Station1` 的 `VariableContinuity` 属性的值来填充 `Station1` 中的变量的值。但是，由于 `Station2` 的 `VariableContinuity` 属性为空，`synchronize` 将使用 `NaN` 值来填充 `Pressure`。

```
BothStations = synchronize(Station1,Station2)
```

BothStations=7×6 timetable						
Date	Tmax	Tmin	PrecipTotal	WXEvent	Pressure	
04-May-2017	60	44	0.2	Hail	995	
05-May-2017	62	45	0	None	1003	
06-May-2017	60	43.333	0	<undefined>	1013	
07-May-2017	58	41.667	0	<undefined>	1018	
08-May-2017	56	40	0	None	1006	
09-May-2017	59	42	0.15	Thunder	NaN	
10-May-2017	60	45	0	None	NaN	

要指示 `Station2.Pressure` 包含连续数据，请指定 `Station2` 的 `VariableContinuity` 属性。虽然 `Station2` 只包含一个变量，但您必须使用元胞数组而非字符串向量来指定 `VariableContinuity`。

```

Station2.Properties.VariableContinuity = {'continuous'};
Station2.Properties

ans =
TimetableProperties with properties:

    Description: "
    UserData: []
    DimensionNames: {'Date' 'Variables'}
    VariableNames: {'Pressure'}
    VariableDescriptions: {}
    VariableUnits: {}
    VariableContinuity: continuous
        RowTimes: [5x1 datetime]
        StartTime: 04-May-2017
        SampleRate: NaN
        TimeStep: 1d
    CustomProperties: No custom properties are set.
    Use addprop and rmprop to modify CustomProperties.

```

同步这两个气象站中的数据。由于 `Station2.Pressure` 包含连续数据，`synchronize` 将会填充 `BothStations.Pressure` 中的值。

```
BothStations = synchronize(Station1,Station2)
```

BothStations=7×6 timetable					
Date	Tmax	Tmin	PrecipTotal	WXEvent	Pressure
04-May-2017	60	44	0.2	Hail	995
05-May-2017	62	45	0	None	1003
06-May-2017	60	43.333	0	<undefined>	1013
07-May-2017	58	41.667	0	<undefined>	1018
08-May-2017	56	40	0	None	1006
09-May-2017	59	42	0.15	Thunder	994
10-May-2017	60	45	0	None	982

如果您将某个方法指定为 `synchronize` 的输入参数，则 `synchronize` 会将该方法应用于所有变量，就像 `retime` 函数所执行的一样。

## 另请参阅

`retime` | `synchronize` | `timetable`

## 相关示例

- “创建时间表”（第 10-2 页）
- “对时间表中的数据进行重采样和聚合”（第 10-5 页）
- “合并时间表并同步其数据”（第 10-8 页）
- “按行时间和变量类型选择时间表数据”（第 10-18 页）
- “清理包含缺失、重复或不均匀时间的时间表”（第 10-23 页）

## 按行时间和变量类型选择时间表数据

时间表是一种表类型，用于将时间与每一行进行关联。您可以使用时间表下标，以不同的方式选择其数据子集。要选择行时间位于给定的时间范围内的时间表行，请使用 `timerange` 函数指定时间范围。由于时间表是一个表，因此您可以使用圆括号或花括号对行和变量建立索引。您可以对特定的行时间进行索引，也可以选择行时间与位于您使用 `withtol` 函数设置的容差范围内的指定时间匹配的行。您也可以为表或时间表建立下标，以选择与使用 `vartype` 函数指定的类型匹配的所有变量。最后，使用 `Variables` 属性将时间表中的数据提取到矩阵中。

### 基于文件创建时间表

基于样本文件 `outages.csv` 创建一个时间表，该文件包含表示美国电力中断的数据。使用 `readtable` 函数读取该文件中的表。将 `T.Cause` 和 `T.Region` 转换为分类数组。然后，使用 `table2timetable` 函数将表转换为时间表。显示时间表的前五行。TT 是一个时间表，其中包含从 2002 年 2 月到 2014 年 1 月的断电数据。

```
T = readtable('outages.csv');
T.Cause = categorical(T.Cause);
T.Region = categorical(T.Region);
TT = table2timetable(T);
TT(1:5,:)
```

	OutageTime	Region	Loss	Customers	RestorationTime	Cause
2002-02-01 12:18	SouthWest	458.98	1.8202e+06	2002-02-07 16:50	winter storm	
2003-01-23 00:49	SouthEast	530.14	2.1204e+05	NaT	winter storm	
2003-02-07 21:15	SouthEast	289.4	1.4294e+05	2003-02-17 08:14	winter storm	
2004-04-06 05:44	West	434.81	3.4037e+05	2004-04-06 06:10	equipment fault	
2002-03-16 06:18	MidWest	186.44	2.1275e+05	2002-03-18 23:23	severe storm	

### 汇总时间表并访问行时间

显示 TT 的摘要。该摘要是一个时间表，其中包含 1468 行和五个变量。

```
summary(TT)
```

RowTimes:

```
OutageTime: 1468x1 datetime
Values:
Min      2002-02-01 12:18
Median   2010-03-18 21:05
Max      2014-01-15 02:41
```

Variables:

```
Region: 1468x1 categorical
```

Values:

```
MidWest    142
NorthEast  557
SouthEast  389
SouthWest  26
```

```
West      354
```

**Loss:** 1468x1 double

Values:

Min	0
Median	180.26
Max	23418
NumMissing	604

**Customers:** 1468x1 double

Values:

Min	0
Median	75765
Max	5.9689e+06
NumMissing	328

**RestorationTime:** 1468x1 datetime

Values:

Min	2002-02-07 16:50
Median	2010-03-31 10:54
Max	2042-09-18 23:31
NumMissing	29

**Cause:** 1468x1 categorical

Values:

attack	294
earthquake	2
energy emergency	188
equipment fault	156
fire	25
severe storm	338
thunder storm	201
unknown	24
wind	95
winter storm	145

访问行时间。行时间不在变量中，行时间向量只是时间表的一个属性。但是，您可以使用圆点语法访问行时间。**TT.OutageTime** 是一个 1468×1 的日期时间值向量。显示 **TT.OutageTime** 的前五行。

**TT.OutageTime(1:5)**

```
ans = 5x1 datetime array
2002-02-01 12:18
2003-01-23 00:49
2003-02-07 21:15
2004-04-06 05:44
2002-03-16 06:18
```

### 对时间范围建立下标

要选择位于某个时间范围内的所有时间表行，请使用 `timerange` 函数创建下标来作为帮助。您指定的开始时间和结束时间不必与时间表中的任何行时间匹配。

选择断电发生在 2002 年 1 月到 2003 年 12 月之间的所有行。显示 TT2 的前五行。

```
TR = timerange('2002-01-01','2003-12-31');
TT2 = TT(TR,:);
TT2(1:5,:)
```

OutageTime	Region	Loss	Customers	RestorationTime	Cause
2002-02-01 12:18	SouthWest	458.98	1.8202e+06	2002-02-07 16:50	winter storm
2003-01-23 00:49	SouthEast	530.14	2.1204e+05	NaT	winter storm
2003-02-07 21:15	SouthEast	289.4	1.4294e+05	2003-02-17 08:14	winter storm
2002-03-16 06:18	MidWest	186.44	2.1275e+05	2002-03-18 23:23	severe storm
2003-06-18 02:49	West	0	0	2003-06-18 10:54	attack

显示 TT2 的后五行。

```
TT2(end-4:end,:)
```

OutageTime	Region	Loss	Customers	RestorationTime	Cause
2003-09-02 19:46	SouthEast	0	0	2003-09-16 22:25	severe storm
2003-09-15 14:56	MidWest	418.7	61045	2003-09-22 04:21	thunder storm
2003-09-24 22:43	SouthWest	2576.9	9.4873e+05	2003-09-25 14:46	severe storm
2003-09-18 10:40	SouthWest	301.8	2.3973e+05	2003-09-27 08:17	severe storm
2003-10-11 19:36	SouthEast	309.8	93582	2003-10-11 19:49	energy emergency

TT2 是一个仅具有 98 行的时间表，其中只包含 2002 年和 2003 年的断电数据。

### 对指定时间进行索引

您可以使用表示 TT.`OutageTime` 中的特定时间的日期时间值或字符向量对 TT 进行索引。但是，当您执行此操作时，所指定的时间必须在时间向量中具有完全匹配项，并且只选择这些时间。根据 TT 的第一行和第三行的时间对 TT 进行索引。

```
TT({'2002-02-01 12:18:00','2003-02-07 21:15:00'},:)
```

OutageTime	Region	Loss	Customers	RestorationTime	Cause
2002-02-01 12:18	SouthWest	458.98	1.8202e+06	2002-02-07 16:50	winter storm
2003-02-07 21:15	SouthEast	289.4	1.4294e+05	2003-02-17 08:14	winter storm

### 使用容差对指定时间进行索引

对时间进行索引时指定一个容差。您可以使用 `withtol` 函数创建下标来作为帮助。通过 `withtol` 的输出，您可以选择在指定容差范围内匹配的行时间。

在指定日期对 TT 进行索引。指定一天的容差，以返回行时间位于指定日期中的一天范围内的行。时间必须位于日期时间或持续时间向量中，或者位于可转换为日期时间或持续时间值的字符串向量元胞数组中。必须使用 **seconds**、**minutes**、**hours** 或 **days** 等函数将容差指定为持续时间。

```
rowTimes = {'2002-02-01','2003-02-07'};
S = withtol(rowTimes,days(1));
TT(S,:)

ans=2×6 timetable
OutageTime    Region    Loss    Customers    RestorationTime    Cause
_____        _____    ____    _____        _____        _____
2002-02-01 12:18    SouthWest    458.98    1.8202e+06    2002-02-07 16:50    winter storm
2003-02-07 21:15    SouthEast    289.4    1.4294e+05    2003-02-17 08:14    winter storm
```

### 按变量类型建立下标

要选择具有给定类型的所有时间表变量，请使用 **vartype** 函数创建下标来作为帮助。您可以指定变量类型，而不必指定变量的名称或变量在时间表中的位置。

选择包含数值数据的所有变量。TT2 只包含变量 **Loss** 和 **Customers**。TT 的其他三个变量为分类变量或日期时间变量。显示 TT2 的前五行。

```
S = vartype('numeric');
TT2 = TT(:,S);
TT2(1:5,:)

ans=5×3 timetable
OutageTime    Loss    Customers
_____        ____    _____
2002-02-01 12:18    458.98    1.8202e+06
2003-01-23 00:49    530.14    2.1204e+05
2003-02-07 21:15    289.4    1.4294e+05
2004-04-06 05:44    434.81    3.4037e+05
2002-03-16 06:18    186.44    2.1275e+05
```

同时根据时间范围和变量类型建立下标。

```
TR = timerange('2002-01-01','2003-12-31');
TT2 = TT(TR,S);
TT2(1:5,:)

ans=5×3 timetable
OutageTime    Loss    Customers
_____        ____    _____
2002-02-01 12:18    458.98    1.8202e+06
2003-01-23 00:49    530.14    2.1204e+05
2003-02-07 21:15    289.4    1.4294e+05
2002-03-16 06:18    186.44    2.1275e+05
2003-06-18 02:49      0         0
```

### 通过 Variables 属性提取数据

表和时间表具有一个 **Variables** 属性，您可以使用该属性将数据从变量提取到矩阵中，只要变量可以串联在一起。

使用 **Variables** 属性从 **TT2** 提取数值数据。**A** 是一个  $1468 \times 2$  的双精度矩阵。当您将数据从时间表提取到数组中时，行时间不包含在内。

```
A = TT2.Variables;
A(1:5,:)
```

```
ans = 5×2
106 ×

0.0005 1.8202
0.0005 0.2120
0.0003 0.1429
0.0002 0.2128
0 0
```

**TT2.Variables** 的结果与使用花括号通过 **TT2{:, :}** 语法提取数据的结果相同。

您可以将 **TT2** 中的变量串联到双精度数组中。但是，**TT** 包含无法串联的数值变量、分类变量和日期时间变量。无法串联变量时，**Variables** 属性将会返回错误。要避免此类错误，您可以在使用 **Variables** 属性之前根据变量类型建立下标。

为 **TT** 建立下标，以选择数值变量并将它们提取到矩阵中。

```
A = TT(:, vartype('numeric')).Variables;
A(1:5,:)
```

```
ans = 5×2
106 ×

0.0005 1.8202
0.0005 0.2120
0.0003 0.1429
0.0004 0.3404
0.0002 0.2128
```

### 另请参阅

[retime](#) | [synchronize](#) | [table2timetable](#) | [timerange](#) | [timetable](#) | [vartype](#) | [withtol](#)

### 相关示例

- “创建时间表”（第 10-2 页）
- “对时间表中的数据进行重采样和聚合”（第 10-5 页）
- “合并时间表并同步其数据”（第 10-8 页）
- “清理包含缺失、重复或不均匀时间的时间表”（第 10-23 页）

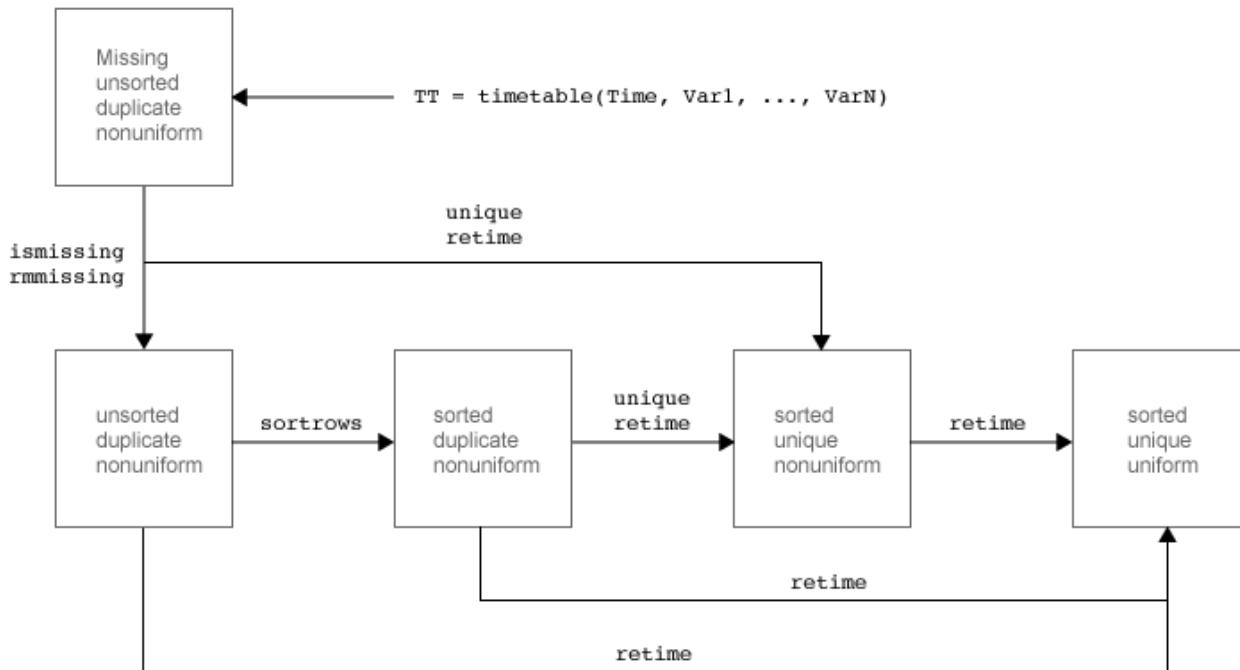
## 清理包含缺失、重复或不均匀时间的时间表

以下示例演示如何根据包含缺失、重复或不均匀时间的时间表来创建规则时间表。时间表是一种表类型，用于将时间戳或行时间与每一行数据进行关联。在规则时间表中，行时间会进行排序并且是唯一的，彼此相差相同的规则时间步。该示例还演示了如何导出时间表中的数据以供其他函数使用。

时间表可以是不规则的。它们可以包含不按行时间排序的行。时间表可以包含多个具有相同行时间的行，但是这些行可以具有不同的数据值。即使行时间已排序并且是唯一的，也会因不同大小的时间步而异。时间表甚至可以包含 `NaT` 或 `Nan` 值以指示缺失的行时间。

时间表提供了很多种不同的方式来解决时间缺失、重复或不均匀的问题，以及对数据进行重采样或将数据聚合为规则行时间。

- 要查找缺失的行时间，请使用 `ismissing`。
- 要删除缺失的时间和数据，请使用 `rmmissing`。
- 要按行时间对时间表进行排序，请使用 `sortrows`。
- 要使时间表具有唯一和已排序的行时间，请使用 `unique` 和 `retime`。
- 要删除重复的时间，请指定一个唯一的时间向量并使用 `retime`。
- 要创建规则时间表，请指定一个规则时间向量并使用 `retime`。



### 加载时间表

从 MAT 文件 `badTimes` 加载一个样本时间表，其中包含在 2016 年 6 月 9 日的几个小时内获取的天气测量值。该时间表包含在当天不定时获取的温度、降雨量和风速测量值。

```
load badTimes
TT

TT=12×4 timetable
Time      Temp   Rain  WindSpeed
-----|-----|-----|-----
09-Jun-2016 06:01:04    73  0.01   2.3
09-Jun-2016 07:59:23    59  0.08   0.9
09-Jun-2016 09:53:57    59  0.03   3.4
09-Jun-2016 09:53:57    67  0.03   3.4
NaT        56  0     0
09-Jun-2016 09:53:57    67  0.03   3.4
09-Jun-2016 08:49:10    62  0.01   2.7
09-Jun-2016 08:49:10    75.8 0.01   2.7
09-Jun-2016 08:49:10    82  0.01   2.7
09-Jun-2016 05:03:11    66.2 0.05   3
09-Jun-2016 08:49:10    67.2 0.01   2.7
09-Jun-2016 04:12:00    58.8  NaN    NaN
```

### 删除包含缺失时间的行

删除将 `NaN` 或缺失值作为行时间的行。要查找行时间向量中的缺失值，请使用 `ismissing` 函数。`ismissing` 会返回一个逻辑向量，只要 `TT.Time` 具有缺失值，该向量即包含 1。为时间表进行向后索引，以仅保留未将缺失值作为行时间的行。将这些行赋给 `TT2`。

```
TF = ismissing(TT.Time);
TT2 = TT(~TF,:);
TT2
```

```
TT2=11×4 timetable
Time      Temp   Rain  WindSpeed
-----|-----|-----|-----
09-Jun-2016 06:01:04    73  0.01   2.3
09-Jun-2016 07:59:23    59  0.08   0.9
09-Jun-2016 09:53:57    59  0.03   3.4
09-Jun-2016 09:53:57    67  0.03   3.4
09-Jun-2016 09:53:57    67  0.03   3.4
09-Jun-2016 08:49:10    62  0.01   2.7
09-Jun-2016 08:49:10    75.8 0.01   2.7
09-Jun-2016 08:49:10    82  0.01   2.7
09-Jun-2016 05:03:11    66.2 0.05   3
09-Jun-2016 08:49:10    67.2 0.01   2.7
09-Jun-2016 04:12:00    58.8  NaN    NaN
```

此方法只会删除包含缺失的行时间的行。表变量仍可能包含缺失的数据值。例如，对于 `Rain` 和 `Windspeed` 变量，`TT2` 的最后一行包含 `NaN` 值。

### 删除包含缺失时间或缺失数据的行

您可以使用 `rmmissing` 函数删除缺失的行时间和缺失的数据值。`rmmissing` 会删除包含缺失的行时间、缺失的数据值或包含两者的任何时间表行。

显示 `TT` 中缺失的行时间和缺失的数据值。然后，从 `TT` 中删除所有缺失值。

```
TT
```

TT=12×4 timetable

Time	Temp	Rain	WindSpeed
09-Jun-2016 06:01:04	73	0.01	2.3
09-Jun-2016 07:59:23	59	0.08	0.9
09-Jun-2016 09:53:57	59	0.03	3.4
09-Jun-2016 09:53:57	67	0.03	3.4
NaN	56	0	0
09-Jun-2016 09:53:57	67	0.03	3.4
09-Jun-2016 08:49:10	62	0.01	2.7
09-Jun-2016 08:49:10	75.8	0.01	2.7
09-Jun-2016 08:49:10	82	0.01	2.7
09-Jun-2016 05:03:11	66.2	0.05	3
09-Jun-2016 08:49:10	67.2	0.01	2.7
09-Jun-2016 04:12:00	58.8	NaN	NaN

TT = rmmissing(TT)

TT=10×4 timetable

Time	Temp	Rain	WindSpeed
09-Jun-2016 06:01:04	73	0.01	2.3
09-Jun-2016 07:59:23	59	0.08	0.9
09-Jun-2016 09:53:57	59	0.03	3.4
09-Jun-2016 09:53:57	67	0.03	3.4
09-Jun-2016 09:53:57	67	0.03	3.4
09-Jun-2016 08:49:10	62	0.01	2.7
09-Jun-2016 08:49:10	75.8	0.01	2.7
09-Jun-2016 08:49:10	82	0.01	2.7
09-Jun-2016 05:03:11	66.2	0.05	3
09-Jun-2016 08:49:10	67.2	0.01	2.7

### 对时间表进行排序并确定是否为规则时间表

确定 TT 是否已排序。然后，使用 sortrows 函数根据行时间对时间表进行排序。

TF = issorted(TT)

TF = logical  
0

TT = sortrows(TT)

TT=10×4 timetable

Time	Temp	Rain	WindSpeed
09-Jun-2016 05:03:11	66.2	0.05	3
09-Jun-2016 06:01:04	73	0.01	2.3
09-Jun-2016 07:59:23	59	0.08	0.9
09-Jun-2016 08:49:10	62	0.01	2.7
09-Jun-2016 08:49:10	75.8	0.01	2.7
09-Jun-2016 08:49:10	82	0.01	2.7
09-Jun-2016 08:49:10	67.2	0.01	2.7

```
09-Jun-2016 09:53:57    59  0.03    3.4
09-Jun-2016 09:53:57    67  0.03    3.4
09-Jun-2016 09:53:57    67  0.03    3.4
```

确定 **TT** 是否为规则时间表。规则时间表的连续行时间之间具有相同的时间间隔。即使是已排序的时间表，也可能具有不均匀的时间步。

**TF = isregular(TT)**

```
TF = logical
0
```

显示两个行时间之差。

**diff(TT.Time)**

```
ans = 9x1 duration array
00:57:53
01:58:19
00:49:47
00:00:00
00:00:00
00:00:00
01:04:47
00:00:00
00:00:00
```

### 删除重复的行

时间表可以包含重复的行。如果多个时间表行具有相同的行时间和相同的数据值，则这些时间表行是重复的。在此示例中，**TT** 的最后两行为重复的行。

要删除重复的行，请使用 **unique** 函数。**unique** 会返回唯一行，并按行时间对这些行进行排序。

**TT = unique(TT)**

```
TT=9×4 timetable
Time      Temp   Rain  WindSpeed
_____
09-Jun-2016 05:03:11  66.2  0.05    3
09-Jun-2016 06:01:04  73    0.01    2.3
09-Jun-2016 07:59:23  59    0.08    0.9
09-Jun-2016 08:49:10  62    0.01    2.7
09-Jun-2016 08:49:10  67.2  0.01    2.7
09-Jun-2016 08:49:10  75.8  0.01    2.7
09-Jun-2016 08:49:10  82    0.01    2.7
09-Jun-2016 09:53:57  59    0.03    3.4
09-Jun-2016 09:53:57  67    0.03    3.4
```

### 查找包含重复时间和不同数据的行

时间表可以具有行时间重复但数据值不同的行。在此示例中，**TT** 有几行的行时间相同但值不同。

查找包含重复的行时间的行。首先，对行时间进行排序并查找其间没有差异的连续时间。其间没有差异的时间为重复的时间。对行时间向量进行向后索引，并返回一组标识 TT 中的重复行时间的唯一时间。

```
dupTimes = sort(TT.Time);
TF = (diff(dupTimes) == 0);
dupTimes = dupTimes(TF);
dupTimes = unique(dupTimes)

dupTimes = 2x1 datetime array
09-Jun-2016 08:49:10
09-Jun-2016 09:53:57
```

为时间表进行索引以显示包含重复的行时间的行。当您对时间进行索引时，输出时间表中会包含具有匹配的行时间的所有行。

**TT(dupTimes,:)**

```
ans=6×4 timetable
Time      Temp   Rain  WindSpeed
_____
09-Jun-2016 08:49:10    62    0.01    2.7
09-Jun-2016 08:49:10    67.2   0.01    2.7
09-Jun-2016 08:49:10    75.8   0.01    2.7
09-Jun-2016 08:49:10    82     0.01    2.7
09-Jun-2016 09:53:57    59     0.03    3.4
09-Jun-2016 09:53:57    67     0.03    3.4
```

### 选择包含重复时间的第一行和最后一行

使用 **unique** 和 **retime** 函数选择包含重复行时间的多行中的第一行和最后一行。

首先，使用 **unique** 函数创建一个由 TT 中的唯一行时间组成的向量。

**uniqueTimes = unique(TT.Time);**

从包含重复时间的每一组行中选择第一行。

**TT2 = retime(TT,uniqueTimes)**

```
TT2=5×4 timetable
Time      Temp   Rain  WindSpeed
_____
09-Jun-2016 05:03:11    66.2   0.05    3
09-Jun-2016 06:01:04    73     0.01    2.3
09-Jun-2016 07:59:23    59     0.08    0.9
09-Jun-2016 08:49:10    62     0.01    2.7
09-Jun-2016 09:53:57    59     0.03    3.4
```

从包含重复时间的每一组行中选择最后一行。为 **retime** 指定 '**previous**' 方法以复制最后一行中的数据。如果您指定 '**previous**'，则 **retime** 会从行时间向量的末尾开始执行，并在遇到重复的行时间时停止。然后，它会复制该行中的数据。

**TT2 = retime(TT,uniqueTimes,'previous')**

TT2=5×4 timetable				
Time	Temp	Rain	WindSpeed	
09-Jun-2016 05:03:11	66.2	0.05	3	
09-Jun-2016 06:01:04	73	0.01	2.3	
09-Jun-2016 07:59:23	59	0.08	0.9	
09-Jun-2016 08:49:10	82	0.01	2.7	
09-Jun-2016 09:53:57	67	0.03	3.4	

### 聚合包含重复时间的所有行中的数据

聚合包含重复的行时间的行中的数据。例如，您可以计算同时获取的具有相同数量的多个测量值的均值。

使用 `retime` 函数计算包含重复行时间的行的平均温度、降雨量和风速。

`TT = retime(TT,uniqueTimes,'mean')`

TT=5×4 timetable				
Time	Temp	Rain	WindSpeed	
09-Jun-2016 05:03:11	66.2	0.05	3	
09-Jun-2016 06:01:04	73	0.01	2.3	
09-Jun-2016 07:59:23	59	0.08	0.9	
09-Jun-2016 08:49:10	71.75	0.01	2.7	
09-Jun-2016 09:53:57	63	0.03	3.4	

### 创建规则时间表

使用 `retime` 创建规则时间表。将数据插入到一个按小时计的规则时间向量中。要使用线性插值，请指定 `'linear'`。TT 中的每个行时间都从整点时刻开始，并且连续的行时间之间的间隔为一小时。

`TT = retime(TT,'hourly','linear')`

TT=6×4 timetable				
Time	Temp	Rain	WindSpeed	
09-Jun-2016 05:00:00	65.826	0.0522	3.0385	
09-Jun-2016 06:00:00	72.875	0.010737	2.3129	
09-Jun-2016 07:00:00	66.027	0.044867	1.6027	
09-Jun-2016 08:00:00	59.158	0.079133	0.9223	
09-Jun-2016 09:00:00	70.287	0.013344	2.8171	
09-Jun-2016 10:00:00	62.183	0.031868	3.4654	

您可以指定自己的时间步，而不是使用诸如 `'hourly'` 等预定义的时间步。要指定 30 分钟的时间步，请使用 `'regular'` 输入参数和 `'TimeStep'` 名称-值对组参数。您可以指定任意大小的时间步作为持续时间或日历持续时间值。

`TT = retime(TT,'regular','linear','TimeStep',minutes(30))`

TT=11×4 timetable				
Time	Temp	Rain	WindSpeed	
09-Jun-2016 05:00:00	65.826	0.0522	3.0385	
09-Jun-2016 05:30:00	66.027	0.044867	1.6027	
09-Jun-2016 06:00:00	72.875	0.010737	2.3129	
09-Jun-2016 06:30:00	59.158	0.079133	0.9223	
09-Jun-2016 07:00:00	70.287	0.013344	2.8171	
09-Jun-2016 07:30:00	62.183	0.031868	3.4654	
09-Jun-2016 08:00:00	65.826	0.0522	3.0385	
09-Jun-2016 08:30:00	72.875	0.010737	2.3129	
09-Jun-2016 09:00:00	66.027	0.044867	1.6027	
09-Jun-2016 09:30:00	59.158	0.079133	0.9223	
09-Jun-2016 10:00:00	70.287	0.013344	2.8171	

09-Jun-2016 05:00:00	65.826	0.0522	3.0385
09-Jun-2016 05:30:00	69.35	0.031468	2.6757
09-Jun-2016 06:00:00	72.875	0.010737	2.3129
09-Jun-2016 06:30:00	69.451	0.027802	1.9578
09-Jun-2016 07:00:00	66.027	0.044867	1.6027
09-Jun-2016 07:30:00	62.592	0.062	1.2625
09-Jun-2016 08:00:00	59.158	0.079133	0.9223
09-Jun-2016 08:30:00	64.722	0.046239	1.8697
09-Jun-2016 09:00:00	70.287	0.013344	2.8171
09-Jun-2016 09:30:00	66.235	0.022606	3.1412
09-Jun-2016 10:00:00	62.183	0.031868	3.4654

### 提取规则时间表数据

您可以导出时间表数据供函数使用，以分析具有固定时间间隔的数据。例如，Econometrics Toolbox™ 和 Signal Processing Toolbox™ 带有可用于进一步分析固定间隔数据的函数。

提取时间表数据作为数组。当表变量可以串联时，您可以使用 **Variables** 属性以数组形式返回数据。

**A = TT.Variables**

**A = 11×3**

65.8260	0.0522	3.0385
69.3504	0.0315	2.6757
72.8747	0.0107	2.3129
69.4507	0.0278	1.9578
66.0266	0.0449	1.6027
62.5923	0.0620	1.2625
59.1579	0.0791	0.9223
64.7224	0.0462	1.8697
70.2868	0.0133	2.8171
66.2348	0.0226	3.1412
⋮		

**TT.Variables** 等效于使用花括号语法 **TT{:, :}** 访问所有变量。

**A2 = TT{:, :}**

**A2 = 11×3**

65.8260	0.0522	3.0385
69.3504	0.0315	2.6757
72.8747	0.0107	2.3129
69.4507	0.0278	1.9578
66.0266	0.0449	1.6027
62.5923	0.0620	1.2625
59.1579	0.0791	0.9223
64.7224	0.0462	1.8697
70.2868	0.0133	2.8171
66.2348	0.0226	3.1412

:

## 另请参阅

`diff` | `fillmissing` | `isregular` | `issorted` | `retime` | `rmmissing` | `sortrows` | `table2timetable` |  
`timetable` | `unique`

## 相关示例

- “创建时间表” (第 10-2 页)
- “对时间表中的数据进行重采样和聚合” (第 10-5 页)
- “合并时间表并同步其数据” (第 10-8 页)
- “使用不同的方法对时间表变量重设时间并进行同步” (第 10-14 页)
- “按行时间和变量类型选择时间表数据” (第 10-18 页)

## 在表和时间表运算中使用行标签

表和时间表提供了标记数据行的方式。在表中，您可以用名称来标记行。在时间表中，您必须用日期、时间或两者来标记行。行名称对于表是可选的，但行时间对于时间表是必需的。这些行标签是表或时间表中的元数据的一部分。在某些函数中，您还可以将行标签用作键变量、分组变量等，就像您可以使用表或时间表中的数据变量一样。这些函数为 `sortrows`、`join`、`innerjoin`、`outerjoin`、`varfun`、`rowfun`、`stack` 和 `unstack`。使用这些表函数以及将行标签用作键变量存在一些限制。

### 按行标签进行排序

例如，您可以根据时间表的行时间、根据时间表的一个或多个数据变量或者同时根据行时间和数据变量来对时间表排序。

使用 `timetable` 函数创建一个时间表。时间表在其第一个维度包含行时间，并为这些行添加标签。行时间为时间表的属性，而非时间表变量。

```
Date = datetime(2016,7,[10;10;11;11;10;10;11;11]);
X = [1;1;1;2;2;2];
Y = {'a';'b';'a';'b';'a';'b';'a';'b'};
Z = [1;2;3;4;5;6;7;8];
TT = timetable(X,Y,Z,'RowTimes',Date)
```

```
TT=8×4 timetable
    Time      X      Y      Z
    _____
    10-Jul-2016  1  {'a'}   1
    10-Jul-2016  1  {'b'}   2
    11-Jul-2016  1  {'a'}   3
    11-Jul-2016  1  {'b'}   4
    10-Jul-2016  2  {'a'}   5
    10-Jul-2016  2  {'b'}   6
    11-Jul-2016  2  {'a'}   7
    11-Jul-2016  2  {'b'}   8
```

重命名第一个维度。默认情况下，时间表的第一个维度的名称为 `Time`。您可以访问 `Properties.DimensionNames` 属性以重命名维度。

```
TT.Properties.DimensionNames{1} = 'Date';
TT.Properties.DimensionNames
ans = 1x2 cell array
{'Date'}  {'Variables'}
```

作为备选方法，您可以将行时间指定为 `timetable` 的第一个输入参数，而不必指定 '`RowTimes`'。`timetable` 函数会在第一个输入参数之后对行时间或第一个维度进行命名，就像在其他输入参数之后对时间表变量进行命名一样。

```
TT = timetable(Date,X,Y,Z)
```

```
TT=8×4 timetable
    Date      X      Y      Z
    _____
    10-Jul-2016  1  {'a'}   1
```

```

10-Jul-2016 1 {'b'} 2
11-Jul-2016 1 {'a'} 3
11-Jul-2016 1 {'b'} 4
10-Jul-2016 2 {'a'} 5
10-Jul-2016 2 {'b'} 6
11-Jul-2016 2 {'a'} 7
11-Jul-2016 2 {'b'} 8

```

按行时间对时间表排序。要对行时间排序，请按名称引用时间表的第一个维度。

**sortrows(TT,'Date')**

```

ans=8×4 timetable
  Date      X      Y      Z
  _____ -  _____ -
10-Jul-2016 1 {'a'} 1
10-Jul-2016 1 {'b'} 2
10-Jul-2016 2 {'a'} 5
10-Jul-2016 2 {'b'} 6
11-Jul-2016 1 {'a'} 3
11-Jul-2016 1 {'b'} 4
11-Jul-2016 2 {'a'} 7
11-Jul-2016 2 {'b'} 8

```

按数据变量 X 和 Y 排序。**sortrows** 会先按 X 进行排序，然后再按 Y 进行排序。

**sortrows(TT,['X' 'Y'])**

```

ans=8×4 timetable
  Date      X      Y      Z
  _____ -  _____ -
10-Jul-2016 1 {'a'} 1
11-Jul-2016 1 {'a'} 3
10-Jul-2016 1 {'b'} 2
11-Jul-2016 1 {'b'} 4
10-Jul-2016 2 {'a'} 5
11-Jul-2016 2 {'a'} 7
10-Jul-2016 2 {'b'} 6
11-Jul-2016 2 {'b'} 8

```

同时按行时间和 X 排序。

**sortrows(TT,['Date' 'X'])**

```

ans=8×4 timetable
  Date      X      Y      Z
  _____ -  _____ -
10-Jul-2016 1 {'a'} 1
10-Jul-2016 1 {'b'} 2
10-Jul-2016 2 {'a'} 5
10-Jul-2016 2 {'b'} 6
11-Jul-2016 1 {'a'} 3
11-Jul-2016 1 {'b'} 4

```

```
11-Jul-2016 2 {'a'} 7
11-Jul-2016 2 {'b'} 8
```

### 将行标签用作分组变量或键变量

当您使用 `rowfun`、`varfun`、`stack` 和 `unstack` 函数将行分组在一起时，可以将行标签指定为分组变量。当您使用 `join`、`innerjoin` 和 `outerjoin` 函数将表或时间表联接在一起时，可以将行标签指定为键变量。

例如，您可以将多个行名称和一个表变量同时用作键变量，将两个表内联在一起。内联仅保留键变量匹配的表行。

创建两个包含患者数据的表。表的第一个维度可以包含行名称，用于为行添加标签，但这不是必需的。这里将患者的姓氏指定为这些表的行名称。添加患者的名字作为表变量。

```
A = table({'Michael';'Louis';'Alice';'Rosemary';'Julie'},[38;43;45;40;49],...
    'VariableNames',{'FirstName' 'Age'},...
    'RowNames',{'Garcia' 'Johnson' 'Wu' 'Jones' 'Picard'})
```

```
A=5×2 table
    FirstName      Age
    _____
    Garcia    {'Michael'}   38
    Johnson   {'Louis'}    43
    Wu        {'Alice'}    45
    Jones     {'Rosemary'}  40
    Picard    {'Julie'}    49
```

```
B = table({'Michael';'Beverly';'Alice'},...
    [64;69;67],...
    [119;163;133],...
    [122 80; 109 77; 117 75],...
    'VariableNames',{'FirstName' 'Height' 'Weight' 'BloodPressure'},...
    'RowNames',{'Garcia' 'Johnson' 'Wu'})
```

```
B=3×4 table
    FirstName      Height    Weight    BloodPressure
    _____
    Garcia    {'Michael'}   64       119      122      80
    Johnson   {'Beverly'}   69       163      109      77
    Wu        {'Alice'}    67       133      117      75
```

如果表包含行名称，则可以按行名称对其进行索引。按行名称进行索引是一种选择表行的便捷方式。按某个患者的姓氏对 `B` 进行索引以检索该患者的相关信息。

```
B('Garcia',:)
```

```
ans=1×4 table
    FirstName      Height    Weight    BloodPressure
    _____
    Garcia    {'Michael'}   64       119      122      80
```

对这两个表执行内联。两个表都使用患者的姓氏作为行名称，并且包含名字作为表变量。这两个表中的某些患者具有匹配的姓氏，但名字不同。要确保姓氏和名字都匹配，请使用行名称和 **FirstName** 作为键变量。要将行名称指定为键变量或分组变量，请使用表的第一个维度的名称。默认情况下，第一个维度的名称为 '**Row**'。

```
C = innerjoin(A,B,'Keys',{'Row','FirstName'})
```

```
C=2×5 table
  FirstName    Age    Height    Weight    BloodPressure
  _____
  Garcia    {'Michael'}   38      64      119      122      80
  Wu        {'Alice'}     45      67      133      117      75
```

如果您重命名表的第一个维度，则可以按该名称来引用行名称，而不是使用 '**Row**'。执行与上面相同的内联，但使用不同的名称来引用行名称。

访问 A 的 **Properties.DimensionNames** 属性以显示其维度名称。

**A.Properties.DimensionNames**

```
ans = 1x2 cell array
  {'Row'}  {'Variables'}
```

使用该表的 **Properties.DimensionNames** 属性以更改其第一个维度的名称。然后，使用新名称作为键变量。

```
A.Properties.DimensionNames{1} = 'LastName';
A.Properties.DimensionNames
```

```
ans = 1x2 cell array
  {'LastName'}  {'Variables'}
```

使用 **LastName** 和 **FirstName** 作为键变量，对 A 和 B 执行内联。

```
B.Properties.DimensionNames{1} = 'LastName';
D = innerjoin(A,B,'Keys',{'LastName','FirstName'})
```

```
D=2×5 table
  FirstName    Age    Height    Weight    BloodPressure
  _____
  Garcia    {'Michael'}   38      64      119      122      80
  Wu        {'Alice'}     45      67      133      117      75
```

### 使用表函数和行标签的说明

- 您无法使用 **stack** 和 **unstack** 函数来堆叠或分叠行标签。但是，您可以使用行标签作为分组变量。
- 当第一个参数为表且第二个参数为时间表时，您无法使用 **join**、**innerjoin** 或 **outerjoin** 函数执行联接。但是，当两个参数都为表、两个参数都为时间表或第一个参数为时间表且第二个参数为表时，则可以执行联接。

- 如果您将行标签指定为键变量，则联接操作的输出可以包含行标签。有关联接操作中的行标签的详细信息，请参阅 `join`、`innerjoin` 和 `outerjoin` 函数的 'Keys'、'LeftKeys' 和 'RightKeys' 参数的相关文档。

## 另请参阅

`innerjoin` | `join` | `outerjoin` | `rowfun` | `sortrows` | `stack` | `unstack` | `varfun`

## 洛马普列塔地震分析

以下示例演示如何分析和以可视方式呈现地震数据。

### 加载地震数据

文件 `quake.mat` 包含圣克鲁斯山脉在 1989 年 10 月 17 日发生的洛马普列塔地震的 200Hz 数据。该数据由加州大学圣克鲁斯分校的 Joel Yellin 通过 Charles F. Richter 地震实验室提供。

首先加载数据。

```
load quake
whos e n v

Name      Size      Bytes Class Attributes
e          10001x1    80008 double
n          10001x1    80008 double
v          10001x1    80008 double
```

工作区中有三个变量，包含由位于加州大学圣克鲁斯分校的自然科学大楼的一个加速计记录的时间跟踪。加速计记录了地震波的主震幅值。变量 `n`、`e`、`v` 指代该工具测量的三个定向分量，工具已调整为与断层平行，其 N 方向指向萨克拉门托方向。数据未针对工具响应进行纠正。

创建一个变量 `Time`，其中包含以 200Hz 采样的时间戳，并且长度与其他向量相同。使用 `seconds` 函数和乘法表示正确的单位以实现该  $\text{Hz} (\text{s}^{-1})$  采样率。这将生成一个适用于表示已用时间的 `duration` 变量。

```
Time = (1/200)*seconds(1:length(e));
whos Time

Name      Size      Bytes Class Attributes
Time     10001x1    80010 duration
```

### 组织时间表中的数据

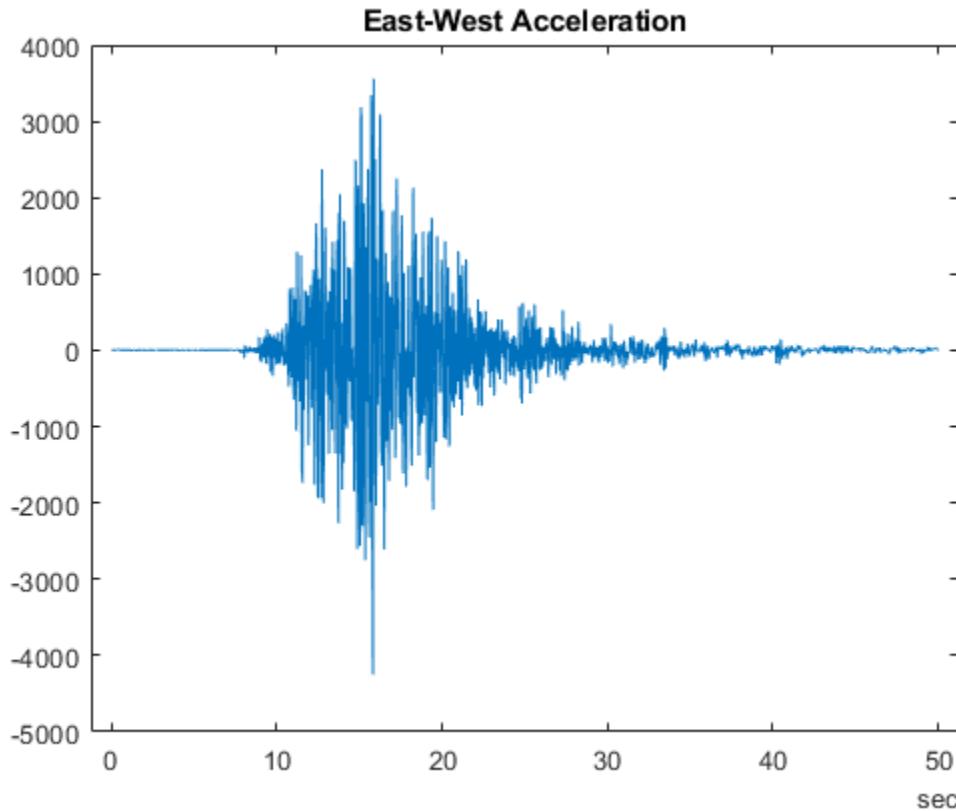
可以将这些单独的变量放到 `table` 或 `timetable` 中，以便于使用。`timetable` 提供了灵活丰富的时间戳数据处理功能。使用时间和三个加速度变量创建 `timetable`，并提供更有意义的变量名称。使用 `head` 函数来显示前八行。

```
varNames = {'EastWest', 'NorthSouth', 'Vertical'};
quakeData = timetable(Time, e, n, v, 'VariableNames', varNames);
head(quakeData)
```

```
ans=8×4 timetable
Time    EastWest  NorthSouth  Vertical
_____|_____|_____|_____
0.005 sec  5       3        0
0.01 sec   5       3        0
0.015 sec  5       2        0
0.02 sec   5       2        0
0.025 sec  5       2        0
0.03 sec   5       2        0
0.035 sec  5       1        0
0.04 sec   5       1        0
```

通过使用圆点下标访问 `timetable` 中的变量来探查数据。（有关圆点下标的详细信息，请参阅访问表中的数据。）我们选择了“东-西”振幅并将其作为持续时间的函数来执行 `plot` 操作。

```
plot(quakeData.Time,quakeData.EastWest)
title('East-West Acceleration')
```



### 缩放数据

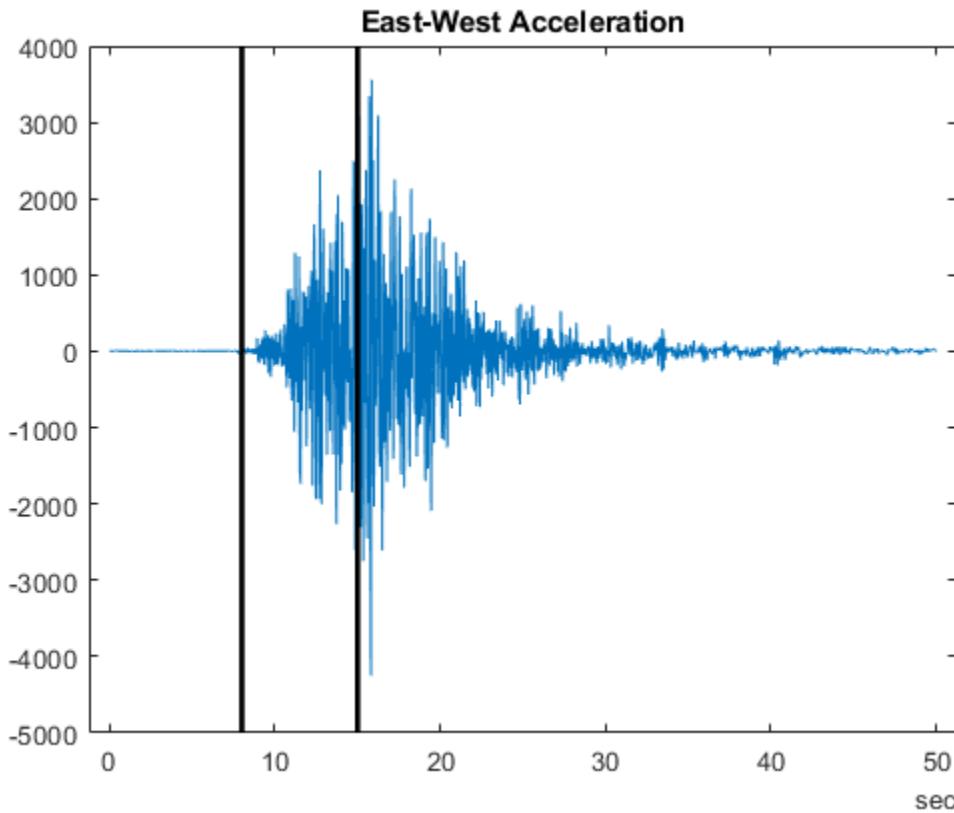
按照重力加速度缩放数据或者将表中的每个变量与常量相乘。由于这些变量都具有相同类型 (`double`)，因此可以使用维度名称 `Variables` 访问所有变量。请注意，`quakeData.Variables` 提供了一种直接修改时间表中数值的方式。

```
quakeData.Variables = 0.098*quakeData.Variables;
```

### 选择要探查的数据子集

我们对冲击波幅值从几乎为零增大到最大水平的时间区域感兴趣。直接观察上图可看到，从 8 至 15 秒这个时间段是我们需要关注的。为了显示更清晰，我们在选定的时间点绘制黑色线条以引起对该时间段的关注。所有后续计算都将涉及此时间段。

```
t1 = seconds(8)*[1;1];
t2 = seconds(15)*[1;1];
hold on
plot([t1 t2],ylim,'k','LineWidth',2)
hold off
```



### 存储相关数据

使用此区间中的数据创建另一个 timetable。使用 timerange 选择感兴趣的行。

```
tr = timerange(seconds(8),seconds(15));
dataOfInterest = quakeData(tr,:);
head(dataOfInterest)

ans=8×4 timetable
Time    EastWest    NorthSouth    Vertical
_____
```

Time	EastWest	NorthSouth	Vertical
8 sec	-0.098	2.254	5.88
8.005 sec	0	2.254	3.332
8.01 sec	-2.058	2.352	-0.392
8.015 sec	-4.018	2.352	-4.116
8.02 sec	-6.076	2.45	-7.742
8.025 sec	-8.036	2.548	-11.466
8.03 sec	-10.094	2.548	-9.8
8.035 sec	-8.232	2.646	-8.134

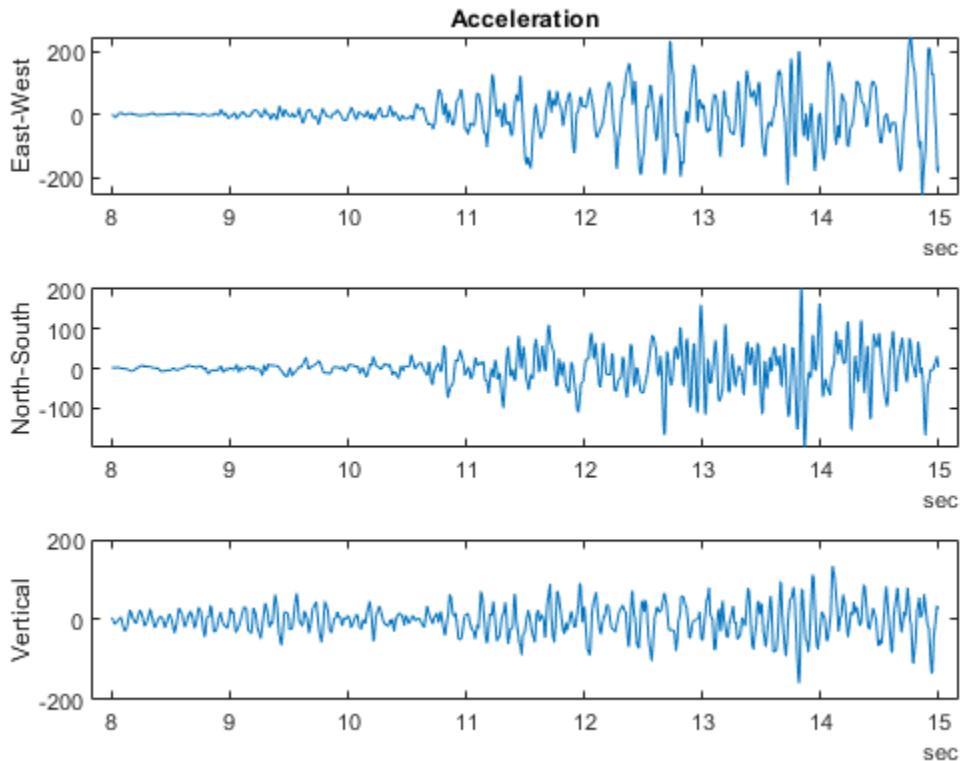
在单独的坐标区上以可视方式呈现三个加速度变量。

```
figure
subplot(3,1,1)
plot(dataOfInterest.Time,dataOfInterest.EastWest)
ylabel('East-West')
```

```

title('Acceleration')
subplot(3,1,2)
plot(dataOfInterest.Time,dataOfInterest.NorthSouth)
ylabel('North-South')
subplot(3,1,3)
plot(dataOfInterest.Time,dataOfInterest.Vertical)
ylabel('Vertical')

```



### 计算摘要统计信息

我们使用 **summary** 函数显示有关数据的统计信息。

```
summary(dataOfInterest)
```

RowTimes:

Time: 1400x1 duration

Values:

Min	8 sec
Median	11.498 sec
Max	14.995 sec
TimeStep	0.005 sec

Variables:

EastWest: 1400x1 double

Values:

```
Min      -255.09
Median    -0.098
Max       244.51
```

NorthSouth: 1400x1 double

Values:

```
Min      -198.55
Median    1.078
Max       204.33
```

Vertical: 1400x1 double

Values:

```
Min      -157.88
Median    0.98
Max       134.46
```

可以使用 `varfun` 计算其他数据统计信息。在对表或时间表中的每个变量应用函数时，这会非常有用。要应用的函数将以函数句柄的形式传递到 `varfun`。下面我们将 `mean` 函数应用于全部三个变量并以表格式输出结果，因为在计算了时间均值之后，时间会变得没有意义。

```
mn = varfun(@mean,dataOfInterest,'OutputFormat','table')
```

```
mn=1×3 table
 mean_EastWest  mean_NorthSouth  mean_Vertical
 _____
 0.9338        -0.10276       -0.52542
```

## 计算速度和位置

为了确定冲击波的传播速度，我们对加速度进行一次积分。我们使用沿时间变量的累积和来获取波前速度。

```
edot = (1/200)*cumsum(dataOfInterest.EastWest);
edot = edot - mean(edot);
```

下面我们将全部三个变量执行积分来计算该速度。可以方便地创建函数，并使用 `varfun` 将该函数应用于 `timetable` 中的变量。在本例中，我们在文件末尾处包括了该函数，并将其命名为 `velFun`。

```
vel = varfun(@velFun,dataOfInterest);
head(vel)

ans=8×4 timetable
 Time    velFun_EastWest  velFun_NorthSouth  velFun_Vertical
 _____
 8 sec      -0.56831      0.44642       1.8173
 8.005 sec   -0.56831      0.45769       1.834
 8.01 sec     -0.5786      0.46945       1.832
 8.015 sec    -0.59869      0.48121       1.8114
 8.02 sec     -0.62907      0.49346       1.7727
 8.025 sec    -0.66925      0.5062        1.7154
```

8.03 sec	-0.71972	0.51894	1.6664
8.035 sec	-0.76088	0.53217	1.6257

将同一函数 velFun 应用于速度以确定位置。

```

pos = varfun(@velFun,vel);
head(pos)

ans=8×4 timetable
Time      velFun_velFun_EastWest    velFun_velFun_NorthSouth    velFun_velFun_Vertical
_____
8 sec      2.1189                  -2.1793                  -3.0821
8.005 sec   2.1161                  -2.177                   -3.0729
8.01 sec    2.1132                  -2.1746                  -3.0638
8.015 sec   2.1102                  -2.1722                  -3.0547
8.02 sec    2.107                   -2.1698                  -3.0458
8.025 sec   2.1037                  -2.1672                  -3.0373
8.03 sec    2.1001                  -2.1646                  -3.0289
8.035 sec   2.0963                  -2.162                   -3.0208

```

请注意 `varfun` 所创建的时间表中的变量名称是如何包含所使用函数的名称的。此方法有助于跟踪已对原始数据执行了哪些运算。使用圆点表示法将变量名称重新调整回其原始值。

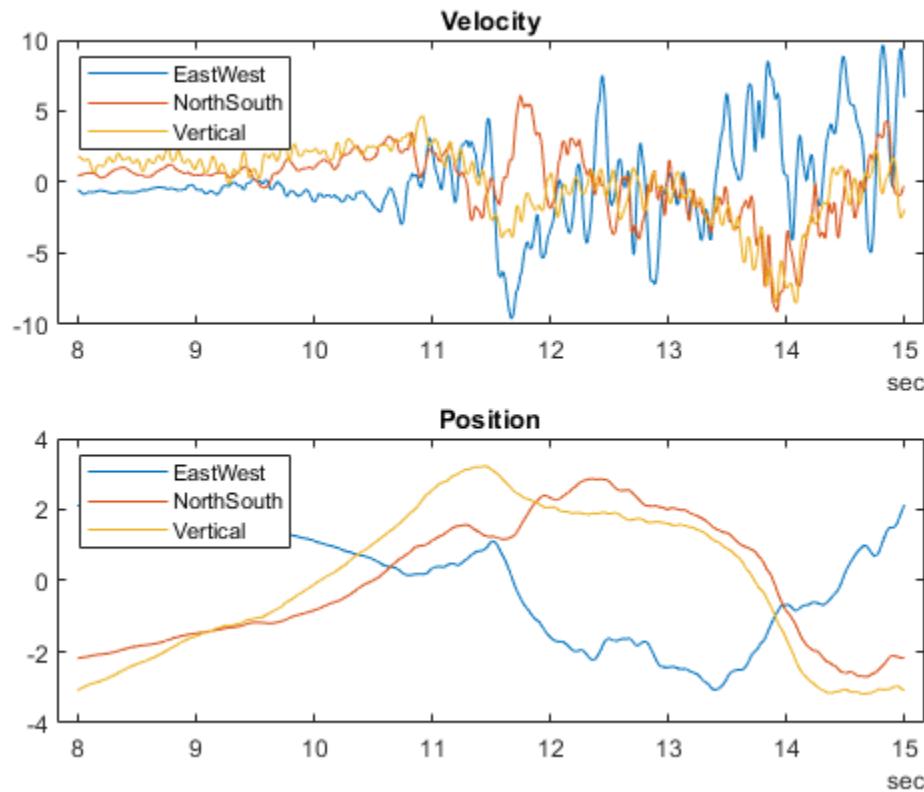
```
pos.Properties.VariableNames = varNames;
```

下面我们分别绘制所关注时间段内速度和位置的 3 个分量。

```

figure
subplot(2,1,1)
plot(vel.Time,vel.Variables)
legend(quakeData.Properties.VariableNames,'Location','NorthWest')
title('Velocity')
subplot(2,1,2)
plot(vel.Time,pos.Variables)
legend(quakeData.Properties.VariableNames,'Location','NorthWest')
title('Position')

```

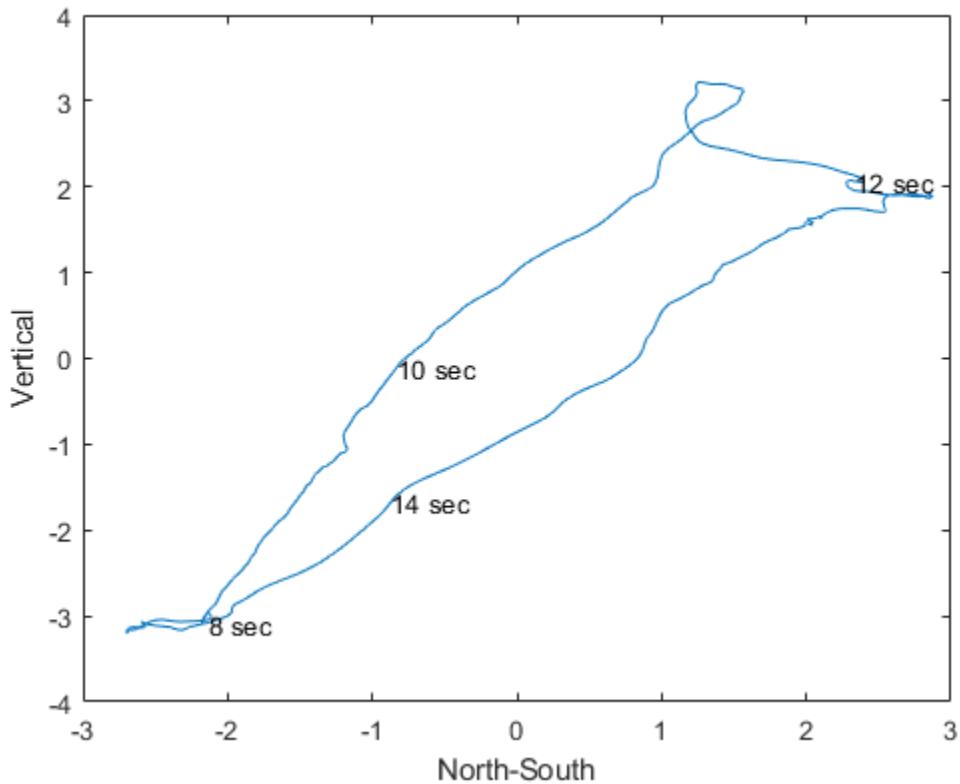


### 以可视方式呈现轨迹

可使用分量值绘制二维或三维轨迹。下面显示了以可视方式呈现这些数据的不同方式。

首先使用二维投影。下面是第一张标注有几个时间值的图。

```
figure
plot(pos.NorthSouth,pos.Vertical)
xlabel('North-South')
ylabel('Vertical')
% Select locations and label
nt = ceil((max(pos.Time) - min(pos.Time))/6);
idx = find(fix(pos.Time/nt) == (pos.Time/nt));
text(pos.NorthSouth(idx),pos.Vertical(idx),char(pos.Time(idx)))
```



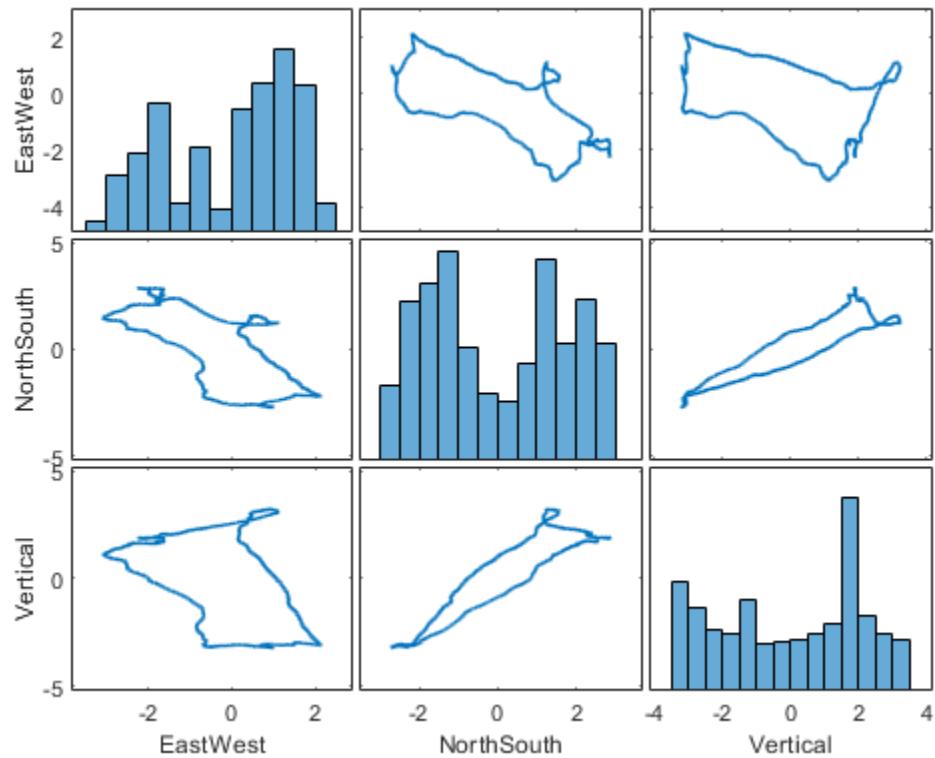
使用 `plotmatrix` 以可视方式呈现一个网格，其中包含每个变量相对另一变量的散点图，对角线上则是每个变量自身的直方图。输出变量 `Ax` 表示网格中的每个坐标区，可用于确定要使用 `xlabel` 和 `ylabel` 标记的坐标区。

```

figure
[S,Ax] = plotmatrix(pos.Variables);

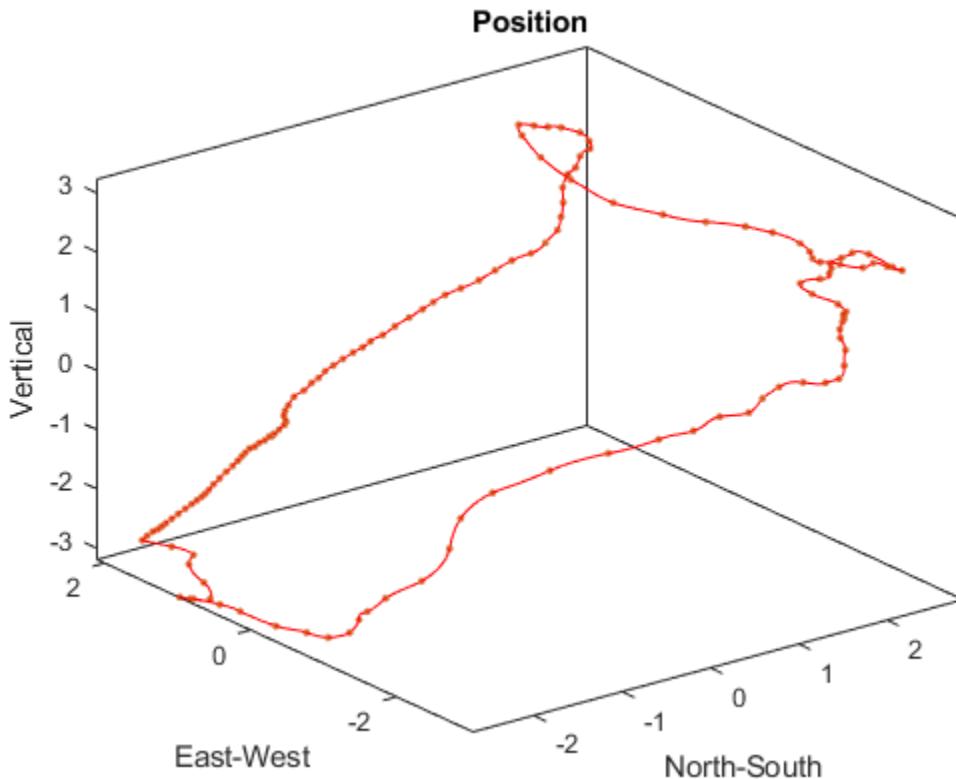
for ii = 1:length(varNames)
    xlabel(Ax(end,ii),varNames{ii})
    ylabel(Ax(ii,1),varNames{ii})
end

```



绘制轨迹的三维视图并每隔十个位置点绘制一个圆点。点之间的间距即表示速度。

```
step = 10;
figure
plot3(pos.NorthSouth,pos.EastWest,pos.Vertical,'r')
hold on
plot3(pos.NorthSouth(1:step:end),pos.EastWest(1:step:end),pos.Vertical(1:step:end),'.')
hold off
box on
axis tight
xlabel('North-South')
ylabel('East-West')
zlabel('Vertical')
title('Position')
```



## 支持函数

下面定义了这些函数。

```
function y = velFun(x)
    y = (1/200)*cumsum(x);
    y = y - mean(y);
end
```

## 另请参阅

[duration](#) | [head](#) | [seconds](#) | [summary](#) | [timerange](#) | [timetable](#) | [varfun](#)

## 相关示例

- “表示 MATLAB 中的日期和时间” (第 7-2 页)
- “创建时间表” (第 10-2 页)
- “清理包含缺失、重复或不均匀时间的时间表” (第 10-23 页)
- “按行时间和变量类型选择时间表数据” (第 10-18 页)
- “访问表中的数据” (第 9-33 页)

## 使用 timetable 预处理和探索时间戳数据

此示例说明如何使用 **timetable** 数据容器组织和预处理时间戳数据，从而基于传感器数据分析自行车交通模式。这些数据来自马萨诸塞州剑桥市百老汇大街上的传感器。剑桥市允许公众访问位于 Cambridge Open Data 站点上的完整数据。

以下示例演示如何执行各种数据清理、整理和预处理任务，例如删除缺失值以及将时间戳数据与不同的时钟同步。此外，还将突出显示数据探查，包括使用 **timetable** 数据容器进行可视化和分组计算，以便：

- 探查每日自行车流量
- 将自行车流量与当地天气状况进行比较
- 分析一周中的不同天和一天中的不同时间的自行车流量

### 将自行车流量数据导入时间表

从逗号分隔的文本文件中导入自行车流量数据样本。**readtable** 函数在表中返回这些数据。使用 **head** 函数来显示前八行。

```
bikeTbl = readtable('BicycleCounts.csv');
head(bikeTbl)
```

Timestamp	Day	Total	Westbound	Eastbound
2015-06-24 00:00:00	{'Wednesday'}	13	9	4
2015-06-24 01:00:00	{'Wednesday'}	3	3	0
2015-06-24 02:00:00	{'Wednesday'}	1	1	0
2015-06-24 03:00:00	{'Wednesday'}	1	1	0
2015-06-24 04:00:00	{'Wednesday'}	1	1	0
2015-06-24 05:00:00	{'Wednesday'}	7	3	4
2015-06-24 06:00:00	{'Wednesday'}	36	6	30
2015-06-24 07:00:00	{'Wednesday'}	141	13	128

这些数据具有时间戳，因此可以使用时间表方便地存储和分析这些数据。时间表与表类似，但包含与数据行关联的时间戳。时间戳（即行时间）由 **datetime** 或 **duration** 值表示。**datetime** 和 **duration** 是建议的数据类型，分别用于表示时间点或已用时间。

使用 **table2timetable** 函数将 **bikeTbl** 转换为时间表。由于 **readtable** 返回的是表，因此必须使用转换函数。**table2timetable** 可将表中的第一个 **datetime** 或 **duration** 变量转换为时间表的行时间。行时间是用来标记行的元数据。但是，当您显示时间表时，行时间和时间表变量以相似方式显示。请注意，该表有五个变量，而时间表只有四个变量。

```
bikeData = table2timetable(bikeTbl);
head(bikeData)
```

Timestamp	Day	Total	Westbound	Eastbound
2015-06-24 00:00:00	{'Wednesday'}	13	9	4
2015-06-24 01:00:00	{'Wednesday'}	3	3	0
2015-06-24 02:00:00	{'Wednesday'}	1	1	0
2015-06-24 03:00:00	{'Wednesday'}	1	1	0
2015-06-24 04:00:00	{'Wednesday'}	1	1	0

```
2015-06-24 05:00:00  {'Wednesday'}    7     3      4
2015-06-24 06:00:00  {'Wednesday'}   36     6     30
2015-06-24 07:00:00  {'Wednesday'}  141    13    128
```

**whos bikeTbl bikeData**

Name	Size	Bytes	Class	Attributes
bikeData	9387x4	1487627	timetable	
bikeTbl	9387x5	1562953	table	

## 访问时间和数据

将 Day 变量转换为分类类型。分类数据类型设计用于由有限的离散值集组成的数据，例如一周中的星期几的名称。列出类别，以便它们按星期几的顺序显示。使用圆点下标按名称访问变量。

```
bikeData.Day = categorical(bikeData.Day,{'Sunday','Monday','Tuesday',...
    'Wednesday','Thursday','Friday','Saturday'});
```

在时间表中，时间与数据变量分开处理。访问时间表的 **Properties** 以显示行时间是时间表的第一个维度，变量是第二个维度。**DimensionNames** 属性显示两个维度的名称，而 **VariableNames** 属性显示沿第二个维度的变量的名称。

**bikeData.Properties**

```
ans =
TimetableProperties with properties:

    Description: "
        UserData: []
        DimensionNames: {'Timestamp' 'Variables'}
        VariableNames: {'Day' 'Total' 'Westbound' 'Eastbound'}
        VariableDescriptions: {}
        VariableUnits: {}
        VariableContinuity: []
        RowTimes: [9387x1 datetime]
        StartTime: 2015-06-24 00:00:00
        SampleRate: NaN
        TimeStep: NaN
        CustomProperties: No custom properties are set.
        Use addprop and rmprop to modify CustomProperties.
```

默认情况下，在将表转换为时间表时，**table2timetable** 将 **Timestamp** 分配为第一个维度名称，因为这是原始表中的变量名称。可以通过 **Properties** 更改维度的名称和其他时间表元数据。

将维度的名称更改为 **Time** 和 **Data**。

```
bikeData.Properties.DimensionNames = {'Time' 'Data'};
bikeData.Properties
```

```
ans =
TimetableProperties with properties:

    Description: "
        UserData: []
        DimensionNames: {'Time' 'Data'}
        VariableNames: {'Day' 'Total' 'Westbound' 'Eastbound'}
```

```

VariableDescriptions: {}
VariableUnits: {}
VariableContinuity: []
RowTimes: [9387x1 datetime]
StartTime: 2015-06-24 00:00:00
SampleRate: NaN
TimeStep: NaN
CustomProperties: No custom properties are set.
Use addprop and rmprop to modify CustomProperties.

```

显示时间表的前八行。

```
head(bikeData)
```

```
ans=8×5 timetable
```

Time	Day	Total	Westbound	Eastbound
2015-06-24 00:00:00	Wednesday	13	9	4
2015-06-24 01:00:00	Wednesday	3	3	0
2015-06-24 02:00:00	Wednesday	1	1	0
2015-06-24 03:00:00	Wednesday	1	1	0
2015-06-24 04:00:00	Wednesday	1	1	0
2015-06-24 05:00:00	Wednesday	7	3	4
2015-06-24 06:00:00	Wednesday	36	6	30
2015-06-24 07:00:00	Wednesday	141	13	128

确定最晚行时间与最早行时间之间经过的天数。在一次引用一个变量时，可以通过圆点表示法访问变量。

```
elapsedTime = max(bikeData.Time) - min(bikeData.Time)
```

```
elapsedTime = duration
```

```
9383:30:00
```

```
elapsedTime.Format = 'd'
```

```
elapsedTime = duration
```

```
390.98 days
```

要确定某给定天的典型自行车计数，请计算总自行车数的均值、向西行进和向东行进的自行车数。

通过使用花括号对 **bikeData** 的内容进行索引，将数值数据以矩阵的形式返回。显示前八行。要访问多个变量，须使用标准表下标。

```
counts = bikeData{:,2:end};
counts(1:8,:)
```

```
ans = 8×3
```

```

13   9   4
3   3   0
1   1   0
1   1   0
1   1   0
7   3   4

```

```
36   6   30
141  13  128
```

由于均值仅适合数值数据，因此可以使用 `vartype` 函数选择数值变量。`vartype` 可能比手动对表或时间表进行索引以选择变量更方便。计算均值并忽略 NaN 值。

```
counts = bikeData{:,vartype('numeric')};
mean(counts,'omitnan')
```

```
ans = 1×3
```

```
49.8860 24.2002 25.6857
```

### 按日期和一天中的时间选择数据

为了确定有多少人在假期中骑自行车，将检查在 7 月 4 日这一假日中的数据。按行时间对 2015 年 7 月 4 日的时间表建立索引。当对行时间建立索引时，必须与时间完全匹配。可以将时间索引指定为 `datetime` 或 `duration` 值，或者指定为可以转换为日期和时间的字符串。可以将多个时间指定为数组。

使用特定的日期和时间对 `bikeData` 进行索引以提取 2015 年 7 月 4 日的数据。如果仅指定日期，则时间假定为午夜，即 00:00:00。

```
bikeData('2015-07-04',:)
```

```
ans=1×5 timetable
Time      Day    Total  Westbound  Eastbound
_____
2015-07-04 00:00:00 Saturday     8       7       1
```

```
d = {'2015-07-04 08:00:00','2015-07-04 09:00:00'};
bikeData(d,:)
```

```
ans=2×5 timetable
Time      Day    Total  Westbound  Eastbound
_____
2015-07-04 08:00:00 Saturday    15       3      12
2015-07-04 09:00:00 Saturday    21       4      17
```

如果使用此策略提取一整天的数据，则是非常繁琐的工作。您也可以指定时间范围，而不对特定的时间建立索引。要创建时间范围下标来作为帮助，请使用 `timerange` 函数。

使用 2015 年 7 月 4 日一整天作为时间范围来对时间表建立下标。指定起始时间为 7 月 4 日午夜，结束时间为 7 月 5 日午夜。默认情况下，`timerange` 涵盖了从起始时间直至结束时间（但不包含结束时间）的所有时间。绘制这一天内各时间段的自行车计数。

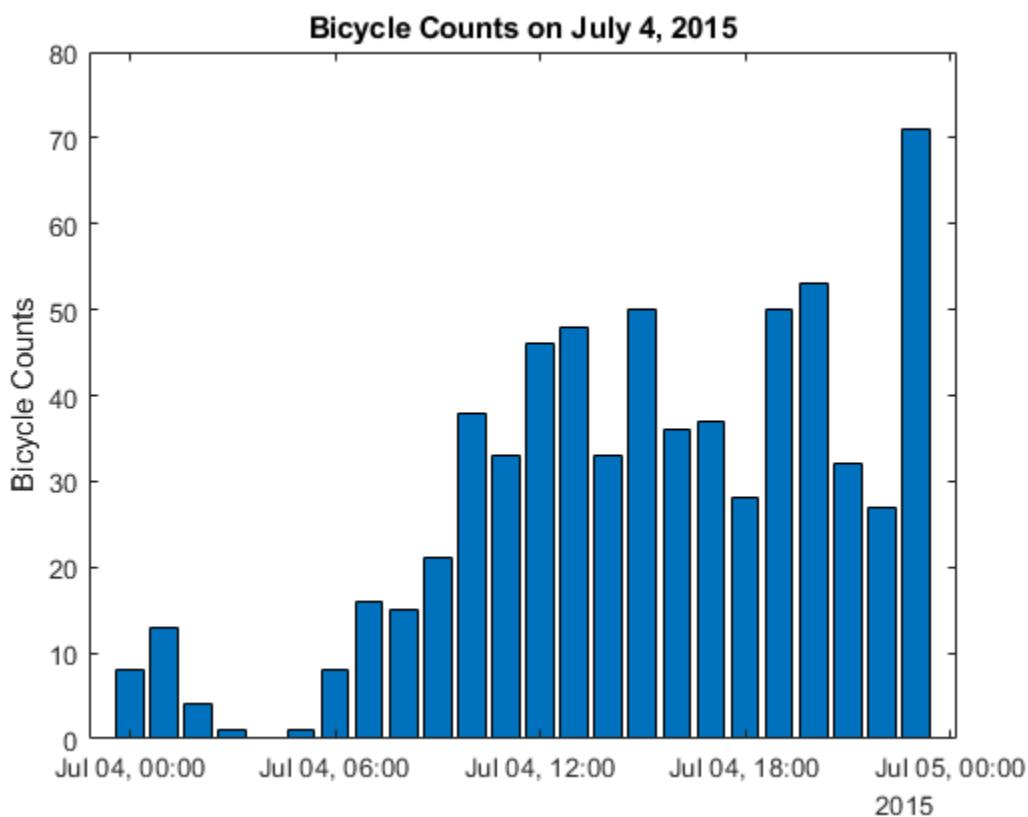
```
tr = timerange('2015-07-04','2015-07-05');
jul4 = bikeData(tr,'Total');
head(jul4)
```

```
ans=8×2 timetable
Time      Total
_____

```

Time	Total
2015-07-04 00:00:00	8
2015-07-04 01:00:00	13
2015-07-04 02:00:00	4
2015-07-04 03:00:00	1
2015-07-04 04:00:00	0
2015-07-04 05:00:00	1
2015-07-04 06:00:00	8
2015-07-04 07:00:00	16

```
bar(jul4.Time,jul4.Total)
ylabel('Bicycle Counts')
title('Bicycle Counts on July 4, 2015')
```

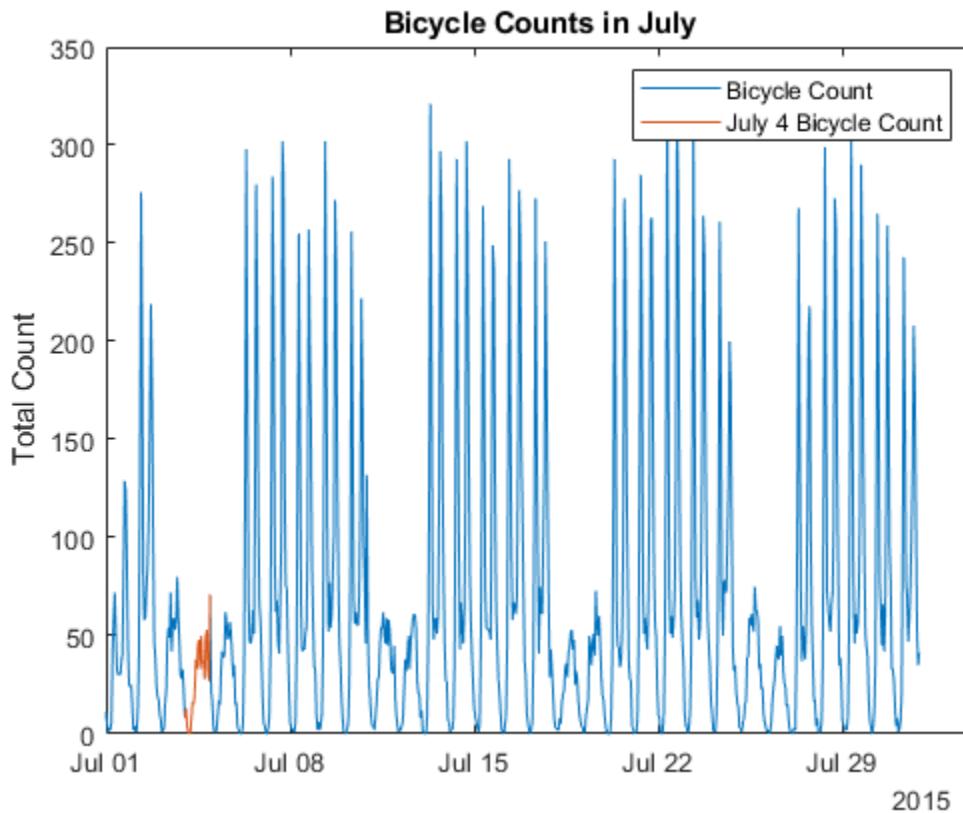


从绘图中可看出，整天的自行车流量较高，下午的流量比较平稳。由于当天非工作日，许多公司未上班，因此该绘图并未反映典型的上下班时间内的交通情况。峰值出现在夜晚较晚时间可能归因于在天黑后进行的烟花燃放庆祝活动。要更准确地确定这些趋势，应将当天的数据与典型一天中的数据进行比较。

将 7 月 4 日的数据与 7 月中其他日期的数据进行比较。

```
jul = bikeData(timerange('2015-07-01','2015-08-01'),:);
plot(jul.Time,jul.Total)
hold on
plot(jul4.Time,jul4.Total)
ylabel('Total Count')
title('Bicycle Counts in July')
```

```
hold off
legend('Bicycle Count','July 4 Bicycle Count')
```



该绘图显示的差异可以归因于工作日与周末之间的流量变化。7月4日和5日的流量模式与周末的流量模式一致。7月5日是周一，但通常作为假日进行观察。通过进一步的预处理和分析，可更准确地确定这些趋势。

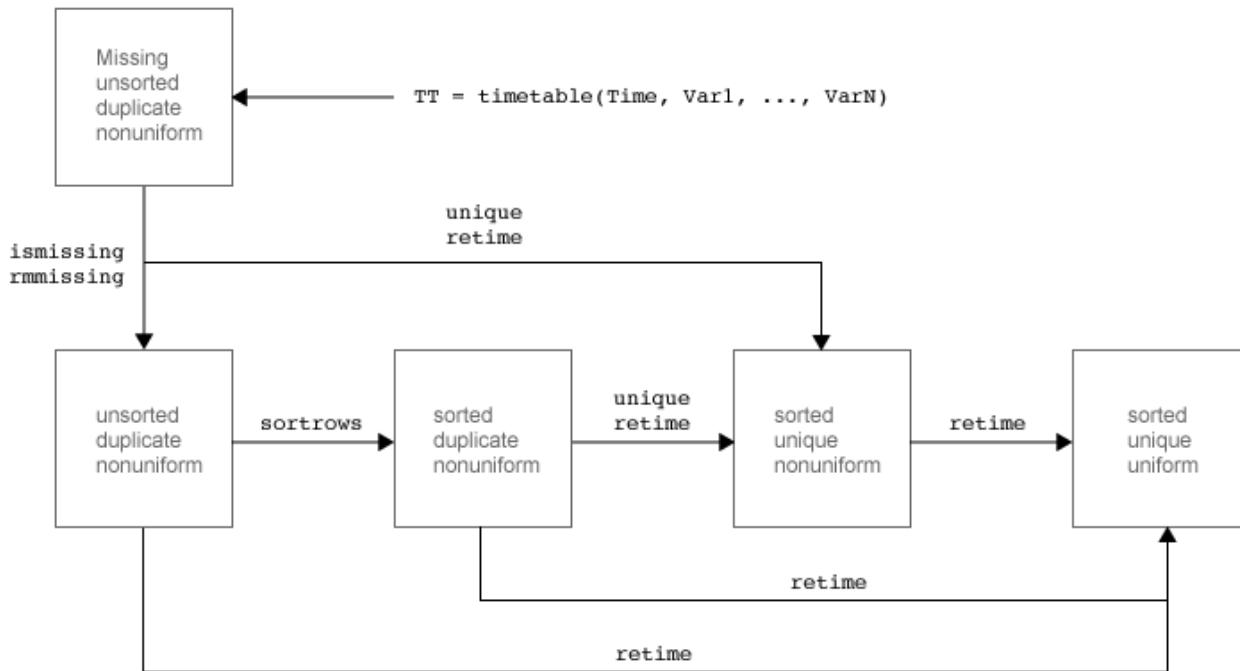
### 使用 timetable 预处理时间和数据

时间戳数据集通常是杂乱的，可能包含异常情况或错误。时间表非常适合解决异常情况和错误。

时间表的行时间无需采用任何特定顺序。它可以包含未按行时间排序的行。时间表也可以包含多个具有相同行时间的行，即使这些行具有不同的数据值也是如此。即使行时间已排序并且是唯一的，也会因不同大小的时间步而异。时间表甚至可以包含 NaT 或 NaN 值以指示缺失的行时间。

**timetable** 数据类型提供了很多种不同的方式来解决时间缺失、重复或不均匀问题。您也可以对数据重采样或聚合数据以创建一个规则时间表。当时间表为规则时间表时，其行时间已排序并且是唯一的，并且行时间之间具有均匀或等间距的时间步。

- 要查找缺失的行时间，请使用 `ismissing`。
- 要删除缺失的时间和数据，请使用 `rmmissing`。
- 要按行时间对时间表进行排序，请使用 `sortrows`。
- 要使时间表具有唯一和已排序的行时间，请使用 `unique` 和 `retime`。
- 要创建规则时间表，请指定一个间距均匀的时间向量并使用 `retime`。



### 按时间顺序排序

确定该时间表是否已排序。如果时间表的行时间按升序列出，则说明它已进行了排序。

```
issorted(bikeData)
```

```
ans = logical  
0
```

对时间表排序。`sortrows` 函数按照行的行时间（从最早时间到最晚时间）对行进行排序。如果存在具有重复行时间的行，则 `sortrows` 会将所有重复项复制到输出。

```
bikeData = sortrows(bikeData);  
issorted(bikeData)
```

```
ans = logical  
1
```

### 确定并删除缺失的时间和数据

时间表可以在其变量或行时间中包含数据缺失指示符。例如，可以将缺失的数值指示为 `Nan`，将缺失的日期时间值指示为 `NaT`。可以分别使用 `standardizeMissing`、`ismissing`、`rmmissing` 和 `fillmissing` 函数来分配、查找、删除和填充缺失值。

查找和统计时间表变量中的缺失值。在下面的示例中，缺失值指示未收集任何数据的情况。

```
missData = ismissing(bikeData);
sum(missData)
```

```
ans = 1×4
```

```
1 3 3 3
```

来自 `ismissing` 的输出是一个大小与表相同的 `logical` 矩阵，将缺失的数据值标识为 `true`。显示具有缺失数据指示符的任何行。

```
idx = any(missData,2);
bikeData(idx,:)
```

```
ans=3×5 timetable
```

Time	Day	Total	Westbound	Eastbound
2015-08-03 00:00:00	Monday	NaN	NaN	NaN
2015-08-03 01:00:00	Monday	NaN	NaN	NaN
NaT	<undefined>	NaN	NaN	NaN

`ismissing(bikeData)` 仅在时间表变量中查找缺失的数据，而并不查找缺失的时间。要查找缺失的行时间，请对行时间调用 `ismissing`。

```
missTimes = ismissing(bikeData.Time);
bikeData(missTimes,:)
```

```
ans=2×5 timetable
```

Time	Day	Total	Westbound	Eastbound
NaT	<undefined>	NaN	NaN	NaN
NaT	Friday	6	3	3

在本示例中，缺失的时间或数据值指示测量有误，可以将这些数据排除。使用 `rmmissing` 删除包含缺失数据值和缺失行时间的表行。

```
bikeData = rmmissing(bikeData);
sum(ismissing(bikeData))
```

```
ans = 1×4
```

```
0 0 0 0
```

```
sum(ismissing(bikeData.Time))
```

```
ans = 0
```

### 删除重复的时间和数据

确定是否存在重复的时间和/或重复的数据行。您可能需要排除多余的重复项，因为这些项也会被视为测量错误。通过找出排序时间的差异完全为零的位置来确定重复的时间。

```
idx = diff(bikeData.Time) == 0;
dup = bikeData.Time(idx)
```

```
dup = 3x1 datetime array
2015-08-21 00:00:00
2015-11-19 23:00:00
2015-11-19 23:00:00
```

三个时间是重复的，并且 2015 年 11 月 19 日重复两次。检查与重复时间关联的数据。

```
bikeData(dup(1,:))
```

```
ans=2×5 timetable
Time Day Total Westbound Eastbound
_____
2015-08-21 00:00:00 Friday 14 9 5
2015-08-21 00:00:00 Friday 11 7 4
```

```
bikeData(dup(2,:))
```

```
ans=3×5 timetable
Time Day Total Westbound Eastbound
_____
2015-11-19 23:00:00 Thursday 17 15 2
2015-11-19 23:00:00 Thursday 17 15 2
2015-11-19 23:00:00 Thursday 17 15 2
```

第一项有重复时间，但是没有重复数据，其他项则完全重复。时间表的不同行如果包含相同的行时间与数据值，则被视为重复。可以使用 `unique` 删除时间表中的重复行。`unique` 函数还会按照行的行时间对行进行排序。

```
bikeData = unique(bikeData);
```

具有重复时间但没有重复数据的行需要提供某些解释。检查围绕这些时间的数据。

```
d = dup(1) + hours(-2:2);
bikeData(d,:)
```

```
ans=5×5 timetable
Time Day Total Westbound Eastbound
_____
2015-08-20 22:00:00 Thursday 40 30 10
2015-08-20 23:00:00 Thursday 25 18 7
2015-08-21 00:00:00 Friday 11 7 4
2015-08-21 00:00:00 Friday 14 9 5
2015-08-21 02:00:00 Friday 6 5 1
```

在本例中，重复的时间可能有误，因为其数据和前后的时间都是连贯的。尽管它看起来应该是 01:00:00，但不能确定它真正的时间。可以累加数据来说明两个时间点的数据。

```
sum(bikeData{dup(1),2:end})
```

```
ans = 1×3
```

```
25 16 9
```

这仅仅是可以手动执行的一种情况。但是，如果有很多行，则可以使用 `retime` 函数来执行此计算。使用 `sum` 函数累加唯一时间对应的数据以计算总和。总和适合于数值数据，但不适合于时间表中的分类数据。使用 `vartype` 确定数值变量。

```
vt = vartype('numeric');
t = unique(bikeData.Time);
numData = retime(bikeData(:,vt),t,'sum');
head(numData)

ans=8×4 timetable
    Time      Total  Westbound  Eastbound
    _____  _____  _____  _____
2015-06-24 00:00:00    13      9      4
2015-06-24 01:00:00     3      3      0
2015-06-24 02:00:00     1      1      0
2015-06-24 03:00:00     1      1      0
2015-06-24 04:00:00     1      1      0
2015-06-24 05:00:00     7      3      4
2015-06-24 06:00:00    36      6     30
2015-06-24 07:00:00   141     13    128
```

不能对分类数据计算总和，但由于一个标签代表一整天，因此取每一天的第一个值。可以使用相同的时间向量再次执行 `retime` 操作，并将时间表串联到一起。

```
vc = vartype('categorical');
catData = retime(bikeData(:,vc),t,'firstvalue');
bikeData = [catData,numData];
bikeData(d,:)

ans=4×5 timetable
    Time      Day      Total  Westbound  Eastbound
    _____  _____  _____  _____  _____
2015-08-20 22:00:00 Thursday    40      30      10
2015-08-20 23:00:00 Thursday    25      18       7
2015-08-21 00:00:00 Friday     25      16       9
2015-08-21 02:00:00 Friday      6       5       1
```

### 检查时间间隔的均匀性

数据似乎具有相同的时间步长，即一小时。要确定是否时间表中的所有行时间都是这种情况，请使用 `isregular` 函数。对于已排序、间隔均匀的时间（单调递增），并且没有重复或缺失的时间（`NaT` 或 `NaN`），`isregular` 函数返回 `true`。

```
isregular(bikeData)
```

```
ans = logical
0
```

输出 `0` 或 `false` 表示时间表中的时间不是间隔均匀的。更详细地探索时间间隔。

```
dt = diff(bikeData.Time);
[min(dt); max(dt)]
```

```
ans = 2x1 duration array
00:30:00
```

03:00:00

要将时间表放到均匀时间间隔中，请使用 `retime` 或 `synchronize` 并指定相关的时间间隔。

### 确定每天自行车数量

使用 `retime` 函数确定每天的自行车计数。使用 `sum` 方法累加每天的计数数据。此操作适合于数值数据，但不适合于时间表中的分类数据。使用 `vartype` 按数据类型标识变量。

```
dayCountNum = retime(bikeData(:,vt),'daily','sum');
head(dayCountNum)
```

```
ans=8×4 timetable
Time      Total  Westbound  Eastbound
_____
2015-06-24 00:00:00  2141    1141     1000
2015-06-25 00:00:00  2106    1123     983
2015-06-26 00:00:00  1748    970      778
2015-06-27 00:00:00  695     346      349
2015-06-28 00:00:00  153     83       70
2015-06-29 00:00:00  1841    978      863
2015-06-30 00:00:00  2170    1145     1025
2015-07-01 00:00:00  997     544      453
```

如上所述，可以再次执行 `retime` 操作，以使用合适方法表示分类数据并将时间表串联到一起。

```
dayCountCat = retime(bikeData(:,vc),'daily','firstvalue');
dayCount = [dayCountCat,dayCountNum];
head(dayCount)
```

```
ans=8×5 timetable
Time      Day      Total  Westbound  Eastbound
_____
2015-06-24 00:00:00  Wednesday  2141    1141     1000
2015-06-25 00:00:00  Thursday   2106    1123     983
2015-06-26 00:00:00  Friday    1748    970      778
2015-06-27 00:00:00  Saturday   695     346      349
2015-06-28 00:00:00  Sunday    153     83       70
2015-06-29 00:00:00  Monday    1841    978      863
2015-06-30 00:00:00  Tuesday   2170    1145     1025
2015-07-01 00:00:00  Wednesday  997     544      453
```

### 同步自行车计数和天气数据

通过使用天气数据比较自行车计数，来确定天气对骑自行车行为的影响。加载天气数据时间表，其中包括马萨诸塞州波士顿市的历史天气数据，包括风暴活动。

```
load BostonWeatherData
head(weatherData)

ans=8×4 timetable
Time      TemperatureF  Humidity  Events
_____

```

01-Jul-2015	72	78	Thunderstorm
02-Jul-2015	72	60	None
03-Jul-2015	70	56	None
04-Jul-2015	67	75	None
05-Jul-2015	72	67	None
06-Jul-2015	74	69	None
07-Jul-2015	75	77	Rain
08-Jul-2015	79	68	Rain

要汇总时间表中的时间和变量，请使用 **summary** 函数。

```
summary(weatherData)
```

RowTimes:

```
Time: 383x1 datetime
Values:
Min      01-Jul-2015
Median   08-Jan-2016
Max      17-Jul-2016
TimeStep  24:00:00
```

Variables:

TemperatureF: 383x1 double

Values:

```
Min      2
Median   55
Max      85
```

Humidity: 383x1 double

Values:

```
Min      29
Median   64
Max      97
```

Events: 383x1 categorical

Values:

```
Fog      7
Hail     1
Rain    108
Rain-Snow 4
Snow    18
Thunderstorm 12
None    233
```

使用 **synchronize** 将自行车数据和天气数据合并到一个公共时间向量。可以使用 **synchronize** 函数参考页上记录的任一方法来重采样或聚合时间表数据。

将两个时间表中的数据同步到一个公共时间向量，即基于两个时间表各自的时间向量的交集来构造。

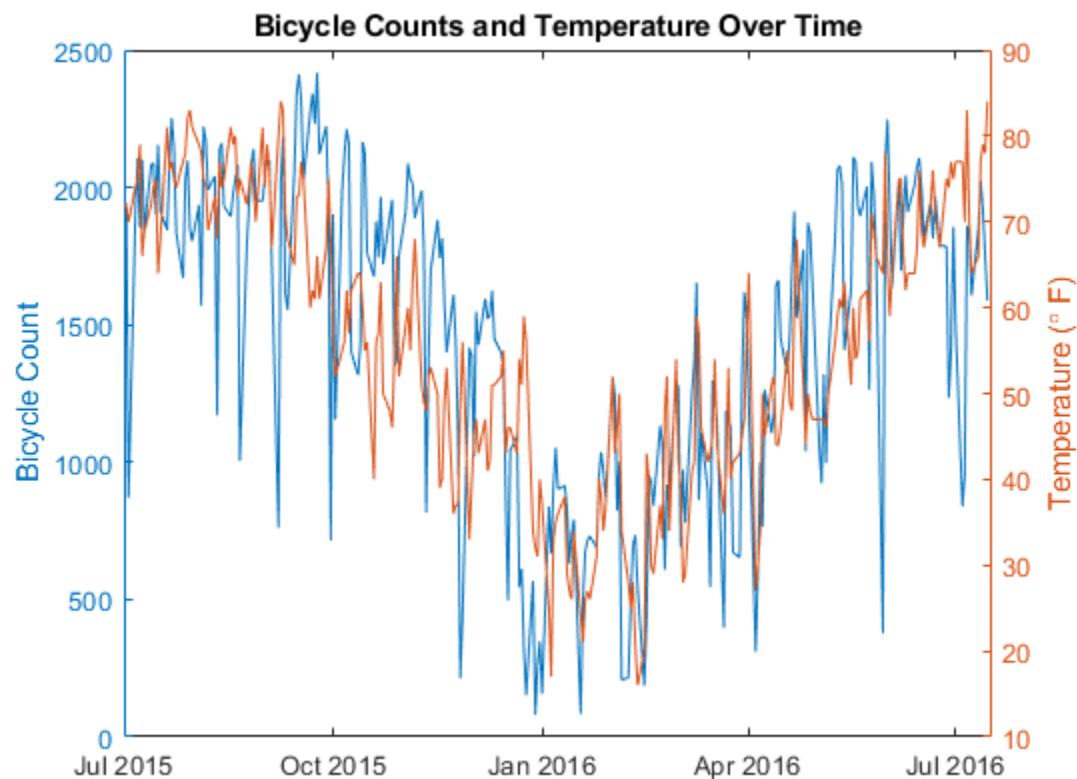
```
data = synchronize(dayCount,weatherData,'intersection');
head(data)
```

ans=8×8 timetable

Time	Day	Total	Westbound	Eastbound	TemperatureF	Humidity	Events
2015-07-01 00:00:00	Wednesday	997	544	453	72	78	Thunderstorm
2015-07-02 00:00:00	Thursday	1943	1033	910	72	60	None
2015-07-03 00:00:00	Friday	870	454	416	70	56	None
2015-07-04 00:00:00	Saturday	669	328	341	67	75	None
2015-07-05 00:00:00	Sunday	702	407	295	72	67	None
2015-07-06 00:00:00	Monday	1900	1029	871	74	69	None
2015-07-07 00:00:00	Tuesday	2106	1140	966	75	77	Rain
2015-07-08 00:00:00	Wednesday	1855	984	871	79	68	Rain

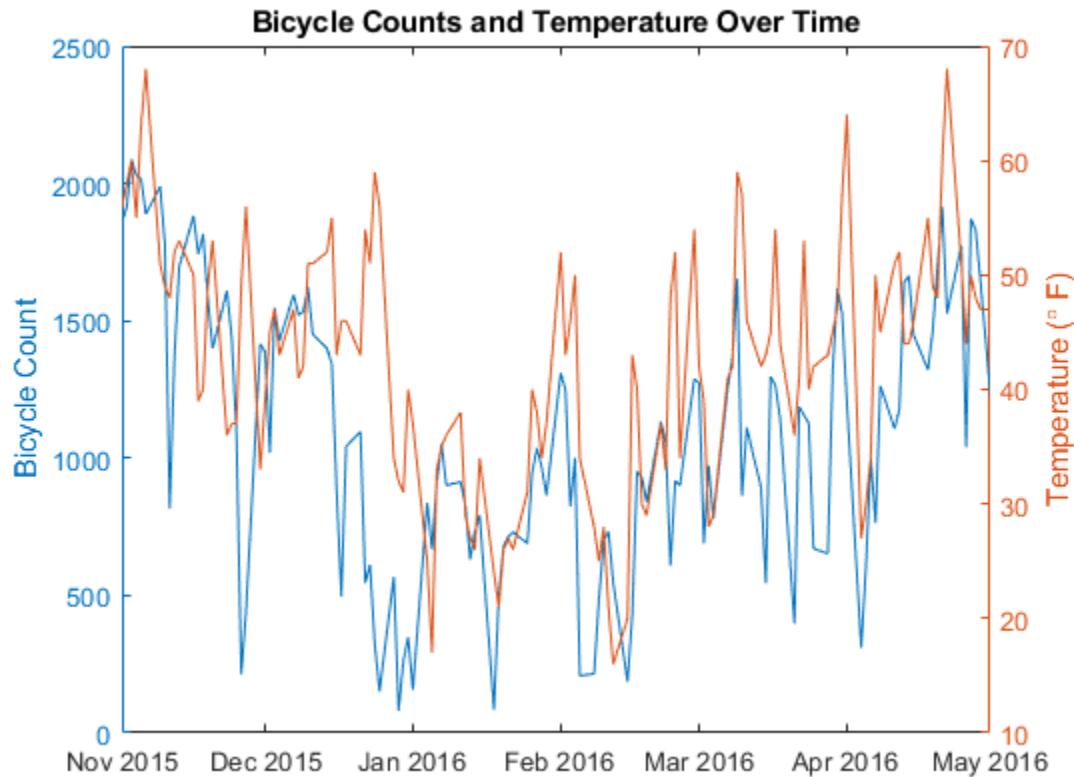
在单独的 y 轴上比较自行车交通计数和室外温度以确定趋势。删除数据中的周末以进行可视化。

```
idx = ~isweekend(data.Time);
weekdayData = data(idx,{'TemperatureF','Total'});
figure
yyaxis left
plot(weekdayData.Time, weekdayData.Total)
ylabel('Bicycle Count')
yyaxis right
plot(weekdayData.Time,weekdayData.TemperatureF)
ylabel('Temperature (\circ F)')
title('Bicycle Counts and Temperature Over Time')
xlim([min(data.Time) max(data.Time)])
```



绘图显示，交通和天气数据可能遵循相似的趋势。放大绘图。

```
xlim([datetime('2015-11-01'),datetime('2016-05-01')])
```



趋势是类似的，指示天气越冷，骑自行车的人越少。

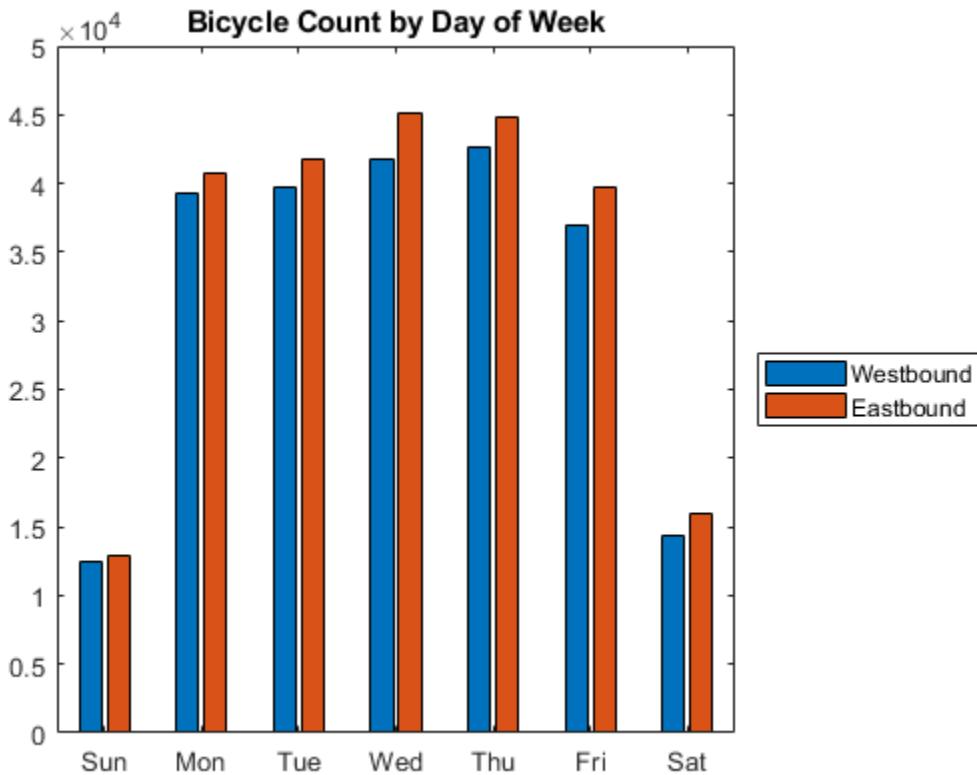
### 按星期几和一天中的时间进行分析

基于不同的时间间隔检查数据，例如星期几和一天中的时间。使用 varfun 确定每天的总计数，以对变量执行分组计算。使用函数句柄指定 sum 函数，使用名称-值对组指定分组变量和首选输出类型。

```
byDay = varfun(@sum,bikeData,'GroupingVariables','Day',...
    'OutputFormat','table')
```

Day	GroupCount	sum_Total	sum_Westbound	sum_Eastbound
Sunday	1344	25315	12471	12844
Monday	1343	79991	39219	40772
Tuesday	1320	81480	39695	41785
Wednesday	1344	86853	41726	45127
Thursday	1344	87516	42682	44834
Friday	1342	76643	36926	39717
Saturday	1343	30292	14343	15949

```
figure
bar(byDay{:,{'sum_Westbound','sum_Eastbound'}})
legend({'Westbound','Eastbound'},'Location','eastoutside')
xticklabels({'Sun','Mon','Tue','Wed','Thu','Fri','Sat'})
title('Bicycle Count by Day of Week')
```



条形图指示，交通在工作日更为拥堵。此外，向东行进和向西行进的流量数据存在差异。这可能指示，人们进入和离开城市时，倾向于采用不同的路线。另外一种可能性是一些人在某一天进入城市，在另一天离开城市。

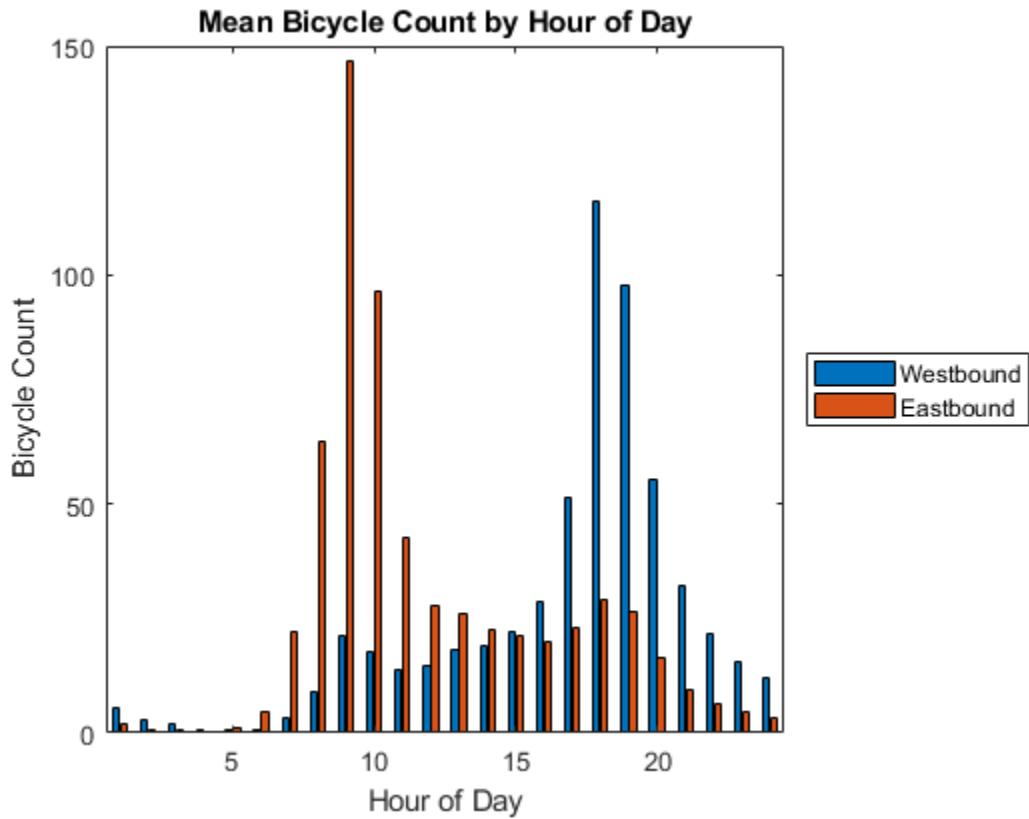
确定一天中的小时并使用 varfun 按组计算。

```
bikeData.HrOfDay = hour(bikeData.Time);
byHr = varfun(@mean,bikeData(:,{'Westbound','Eastbound','HrOfDay'}),...
    'GroupingVariables','HrOfDay','OutputFormat','table');
head(byHr)
```

```
ans=8×4 table
  HrOfDay GroupCount mean_Westbound mean_Eastbound
  _____ _____ _____ _____
  0         389      5.4396     1.7686
  1         389      2.7712     0.87147
  2         391      1.8696     0.58312
  3         391      0.7468     0.289
  4         391      0.52685    1.0026
  5         391      0.70588    4.7494
  6         391      3.1228    22.097
  7         391      9.1176    63.54
```

```
bar(byHr{:,'mean_Westbound','mean_Eastbound'})
legend('Westbound','Eastbound','Location','eastoutside')
xlabel('Hour of Day')
```

```
ylabel('Bicycle Count')
title('Mean Bicycle Count by Hour of Day')
```



在上午约 9:00 和下午约 5:00 是流量高峰期。此外，向东行进和向西行进的趋势是明显不同的。通常情况下，向东行进的目标是围绕剑桥区的居住区和大学。向西行进的目标是波士顿。

在一天中的晚些时候，相对于向东方向，向西方向的交通更为拥堵。这可能指示，由于餐馆位于该区域中，这可能是大学学生活动和交通导致。按星期几和一天中小时确定趋势。

```
byHrDay = varfun(@sum,bikeData,'GroupingVariables',{'HrOfDay','Day'},...
    'OutputFormat','table');
head(byHrDay)

ans=8×6 table
  HrOfDay    Day    GroupCount    sum_Total    sum_Westbound    sum_Eastbound
  _____    ____    _____    _____    _____    _____
  0    Sunday      56      473      345      128
  0    Monday       55      202      145       57
  0    Tuesday      55      297      213       84
  0    Wednesday     56      374      286       88
  0    Thursday      56      436      324      112
  0    Friday        55      442      348       94
  0    Saturday      56      580      455      125
  1    Sunday        56      333      259       74
```

若要安排时间表以便将星期几作为变量，可使用 `unstack` 函数。

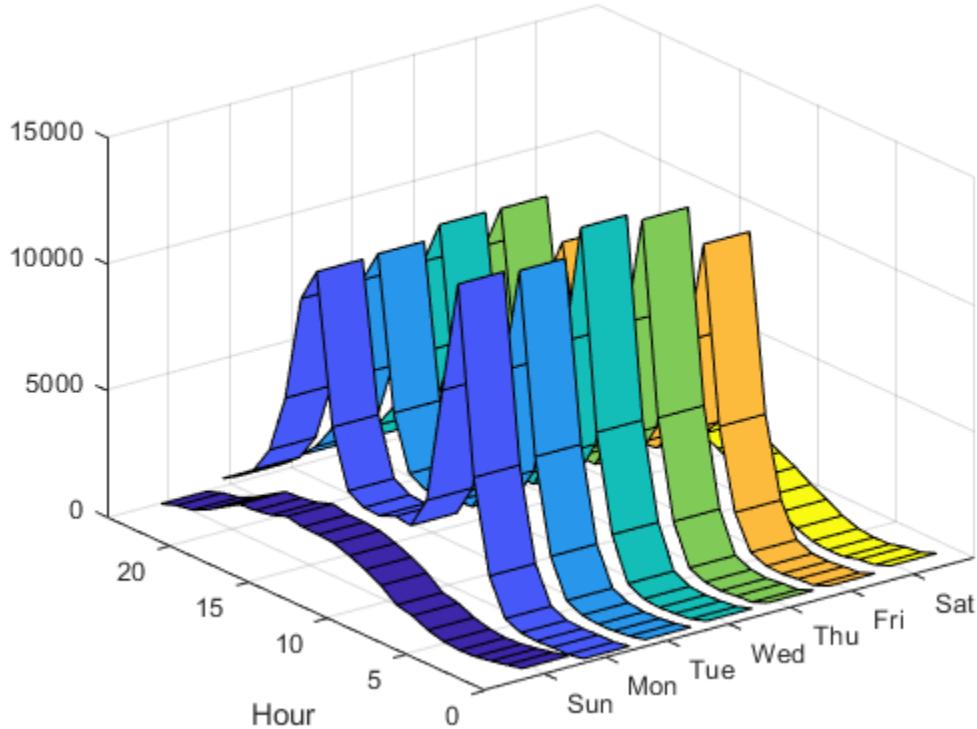
```
hrAndDayWeek = unstack(byHrDay(:,{'HrOfDay','Day','sum_Total'}),'sum_Total','Day');
head(hrAndDayWeek)
```

ans=8×8 table

	HrOfDay	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
0	473	202	297	374	436	442	580	
1	333	81	147	168	173	183	332	
2	198	77	68	93	128	141	254	
3	86	41	43	44	50	61	80	
4	51	81	117	101	108	80	60	
5	105	353	407	419	381	340	128	
6	275	1750	1867	2066	1927	1625	351	
7	553	5355	5515	5818	5731	4733	704	

```
ribbon(hrAndDayWeek.HrOfDay,hrAndDayWeek{:,2:end})
ylim([0 24])
xlim([0 8])
xticks(1:7)
xticklabels({'Sun','Mon','Tue','Wed','Thu','Fri','Sat'})
ylabel('Hour')
title('Bicycle Count by Hour and Day of Week')
```

Bicycle Count by Hour and Day of Week



从周一到周五的正常工作日也有类似的趋势，在上下班高峰时段达到峰值，夜间交通逐渐减少。周五的交通拥堵情况较少，尽管这一天的总体趋势与其他工作日类似。周六与周日的情况大同小异，无高峰时段峰值，日间晚些时候交通流量有所增加。从周一到周五深夜时段的趋势类似，周五深夜交通流量较少。

### 分析高峰时段中的交通

要确定一天中的时间的总体趋势，请按高峰时段划分数据。使用 `discretize` 函数时，可以使用一天中的不同时间或不同时间单位。例如，按 AM、AMRush、Day、PMRush、PM 将数据划分为组。然后，使用 `varfun` 按组计算均值。

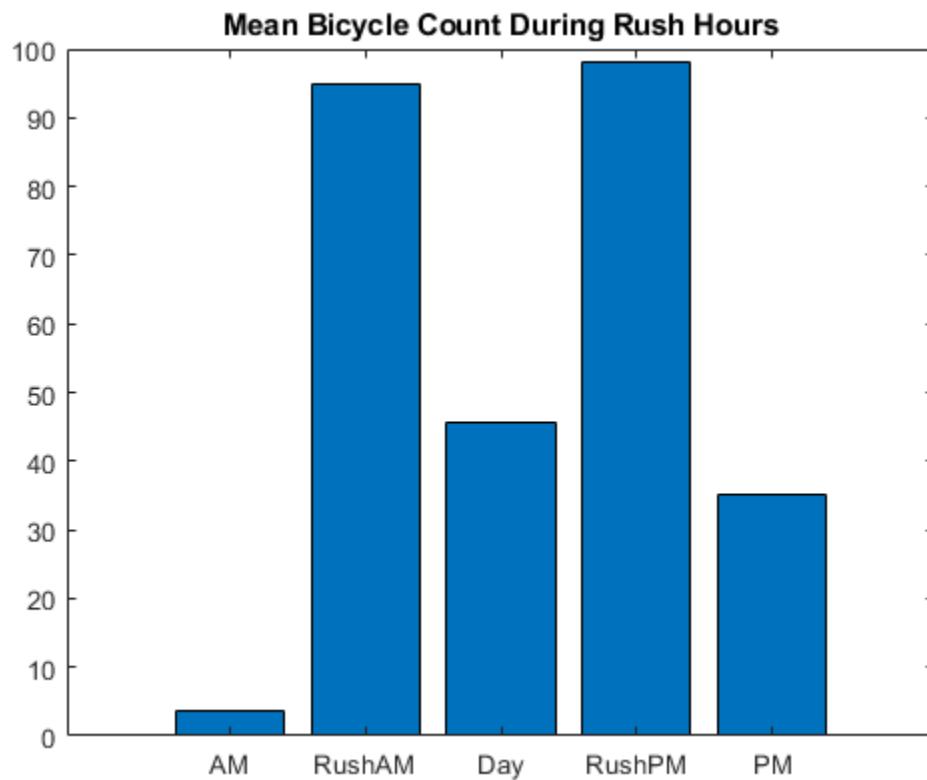
```

bikeData.HrLabel = discretize(bikeData.HrOfDay,[0,6,10,15,19,24],'categorical',...
    {'AM','RushAM','Day','RushPM','PM'});
byHrBin = varfun(@mean,bikeData(:,{:,'Total','HrLabel'}),'GroupingVariables','HrLabel',...
    'OutputFormat','table')

byHrBin=5×3 table
    HrLabel    GroupCount    mean_Total
    _____    _____    _____
    AM          2342      3.5508
    RushAM      1564      94.893
    Day          1955      45.612
    RushPM      1564      98.066
    PM          1955      35.198

bar(byHrBin.mean_Total)
cats = categories(byHrBin.HrLabel);
xticklabels(cats)
title('Mean Bicycle Count During Rush Hours')

```



通常情况下，在此区域中早晚高峰时段的交通量是一天中其他时间的交通量的两倍。在此区域中的清早的交通量很少，但在傍晚和晚间的交通量仍较大，相当于一天中除早晚高峰时段外的交通量。

## 另请参阅

`datetime` | `head` | `retime` | `rmmissing` | `sortrows` | `summary` | `table2timetable` | `timerange` |  
`timetable` | `unstack` | `varfun`

## 相关示例

- “表示 MATLAB 中的日期和时间”（第 7-2 页）
- “创建时间表”（第 10-2 页）
- “对时间表中的数据进行重采样和聚合”（第 10-5 页）
- “清理包含缺失、重复或不均匀时间的时间表”（第 10-23 页）
- “按行时间和变量类型选择时间表数据”（第 10-18 页）



# 结构体

---

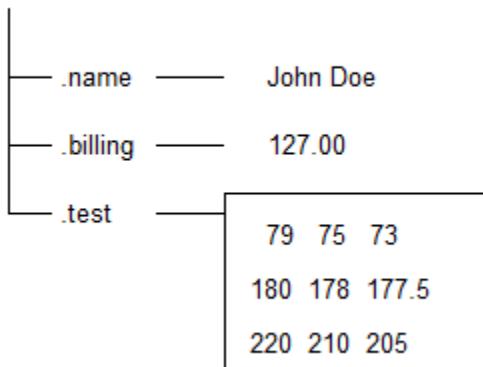
- “创建结构体数组”（第 11-2 页）
- “访问结构体数组中的数据”（第 11-5 页）
- “串联结构体”（第 11-9 页）
- “基于变量生成字段名称”（第 11-11 页）
- “访问嵌套结构体中的数据”（第 11-12 页）
- “访问非标量结构体数组的元素”（第 11-14 页）
- “结构体数组中数据的组织方法”（第 11-16 页）
- “结构体数组的内存要求”（第 11-18 页）

## 创建结构体数组

下面的示例说明了如何创建结构体数组。结构体是使用被称为字段的数据容器将相关数据组合在一起的一种数据类型。每个字段都可以包含任意类型或任意大小的数据。

将患者记录存储在含有字段 `name`、`billing` 和 `test` 的标量结构体中。

`patient`



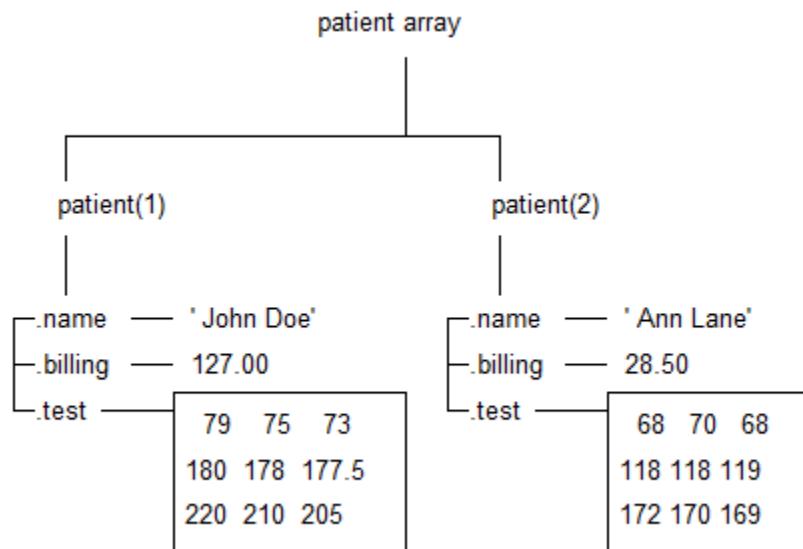
```

patient(1).name = 'John Doe';
patient(1).billing = 127.00;
patient(1).test = [79, 75, 73; 180, 178, 177.5; 220, 210, 205];
patient
  
```

```

patient = struct with fields:
  name: 'John Doe'
  billing: 127
  test: [3x3 double]
  
```

通过在数组名称后添加下标，可在此数组中添加其他患者的记录。



```
patient(2).name = 'Ann Lane';
patient(2).billing = 28.50;
patient(2).test = [68, 70, 68; 118, 118, 119; 172, 170, 169];
patient

patient=2×3 struct
  name
  billing
  test
```

数组中的每条患者记录都是 `struct` 类的结构体。由结构体构成的数组通常称为结构体数组。与其他 MATLAB 数组类似，结构体数组可以具有任意维度。

结构体数组具有下列属性：

- 数组中的所有结构体都具有相同数目的字段。
- 所有结构体都具有相同的字段名称。
- 不同结构体中的同名字段可包含不同类型或大小的数据。

数组中新结构体的任何未指定字段均包含空数组。

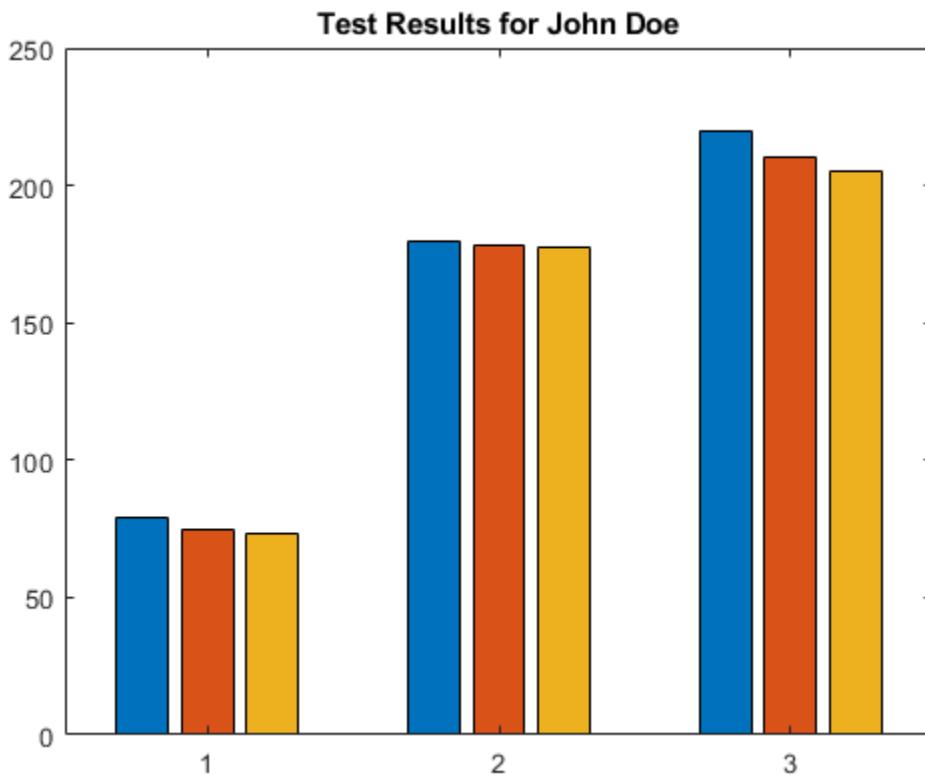
```
patient(3).name = 'New Name';
patient(3)
```

```
ans = struct with fields:
  name: 'New Name'
  billing: []
  test: []
```

访问结构体数组中的数据，看一下第一名患者有多少欠款，并根据其测试结果创建一个条形图。

```
amount_due = patient(1).billing
amount_due = 127

bar(patient(1).test)
title(['Test Results for ', patient(1).name])
```



### 另请参阅

#### 相关示例

- “访问结构体数组中的数据” (第 11-5 页)
- “创建元胞数组” (第 12-3 页)
- “创建和使用表” (第 9-2 页)

#### 详细信息

- “元胞数组与结构体数组” (第 12-16 页)
- “使用表的好处” (第 9-55 页)

## 访问结构体数组中的数据

此示例演示了如何访问结构体数组的内容。为了运行此示例中的代码，先将多个变量加载到名为 S 的结构体中。

### 访问标量结构体中的数据

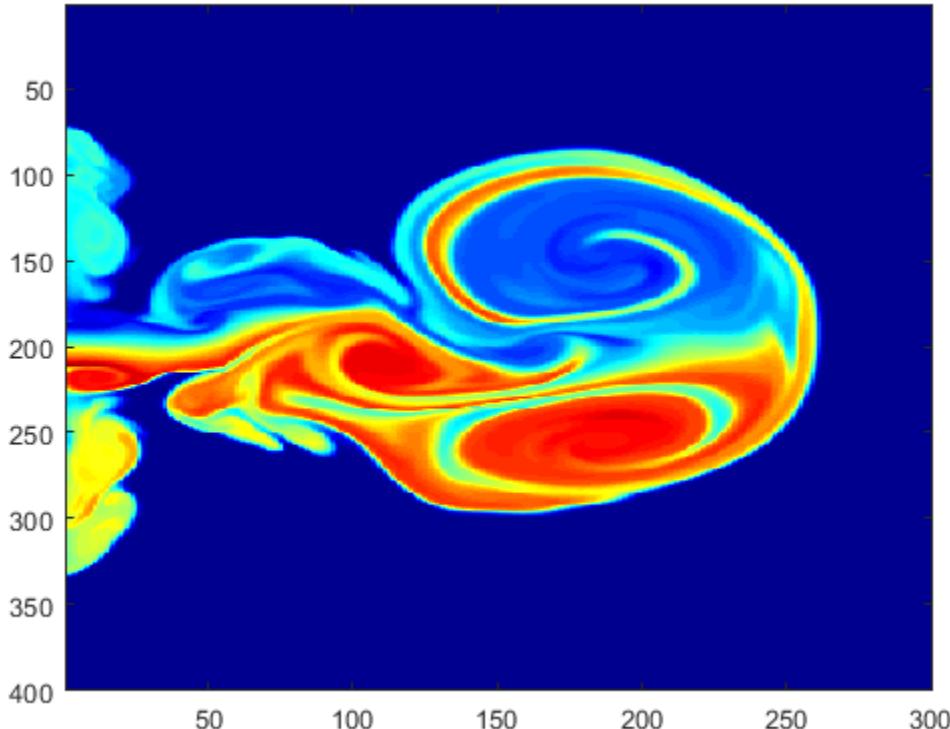
首先，将数据从 `flujet.mat` 加载到标量结构体 S 中。文件 `flujet.mat` 包含一幅仿真天体物理射流经历紊流的图像。

```
S = load('flujet.mat')  
  
S = struct with fields:  
    X: [400x300 double]  
    map: [64x3 double]  
    caption: [2x32 char]
```

该文件中的变量（X、`caption` 和 `map`）现在为结构体中的字段。

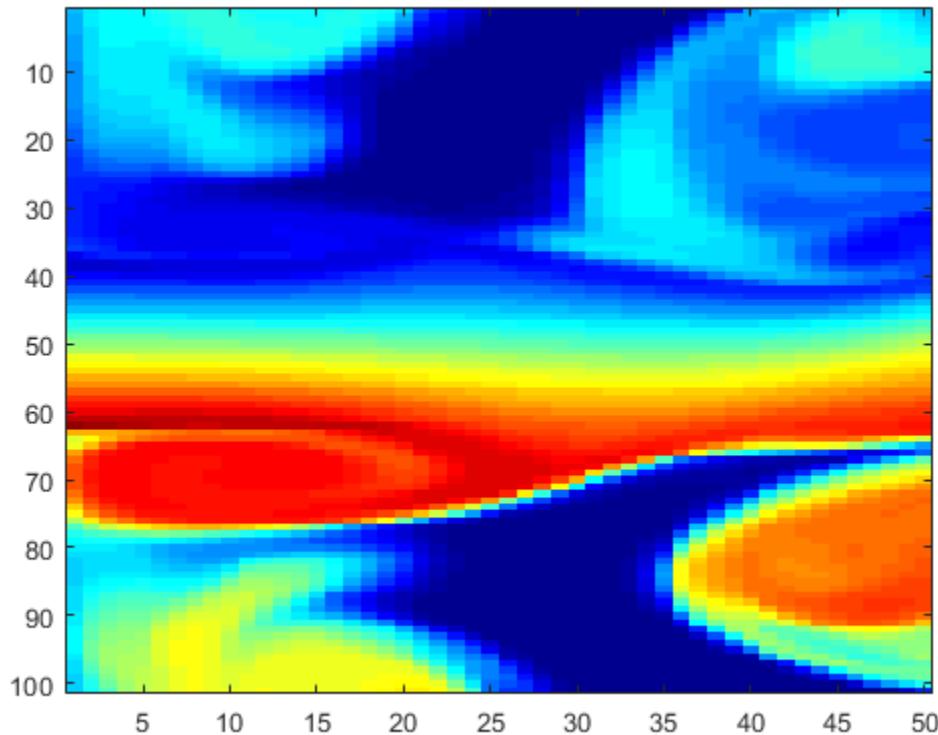
使用 `structName.fieldName` 形式的圆点表示法访问数据。例如，将字段 X 中的数值数据传递给 `image` 函数：

```
image(S.X)  
colormap(S.map)
```



要访问字段的一部分内容，请添加适合字段中数据的大小和类型的索引。例如，将 X 的左中部分传递给 `image` 函数。

```
centerLeft = S.X(150:250,1:50);
image(centerLeft)
```



如果一个字段包含元胞数组，请使用花括号访问数据，例如 `S.cellField{1:50,1:80}`。

## 通过对结构体数组进行索引来访问数据

通过将文件 `cape.mat` 中的数据加载到数组 S 的第二个元素中来创建一个非标量数组。文件 `cape.mat` 包含马萨诸塞州鳕鱼角的图像。

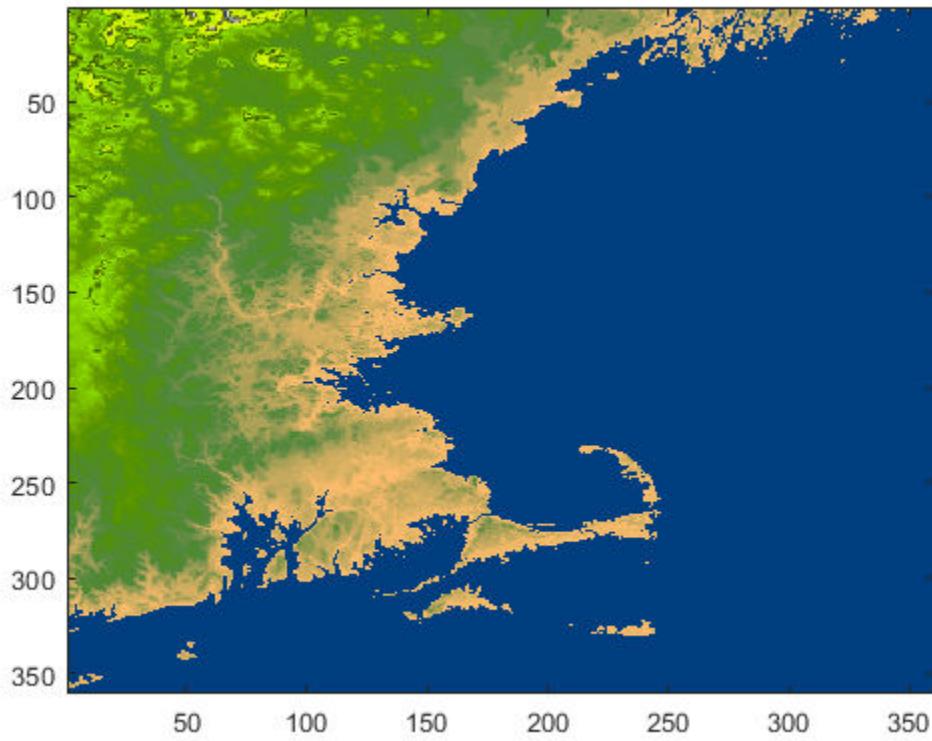
一个结构体数组的每个元素都必须具有相同的字段。`flujet.mat` 和 `cape.mat` 都包含变量 X、map 和 caption。S 是  $1 \times 2$  数组。

```
S(2) = load('cape.mat')
```

```
S=2×3 struct
  X
  map
  caption
```

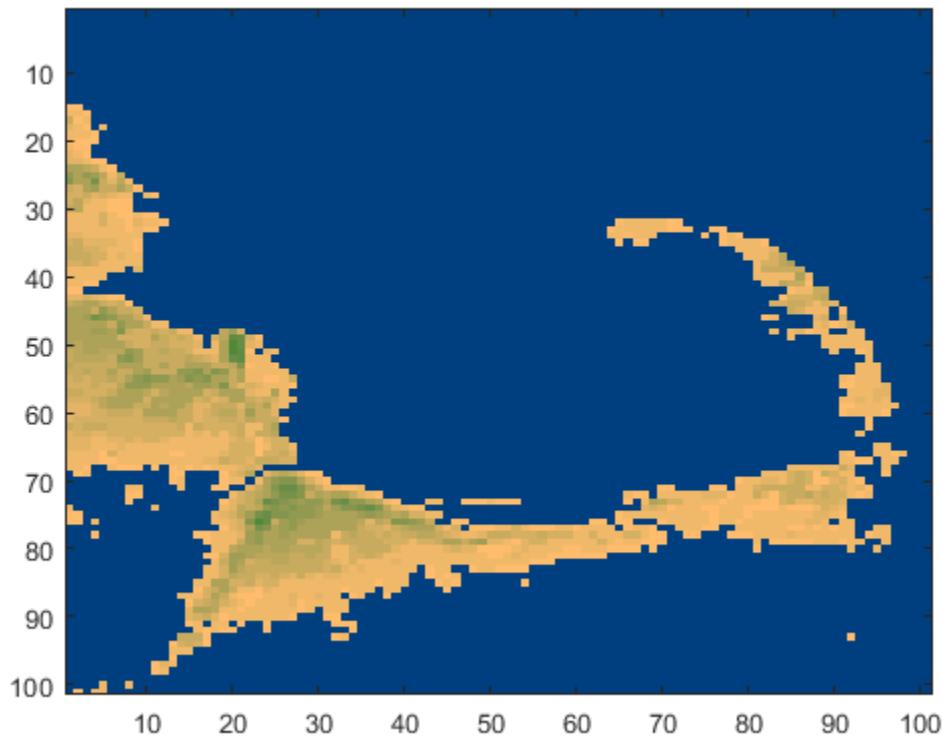
对于非标量结构体，访问字段的语法为 `structName(indices).fieldName`。显示鳕鱼角的图像，将 2 指定为 S 的索引。

```
image(S(2).X)
colormap(S(2).map)
```



添加索引以选择和显示 S(2).X 的一小部分。

```
capeSection = S(2).X(200:300,150:250);
image(capeSection)
```



**注意** 仅当引用结构体数组的单个元素时，才能为字段的部分内容建立索引。MATLAB 不支持诸如 `S(1:2).X(1:50,1:80)` 的语句，后者尝试为结构体的多个元素的字段建立索引。

## 另请参阅

### 相关示例

- “访问嵌套结构体中的数据” (第 11-12 页)
- “访问非标量结构体数组的元素” (第 11-14 页)
- “基于变量生成字段名称” (第 11-11 页)

## 串联结构体

此示例演示了如何使用 `[]` 运算符串联结构体数组。要串联结构体，他们必须具有相同的字段集，但这些字段无需包含相同的大小或数据类型。

创建标量 ( $1 \times 1$ ) 结构体数组 `struct1` 和 `struct2`，并且每个数组都具有字段 `a` 和 `b`：

```
struct1.a = 'first';
struct1.b = [1,2,3];
struct2.a = 'second';
struct2.b = rand(5);
struct1,struct2

struct1 = struct with fields:
  a: 'first'
  b: [1 2 3]
```

```
struct2 = struct with fields:
  a: 'second'
  b: [5x5 double]
```

正如串联两个标量值（如 `[1,2]`）会创建一个  $1 \times 2$  数值数组一样，串联 `struct1` 和 `struct2` 也会创建一个  $1 \times 2$  结构体数组。

```
combined = [struct1,struct2]

combined=2×2 struct
  a
  b
```

当要访问特定字段的内容时，请指定数组中的结构体的索引。例如，访问第一个结构体的字段 `a`。

```
combined(1).a

ans =
'first'
```

串联也适用于非标量结构体数组。例如，创建一个名为 `new` 的  $2 \times 2$  结构体数组。由于  $1 \times 2$  结构体 `combined` 和  $2 \times 2$  结构体 `new` 都包含两列，因此您可以使用分号分隔符垂直串联它们。

```
new(1,1).a = 1;
new(1,1).b = 10;
new(1,2).a = 2;
new(1,2).b = 20;
new(2,1).a = 3;
new(2,1).b = 30;
new(2,2).a = 4;
new(2,2).b = 40;

larger = [combined; new]

larger=3×2 struct
  a
  b
```

访问结构体 `larger(2,1)` 的字段 `a`。它与 `new(1,1).a` 包含相同的值。

```
larger(2,1).a
```

```
ans = 1
```

### 另请参阅

#### 相关示例

- “创建、串联和扩展矩阵”
- “访问结构体数组中的数据” (第 11-5 页)
- “访问非标量结构体数组的元素” (第 11-14 页)

## 基于变量生成字段名称

此示例演示了如何在运行时从变量或表达式获取结构体字段名称。一般语法为

```
structName.(dynamicExpression)
```

其中 **dynamicExpression** 是一个变量或表达式，它在求值后返回字符向量，从 R2017b 开始，还可以返回字符串标量。使用表达式引用的字段名称称为动态字段名（有时称为动态字段名称）。

例如，基于当前日期创建一个字段名称：

```
currentDate = datestr(now,'mmmdy');
myStruct.(currentDate) = [1,2,3]
```

如果您的系统报告的当前日期为 2 月 29 日，则此代码会将数据分配给名为 **Feb29** 的字段。

```
myStruct =
  Feb29: [1 2 3]
```

动态字段名可以返回字符向量或字符串标量。例如，您可以使用单引号或从 R2017b 开始使用双引号指定字段 **Feb29**。

```
myStruct.('Feb29')
```

```
ans =
  1   2   3
```

```
myStruct.("Feb29")
```

```
ans =
  1   2   3
```

字段名称与变量名称一样，都必须以字母开头，可以包含字母、数字或下划线字符，并且区分大小写。为了避免可能的冲突，请勿使用现有变量或函数的名称作为字段名称。

## 另请参阅

[fieldnames](#) | [getfield](#) | [setfield](#) | [struct](#)

## 相关示例

- “变量名称”（第 1-5 页）
- “创建结构体数组”（第 11-2 页）
- “访问结构体数组中的数据”（第 11-5 页）

## 访问嵌套结构体中的数据

此示例演示了如何为嵌套于另一个结构体中的结构体建立索引。访问特定字段中的数据的一般语法为

```
structName(index).nestedStructName(index).fieldName(indices)
```

当结构体为标量 ( $1 \times 1$ ) 时，无需包括索引以引用单个元素。例如，创建一个标量结构体 **s**，其中字段 **n** 是一个嵌套的标量结构体，其中包含字段 **a**、**b** 和 **c**：

```
s.n.a = ones(3);  
s.n.b = eye(4);  
s.n.c = magic(5);
```

访问字段 **b** 的第三行：

```
third_row_b = s.n.b(3,:)
```

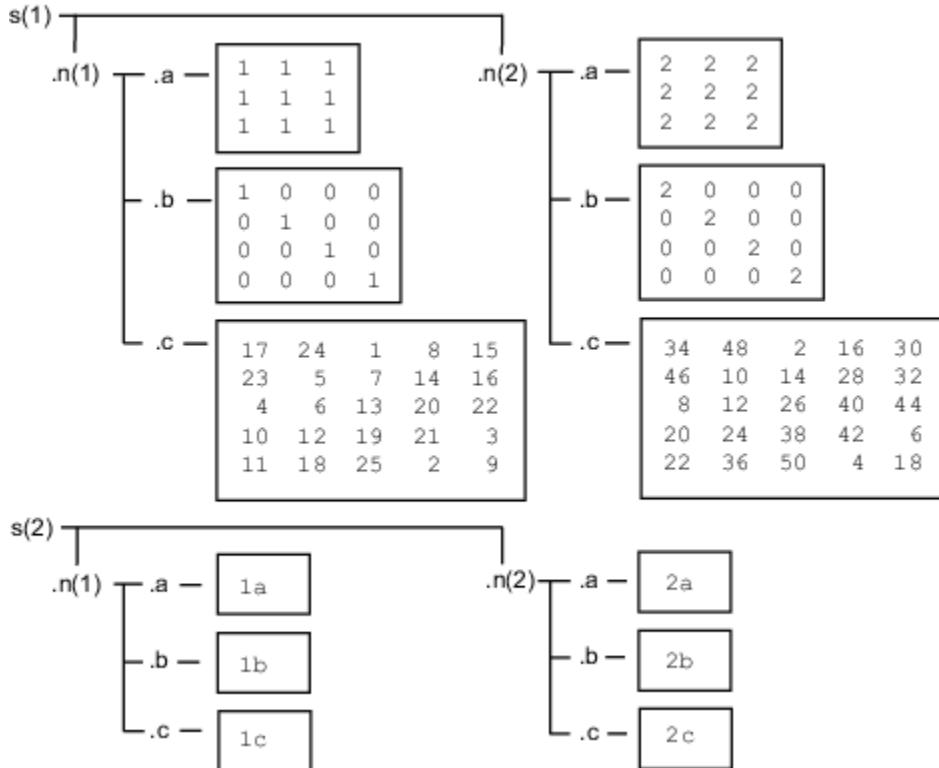
变量 **third\_row\_b** 包含 **eye(4)** 的第三行。

```
third_row_b =  
0 0 1 0
```

扩展 **s**，以便 **s** 和 **n** 都为非标量 ( $1 \times 2$ )：

```
s(1).n(2).a = 2*ones(3);  
s(1).n(2).b = 2*eye(4);  
s(1).n(2).c = 2*magic(5);  
  
s(2).n(1).a = '1a';  
s(2).n(2).a = '2a';  
s(2).n(1).b = '1b';  
s(2).n(2).b = '2b';  
s(2).n(1).c = '1c';  
s(2).n(2).c = '2c';
```

结构体 **s** 现在包含下图中所示的数据。



访问 `n` (位于 `s` 的第一个元素中) 的第二个元素的字段 `b` 中的数组部分内容:

`part_two_eye = s(1).n(2).b(1:2,1:2)`

这将返回 `2*eye(4)` 的左上角  $2 \times 2$  的部分:

```
part_two_eye =
  2   0
  0   2
```

## 另请参阅

`getfield` | `setfield` | `struct`

## 相关示例

- “创建结构体数组” (第 11-2 页)
- “访问结构体数组中的数据” (第 11-5 页)
- “结构体数组中数据的组织方法” (第 11-16 页)

## 访问非标量结构体数组的元素

此示例演示了如何访问和处理一个非标量结构体数组的多个元素中的数据。

创建一个  $1 \times 3$  的结构体 **s**, 其中包含字段 **f**:

```
s(1).f = 1;
s(2).f = 'two';
s(3).f = 3 * ones(3);
```

尽管数组中的每个结构体都必须具有相同的字段数和相同的字段名称, 但字段的类型和大小可以不同。当您引用多个元素的字段 **f** 时, 例如

**s(1:3).f**

或

**s.f**

MATLAB 以逗号分隔列表的形式返回元素中的数据, 如下所示:

```
ans =
1

ans =
two

ans =
3 3 3
3 3 3
3 3 3
```

您不能使用 **v = s.f** 语法将该列表分配给单个变量, 因为各字段包含的数据类型可能不同。但是, 您可以将列表项分配给相同数量的变量, 例如

**[v1, v2, v3] = s.f;**

或者分配给元胞数组, 例如

**c = {s.f};**

如果所有字段均包含相同类型的数据并且可以构成超矩形, 则可以串联列表项。例如, 创建一个包含字段 **f** (具有数值标量) 的结构体 **nums**, 然后串联字段中的数据:

```
nums(1).f = 1;
nums(2).f = 2;
nums(3).f = 3;

allNums = [nums.f]
```

该代码返回

```
allNums =
1 2 3
```

如果要使用相同运算处理数组的每个元素, 请使用 **arrayfun** 函数。例如, 统计数组 **s** 中每个结构体的字段 **f** 的元素数。

```
numElements = arrayfun(@(x) numel(x.f), s)
```

语法 `@(x)` 可以创建匿名函数。此代码对数组 `s` 的每个元素调用 `numel` 函数，例如 `numel(s(1).f)`，并返回

```
numElements =  
1 3 9
```

要了解相关信息，请参阅：

- “逗号分隔的列表”（第 2-73 页）
- “匿名函数”（第 20-19 页）

## 结构体数组中数据的组织方法

至少有两种方法可用来组织结构体数组中的数据：平面组织和按元素组织。最适合您的数据的方法取决于您计划访问数据的方式，而对于大型数据集，取决于是否存在系统内存约束。

使用平面组织更易于访问字段中的所有值。使用按元素组织更易于访问与单个元素或记录相关的所有信息。以下部分包括每种组织类型的示例：

- “平面组织”（第 11-16 页）
- “按元素组织”（第 11-17 页）

当创建结构体数组时，MATLAB 将有关每个元素和字段的信息存储于数组标头中。因此，具有多个元素和字段的结构体比包含相同数据的简单结构体需要更多的内存。

### 平面组织

以具有与颜色强度值对应的三个数组的 RGB 图像为例。

	Blue intensity values	0.689 0.706 0.118 0.884 ... 0.535 0.532 0.653 0.925 ... 0.314 0.265 0.159 0.101 ... 0.553 0.633 0.528 0.493 ... 0.441 0.465 0.512 0.512 ... 0.398 0.401 0.421 0.398 ...
	Green intensity values	0.342 0.647 0.515 0.816 ... 0.111 0.300 0.205 0.526 ... 0.523 0.428 0.712 0.929 ... 0.214 0.604 0.918 0.344 ... 0.100 0.121 0.113 0.126 ... 0.288 0.187 0.204 0.175 ...
Red intensity values		0.112 0.986 0.234 0.432 ... 0.765 0.128 0.863 0.521 ... 1.000 0.985 0.761 0.698 ... 0.455 0.783 0.224 0.395 ... 0.021 0.500 0.311 0.123 ... 1.000 1.000 0.867 0.051 ... 1.000 0.945 0.998 0.693 ... 0.990 0.941 1.000 0.875 ... 0.902 0.867 0.834 0.798 ... .

如果工作区中存在数组 RED、GREEN 和 BLUE，则以下命令将创建一个使用平面组织、名为 img 的标量结构体：

```
img.red = RED;
img.green = GREEN;
img.blue = BLUE;
```

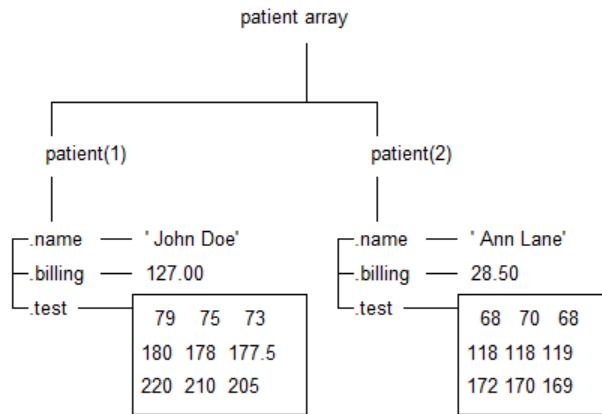
使用平面组织更易于提取整个图像平面以进行显示、筛选或其他处理。例如，将红色强度值乘以 0.9：

```
adjustedRed = .9 * img.red;
```

如果您有多幅图像，可以将它们添加到 img 结构体，一个元素 img(1),...,img(n) 包含一幅图像。对于将元素添加到结构体的示例，请参阅以下部分。

## 按元素组织

以包含患者信息的一个数据库为例。每条记录包含一位患者的姓名、测试结果和帐单金额的数据。



以下语句在名为 **patient** 的结构体数组中创建一个元素：

```
patient(1).name = 'John Doe';
patient(1).billing = 127.00;
patient(1).test = [79, 75, 73; 180, 178, 177.5; 220, 210, 205];
```

附加患者对应于结构体中的新元素。例如，为第二位患者添加一个元素：

```
patient(2).name = 'Ann Lane';
patient(2).billing = 28.50;
patient(2).test = [68, 70, 68; 118, 118, 119; 172, 170, 169];
```

按元素组织支持简单索引以访问特定患者的数据。例如，计算第一位患者的测试结果的平均值，按行（维度 2）而不是按列计算。

```
aveResultsDoe = mean(patient(1).test,2)
```

该代码返回

```
aveResultsDoe =
75.6667
178.5000
212.0000
```

## 另请参阅

**struct**

## 详细信息

- “访问结构体数组中的数据”（第 11-5 页）
- “访问非标量结构体数组的元素”（第 11-14 页）
- “结构体数组的内存要求”（第 11-18 页）

## 结构体数组的内存要求

结构体数组不要求完全连续的内存。但是，每个字段都要求连续的内存，MATLAB 创建的用来描述数组的标头也要求连续的内存。对于大型数组，不断增加字段的数量或字段中元素的数量会导致 **Out of Memory** 错误。

通过使用 `struct` 函数预分配初始值来为这些内容预分配内存，例如

```
newStruct(1:25,1:50) = struct('a',ones(20),'b',zeros(30),'c',rand(40));
```

此代码创建并填充一个  $25 \times 50$  的结构体数组 S，其中包含字段 a、b 和 c。

如果不希望分配初始值，可以通过为结构体数组中的最后一个元素的每个字段分配空数组来初始化结构体数组，例如

```
newStruct(25,50).a = [];
newStruct(25,50).b = [];
newStruct(25,50).c = [];
```

或者采用以下等效命令，

```
newStruct(25,50) = struct('a',[],'b',[],'c',[]);
```

但是，在这种情况下，MATLAB 仅为标头分配内存，而不为数组内容分配内存。

有关详细信息，请参阅：

- “重构和重新排列数组”
- “MATLAB 如何分配内存”（第 30-9 页）

# 元胞数组

---

- “什么是元胞数组？”（第 12-2 页）
- “创建元胞数组”（第 12-3 页）
- “访问元胞数组中的数据”（第 12-5 页）
- “将元胞添加到元胞数组”（第 12-8 页）
- “删除元胞数组中的数据”（第 12-9 页）
- “合并元胞数组”（第 12-10 页）
- “将元胞数组的内容传递给函数”（第 12-11 页）
- “为元胞数组预分配内存”（第 12-15 页）
- “元胞数组与结构体数组”（第 12-16 页）
- “访问部分元胞的多级索引”（第 12-20 页）

### 什么是元胞数组？

元胞数组是包含称为元胞的索引数据容器的数据类型。每个元胞可以包含任意类型的数据。元胞数组通常包含文本块、来自电子表格或文本文件的文本和数字的组合，或者不同大小的数值数组。

引用元胞数组的元素有两种方法。将索引括在圆括号 () 中以引用元胞集，例如，用于定义一个数组子集。将索引括在花括号 {} 中以引用各个元胞中的文本、数字或其他数据。

有关详细信息，请参阅：

- “创建元胞数组”（第 12-3 页）
- “访问元胞数组中的数据”（第 12-5 页）

## 创建元胞数组

此示例说明如何使用 {} 运算符或 `cell` 函数创建元胞数组。

当要将数据放入一个元胞数组中时，请使用元胞数组构造运算符 {} 创建该数组。

```
myCell = {1, 2, 3;
          'text', rand(5,10,2), {11; 22; 33}}
```

```
myCell=2×3 cell
{[ 1]} {[ 2]} {[ 3]}
{'text'} {5x10x2 double} {3x1 cell}
```

与所有 MATLAB® 数组一样，元胞数组也是矩形，每一行中具有相同的元胞数。`myCell` 是一个  $2 \times 3$  元胞数组。

您也可以使用 {} 运算符创建一个空的  $0 \times 0$  元胞数组。

```
C = {}
```

```
C =
```

```
0x0 empty cell array
```

要随时间推移或要循环向元胞数组添加值，请使用 `cell` 函数创建一个空的 N 维数组。

```
emptyCell = cell(3,4,2)
```

```
emptyCell = 3x4x2 cell array
emptyCell(:,:,1) =
```

```
{0x0 double} {0x0 double} {0x0 double} {0x0 double}
{0x0 double} {0x0 double} {0x0 double} {0x0 double}
{0x0 double} {0x0 double} {0x0 double} {0x0 double}
```

```
emptyCell(:,:,2) =
```

```
{0x0 double} {0x0 double} {0x0 double} {0x0 double}
{0x0 double} {0x0 double} {0x0 double} {0x0 double}
{0x0 double} {0x0 double} {0x0 double} {0x0 double}
```

`emptyCell` 是一个  $3 \times 4 \times 2$  的元胞数组，其中每个元胞包含一个空的数组 []。

### 另请参阅

`cell`

### 相关示例

- “访问元胞数组中的数据”（第 12-5 页）
- “创建结构体数组”（第 11-2 页）
- “创建和使用表”（第 9-2 页）

### 详细信息

- “元胞数组与结构体数组” (第 12-16 页)
- “使用表的好处” (第 9-55 页)

## 访问元胞数组中的数据

此示例说明如何在元胞数组中读取和写入数据。

创建一个由文本和数值数据组成的  $2 \times 3$  元胞数组。

```
C = {'one', 'two', 'three';
      1, 2, 3};

C=2×3 cell
{'one'}  {'two'}  {'three'}
{[ 1]}  {[ 2]}  {[ 3]}
```

引用元胞数组的元素有两种方法。将索引括在圆括号 () 中以引用元胞集，例如，用于定义一个数组子集。将索引括在花括号 {} 中以引用各个元胞中的文本、数字或其他数据。

### 使用圆括号 () 的元胞索引

圆括号中的元胞数组索引引用元胞集。例如，要创建一个属于 C 的子集的  $2 \times 2$  元胞数组，请使用圆括号。

```
upperLeft = C(1:2,1:2);

upperLeft=2×2 cell
{'one'}  {'two'}
{[ 1]}  {[ 2]}
```

通过将元胞集替换为相同数量的元胞来更新这些元胞集。例如，将 C 的第一行中的元胞替换为大小相等 ( $1 \times 3$ ) 的元胞数组。

```
C(1,1:3) = {'first','second','third'};

C=2×3 cell
{'first'}  {'second'}  {'third'}
{[ 1]}  {[ 2]}  {[ 3]}
```

如果数组中的元胞包含数值数据，可以使用 `cell2mat` 函数将这些元胞转换为数值数组。

```
numericCells = C(2,1:3);

numericCells=1×3 cell
{[1]}  {[2]}  {[3]}

numericVector = cell2mat(numericCells)

numericVector = 1×3

1    2    3
```

`numericCells` 是一个  $1 \times 3$  的元胞数组，但 `numericVector` 是一个 `double` 类型的  $1 \times 3$  数组。

### 使用花括号 {} 的内容索引

通过使用花括号进行索引来访问元胞的内容，即元胞中的数字、文本或其他数据。例如，要访问 C 的最后一个元胞的内容，请使用花括号。

```
last = C{2,3}
```

```
last = 3
```

last 为一个 **double** 类型的数值变量，因为该元胞包含 **double** 值。

同样，您也可以使用花括号进行索引来替换元胞的内容。

```
C{2,3} = 300
```

```
C=2×3 cell
{'first'}  {'second'}  {'third'}
{[ 1]}  {[ 2]}  {[ 300]}
```

您可以使用花括号进行索引来访问多个元胞的内容。MATLAB® 会以逗号分隔的列表形式返回这些元胞的内容。因为每个元胞可以包含不同类型的数据，所以无法将此列表分配给单个变量。但是，您可以将此列表分配给与元胞数量相同的变量。MATLAB® 将按列顺序赋给变量。

将 C 的四个元胞的内容赋给四个变量。

```
[r1c1, r2c1, r1c2, r2c2] = C{1:2,1:2}
```

```
r1c1 =
'first'
```

```
r2c1 = 1
```

```
r1c2 =
'second'
```

```
r2c2 = 2
```

如果每个元胞都包含相同类型的数据，则可以通过将数组串联运算符 [] 应用于逗号分隔的列表来创建单个变量。

将第二行的内容串联到数值数组中。

```
nums = [C{2,:}]
```

```
nums = 1×3
```

```
1 2 300
```

### 另请参阅

[cell](#) | [cell2mat](#)

### 相关示例

- “[创建元胞数组](#)”（第 12-3 页）
- “[访问部分元胞的多级索引](#)”（第 12-20 页）

- “逗号分隔的列表” (第 2-73 页)

## 将元胞添加到元胞数组

此示例演示了如何将元胞添加到元胞数组。

创建一个  $1 \times 3$  元胞数组。

```
C = {1, 2, 3}
```

```
C=1×3 cell
{[1]} {[2]} {[3]}
```

将数据分配给当前维度之外的元胞。MATLAB® 将元胞数组扩展到包括指定下标的矩阵。任何中间元胞都包含空数组。

```
C{4,4} = 44
```

```
C=4×4 cell
{[    1]} {[    2]} {[    3]} {0x0 double}
{0x0 double} {0x0 double} {0x0 double} {0x0 double}
{0x0 double} {0x0 double} {0x0 double} {0x0 double}
{0x0 double} {0x0 double} {0x0 double} {[    44]}
```

通过将一个空数组指定为元胞的内容来添加元胞而不指定值。C 现在是一个  $5 \times 5$  元胞数组。

```
C{5,5} = []
```

```
C=5×5 cell
Columns 1 through 4
{[    1]} {[    2]} {[    3]} {0x0 double}
{0x0 double} {0x0 double} {0x0 double} {0x0 double}
{0x0 double} {0x0 double} {0x0 double} {0x0 double}
{0x0 double} {0x0 double} {0x0 double} {[    44]}
{0x0 double} {0x0 double} {0x0 double} {0x0 double}

Column 5
{0x0 double}
{0x0 double}
{0x0 double}
{0x0 double}
{0x0 double}
```

## 另请参阅

### 相关示例

- “访问元胞数组中的数据”（第 12-5 页）
- “合并元胞数组”（第 12-10 页）
- “删除元胞数组中的数据”（第 12-9 页）

## 删除元胞数组中的数据

此示例演示了如何删除单个元胞中的数据，以及如何删除元胞数组中的全部元胞。

创建一个  $3 \times 3$  元胞数组

```
C = {1, 2, 3; 4, 5, 6; 7, 8, 9};
```

C=3×3 cell

```
{[1]} {[2]} {[3]}\n{[4]} {[5]} {[6]}\n{[7]} {[8]} {[9]}
```

通过将一个空数组赋给元胞并使用花括号建立内容索引 {} 来删除特定元胞的内容。

```
C{2,2} = []
```

C=3×3 cell

```
{[1]} {[2]} {[3]}\n{[4]} {0x0 double} {[6]}\n{[7]} {[8]} {[9]}
```

通过使用圆括号 () 建立的标准数组索引来删除元胞集。例如，删除 C 的第二行。

```
C(2,:) = []
```

C=2×3 cell

```
{[1]} {[2]} {[3]}\n{[7]} {[8]} {[9]}
```

## 另请参阅

### 相关示例

- “将元胞添加到元胞数组”（第 12-8 页）
- “访问元胞数组中的数据”（第 12-5 页）

## 合并元胞数组

此示例演示了如何通过串联或嵌套来合并元胞数组。为了运行此示例中的代码，使用相同数量的列创建多个元胞数组：

```
C1 = {1, 2, 3};  
C2 = {'A', 'B', 'C'};  
C3 = {10, 20, 30};
```

使用数组串联运算符 `[]` 串联元胞数组。在此示例中，通过使用分号分隔元胞数组来垂直串联元胞数组：

```
C4 = [C1; C2; C3]
```

C4 是一个  $3 \times 3$  元胞数组：

```
C4 =  
[ 1] [ 2] [ 3]  
'A' 'B' 'C'  
[10] [20] [30]
```

使用元胞数组构造运算符 `{}` 创建一个嵌套元胞数组：

```
C5 = {C1; C2; C3}
```

C5 是一个  $3 \times 1$  元胞数组，其中每个元胞都包含一个元胞数组：

```
C5 =  
{1x3 cell}  
{1x3 cell}  
{1x3 cell}
```

要将字符向量元胞数组合并到一个字符向量中，请使用 `strjoin` 函数。

### 另请参阅

[strjoin](#)

### 相关示例

- “创建、串联和扩展矩阵”

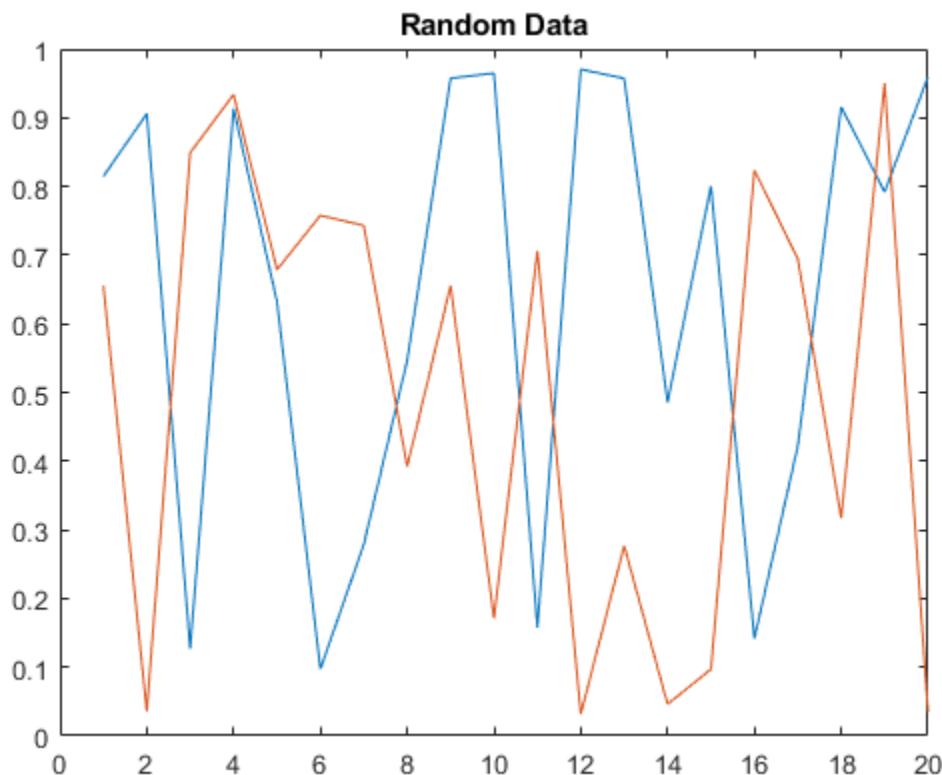
## 将元胞数组的内容传递给函数

以下这些示例演示了将元胞数组中的数据传递给无法将元胞数组识别为输入的 MATLAB® 函数的多种方法。

**通过使用花括号 {} 进行索引来传递单个元胞的内容。**

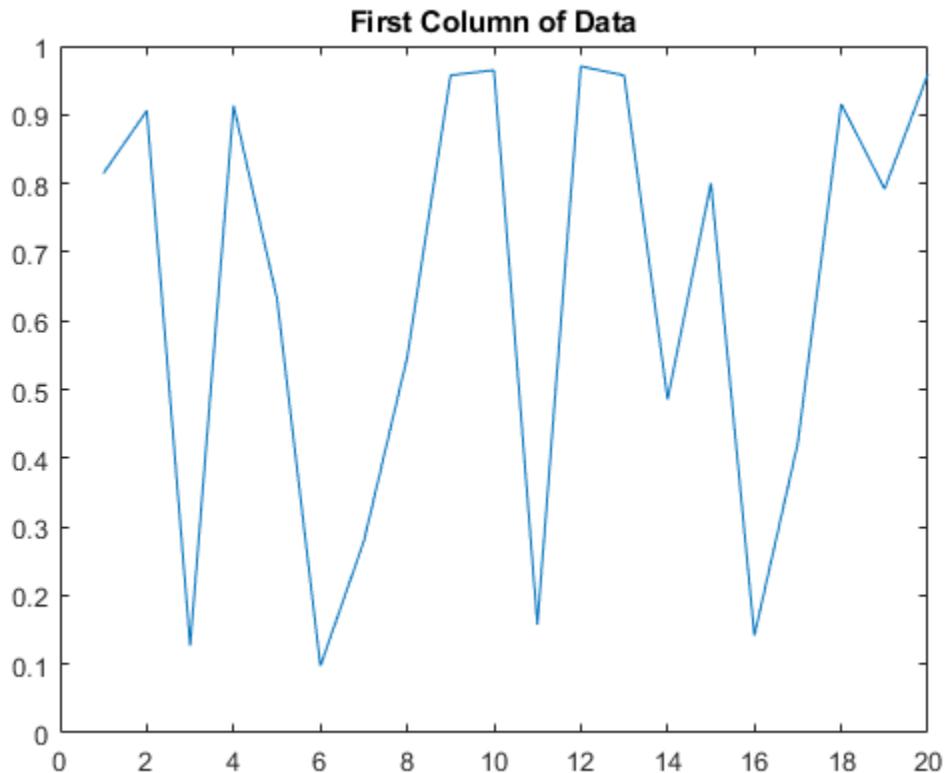
此示例创建一个包含文本的元胞数组和一个由随机数组成的  $20 \times 2$  数组。

```
randCell = {'Random Data', rand(20,2)};
plot(randCell{1,2})
title(randCell{1,1})
```



通过进一步对内容进行索引（多级索引）来仅绘制数据的第一列。

```
figure
plot(randCell{1,2}{:,1})
title('First Column of Data')
```



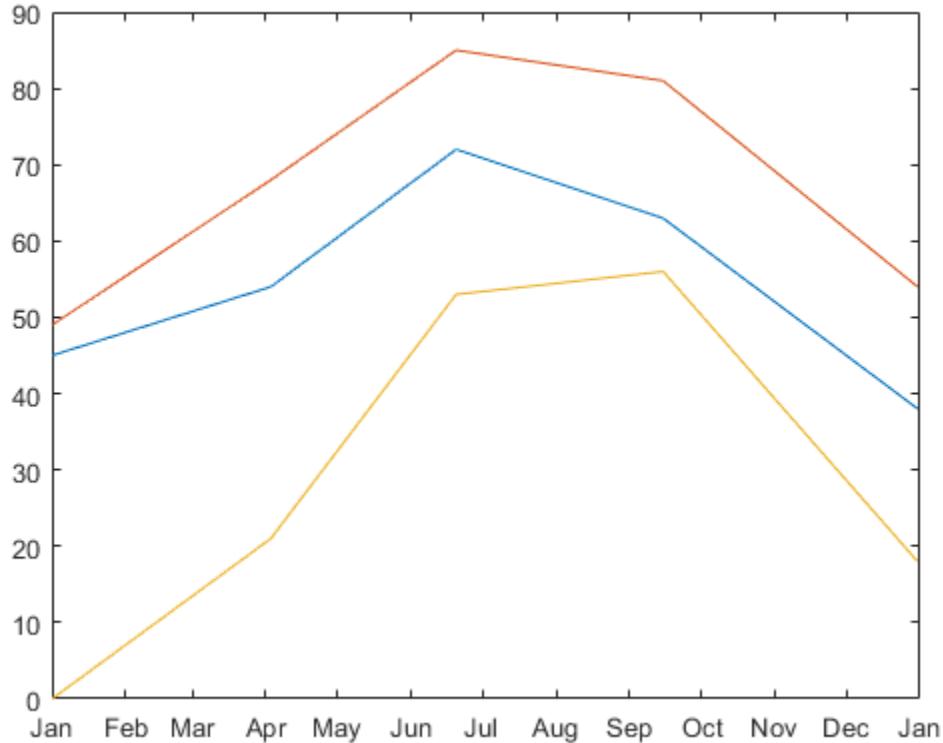
**使用 cell2mat 函数合并多个元胞中的数值数据。**

此示例创建一个  $5 \times 2$  元胞数组（存储三个城市的温度数据）以及按日期绘制每个城市的温度图。

```
temperature(1,:) = {'01-Jan-2010', [45, 49, 0]};
temperature(2,:) = {'03-Apr-2010', [54, 68, 21]};
temperature(3,:) = {'20-Jun-2010', [72, 85, 53]};
temperature(4,:) = {'15-Sep-2010', [63, 81, 56]};
temperature(5,:) = {'31-Dec-2010', [38, 54, 18]};

allTemps = cell2mat(temperature(:,2));
dates = datenum(temperature(:,1), 'dd-mmm-yyyy');

plot(dates, allTemps)
datetick('x','mmm')
```



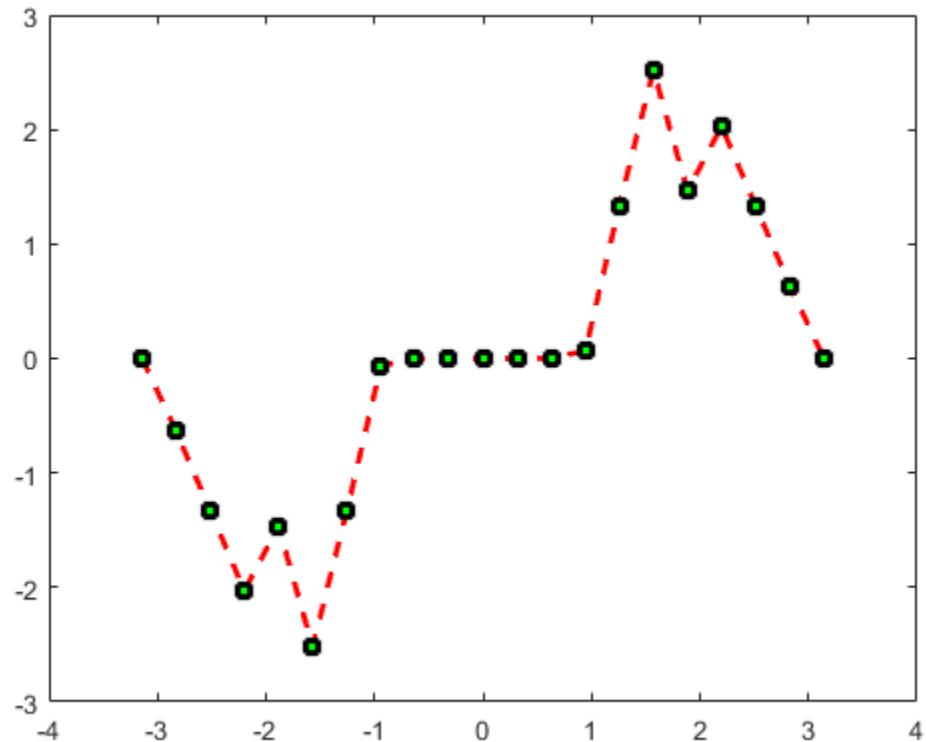
将多个元胞的内容以逗号分隔列表的形式传递给接受多个输入的函数。

此示例绘制 X 对 Y 的图，并基于一个  $2 \times 3$  元胞数组 C 应用线型。

```
X = -pi:pi/10:pi;
Y = tan(sin(X)) - sin(tan(X));

C(:,1) = {'LineWidth', 2};
C(:,2) = {'MarkerEdgeColor', 'k'};
C(:,3) = {'MarkerFaceColor', 'g'};

plot(X, Y, '-rs', C{:})
```



## 另请参阅

### 详细信息

- “访问元胞数组中的数据”（第 12-5 页）
- “访问部分元胞的多级索引”（第 12-20 页）
- “逗号分隔的列表”（第 2-73 页）

# 为元胞数组预分配内存

此示例演示了如何为元胞数组初始化和分配内存。

元胞数组不需要完全连续的内存。但是，每个元胞都要求连续的内存，与 MATLAB 创建用于描述数组的元胞数组头一样。对于大型数组，以递增方式增加元胞的数量或元胞中元素的数量会导致 **Out of Memory** 错误。

通过调用 `cell` 函数或分配给最后一个元素，来初始化元胞数组。例如，以下语句是等效的：

```
C = cell(25,50);  
C{25,50} = [];
```

MATLAB 为一个  $25 \times 50$  元胞数组创建头。但是，MATLAB 不会为每个元胞的内容分配任何内存。

## 另请参阅

`cell`

## 相关示例

- “重构和重新排列数组”
- “MATLAB 如何分配内存”（第 30-9 页）

## 元胞数组与结构体数组

此示例比较元胞和结构体数组，并说明如何在每种类型的数组中存储数据。使用元胞和结构体数组都可以存储不同类型和大小的数据。

### 结构体数组

结构体数组中数据包含在可按名称访问的字段中。

例如，将患者记录存储在一个结构体数组中。

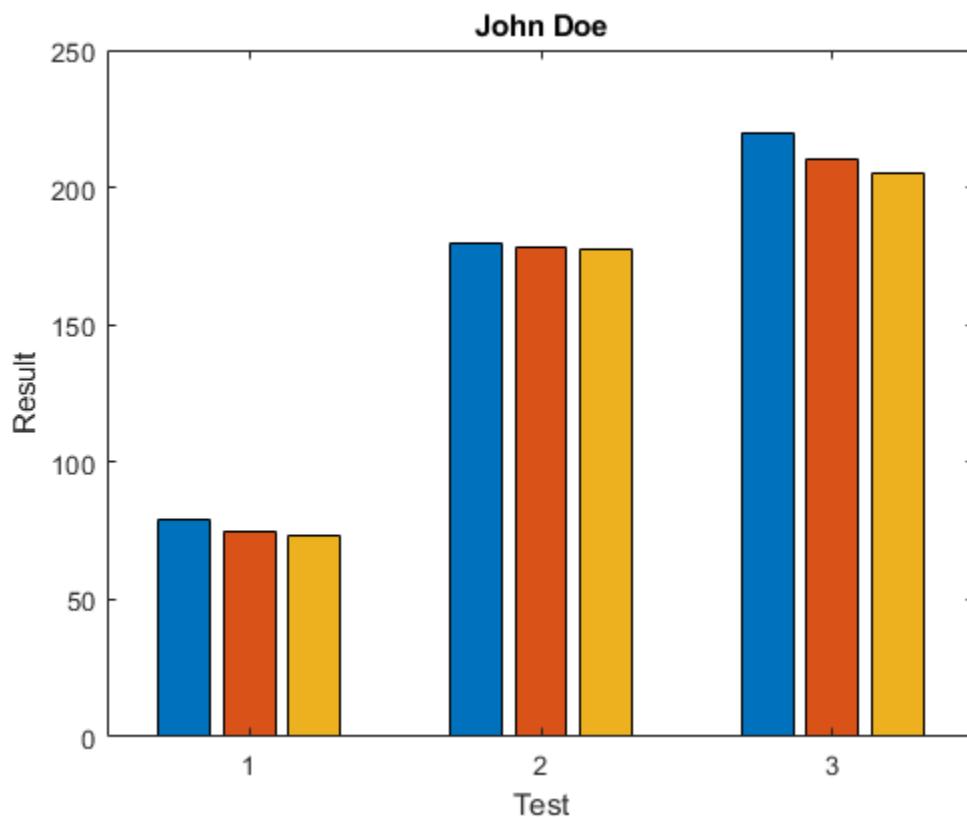
```
patient(1).name = 'John Doe';
patient(1).billing = 127.00;
patient(1).test = [79, 75, 73; 180, 178, 177.5; 220, 210, 205];

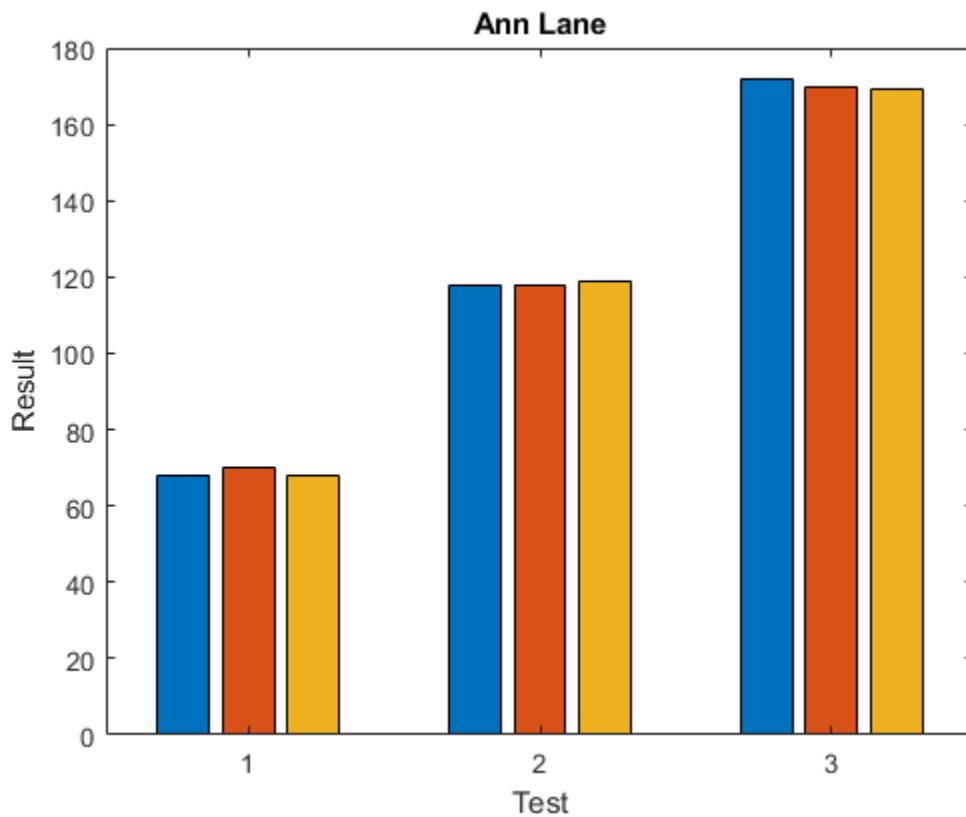
patient(2).name = 'Ann Lane';
patient(2).billing = 28.50;
patient(2).test = [68, 70, 68; 118, 118, 119; 172, 170, 169];

patient
```

为每位患者的测试结果创建一个条形图。

```
numPatients = numel(patient);
for p = 1:numPatients
    figure
    bar(patient(p).test)
    title(patient(p).name)
    xlabel('Test')
    ylabel('Result')
end
```





### 元胞数组

元胞数组中数据包含在可按数值索引访问的元胞中。元胞数组的常见应用包括存储单独的文本段，以及存储电子表格中的异类数据。

例如，在一个元胞数组中存储一段时间中三个城市的温度数据。

```
temperature(1,:) = {'2009-12-31', [45, 49, 0]};
temperature(2,:) = {'2010-04-03', [54, 68, 21]};
temperature(3,:) = {'2010-06-20', [72, 85, 53]};
temperature(4,:) = {'2010-09-15', [63, 81, 56]};
temperature(5,:) = {'2010-12-09', [38, 54, 18]};
```

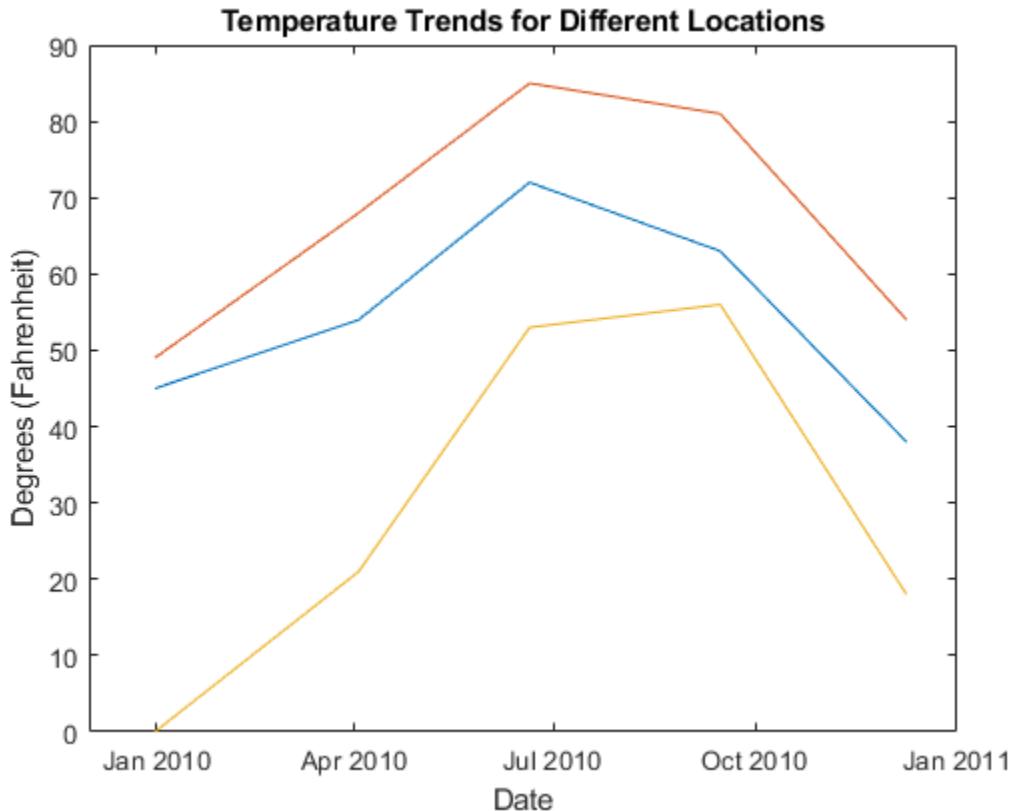
**temperature**

```
temperature=5×2 cell
{'2009-12-31'}  {1x3 double}
{'2010-04-03'}  {1x3 double}
{'2010-06-20'}  {1x3 double}
{'2010-09-15'}  {1x3 double}
{'2010-12-09'}  {1x3 double}
```

按日期绘制每个城市的温度。

```
allTemps = cell2mat(temperature(:,2));
dates = datetime(temperature(:,1));
```

```
plot(dates,allTemps)
title('Temperature Trends for Different Locations')
xlabel('Date')
ylabel('Degrees (Fahrenheit)')
```



## 其他容器数组

结构体和元胞数组是存储异类数据的最常用容器。可以方便地使用表来存储异构的列向数据或表格数据。或者，使用映射容器或创建自己的类。

## 另请参阅

[cell](#) | [cell2mat](#) | [containers.Map](#) | [datetime](#) | [plot](#) | [struct](#) | [table](#)

## 相关示例

- “访问元胞数组中的数据”（第 12-5 页）
- “访问结构体数组中的数据”（第 11-5 页）
- “访问表中的数据”（第 9-33 页）

## 详细信息

- “使用表的好处”（第 9-55 页）

## 访问部分元胞的多级索引

此示例说明了访问存储于元胞数组的元胞中的数组数据的方法。

创建一个样本元胞数组。

```
myNum = [1, 2, 3];
myCell = {'one', 'two'};
myStruct.Field1 = ones(3);
myStruct.Field2 = 5*ones(5);

C = {myNum, 100*myNum;
      myCell, myStruct}

C=2×2 cell
  {1x3 double}  {1x3 double}
  {1x2 cell }  {1x1 struct}
```

使用花括号 {} 访问特定元胞的完整内容。例如，从包含某个数值向量的元胞返回该向量。

```
C{1,2}
ans = 1×3

100 200 300
```

通过追加索引，并使用与内容的数据类型匹配的语法，来访问元胞的部分内容。

使用圆括号将数值索引括起来。例如，`C{1,1}` 返回一个  $1 \times 3$  数值向量 [1 2 3]。使用圆括号访问该向量的第二个元素。

```
C{1,1}(1,2)
ans = 2
```

将元胞数组索引括在花括号中。例如，`C{2,1}` 返回元胞数组 {'one','two'}。使用花括号访问该元胞数组中的第二个元胞的内容。

```
C{2,1}{1,2}
ans =
'two'
```

使用圆点表示法引用结构体数组的字段，并按对数值和元胞数组的说明为数组建立索引。例如，`C{2,2}` 返回一个结构体数组，其中 `Field2` 包含一个由 5 构成的  $5 \times 5$  数值数组。使用圆点表示法和圆括号访问该字段的第 5 行和第 1 列中的元素。

```
C{2,2}.Field2(5,1)
ans = 5
```

您可以嵌套任何数量的元胞和结构体数组。例如，将嵌套的元胞和结构体添加到 C。

```
C{2,1}{2,2} = {pi, eps};
C{2,2}.Field3 = struct('NestedField1', rand(3), ...
    'NestedField2', magic(4), ...
    'NestedField3', {'text'; 'more text'} );
```

使用花括号、圆括号或圆点表示法访问部分新数据。

```
copy_pi = C{2,1}{2,2}{1,1}
copy_pi = 3.1416
part_magic = C{2,2}.Field3.NestedField2(1:2,1:2)
part_magic = 2×2
16    2
      5    11
```

```
nested_cell = C{2,2}.Field3.NestedField3{2,1}
nested_cell =
'more text'
```

## 另请参阅

### 相关示例

- “访问元胞数组中的数据”（第 12-5 页）



# 函数句柄

---

- “创建函数句柄” (第 13-2 页)
- “将一个函数传递到另一个函数” (第 13-5 页)
- “使用函数句柄调用局部函数” (第 13-6 页)
- “比较函数句柄” (第 13-8 页)

## 创建函数句柄

### 本节内容

- “什么是函数句柄？”（第 13-2 页）
- “创建函数句柄”（第 13-2 页）
- “匿名函数”（第 13-3 页）
- “由函数句柄组成的数组”（第 13-4 页）
- “保存和加载函数句柄”（第 13-4 页）

您可以为已命名函数和匿名函数创建函数句柄。您可以将多个函数句柄存储在数组中，保存并加载它们，方法与对任何其他变量一样。

### 什么是函数句柄？

函数句柄是一种存储指向函数的关联关系的 MATLAB 数据类型。间接调用函数使您在调用该函数时无需考虑调用位置。函数句柄的典型用法包括：

- 将一个函数传递到另一个函数（通常称为功能函数）。例如，将函数传递到 `integral` 和 `fzero` 等积分和优化函数。
- 指定回调函数。例如，响应 UI 事件或与数据采集硬件交互的回调。
- 构造以内联方式定义而非存储在程序文件（匿名函数）中的函数的句柄。
- 从主函数外调用局部函数。

您可以使用 `isa(h,'function_handle')` 来查看变量 `h` 是否为函数句柄。

### 创建函数句柄

通过在函数名称前添加一个 @ 符号来为函数创建句柄。例如，如果您有一个名为 `myfunction` 的函数，请按如下所示创建一个名为 `f` 的句柄：

```
f = @myfunction;
```

使用句柄调用函数的方式与直接调用函数一样。例如，假设您有一个名为 `computeSquare` 的函数，该函数定义为：

```
function y = computeSquare(x)
y = x.^2;
end
```

创建句柄并调用该函数以计算 4 的平方。

```
f = @computeSquare;
a = 4;
b = f(a)
```

```
b =
```

16

如果该函数不需要任何输入，则您可以使用空括号调用该函数，例如

```
h = @ones;
a = h()
```

a =

1

如果不使用括号，则该赋值会创建另一个函数句柄。

```
a = h
```

a =

@ones

函数句柄是您可传递给其他函数的变量。例如，计算  $x^2$  在区间  $[0,1]$  上的积分。

```
q = integral(f,0,1);
```

函数句柄会存储其绝对路径，因此如果您有有效句柄，则可以从任意位置调用该函数。您不必在创建句柄时指定函数路径，只需指定函数名。

创建函数句柄时应牢记以下几点：

- 名称长度 - 函数名称（包括包名称和类名称）的每个部分都必须小于 **namelengthmax** 指定的数值。否则 MATLAB 会截断该名称的后面一部分。
- 范围 - 在您创建句柄时，该函数必须处于范围内。因此，该函数必须在 MATLAB 路径上或位于当前文件夹中。或者，对于局部或嵌套函数句柄，这些函数必须位于当前文件中。
- 优先级 - 当多个函数采用同一名称时，MATLAB 会使用与调用函数相同的优先级规则来定义函数句柄。有关详细信息，请参阅“[函数优先顺序](#)”（第 20-34 页）。
- 重载 - 如果您指定的函数在非基础 MATLAB 类的类中重载函数，那么在构造该函数的句柄时，函数不会与该函数句柄关联。相反，MATLAB 会考虑输入参数并在计算时确定要调用的实现。

## 匿名函数

您可以创建指向匿名函数的句柄。匿名函数是基于单行表达式的 MATLAB 函数，不需要程序文件。构造指向匿名函数的句柄，方法是定义 **anonymous\_function** 函数主体，以及指向匿名函数 **arglist** 的以逗号分隔的输入参数列表。语法为：

```
h = @(arglist)anonymous_function
```

例如，创建一个指向用于计算平方数的匿名函数的句柄 **sqr**，并使用其句柄调用该匿名函数。

```
sqr = @(n) n.^2;
x = sqr(3)
```

x =

9

有关详细信息，请参阅“[匿名函数](#)”（第 20-19 页）。

## 由函数句柄组成的数组

您可以通过将函数句柄收集到一个元胞数组或结构体数组中，来创建由这些函数句柄组成的数组。例如，使用元胞数组：

```
C = {@sin, @cos, @tan};  
C{2}(pi)
```

```
ans =
```

```
-1
```

或使用结构体数组：

```
S.a = @sin; S.b = @cos; S.c = @tan;  
S.a(pi/2)
```

```
ans =
```

```
1
```

## 保存和加载函数句柄

您可以在 MATLAB 中保存和加载函数句柄，就像处理任何其他变量一样。换言之，使用 `save` 和 `load` 函数。如果您保存函数句柄，则 MATLAB 不会保存路径信息。如果您加载函数句柄，并且函数文件不再位于该路径上，则该句柄无效。如果文件位置或文件名在您创建句柄后进行了更改，则该句柄将变得无效。如果句柄无效，MATLAB 可能会在您加载文件时显示警告。当您调用无效句柄时，MATLAB 将引发错误。

## 另请参阅

[func2str](#) | [functions](#) | [isa](#) | [str2func](#)

## 相关示例

- “将一个函数传递到另一个函数”（第 13-5 页）

## 详细信息

- “匿名函数”（第 20-19 页）

## 将一个函数传递到另一个函数

您可以使用函数句柄作为其他函数（称为功能函数）的输入参数。这些函数基于某个范围内的值计算数学表达式。典型的功能函数包括 `integral`、`quad2d`、`fzero` 和 `fminbnd`。

例如，要计算自然对数从 0 到 5 范围内的积分，请将指向 `log` 函数的句柄传递到 `integral`。

```
a = 0;  
b = 5;  
q1 = integral(@log,a,b)  
  
q1 = 3.0472
```

同样地，要计算 `sin` 函数和 `exp` 函数的积分，请将指向这些函数的句柄传递到 `integral`。

```
q2 = integral(@sin,a,b)  
  
q2 = 0.7163  
  
q3 = integral(@exp,a,b)  
  
q3 = 147.4132
```

此外，您还可以将指向匿名函数的句柄传递给功能函数。匿名函数是基于单行表达式的 MATLAB® 函数，不需要程序文件。例如，计算  $x/(e^x - 1)$  在  $[0, \text{Inf}]$  范围内的积分：

```
fun = @(x)x./(exp(x)-1);  
q4 = integral(fun,0,Inf)  
  
q4 = 1.6449
```

将函数作为输入项（称为功能函数）的函数预期与函数句柄关联的函数应具有特定数量的输入变量。例如，如果您调用 `integral` 或 `fzero`，则与函数句柄关联的函数必须恰好只有一个输入变量。如果您调用 `integral3`，则与函数句柄关联的函数必须具有三个输入变量。有关调用具有更多变量的功能函数的信息，请参阅“参数化函数”。

## 另请参阅

### 相关示例

- “创建函数句柄”（第 13-2 页）
- “参数化函数”

### 详细信息

- “匿名函数”（第 20-19 页）

## 使用函数句柄调用局部函数

下面的示例演示如何创建指向局部函数的句柄。如果函数返回指向局部函数的句柄，则可以在主函数外部调用局部函数。此方法可让您在单个文件中包含多个可调用函数。

在您的工作文件夹下的 `ellipseVals.m` 文件中创建以下函数。此函数会返回一个包含指向局部函数的句柄的结构体。

```
% Copyright 2015 The MathWorks, Inc.

function fh = ellipseVals
fh.focus = @computeFocus;
fh.eccentricity = @computeEccentricity;
fh.area = @computeArea;
end

function f = computeFocus(a,b)
f = sqrt(a^2-b^2);
end

function e = computeEccentricity(a,b)
f = computeFocus(a,b);
e = f/a;
end

function ae = computeArea(a,b)
ae = pi*a*b;
end
```

调用该函数以获取指向局部函数的句柄 `struct`。

```
h = ellipseVals

h =
struct with fields:

    focus: @computeFocus
    eccentricity: @computeEccentricity
    area: @computeArea
```

使用某局部函数的句柄调用该局部函数来计算椭圆面积。

```
h.area(3,1)
```

```
ans =
9.4248
```

您也可以使用 **localfunctions** 函数从所有局部函数自动创建由函数句柄组成的元胞数组。如果您需要添加、删除或修改局部函数的名称，则此方法很方便。

## 另请参阅

**localfunctions**

## 相关示例

- “创建函数句柄” (第 13-2 页)

## 详细信息

- “局部函数” (第 20-24 页)

## 比较函数句柄

### 比较根据命名函数构造的句柄

MATLAB® 会将根据同一命名函数构造的函数句柄视为相同的函数句柄。`isequal` 函数在比较以下类型的句柄时会返回 `true` 值。

```
fun1 = @sin;
fun2 = @sin;
isequal(fun1,fun2)
```

```
ans =
logical
1
```

如果将这些句柄保存为 MAT 文件，以后将它们加载回工作区时它们仍然是相等的。

### 比较指向匿名函数的句柄

与指向命名函数的句柄不同，表示同一个匿名函数的函数句柄不相等。之所以将其视为不等，是因为 MATLAB 不能保证非参数变量的冻结值相同。例如，在本例中，`A` 是一个非参数变量。

```
A = 5;
h1 = @(x)A * x.^2;
h2 = @(x)A * x.^2;
isequal(h1,h2)
```

```
ans =
logical
0
```

如果您创建匿名函数句柄的副本，则副本与原始句柄相等。

```
h1 = @(x)A * x.^2;
h2 = h1;
isequal(h1,h2)
```

```
ans =
logical
1
```

### 比较嵌套函数的句柄

仅当您的代码在对包含嵌套函数的函数的同一调用中构造这些句柄时，MATLAB 才会将同一嵌套函数的函数句柄视为相等。此函数会构造两个指向同一嵌套函数的句柄。

```
function [h1,h2] = test_eq(a,b,c)
h1 = @findZ;
h2 = @findZ;

function z = findZ
z = a.^3 + b.^2 + c';
end
end
```

根据同一嵌套函数且在对其父函数的同一调用中构建的函数句柄被视为相等。

```
[h1,h2] = test_eq(4,19,-7);
isequal(h1,h2)
```

```
ans =
logical
1
```

使用不同调用构造的函数句柄被视为不相等。

```
[q1,q2] = test_eq(4,19,-7);
isequal(h1,q1)
```

```
ans =
logical
0
```

## 另请参阅

[isequal](#)

## 相关示例

- “[创建函数句柄](#)” (第 13-2 页)



# 映射容器

---

- “映射数据结构体概述” (第 14-2 页)
- “Map 类的说明” (第 14-3 页)
- “创建映射对象” (第 14-5 页)
- “检查映射的内容” (第 14-7 页)
- “使用键索引读取和写入” (第 14-8 页)
- “修改映射中的键和值” (第 14-11 页)
- “映射到不同值类型” (第 14-13 页)

## 映射数据结构体概述

映射是一种快速的键查找数据结构体，可用于灵活地对其单个元素进行索引。与 MATLAB 软件中仅允许通过整数索引获取元素的大多数数组数据结构体不同，映射的索引几乎可以是任何数值标量或字符向量。

指向映射元素的索引称为键。这些键以及与其相关的数据值都存储在映射内。映射的每个条目都包含一个唯一键及其相应的值。对下面所示降雨量统计量的映射进行索引，用一个字符向量表示八月，获取在内部与该月份相关联的值 37.3。

KEYS	VALUES
Jan	327.2
Feb	368.2
Mar	197.6
Apr	178.4
May	100.0
Jun	69.9
Jul	32.3
Aug	37.3
Sep	19.0
Oct	37.0
Nov	73.2
Dec	110.9
Annual	1551.0

Aug → 37.3

### 月平均降雨量统计量 (mm)

键不限于整数，因为它们也用于其他数组。具体而言，键可以是以下任一类型：

- $1 \times N$  字符数组
- 实数标量 `double` 或 `single`
- 有符号或无符号的整数标量

存储于映射中的值可以为任意类型。这包括数值、结构体、元胞、字符数组、对象或其他映射组成的数据组。

**注意** 其中存储的数据是标量数字或字符数组时映射最有效。

### 另请参阅

`containers.Map` | `keys` | `values`

### 相关示例

- “Map 类的说明”（第 14-3 页）
- “创建映射对象”（第 14-5 页）
- “检查映射的内容”（第 14-7 页）

# Map 类的说明

映射实际上是称作 **Map** 的 MATLAB 类的对象或实例。它还是句柄对象，因此其表现类似于任何其他的 MATLAB 句柄对象。此部分简要介绍了 **Map** 类。有关详细信息，请参阅 [containers.Map](#) 参考页。

## Map 类的属性

**Map** 类的所有对象都有三种属性。您不能直接向任何一种属性中写入内容，只能通过 **Map** 类的方法更改它们。

属性	说明	默认值
<b>Count</b>	无符号 64 位整数，表示 <b>Map</b> 对象中包含的键/值对组的总数。	0
<b>KeyType</b>	表示 <b>Map</b> 对象中包含的所有键类型的字符向量。 <b>KeyType</b> 可以是以下任一类型： <b>double</b> 、 <b>single</b> 、 <b>char</b> 以及有符号或无符号 32 位或 64 位整数。如果您尝试添加不受支持类型的键，例如 <b>int8</b> ，MATLAB 将它们视为 <b>double</b> 。	<b>char</b>
<b>ValueType</b>	表示 <b>Map</b> 对象中包含的值类型的字符向量。如果 <b>Map</b> 中的值都是同一类型的标量数字， <b>ValueType</b> 也设置为此类型。如果这些值都是字符串数组，则 <b>ValueType</b> 为 ' <b>char</b> '。否则， <b>ValueType</b> 为 ' <b>any</b> '。	<b>any</b>

要检查其中的某种属性，请在映射对象后添加圆点，然后再接属性名。例如，要查看映射 **mapObj** 中使用的键类型，请使用

```
mapObj.KeyType
```

一个映射就是一个句柄对象。因此，如果您复制映射对象，MATLAB 不会创建新映射，而是为您指定的现有映射创建一个新句柄。如果您更改这个新句柄涉及的映射的内容，MATLAB 也会将您所做的更改应用于原始映射。但您可以删除新句柄，而不会影响原始映射。

## Map 类的方法

**Map** 类实现以下方法。它们的用法将在本文档后面的部分以及函数参考页中介绍。

方法	说明
<b>isKey</b>	检查映射是否包含指定键
<b>keys</b>	映射中的所有键的名称
<b>length</b>	映射长度
<b>remove</b>	从映射中删除键及其值
<b>size</b>	映射的维度
<b>values</b>	映射中包含的值

## 另请参阅

[containers.Map](#) | [isKey](#) | [keys](#) | [length](#) | [remove](#) | [size](#) | [values](#)

## 相关示例

- “映射数据结构体概述”（第 14-2 页）

- “创建映射对象” (第 14-5 页)
- “检查映射的内容” (第 14-7 页)

## 创建映射对象

映射是 **Map** 类的对象。它在称为 **containers** 的 MATLAB 包中定义。与任何类一样，使用其构造函数创建它的任何新实例。调用构造函数时必须包含包名称：

```
newMap = containers.Map(optional_keys_and_values)
```

### 构造空的映射对象

在不带输入参数的情况下调用映射构造函数时，MATLAB 构造空的 **Map** 对象。如果命令不以分号结尾，MATLAB 会显示以下有关已构造对象的信息：

```
newMap = containers.Map
```

```
newMap =
```

**Map with properties:**

```
Count: 0
KeyType: char
ValueType: any
```

空 **Map** 对象的属性设置为它们的默认值：

- **Count** = 0
- **KeyType** = 'char'
- **ValueType** = 'any'

构造空的映射对象后，您可以使用 **keys** 和 **values** 方法填充它。有关可与映射对象一起使用的 MATLAB 函数的汇总，请参阅“**Map** 类的方法”（第 14-3 页）

### 构造已初始化的映射对象

通常，您在构造至少具有一些键和值的映射时需要对其进行初始化。可以使用下面显示的语法输入一组或多组键和值。如果仅输入一个键/值对组，则不需要花括号运算符 ({}):

```
mapObj = containers.Map({key1, key2, ...}, {val1, val2, ...});
```

对于字符串类型的键和值，确保在指定时将它们引在单引号内。例如，构造包含字符串类型的键的映射时，使用

```
mapObj = containers.Map...
{'keystr1', 'keystr2', ...}, {val1, val2, ...});
```

下面是构造已初始化的 **Map** 对象的示例，我们为从本节前面显示的月降雨量获取的以下键/值对组创建一个新的映射。

KEYS	VALUES
Jan	327.2
Feb	368.2
Mar	197.6
Apr	178.4
May	100.0
Jun	69.9
Jul	32.3
Aug	37.3
Sep	19.0
Oct	37.0
Nov	73.2
Dec	110.9
Annual	1551.0

```
k = {'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', ...
'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec', 'Annual'};
```

```
v = {327.2, 368.2, 197.6, 178.4, 100.0, 69.9, ...
32.3, 37.3, 19.0, 37.0, 73.2, 110.9, 1551.0};
```

```
rainfallMap = containers.Map(k, v)
```

```
rainfallMap =
```

Map with properties:

```
Count: 13
KeyType: char
ValueType: double
```

Count 属性现已设置为映射内的键/值对组数量 13，KeyType 是 char，ValueType 是 double。

## 组合映射对象

可以使用串联垂直组合 Map 对象。但是，生成的不是映射向量，而是包含相关映射的所有键/值对组的单个 Map 对象。不允许使用水平映射向量。请参阅下面的“使用串联构建映射”（第 14-9 页）。

## 另请参阅

`containers.Map | keys | values`

## 相关示例

- “映射数据结构体概述”（第 14-2 页）
- “Map 类的说明”（第 14-3 页）
- “检查映射的内容”（第 14-7 页）

## 检查映射的内容

映射中的每个条目包含两个部分：唯一键及其相应值。要查找映射中的所有键，请使用 `keys` 方法。要查找所有值，请使用 `values` 方法。

创建称为 `ticketMap` 的新映射，将机票编号映射到这些机票的持有者。构造带有四个键/值对组的映射：

```
ticketMap = containers.Map(...  
    {'2R175', 'B7398', 'A479GY', 'NZ1452'}, ...  
    {'James Enright', 'Carl Haynes', 'Sarah Latham', ...  
     'Bradley Reid'});
```

使用 `keys` 方法显示映射中的所有键。MATLAB 按字母顺序列出 `char` 类型的键，按数字顺序列出任何数值类型的键：

```
keys(ticketMap)
```

```
ans =
```

```
'2R175'  'A479GY'  'B7398'  'NZ1452'
```

接下来，显示与映射中的这些键相关的值。值顺序由与其相关的键顺序决定。

下表显示了按字母顺序列出的键：

键	值
2R175	James Enright
A479GY	Sarah Latham
B7398	Carl Haynes
NZ1452	Bradley Reid

`values` 方法使用相同顺序的值：

```
values(ticketMap)
```

```
ans =
```

```
'James Enright'  'Sarah Latham'  'Carl Haynes'  'Bradley Reid'
```

### 另请参阅

`containers.Map` | `isKey` | `keys` | `length` | `remove` | `size` | `values`

### 相关示例

- “创建映射对象”（第 14-5 页）
- “使用键索引读取和写入”（第 14-8 页）
- “修改映射中的键和值”（第 14-11 页）
- “映射到不同值类型”（第 14-13 页）

## 使用键索引读取和写入

从映射中读取时，使用您已定义并与特定值相关的相同键。向映射写入新条目时需要您向要存储的每个值提供键。

---

**注意** 对于大型映射，键加值的方法占用许多内存，因为其输出是元胞数组。

---

### 从映射中读取

构造并填充映射后，您可以开始使用它存储并检索数据。按照和数组相同的方式使用映射，不同的是不限于使用整数索引。用于查找特定键 (`keyN`) 的值 (`valueN`) 的常规语法如下所示。如果键是字符串向量，请将其引在单引号中：

```
valueN = mapObj(keyN);
```

从映射 `ticketMap` 开始：

```
ticketMap = containers.Map(...  
    {'2R175', 'B7398', 'A479GY', 'NZ1452'}, ...  
    {'James Enright', 'Carl Haynes', 'Sarah Latham', ...  
     'Bradley Reid'});
```

可以通过恰当的键索引映射来查找任何单个值：

```
passenger = ticketMap('2R175')
```

```
passenger =
```

**James Enright**

查找机票 A479GY 的持有者：

```
sprintf(' Would passenger %s please come to the desk?\n', ...  
       ticketMap('A479GY'))
```

```
ans =
```

**Would passenger Sarah Latham please come to the desk?**

要访问多个键的值，请使用 `values` 方法并指定元胞数组中的键：

```
values(ticketMap, {'2R175', 'B7398'})
```

```
ans =
```

**'James Enright' 'Carl Haynes'**

映射容器仅支持标量索引。不能像对其他 MATLAB 类一样使用冒号运算符访问一系列键。例如，以下语句引发错误：

```
ticketMap('2R175':'B7398')  
ticketMap(:)
```

## 添加键/值对组

和其他数组类型不同，映射中的每个条目包含两项：值及其键。向映射写入新值时，还必须提供其键。该键的类型必须与映射中的任何其他键一致。

使用以下语法在映射中插入其他元素：

```
existingMapObj(newKeyName) = newValue;
```

从映射 ticketMap 开始：

```
ticketMap = containers.Map(...  
    {'2R175', 'B7398', 'A479GY', 'NZ1452'}, ...  
    {'James Enright', 'Carl Haynes', 'Sarah Latham', ...  
     'Bradley Reid'});
```

为 ticketMap 映射增加两个条目：验证 ticketMap 现在拥有六个键/值对组：

```
ticketMap('947F4') = 'Susan Spera';  
ticketMap('417R93') = 'Patricia Hughes';
```

**ticketMap.Count**

```
ans =
```

6

列出 ticketMap 中的所有键和值：

```
keys(ticketMap), values(ticketMap)  
ans =  
'2R175'  '417R93'  '947F4'  'A479GY'  'B7398'  'NZ1452'  
  
ans =  
'James Enright'  'Patricia Hughes'  'Susan Spera'  'Sarah Latham'  'Carl Haynes'  'Bradley Reid'
```

## 使用串联构建映射

可以使用串联向映射中添加键/值对组。映射对象的串联与其他类不同。MATLAB 不会构建映射对象的向量，而是返回一个映射，该映射含有来自每个相关映射对象的键/值对组。

映射对象串联的规则是：

- 仅允许使用映射对象的垂直向量。不能创建映射对象的  $m \times n$  数组或水平向量。出于这个原因，系统支持映射对象的 `vertcat`，但不支持 `horzcat`。
- 串联的每个映射中的所有键必须属于同一类。
- 可以将具有不同数量的键/值对组的映射组合在一起。这将生成一个映射对象，其中包含来自每个相关映射对象的键/值对组：

```
tMap1 = containers.Map({'2R175', 'B7398', 'A479GY'}, ...  
    {'James Enright', 'Carl Haynes', 'Sarah Latham'});
```

```
tMap2 = containers.Map({'417R93', 'NZ1452', '947F4'}, ...  
    {'Patricia Hughes', 'Bradley Reid', 'Susan Spera'});
```

```
% Concatenate the two maps:
ticketMap = [tMap1; tMap2];
```

该串联的结果是生成一个 6 元素映射，与前一部分中所构造的映射相同：

```
ticketMap.Count
ans =
6
keys(ticketMap), values(ticketMap)
ans =
'2R175' '417R93' '947F4' 'A479GY' 'B7398' 'NZ1452'
ans =
'James Enright' 'Patricia Hughes' 'Susan Spera' 'Sarah Latham' 'Carl Haynes' 'Bradley Reid'
```

- 串联在生成的映射对象中不包括重复键或其值。

在下面的示例中，对象 m1 和 m2 都使用键 8。在映射 m1 中，8 是值 C 的键；在 m2 中，它是值 X 的键：

```
m1 = containers.Map({1, 5, 8}, {'A', 'B', 'C'});
m2 = containers.Map({8, 9, 6}, {'X', 'Y', 'Z'});
```

组合 m1 与 m2 以形成新的映射对象 m：

```
m = [m1; m2];
```

生成的映射对象 m 仅有五个键/值对组。值 C 已从该串联中删除，因为其键不是唯一的：

```
keys(m), values(m)
```

```
ans =
```

```
[1] [5] [6] [8] [9]
```

```
ans =
```

```
'A' 'B' 'Z' 'X' 'Y'
```

## 另请参阅

[containers.Map](#) | [isKey](#) | [keys](#) | [values](#)

## 相关示例

- “创建映射对象”（第 14-5 页）
- “检查映射的内容”（第 14-7 页）
- “修改映射中的键和值”（第 14-11 页）
- “映射到不同值类型”（第 14-13 页）

## 修改映射中的键和值

**注意** 请记住，如果一个映射有多个句柄，修改该句柄还会更改原始映射。请参阅下面的“修改映射副本”（第 14-12 页）。

### 从映射中删除键和值

使用 `remove` 方法从映射中删除任何条目。调用该方法时，指定要删除的 `Map` 对象名称和键名称。MATLAB 从映射中删除键及其相关值。

删除方法的语法是

```
remove(mapName, 'keyname');
```

从映射 `ticketMap` 开始：

```
ticketMap = containers.Map(...  
    {'2R175', 'B7398', 'A479GY', 'NZ1452'}, ...  
    {'James Enright', 'Carl Haynes', 'Sarah Latham', ...  
    'Bradley Reid'});
```

从 `Map` 对象中删除一个条目（指定的键及其值）：

```
remove(ticketMap, 'NZ1452');  
values(ticketMap)  
  
ans =  
  
'James Enright'  'Sarah Latham'  'Carl Haynes'
```

### 修改值

您只能通过覆盖当前值修改映射中的任何值。持有机票 A479GY 的乘客确定为 `Sarah Latham`：

```
ticketMap('A479GY')
```

```
ans =
```

`Sarah Latham`

通过覆盖 `A479GY` 键的原始值将乘客的名字更改为 `Anna Latham`：

```
ticketMap('A479GY') = 'Anna Latham';
```

验证更改：

```
ticketMap('A479GY')
```

```
ans =
```

`Anna Latham`

## 修改键

要在修改键时保持值不变，请先从映射中删除键及其值。然后创建一个新条目，这次使用更正后的键名称。

修改属于乘客 James Enright 的机票编号：

```
remove(ticketMap, '2R175');
ticketMap('2S185') = 'James Enright';

k = keys(ticketMap); v = values(ticketMap);
str1 = '%s has been assigned a new\n';
str2 = ' ticket number: %s.\n';

fprintf(str1, v{1})
fprintf(str2, k{1})

'James Enright' has been assigned a new
ticket number: 2S185.
```

## 修改映射副本

由于 `ticketMap` 是句柄对象，因此您在创建映射副本时需要小心。请记住，复制映射对象时，您只是在创建同一对象的另一个句柄。对该句柄所做的任何更改也会应用于原始映射。

创建 `ticketMap` 映射的副本：向该副本写入内容时，请注意更改会应用于原始映射对象本身：

```
copiedMap = ticketMap;

copiedMap('AZ12345') = 'unidentified person';
ticketMap('AZ12345')

ans =
```

**unidentified person**

清理：

```
remove(ticketMap, 'AZ12345');
clear copiedMap;
```

## 另请参阅

`containers.Map` | `isKey` | `keys` | `length` | `remove` | `size` | `values`

## 相关示例

- “创建映射对象”（第 14-5 页）
- “检查映射的内容”（第 14-7 页）
- “使用键索引读取和写入”（第 14-8 页）
- “映射到不同值类型”（第 14-13 页）

## 映射到不同值类型

在映射结构体中存储其他类很常见，例如结构体或元胞数组。但是，当映射中存储的数据属于双精度值、字符、整数、逻辑值等基本 MATLAB 类型之一时，映射使用内存的效率最高。

### 映射到结构体数组

以下示例将飞机座位号映射到包含机票编号和目的地的结构体。从映射 `ticketMap` 开始，该映射将机票编号映射到乘客：

```
ticketMap = containers.Map(...  
    {'2R175', 'B7398', 'A479GY', 'NZ1452'}, ...  
    {'James Enright', 'Carl Haynes', 'Sarah Latham', ...  
     'Bradley Reid'});
```

然后创建以下结构体数组，其中包含机票编号和目的地：

```
s1.ticketNum = '2S185'; s1.destination = 'Barbados';  
s1.reserved = '06-May-2008'; s1.origin = 'La Guardia';  
s2.ticketNum = '947F4'; s2.destination = 'St. John';  
s2.reserved = '14-Apr-2008'; s2.origin = 'Oakland';  
s3.ticketNum = 'A479GY'; s3.destination = 'St. Lucia';  
s3.reserved = '28-Mar-2008'; s3.origin = 'JFK';  
s4.ticketNum = 'B7398'; s4.destination = 'Granada';  
s4.reserved = '30-Apr-2008'; s4.origin = 'JFK';  
s5.ticketNum = 'NZ1452'; s5.destination = 'Aruba';  
s5.reserved = '01-May-2008'; s5.origin = 'Denver';
```

将五个座位映射到上述结构体：

```
seatingMap = containers.Map( ...  
    {'23F', '15C', '15B', '09C', '12D'}, ...  
    {s5, s1, s3, s4, s2});
```

使用该映射对象，查找有关预定了座位 09C 的乘客的信息：

```
seatingMap('09C')
```

```
ans =
```

```
ticketNum: 'B7398'  
destination: 'Granada'  
reserved: '30-Apr-2008'  
origin: 'JFK'
```

同时使用 `ticketMap` 和 `seatingMap`，可以查找预订了座位 15B 的人员的姓名：

```
ticket = seatingMap('15B').ticketNum;  
passenger = ticketMap(ticket)  
  
passenger =  
Sarah Latham
```

## 映射到元胞数组

与结构体一样，您还可以映射到映射对象中的元胞数组。继续使用前面部分中的航空公司示例，该航班的一些乘客拥有该航空公司的“飞行常客”帐户。将这些乘客的姓名映射到他们已使用和仍可用的飞行里程数记录：

```
accountMap = containers.Map( ...
    {'Susan Spera','Carl Haynes','Anna Latham'}, ...
    {{247.5, 56.1}, {0, 1342.9}, {24.6, 314.7}});
```

使用映射检索有关乘客的帐户信息：

```
name = 'Carl Haynes';
acct = accountMap(name);

fprintf('%s has used %.1f miles on his/her account,\n', ...
    name, acct{1})
fprintf(' and has %.1f miles remaining.\n', acct{2})
```

```
Carl Haynes has used 0.0 miles on his/her account,
and has 1342.9 miles remaining.
```

### 另请参阅

[cell](#) | [containers.Map](#) | [isKey](#) | [keys](#) | [struct](#) | [values](#)

### 相关示例

- “[创建映射对象](#)” (第 14-5 页)
- “[创建结构体数组](#)” (第 11-2 页)
- “[创建元胞数组](#)” (第 12-3 页)
- “[检查映射的内容](#)” (第 14-7 页)
- “[使用键索引读取和写入](#)” (第 14-8 页)
- “[修改映射中的键和值](#)” (第 14-11 页)

# 合并不同类

---

- “不同类的有效合并” (第 15-2 页)
- “合并不同的整数类型” (第 15-3 页)
- “合并整数与非整数数据” (第 15-5 页)
- “合并元胞数组与非元胞数组” (第 15-6 页)
- “空矩阵” (第 15-7 页)
- “串联示例” (第 15-8 页)

## 不同类的有效合并

只要矩阵中的所有元素属于同一类型，矩阵和数组可以由大多数 MATLAB 数据类型的元素组成。如果您在构造矩阵时纳入了不同类的元素，MATLAB 会转换一些元素，以使生成的矩阵中包含的所有元素都为同一类型。

数据类型转换与预设的类优先级相关。下表显示了您可以进行异类串联而不出错的 5 个类（字符和逻辑值间除外）。

类型	字符	整数	单精度值	双精度值	逻辑值
字符	字符	字符	字符	字符	无效
整数	字符	整数	整数	整数	整数
单精度值	字符	整数	单精度值	单精度值	单精度值
双精度值	字符	整数	单精度值	双精度值	双精度值
逻辑值	无效	整数	单精度值	双精度值	逻辑值

例如，串联 `double` 和 `single` 矩阵始终都会生成 `single` 类型的矩阵。MATLAB 将 `double` 元素转换为 `single` 以完成该过程。

## 另请参阅

### 详细信息

- “合并不同的整数类型”（第 15-3 页）
- “合并整数与非整数数据”（第 15-5 页）
- “合并元胞数组与非元胞数组”（第 15-6 页）
- “串联示例”（第 15-8 页）
- “Concatenating Objects of Different Classes”

# 合并不同的整数类型

## 本节内容

- “概述”（第 15-3 页）
- “合并不同大小的整数的示例”（第 15-3 页）
- “合并有符号与无符号整数的示例”（第 15-3 页）

## 概述

如果您在矩阵中合并不同的整数类型（例如有符号与无符号，或 8 位整数与 16 位整数），MATLAB 将返回所有元素都属于一个公共类型的矩阵。MATLAB 将生成矩阵的所有元素设置为输入矩阵中的最左侧元素的数据类型。例如，以下串联生成由 3 个 16 位有符号整数组成的向量：

```
A = [int16(450) uint8(250) int32(1000000)]
```

## 合并不同大小的整数的示例

禁用如上所示的整数串联警告后，一次串联下面的两个数字，然后转换它们的顺序。返回值取决于整数的串联顺序。最左侧的类型决定着向量中的所有元素的数据类型：

```
A = [int16(5000) int8(50)]
A =
 5000  50
```

```
B = [int8(50) int16(5000)]
B =
 50  127
```

第一个操作返回由 16 位整数组成的向量。第二个操作返回由 8 位整数组成的向量。元素 `int16(5000)` 设置为 127，即 8 位有符号整数的最大值。

相同规则也适用于垂直串联：

```
C = [int8(50); int16(5000)]
C =
 50
127
```

---

**注意** 您可以使用 `intmax` 和 `intmin` 函数查找任何 MATLAB 整数类型的最大值或最小值。对于浮点类型，请使用 `realmax` 和 `realmin`。

---

## 合并有符号与无符号整数的示例

现在使用有符号与无符号整数做相同的练习。同样，最左侧的元素决定着生成矩阵中的所有元素的数据类型：

```
A = [int8(-100) uint8(100)]
A =
 -100  100
```

```
B = [uint8(100) int8(-100)]
```

```
B =  
100 0
```

元素 `int8(-100)` 设为零，因为它不再有符号。

MATLAB 在将每个元素串联为一个合并数组之前计算每个元素。换句话说，在合并两个元素前，以下语句的计算结果为一个 8 位有符号整数（等于 50）和一个 8 位无符号整数（无符号的 -50 设为零）。按照该串联，第二个元素保留其零值，但采用无符号 `int8` 类型：

```
A = [int8(50), uint8(-50)]  
A =  
50 0
```

## 合并整数与非整数数据

如果您合并 `double`、`single` 或 `logical` 类的整数，将为生成矩阵的所有元素提供最左侧整数的数据类型。例如，以下向量的所有元素设置为 `int32`：

```
A = [true pi int32(1000000) single(17.32) uint8(250)]
```

## 合并元胞数组与非元胞数组

合并多个数组时，若其中有一个或多个为元胞数组，将返回一个新元胞数组。每个原始数组占用新数组中的一个元胞：

```
A = [100, {uint8(200)}, 300, 'MATLAB'];  
whos A
```

Name	Size	Bytes	Class	Attributes
A	1x4	477	cell	

合并的数组中的每个元素都保持其原始类：

```
fprintf('Classes: %s %s %s %s\n',...
    class(A{1}),class(A{2}),class(A{3}),class(A{4}))
```

```
Classes: double uint8 double char
```

## 空矩阵

如果您使用空矩阵元素构造矩阵，生成矩阵中会忽略空矩阵：

```
A = [5.36; 7.01; []; 9.44]  
A =  
    5.3600  
    7.0100  
    9.4400
```

## 串联示例

### 本节内容

- “合并单精度与双精度类型值”（第 15-8 页）
- “合并整数与双精度类型值”（第 15-8 页）
- “合并字符与双精度类型值”（第 15-8 页）
- “合并逻辑值与双精度类型值”（第 15-8 页）

### 合并单精度与双精度类型值

合并 `single` 值与 `double` 值会生成 `single` 矩阵。请注意， $5.73 \times 10^{300}$  太大，无法作为 `single` 存储，因此从 `double` 转换为 `single` 时将其设置为无穷大。（本示例中使用的 `class` 函数返回输入值的数据类型）。

```
x = [single(4.5) single(-2.8) pi 5.73*10^300]
x =
    4.5000   -2.8000   3.1416      Inf

class(x)      % Display the data type of x
ans =
    single
```

### 合并整数与双精度类型值

合并整数值与 `double` 值会生成整数矩阵。请注意，`pi` 的小数部分舍入到最接近的整数。（本示例中使用的 `int8` 函数将其数值参数转换为一个 8 位整数）。

```
x = [int8(21) int8(-22) int8(23) pi 45/6]
x =
    21  -22  23   3   8
class(x)
ans =
    int8
```

### 合并字符与双精度类型值

合并 `character` 值与 `double` 值会生成 `character` 矩阵。MATLAB 将本示例中的 `double` 元素转换为它们的 `character` 等效值：

```
x = ['A' 'B' 'C' 68 69 70]
x =
    ABCDEF

class(x)
ans =
    char
```

### 合并逻辑值与双精度类型值

合并 `logical` 值与 `double` 值会生成 `double` 矩阵。MATLAB 将本示例中的 `logicaltrue` 和 `false` 元素转换为它们的 `double` 等效值：

```
x = [true false false pi sqrt(7)]  
x =  
    1.0000      0      0   3.1416   2.6458  
  
class(x)  
ans =  
    double
```



# 使用对象

---

# 对象行为

## 本节内容

- “两种复制行为” (第 16-2 页)
- “句柄对象复制” (第 16-2 页)
- “值对象复制行为” (第 16-2 页)
- “句柄对象复制行为” (第 16-3 页)
- “测试句柄或值类” (第 16-5 页)

## 两种复制行为

存在两种基本的 MATLAB 对象 - 句柄和值。

在复制操作方面，值对象的行为与 MATLAB 基础类型类似。副本是独立值。您对一个对象执行的操作不会影响该对象的副本。

句柄对象通过其句柄变量进行引用。句柄变量的副本引用同一个对象。您对句柄对象执行的操作对引用该对象的所有句柄变量可见。

## 句柄对象复制

如果您要定义类并想支持句柄对象复制，请参阅 “[Implement Copy for Handle Classes](#)”。

## 值对象复制行为

MATLAB 数值变量属于值对象。例如，您将 **a** 复制为变量 **b** 时，这两个变量彼此独立。更改 **a** 的值并不会更改 **b** 的值：

```
a = 8;
b = a;
```

现在重新为 **a** 赋值，而 **b** 不变：

```
a = 6;
b

b =
8
```

清除 **a** 不会影响 **b**：

```
clear a
b

b =
8
```

## 值对象属性

作为值对象中的属性存储的值的复制行为与数值变量相同。例如，假定 **vobj1** 是带有属性 **a** 的值对象：

```
vobj1.a = 8;
```

如果您将 `vobj1` 复制到 `vobj2`, 然后更改 `vobj1` 属性 `a` 的值, 则复制对象的属性 `vobj2.a` 的值将保持不变:

```
vobj2 =vobj1;
vobj1.a = 5;
vobj2.a

ans =
8
```

## 句柄对象复制行为

此处提供的句柄类称为 `HdClass`, 它定义称为 `Data` 的属性。

```
classdef HdClass < handle
properties
    Data
end
methods
    function obj = HdClass(val)
        if nargin > 0
            obj.Data = val;
        end
    end
end
end
```

创建此类的对象:

```
hobj1 = HdClass(8)
```

由于该语句不是通过分号终止的, 因此 MATLAB 显示有关该对象的信息:

```
hobj1 =
```

`HdClass` with properties:

  Data: 8

变量 `hobj1` 是引用所创建对象的句柄。将 `hobj1` 复制到 `hobj2` 会生成引用同一对象的另一个句柄:

```
hobj2 = hobj1
```

```
hobj2 =
```

`HdClass` with properties:

  Data: 8

由于句柄引用该对象, 复制句柄会将句柄复制为一个新的变量名称, 但该句柄仍然引用同一对象。例如, 假定 `hobj1` 是带有属性 `Data` 的句柄对象:

```
hobj1.Data
```

```
ans =
```

8

更改 `hobj1` 的 `Data` 属性的值, 则所复制对象的 `Data` 属性值也会随之更改:

```
hobj1.Data = 5;
hobj2.Data
```

```
ans =
```

```
5
```

由于 `hobj2` 和 `hobj1` 是同一对象的句柄，更改副本 `hobj2` 还会更改您通过句柄 `hobj1` 访问的数据：

```
hobj2.Data = 17;
hobj1.Data
```

```
ans =
```

```
17
```

### 为句柄变量重新赋值

为句柄变量重新赋值会生成与为任何 MATLAB 变量重新赋值相同的结果。创建一个对象并将其赋给 `hobj1` 时：

```
hobj1 = HdClass(3.14);
```

`hobj1` 引用新对象，而不是以前引用的同一对象（并且仍是由 `hobj2` 引用）。

### 清除句柄变量

从工作区中清除句柄时，MATLAB 会删除变量，但不会删除另一个句柄引用的对象。但是，如果没有指向对象的引用，则 MATLAB 会销毁该对象。

假定 `hobj1` 和 `hobj2` 都引用同一对象，您可以清除其中任一句柄而不会影响该对象：

```
hobj1.Data = 2^8;
clear hobj1
hobj2
```

```
hobj2 =
```

```
HdClass with properties:
```

```
Data: 256
```

如果您同时清除 `hobj1` 和 `hobj2`，则将不存在指向该对象的引用。MATLAB 将销毁该对象并释放该对象使用的内存。

### 删除句柄对象

要删除由任意数量的句柄引用的对象，请使用 `delete`。假定 `hobj1` 和 `hobj2` 都引用同一对象，请删除其中任一句柄。MATLAB 会删除该对象：

```
hobj1 = HdClass(8);
hobj2 = hobj1;
delete(hobj1)
hobj2

hobj2 =
handle to deleted HdClass
```

使用 `clear` 从工作区中删除变量。

## 修改对象

将对象传递给函数时，MATLAB 会将该对象的副本传递到函数工作区。如果该函数修改了对象，MATLAB 将仅修改函数工作区中对象的副本。句柄与值类之间的复制行为的区别在以下情况中很重要：

- 值对象 - 函数必须返回对象的已修改副本。要在调用方工作区中修改该对象，请将函数输出分配具有相同名称的变量
- 句柄对象 - 函数工作区中的副本引用同一个对象。因此，该函数无需返回已修改的副本。

## 测试句柄或值类

要确定对象是否为句柄对象，请使用 `isa` 函数。如果 `obj` 为某个类的对象，则此语句将确定 `obj` 是否为句柄：

```
isa(obj,'handle')
```

例如，`containers.Map` 类创建句柄对象：

```
hobj = containers.Map({'Red Sox','Yankees'},{'Boston','New York'});
isa(hobj,'handle')
```

```
ans =
```

```
1
```

`hobj` 还是 `containers.Map` 对象：

```
isa(hobj,'containers.Map')
```

```
ans =
```

```
1
```

查询 `hobj` 的类会显示它是 `containers.Map` 对象：

```
class(hobj)
```

```
ans =
```

```
containers.Map
```

`class` 函数会返回特定类的对象。

## 另请参阅

### 相关示例

- “Implement Copy for Handle Classes”



# 定义您自己的类

---

所有的 MATLAB 数据类型都是作为面向对象的类实现的。您可以通过创建其他类将您自己的数据类型添加到 MATLAB 环境中。这些用户定义的类定义新数据类型的结构体，您为每个类编写的函数或方法定义该数据类型的行为。

这些方法还可以定义各种 MATLAB 运算符（包括算术运算、下标引用和串联）应用于新数据类型的方式。例如，称为 **polynomial** 的类可能重新定义加号 (+)，这样它对多项式正确执行加法运算。

通过 MATLAB 类，您可以

- 创建用于重载现有 MATLAB 功能的方法
- 限制某类对象允许的操作
- 通过继承自相同父类来强制相关类之间的常见行为
- 显著提高代码的重用率

有关详细信息，请参阅“[类在 MATLAB 中的角色](#)”。



# **脚本和函数**



# 脚本

---

- “创建脚本” (第 18-2 页)
- “向程序中添加注释” (第 18-3 页)
- “代码节” (第 18-5 页)
- “脚本与函数” (第 18-12 页)
- “向脚本中添加函数” (第 18-13 页)

## 创建脚本

脚本是最简单的程序文件类型，因为它们没有输入或输出参数。它们可用于自动执行一系列 MATLAB 命令，例如您必须从命令行重复执行的计算或必须引用的一系列命令。

您可以通过以下方式创建新脚本：

- 高亮显示“命令历史记录”中的命令，右键点击，然后选择**创建脚本**。
- 点击主页选项卡上的**新建脚本**  按钮。
- 使用 `edit` 函数。例如，`edit new_file_name` 会创建（如果不存在相应文件）并打开 `new_file_name` 文件。如果 `new_file_name` 未指定，MATLAB 将打开一个称为 `Untitled` 的新文件。

创建脚本后，您可以向其中添加代码并保存代码。例如，您可以将生成从 0 到 100 的随机数的代码保存为名为 `numGenerator.m` 的脚本。

```
columns = 10000;
rows = 1;
bins = columns/100;

rng(now);
list = 100*rand(rows,columns);
histogram(list,bins)
```

保存脚本并使用以下方法之一运行代码：

- 在命令行上键入脚本名称并按 **Enter**。例如，要运行 `numGenerator.m` 脚本，请键入 `numGenerator`。
- 点击编辑器选项卡上的**运行**  按钮

您还可以从第二个程序文件运行代码。为此，请向第二个程序文件中添加一行含脚本名称的代码。例如，要从第二个程序文件运行 `numGenerator.m` 脚本，请将 `numGenerator;` 行添加到该文件中。MATLAB 会在您运行第二个文件时运行 `numGenerator.m` 中的代码。

脚本执行完毕后，这些变量会保留在 MATLAB 工作区中。在 `numGenerator.m` 示例中，变量 `columns`、`rows`、`bins` 和 `list` 仍位于工作区中。要查看变量列表，请在命令提示符下键入 `whos`。脚本与您的交互式 MATLAB 会话和其他脚本共享基础工作区。

## 另请参阅

### 详细信息

- “代码节”（第 18-5 页）
- “脚本与函数”（第 18-12 页）
- “基础和函数工作区”（第 20-8 页）
- “在实时编辑器中创建实时脚本”（第 19-6 页）

## 向程序中添加注释

编写代码时，最好添加描述代码的注释。注释有助于其他人员理解您的代码，并且有助您在稍后返回代码时再度记起。在程序开发和测试期间，您还可以使用注释来注释掉任何不需要运行的代码。

在实时编辑器中，您可以在代码前后插入文本行来描述过程或代码。文本行还提供其他灵活的功能，例如标准格式设置选项以及插入图像、超链接和方程。有关详细信息，请参阅“在实时编辑器中创建实时脚本”（第 19-6 页）。

---

**注意** 如果您 MATLAB 代码文件 (.m) 中的文本所包含的字符编码与您的平台编码不同，则在您保存或发布文件时，MATLAB 会将这些字符显示为乱码。实时脚本和函数 (.mlx) 支持所有区域设置的存储和字符显示。

要向 MATLAB 代码中添加注释，请使用百分比 (%) 符号。注释行可以显示在程序文件中的任何位置，您也可以在代码行末尾附加注释。

例如：

```
% Add up all the vector elements.
y = sum(x)      % Use the sum function.
```

要注释掉多个代码行，请使用块注释运算符 %{ 和 %}。%{ 和 %} 运算符必须单独显示在帮助文本块前后紧邻的行上。不要在这些行中包括任何其他文本。

例如：

```
a = magic(3);
%{
sum(a)
diag(a)
sum(diag(a))
%}
sum(diag(fliplr(a)))
```

要注释掉所选内容，请选择代码行，转到编辑器或实时编辑器选项卡，然后按  按钮。也可以键入 **Ctrl + R**。要取消注释所选代码行，请按  按钮或键入 **Ctrl + T**。

要注释掉跨多行的部分语句，请使用省略号 (...) 代替百分比符号。例如：

```
header = ['Last Name, ', ...
          'First Name, ', ...
          ... 'Middle Initial, ', ...
          'Title']
```

MATLAB 编辑器包括许多工具和上下文菜单项，用来帮助您为 MATLAB、Java 和 C/C++ 代码添加、删除或更改注释格式。例如，假设您在注释的行中输入以下长文本。

```
% This is a program that has a comment that is a little more than 75 columns wide.
disp('Hello, world')
```

将光标放在该行上，转到编辑器选项卡，然后在编辑部分按  按钮。编辑器会对注释进行换行：

```
% This is a program that has a comment that is a little more than 75
% columns wide.
disp('Hello, world')
```



按钮在实时编辑器或 MATLAB Online 中不可用。

默认情况下，您在编辑器中键入注释时，文本在列宽度达到 75 时换行。要更改注释文本换行位置所在的列或者要禁用自动注释换行，请转到[主页](#)选项卡，然后在[环境](#)部分，点击 预设。选择 **MATLAB > 编辑器/调试器 > 语言**，然后调整[注释格式设置](#)预设项。

编辑器不会对以下注释换行：

- 代码节标题（以 %% 开头的注释）
- 长的连续文本，例如 URL
- 前一行含项目符号列表项（以 \* 或 # 开头的文本）

### 另请参阅

#### 相关示例

- “为程序添加帮助”（第 20-5 页）
- “创建脚本”（第 18-2 页）
- “在实时编辑器中创建实时脚本”（第 19-6 页）
- “编辑器/调试器预设项”

# 代码节

## 本节内容

- “将您的文件分为多个代码节” (第 18-5 页)
- “执行代码节” (第 18-5 页)
- “在文件中的各代码节之间导航” (第 18-6 页)
- “执行代码节的示例” (第 18-6 页)
- “更改代码节的外观” (第 18-8 页)
- “同时使用代码节与控制语句和函数” (第 18-9 页)

## 将您的文件分为多个代码节

MATLAB 文件通常包含多个命令。您通常一次仅专注于程序中的某个部分，分块操作代码。同样，向其他人解释您的文件时，通常分块介绍您的程序。为帮助介绍这些过程，使用代码节，也称为代码单元或单元模式。代码节包含 MATLAB 脚本中您想要作为一组求值的连续代码行，以两个注释字符 (%%) 开头。

要显式定义代码节界限，使用以下方法插入分节符：

- 在**编辑器**选项卡上的**编辑**部分中，点击**插入**按钮组中的 。
- 在您想要开始新的代码节的行首输入两个百分比符号 (%%)。

与 %% 位于同一行中的文本称为节标题。可以选择是否包含节标题，不过，节标题可提高文件的可读性并在您发布代码时显示为题头。

## 执行代码节

编写 MATLAB 程序文件时，可以使用“编辑器”部分的功能逐节执行程序文件。该方法有助于您试用和微调您的程序。您可以在各节之间导航并单独执行每节。要执行某节，该节必须包含它需要的所有值，或这些值必须存在于 MATLAB 工作区中。

节执行功能运行当前以黄色高亮显示的代码节。MATLAB 在执行某个代码节时不会自动保存您的文件。文件无需位于您的搜索路径下。

该表提供有关执行代码节的说明。

操作	说明
运行当前节中的代码。	<ul style="list-style-type: none"> <li>• 将光标置于代码节中。</li> <li>• 在<b>编辑器</b>选项卡上的<b>运行</b>部分中，点击  <b>运行节</b>。</li> </ul>
运行当前节中的代码，然后移至下一节。	<ul style="list-style-type: none"> <li>• 将光标置于代码节中。</li> <li>• 在<b>编辑器</b>选项卡中，点击<b>运行</b>部分中的  <b>运行并前进</b></li> </ul>
运行文件中的所有代码。	<ul style="list-style-type: none"> <li>• 在命令行窗口中键入已保存脚本的名称。</li> <li>• 在<b>编辑器</b>选项卡上，点击<b>运行</b>部分中的  <b>运行</b>。</li> </ul>

### 递增代码节中的值

您可以逐步递增某节内的数字，在每次更改后重新运行该节。这有助于您微调并试用您的代码。

要逐步递增或递减节内的数字，请执行以下操作：

- 1 高亮显示数字或将光标放在数字旁边。
- 2 右键点击以打开上下文菜单。
- 3 选择**递增值和运行节**。随即出现一个小型对话框。



- 4 在  $-/+$  文本框或  $\div/\times$  文本框中输入合适的值。
- 5 点击  $+$ 、 $-$ 、 $\times$  或  $\div$  按钮以在节中增加、减去、乘以或除以所选的数值。

MATLAB 在每次点击后运行该节。

**注意** MATLAB 软件不自动保存您对脚本中的数字所做的更改。

### 在文件中的各代码节之间导航

您可以在文件中的各代码节之间导航，无需执行这些节内的代码。这有助于在文件中的各节之间快速跳转。例如，您可能在大文件中这样做以查找特定代码。

操作	说明
移至下一节。	<ul style="list-style-type: none"> <li>在<b>编辑器</b>选项卡上，点击<b>运行</b>部分中的  前进。</li> </ul>
移至上一节。	<ul style="list-style-type: none"> <li>按 <b>Ctrl + 向上箭头</b>。</li> </ul>
移至特定节。	<ul style="list-style-type: none"> <li>在<b>编辑器</b>选项卡中，使用<b>导航</b>部分中的  转至 <input type="text"/>，将光标移至所选节。</li> </ul>

### 执行代码节的示例

本示例定义 `sine_wave.m` 文件中的两个代码节，然后递增参数以调整所创建的绘图。要在编辑器中打开该文件，请运行以下命令，然后将文件保存到本地文件夹中：

```
edit(fullfile(matlabroot,'help','techdoc','matlab_env',...
'examples','sine_wave.m'))
```

文件在编辑器中打开后：

- 1 在文件的第一行插入分节符和以下标题。

```
%% Calculate and Plot Sine Wave
```

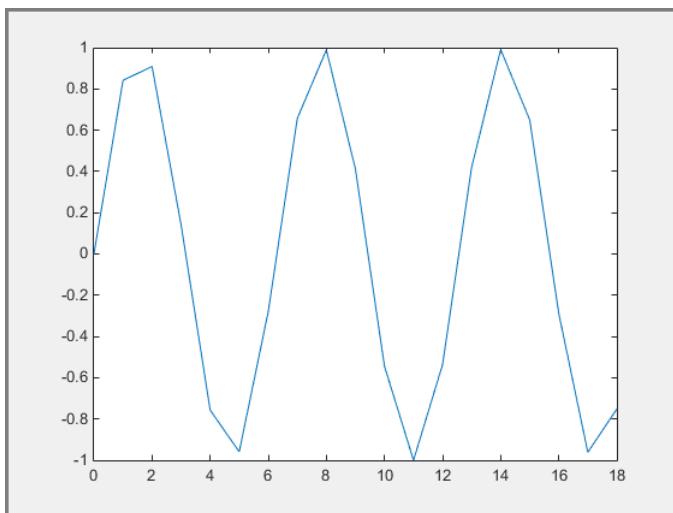
- 2 在 `plot(x,y)` 之后插入空白行和第二个分节符。添加节标题 **Modify Plot Properties**，这样整个文件都包含该代码：

```
%% Calculate and Plot Sine Wave
% Define the range for x.
% Calculate and plot y = sin(x).
x = 0:1:6*pi;
y = sin(x);
plot(x,y)
```

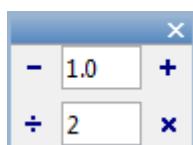
```
%% Modify Plot Properties
title('Sine Wave')
xlabel('x')
ylabel('sin(x)')
fig = gcf;
fig.MenuBar = 'none';
```

- 3 保存文件。  
 4 将光标放在标题为 Calculate and Plot Sine Wave 的节中。在编辑器选项卡上的运行部分中，点击  运行节。

随即出现一个显示  $\sin(x)$  轨迹图的图窗。

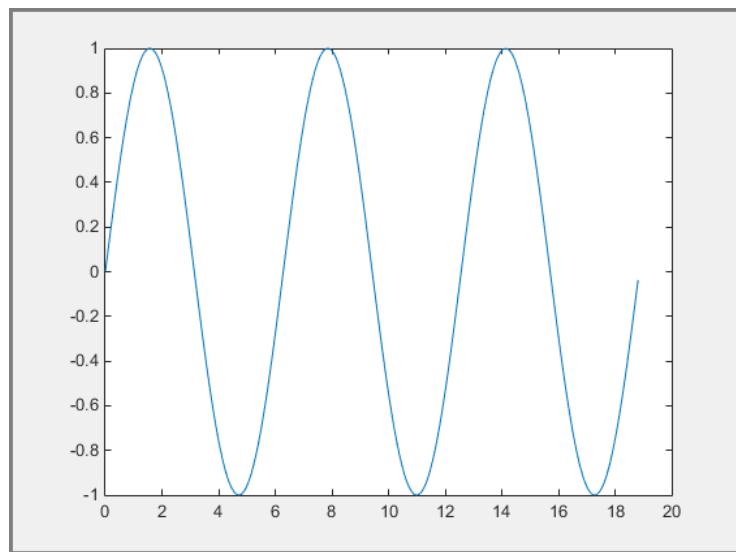


- 5 对该正弦图进行平滑处理。
- 1 高亮显示  $x = 0:1:6*pi;$  语句中的 1。
  - 2 右键点击并选择递增值和运行节。随即出现一个小型对话框。

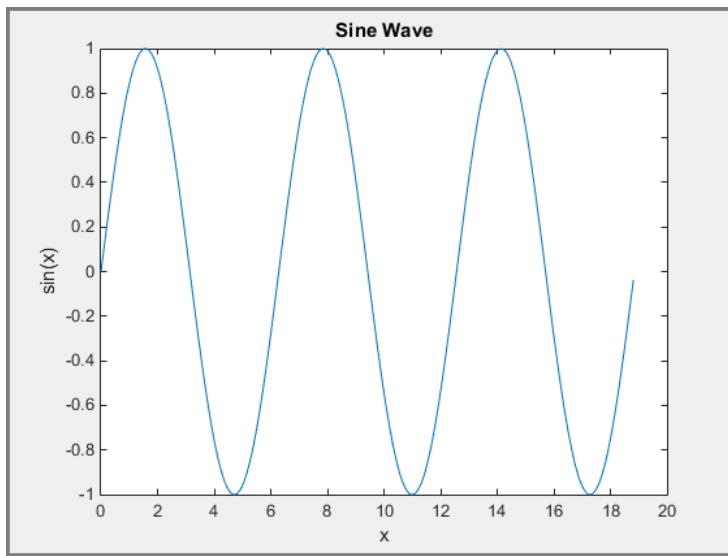


- 3 在  $\div / \times$  文本框中键入 2。
- 4 点击  $\div$  按钮多次。

每次点击后正弦图都会变得更平滑。



- 5 关闭图窗并保存文件。  
 6 运行整个 `sine_wave.m` 文件。带有标题的平滑正弦图显示在新图窗中。



## 更改代码节的外观

可以更改代码节如何在 MATLAB 编辑器中显示。MATLAB 默认以黄色高亮显示代码节，并用水平线将它们分开。光标定位在节内的任何行中时，编辑器会高亮显示整个节。

要更改代码节的外观，请执行以下操作：

- 1 在主页选项卡上，点击环境部分中的  预设。此时将显示“预设项”对话框。
- 2 在左窗格中，选择 MATLAB > 颜色 > 编程工具。
- 3 在节显示选项下方，选择代码节的外观。



您可以选择是否高亮显示节、高亮显示的颜色以及代码节之间是否显示分界线。

## 同时使用代码节与控制语句和函数

在控制语句和函数内使用代码节时会产生意外结果，因为 MATLAB 自动插入编辑器中未显示的分节符，除非您显式插入分节符。涉及嵌套代码时尤其如此。在控制语句或函数的范围内放置另一个控制语句或函数时产生嵌套代码。

MATLAB 根据以下条件自动定义代码块中的节界限：

- MATLAB 在文件顶端和底端插入分节符，创建包含整个文件的代码节。但是，编辑器不会高亮显示包含整个文件的生成节，除非您将一个或多个显式代码节添加到文件中。
- 如果您定义控制流语句（例如 **if** 或 **while** 语句）内的分节符，MATLAB 会自动将分节符插入包含语句开始和结束的行。
- 如果您在函数内定义分节符，MATLAB 会将分节符插入函数声明和函数结束语句中。如果您没有用 **end** 语句结束函数，MATLAB 将视该函数的结尾紧发生在下个函数的开头前。

如果在您插入分节符的同一行中自动插入分节符，它们会折叠为一个分节符。

### 嵌套的代码节分节符

以下代码阐释了嵌套代码节的概念：

```
t = 0:.1:pi*4;
y = sin(t);

for k = 3:2:9
    %%
    y = y + sin(k*t)/k;
    if ~mod(k,3)
        %%
        display(sprintf('When k = %.1f',k));
        plot(t,y)
    end
end
```

如果您将该代码复制并粘贴到 MATLAB 编辑器，会看到两个分节符创建了三个嵌套级别：

- **在嵌套的最外层**，一个节涵盖整个文件。

```

1 - t = 0:.1:pi*4;
2 - y = sin(t);
3
4 - for k = 3:2:9
5 - %%
6 - y = y + sin(k*t)/k;
7 - if ~mod(k,3)
8 - %%
9 - display(sprintf('When k = %.1f',k));
10 - plot(t,y)
11 - end
12 - end

```

只有您在代码块内的同级别指定分节符，MATLAB 才会在该代码块内定义节。因此，MATLAB 将该光标视为位于包含整个文件的节内。

- 在嵌套的第二级，一个节位于 **for** 循环内。

```

1 - t = 0:.1:pi*4;
2 - y = sin(t);
3
4 - for k = 3:2:9
5 - %%
6 - y = y + sin(k*t)/k;
7 - if ~mod(k,3)
8 - %%
9 - display(sprintf('When k = %.1f',k));
10 - plot(t,y)
11 - end
12 - end

```

- 在嵌套的第三级，一个节位于 **if** 语句内。

```

1 - t = 0:.1:pi*4;
2 - y = sin(t);
3
4 - for k = 3:2:9
5 - %%
6 - y = y + sin(k*t)/k;
7 - if ~mod(k,3)
8 - %%
9 - display(sprintf('When k = %.1f',k));
10 - plot(t,y)
11 - end
12 - end

```

## 另请参阅

### 详细信息

- “创建脚本” (第 18-2 页)
- “在实时编辑器中创建实时脚本” (第 19-6 页)
- “脚本与函数” (第 18-12 页)

## 脚本与函数

本主题讨论脚本与函数的区别并演示如何将脚本转换为函数。

脚本和函数都允许您通过将命令序列存储在程序文件中来重用它们。脚本是最简单的程序类型，因为它们存储命令的方式与您在命令行中键入命令完全相同。但是，函数更灵活，更容易扩展。

在名为 `triarea.m` 的文件中创建一个脚本以计算三角形的面积：

```
b = 5;
h = 3;
a = 0.5*(b.*h)
```

保存文件后，您可以从命令行中调用该脚本：

```
triarea
```

```
a =
7.5000
```

要使用同一脚本计算另一三角形区域，您可以更新 **b** 和 **h** 在脚本中的值并返回值。每次运行脚本时，它都会将结果存储在名为 **a** 的变量（位于基础工作区中）中。

但是，您可以通过将脚本转换为函数来以提升程序的灵活性，无需每次手动更新脚本。用函数声明语句替换向 **b** 和 **h** 赋值的语句。声明包括 **function** 关键字、输入和输出参数的名称以及函数名称。

```
function a = triarea(b,h)
a = 0.5*(b.*h);
end
```

保存该文件后，您可以从命令行调用具有不同的基值和高度值的函数，不用修改脚本：

```
a1 = triarea(1,5)
a2 = triarea(2,10)
a3 = triarea(3,6)
```

```
a1 =
2.5000
a2 =
10
a3 =
9
```

函数具有它们自己的工作区，与基础工作区隔开。因此，对函数 `triarea` 的任何调用都不会覆盖 **a** 在基础工作区中的值。但该函数会将结果指定给变量 **a1**、**a2** 和 **a3**。

## 另请参阅

### 详细信息

- “[创建脚本](#)”（第 18-2 页）
- “[在文件中创建函数](#)”（第 20-2 页）
- “[向脚本中添加函数](#)”（第 18-13 页）
- “[基础和函数工作区](#)”（第 20-8 页）

# 向脚本中添加函数

MATLAB 脚本（包括实时脚本）可以包含用于定义函数的代码。这些函数称为局部函数。局部函数在您要重用脚本中的代码时很有用。通过添加局部函数，可以避免创建和管理单独的函数文件。在尝试不同函数时，局部函数也很有用，可根据需要轻松添加、修改和删除它们。R2016b 或更高版本支持脚本中的函数。

## 添加局部函数

局部函数仅在定义了这些函数的文件中对脚本代码和文件中的其他局部函数可见。它们对其他文件中的函数不可见，并且不能通过命令行来调用。它们等效于其他编程语言的子例程，有时被称为子函数。

要在脚本中添加局部函数，请首先创建脚本。转至[主页](#)选项卡并选择[新建 > 脚本](#)。有关创建脚本的详细信息，请参阅“[创建脚本](#)”（第 18-2 页）。您也可以“[在实时编辑器中创建实时脚本](#)”（第 19-6 页）。

创建脚本之后，请在脚本中添加并保存代码。例如，添加以下代码并将其保存为名为 `mystats.m` 的脚本。此代码用于声明一个数组、确定该数组的长度并将两个值均传递给局部函数 `mymean` 和 `mymedian`。局部函数 `mymean` 和 `mymedian` 会计算输入列表的平均值和中位数并返回结果。

---

**注意** 在脚本中包含函数需要 MATLAB R2016b 或更高版本。

---

```
x = 1:10;
n = length(x);
avg = mymean(x,n);
med = mymedian(x,n);

function a = mymean(v,n)
% MYMEAN Example of a local function.

    a = sum(v)/n;
end

function m = mymedian(v,n)
% MYMEDIAN Another example of a local function.

    w = sort(v);
    if rem(n,2) == 1
        m = w((n + 1)/2);
    else
        m = (w(n/2) + w(n/2 + 1))/2;
    end
end
```

您可以按照任意顺序添加局部函数，只要它们全部出现在其余的脚本代码之后。每个函数都以其自己的函数定义语句开头，以 `end` 关键字结尾。定义语句是任何函数的第一个可执行代码行，例如 `function a = mymean(v,n)`。有关函数定义语句的详细信息（包括如何创建它们），请参阅“[在文件中创建函数](#)”（第 20-2 页）。

## 访问帮助

虽然您不能从命令行或其他文件中的函数调用局部函数，但您可以使用 `help` 命令访问其帮助。同时指定脚本和局部函数的名称，用 `>` 字符将它们隔开：

```
help mystats>mymean
```

mymean Example of a local function.

## 运行代码

要运行脚本（包括所有局部函数），请点击  **运行**（对于脚本）或  **全部运行**（对于实时脚本）按钮，或者在命令行窗口中键入已保存脚本的名称。您也可以通过点击  **运行节**按钮来运行脚本中的单个节。

当前文件中的局部函数优先于其他文件中的函数。即，当您在程序文件内调用某个函数时，MATLAB 会先检查该函数是否为局部函数，然后再查找其他函数。这样您可以在创建特定函数的备用版本的同时，将原始版本保留在另一文件中。

脚本可在基础工作区中创建变量并进行访问。局部函数与其他函数类似，具有与基础工作区分开的专属工作区。局部函数无法访问其他函数的工作区或基础工作区中的变量，除非您将这些变量作为参数传递。有关详细信息，请参阅“基础和函数工作区”（第 20-8 页）。

## 在实时脚本中添加和运行节

实时编辑器不支持在局部函数中运行各节。因此，您不能在实时脚本的局部函数中添加分节符。当您在实时脚本中添加局部函数时，MATLAB 会自动在第一个局部函数的定义之前添加一个分节符，并删除该函数后面的所有分节符。但是，您可以在脚本代码中运行各个节，即使它们包含对文件中的某个局部函数的调用也一样。有关分节符和运行实时脚本的详细信息，请参阅“在实时脚本中运行节”（第 19-11 页）。

## 另请参阅

### 详细信息

- “在文件中创建函数”（第 20-2 页）
- “函数优先顺序”（第 20-34 页）

# 实时脚本和函数

---

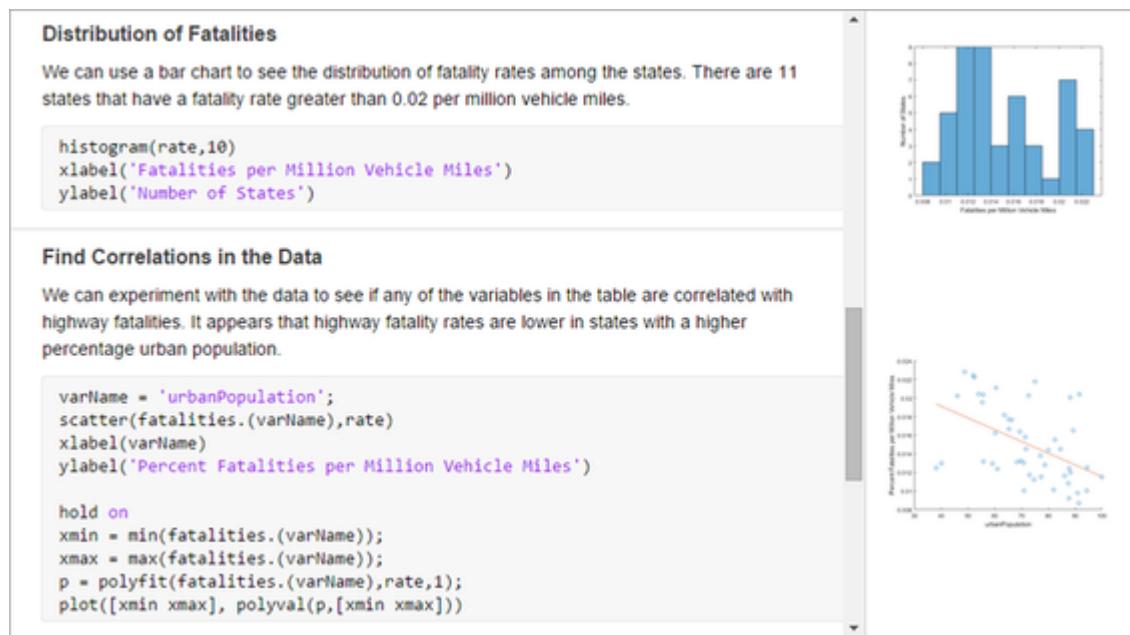
## 什么是实时脚本或实时函数？

MATLAB 实时脚本和实时函数是交互式文档，它们在一个称为实时编辑器的环境中将 MATLAB 代码与格式化文本、方程和图像组合到一起。此外，实时脚本可存储输出，并将其显示在创建它的代码旁。

实时脚本和函数可用于：

### 直观浏览和分析问题

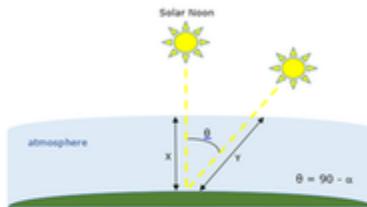
- 在单个交互式环境中编写、执行和测试代码。
- 逐个运行代码段或作为整个文件运行，查看结果和图形以及生成它们的对应源代码。



### 共享富文本格式的可执行记叙脚本

- 添加标题、题头和格式化文本以描述相应过程，并纳入方程、图像和超链接作为支持材料。
- 将您的记叙脚本另存为富文本格式的可执行文档，并与同事或 MATLAB 社区共享它们，或者将其转换为 HTML、PDF、Microsoft Word 或 LaTeX 文档以供发布。

### Air Mass and Solar Radiation



The larger the air mass, the less radiation reaches the ground. The air mass can be calculated from the equation

$$AM = \frac{1}{\cos(90 - \alpha) + 0.5057(6.0799 + \alpha)^{-1.6364}}.$$

Then the solar radiation (in  $\text{kW/m}^2$ ) reaching the ground can be calculated from the empirical equation

$$sRad = 1.353 * 0.7^{AM^{0.678}}.$$

```
AM = 1/(cosd(90-alpha) + 0.50572*(6.07955+alpha)^-1.6354);
sRad = 1.353*0.7^(AM^0.678);
% kW/m^2
disp(['Air Mass = ' num2str(AM) ' Solar Radiation = ' num2str(sRad) ' kW/m^2'])
```

### 创建交互式教学课件

- 将代码和结果与格式化文本和数学方程结合使用。
- 创建分步式课件并逐步进行计算以说明教学主题。
- 随时修改代码以回答问题或探讨相关主题。
- 将课件作为交互式文档与学生共享或以硬拷贝形式共享，将部分完成的文件作为作业发给学生。

#### Homework

Use the techniques described above to complete the following exercises:

Exercise 1: Write MATLAB code to calculate the 3 cube roots of i.

% Put your code here

Exercise 2: Write MATLAB code to calculate the 5 fifth roots of -1.

% Put your code here

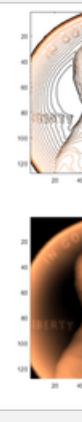
Exercise 3: Describe the mathematical approach you would use to calculate the  $n$ th roots of an arbitrary complex number. Include the equations you used in your approach.

(Describe your approach here)

### 与纯代码脚本和函数的差异

实时脚本和实时函数在几个方面与纯代码脚本和函数存在差别。此表对主要差别进行了汇总。

	实时脚本和函数	纯代码脚本和函数
文件格式	实时代码文件格式。有关详细信息，请参阅“实时代码文件格式 (.mlx)”（第 19-59 页）	普通文本文件格式
文件扩展名	.mlx	.m

	实时脚本和函数	纯代码脚本和函数
输出显示	在实时编辑器中，与代码一起显示（仅限实时脚本）	在命令行窗口中
国际化	可跨区域设置互操作	非 7 位 ASCII 字符并非兼容所有区域设置
文本格式设置	在实时编辑器中添加和查看格式化文本	使用发布标记添加格式化文本，发布到视图
视觉表示	<p><b>Viewing a Penny</b></p> <p>This example shows four techniques to visualize the surface data of a penny. The file PENNY.MAT contains measurements made at the National Institute of Standards and Technology of the depth of the mold used to mint a U. S. penny, sampled on a 128-by-128 grid.</p> <pre>% Copyright 1984-2014 The MathWorks, Inc.  <b>Drawing a Contour Plot</b> Draw a contour plot with 15 copper colored contour lines.  load penny.mat contour(P,15) colormap(copper) axis ij square</pre> <p><b>Drawing a Pseudocolor Plot</b> Draw a pseudocolor plot with brightness proportional to height.</p> <pre>color(P) axis ij square shading flat</pre>	 <pre>% Viewing a Penny % This example shows four techniques to visualize the surface ... % penny. The file PENNY.MAT contains measurements made at the ... % Institute of Standards and Technology of the depth of the ... % mint a U. S. penny, sampled on a 128-By-128 grid.  % Copyright 1984-2014 The MathWorks, Inc.  %% Drawing a Contour Plot % Draw a contour plot with 15 copper colored contour lines.  load penny.mat contour(P,15) colormap(copper) axis ij square  %% Drawing a Pseudocolor Plot % Draw a pseudocolor plot with brightness proportional to height.  color(P) axis ij square shading flat</pre>

## 要求

- MATLAB R2016a - MATLAB 支持 R2016a 及更高版本中的实时脚本，以及 R2018a 及更高版本中的实时函数。
- 操作系统 - 在 MATLAB 支持的多数操作系统中，MATLAB 都支持实时编辑器。有关详细信息，请参阅系统要求。有少数几个 MATLAB 支持的操作系统不支持实时编辑器。

不支持的版本包括：

- Red Hat Enterprise Linux 6。
- Red Hat Enterprise Linux 7。
- SUSE Linux Enterprise Desktop 版本 13.0 及更早版本。
- Debian 7.6 及更早版本。

某些操作系统需要额外配置才能运行实时编辑器。如果您无法在系统中运行实时编辑器，请联系技术支持以了解有关如何配置系统的信息。

## 不支持的功能

在决定是否创建实时脚本或函数时，请务必注意在实时编辑器中不受支持的几项功能：

- 类 - 实时编辑器不支持类。应将类创建为纯代码文件 (.m)。然后，您可以在您的实时脚本或函数中使用这些类。
- 编辑器预设 - 实时编辑器会忽略大多数编辑器预设，包括自定义键盘快捷方式和 Emacs 式键盘快捷方式。

## 将实时脚本和函数另存为纯代码

要将实时脚本或函数另存为纯代码文件 (.m)，请执行下列操作：

- 1 在实时编辑器选项卡上的**文件**部分中，选择**保存 > 另存为...**。
- 2 在显示的对话框中，选择“MATLAB 代码文件 (\*.m)”作为**保存类型**。
- 3 点击**保存**。

保存时，MATLAB 会将所有格式化内容转换为发布标记。

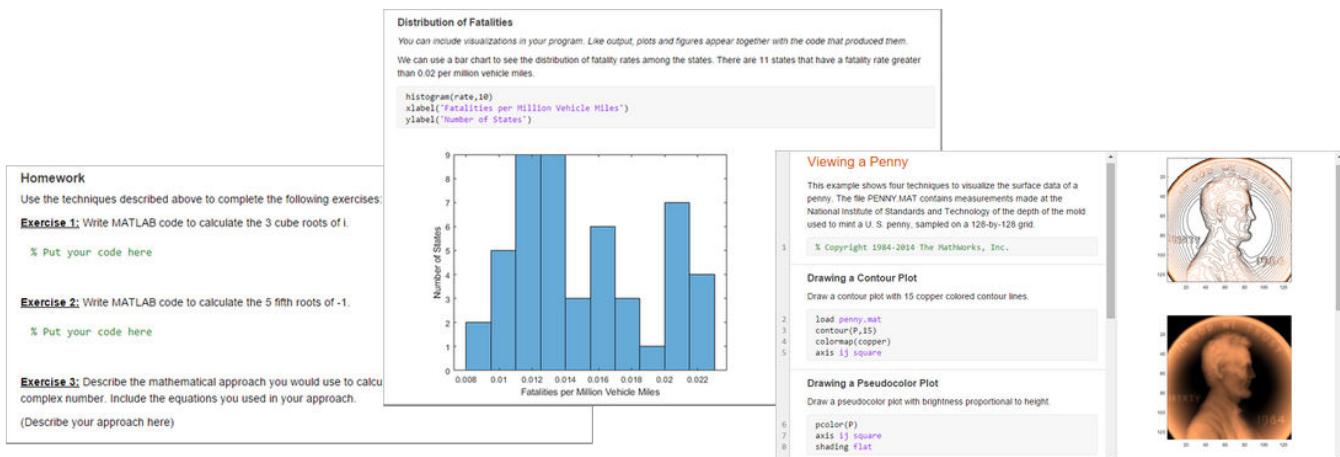
## 另请参阅

### 相关示例

- “在实时编辑器中创建实时脚本”（第 19-6 页）
- “实时代码文件格式 (.mlx)”（第 19-59 页）
- MATLAB 实时脚本库

## 在实时编辑器中创建实时脚本

实时脚本是在一个称为实时编辑器的交互式环境中同时包含代码、输出和格式化文本的程序文件。在实时脚本中，您可以编写代码并查看生成的输出和图形以及相应的源代码。添加格式化文本、图像、超链接和方程，以创建可与其他人共享的交互式记叙脚本。



## 创建实时脚本

要在实时编辑器中创建实时脚本，请转到[主页](#)选项卡并点击[新建实时脚本](#) 。您也可以在命令行窗口中使用 `edit` 函数。例如，键入 `edit penny mlx` 以打开或创建文件 `penny mlx`。为确保创建实时脚本，请指定 `.mlx` 扩展名。如果未指定扩展名，则 MATLAB 会默认文件的扩展名为 `.m`，这种扩展名仅支持纯代码。

### 以实时脚本方式打开现有脚本

如果您已有一个脚本，可以将其以实时脚本方式在实时编辑器中打开。以实时脚本方式打开脚本会创建一个文件副本，并保持原始文件不变。MATLAB 会将原始脚本中的发布标记转换为新实时脚本中的格式化内容。

要通过编辑器将现有脚本 (`.m`) 以实时脚本 (`.mlx`) 方式打开，请右键点击文档选项卡，然后从上下文菜单中选择将 `scriptName` 以实时脚本方式打开。您还可以转至[编辑器](#)选项卡，点击[保存](#) ，然后选择[另存为](#)。然后，将[保存类型](#): 设置为 “**MATLAB 实时代码文件 (\*.mlx)**” 并点击[保存](#)。

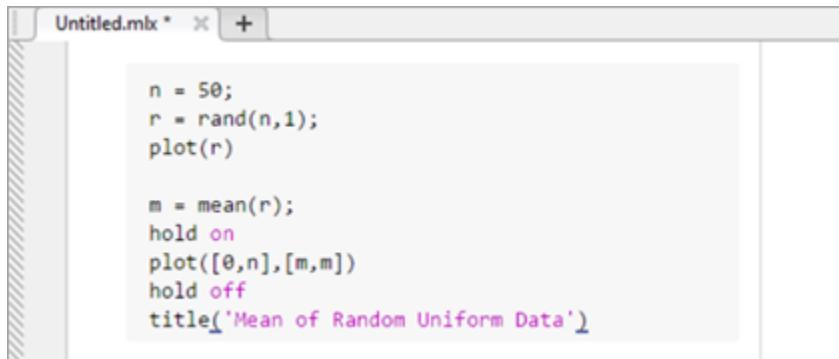
**注意** 您必须使用所述的转换方法之一将脚本转换为实时脚本。仅使用 `.mlx` 扩展名重命名该脚本行不通，并可能损坏文件。

## 添加代码

创建实时脚本后，您可以添加并运行代码。例如，添加以下代码，以绘制随机数据向量图，并在绘图中的均值处绘制一条水平线。

```
n = 50;
r = rand(n,1);
plot(r)
```

```
m = mean(r);
hold on
plot([0,n],[m,m])
hold off
title('Mean of Random Uniform Data')
```



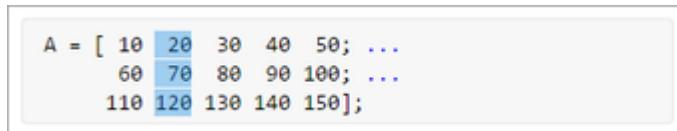
默认情况下，在实时编辑器中输入代码时，MATLAB 会自动补全块结尾、括号和引号。例如，键入 **if**，然后按 **Enter** 键。MATLAB 会自动添加 **end** 语句。



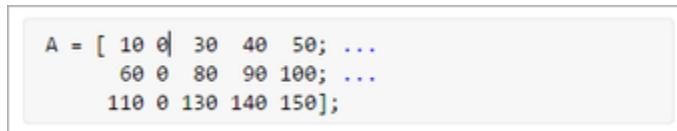
当拆分为两行时，MATLAB 还会自动补全注释、字符向量、字符串和圆括号。要退出自动补全，请按 **Ctrl + Z** 或撤消 按钮。默认情况下会启用自动补全。要禁用它们，请参阅“编辑器/调试器自动编码预设项”。

添加或编辑代码时，您可以选择和编辑一个矩形区域的代码（也称为列选择或块编辑）。如果要复制或删除多列数据（而不是若干行），或者要一次性编辑多行，该功能非常有用。要选择一个矩形区域，请在进行选择时按 **Alt** 键。

例如，选择 A 中的第二列数据。



键入 **0** 可将所有选定的值设置为 0。



## 运行代码

要运行代码，请点击代码左侧的斜纹竖条。也可以转到**实时编辑器**选项卡并点击**运行**。当您的程序正在运行时，系统会在编辑器窗口左上方显示一个状态指示符 。代码行左侧的灰色闪烁条指示 MATLAB 正在计算的行。要导航至 MATLAB 正在计算的行，请点击状态指示符。

如果在 MATLAB 运行程序时出错，或者 MATLAB 检测到您的代码中存在重大问题，则状态指示符会变为错误图标 。要导航至相应错误，请点击该图标。代码行右侧的错误图标 指示该错误。相应的错误消息显示为输出。

您不需要保存实时脚本即可运行它。当您确实要保存实时脚本时，MATLAB 会自动使用 .mlx 扩展名保存它。例如，转到**实时编辑器**选项卡，点击 **保存**，然后输入名称 **plotRand**。MATLAB 会将实时脚本另存为 **plotRand mlx**。

## 显示输出

默认情况下，MATLAB 会在代码右侧显示输出。每个输出都会随创建它的代码行并排显示，就像在命令行窗口中一样。

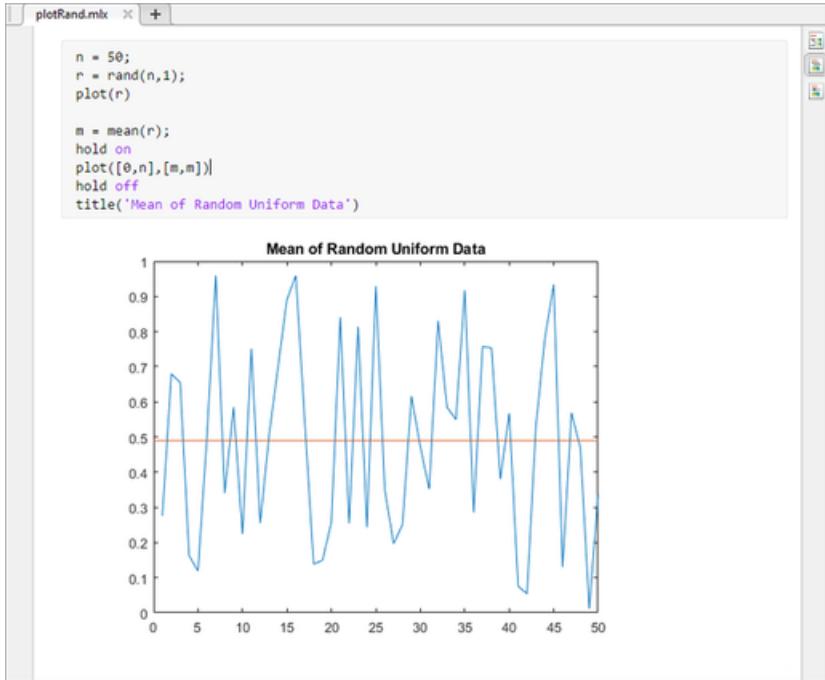


您可以向左或向右拖动代码和输出之间的调整大小栏，以更改输出显示面板的大小。

要清除输出，请右键点击输出或创建输出的代码行，并选择**清除输出**。要清除全部输出，请右键点击脚本中的任意位置，并选择**清除全部输出**。或者，转到**视图**选项卡，并在**输出**部分中，点击**清除全部输出**按钮。

滚动时，MATLAB 会将输出与用于生成输出的代码对齐。要禁用输出与代码对齐模式，请右键点击输出部分，并选择**禁用同步滚动**。

要使输出内嵌在代码中，请点击实时脚本右侧的 **输出内嵌**按钮。您也可以转到**视图**选项卡，然后在**视图**部分中点击 **内嵌输出**按钮。



要仅显示输出、控件和格式化文本并隐藏代码，请点击隐藏代码 按钮。要再次显示代码，请点击输出内嵌 按钮或右侧的 按钮上的输出。

要修改输出中的图窗，请使用图窗坐标区右上角或图窗工具条中的工具。您可以使用这些工具来探查图窗中的数据，并添加格式设置和注释。有关详细信息，请参阅“修改实时脚本中的图窗”（第 19-20 页）。

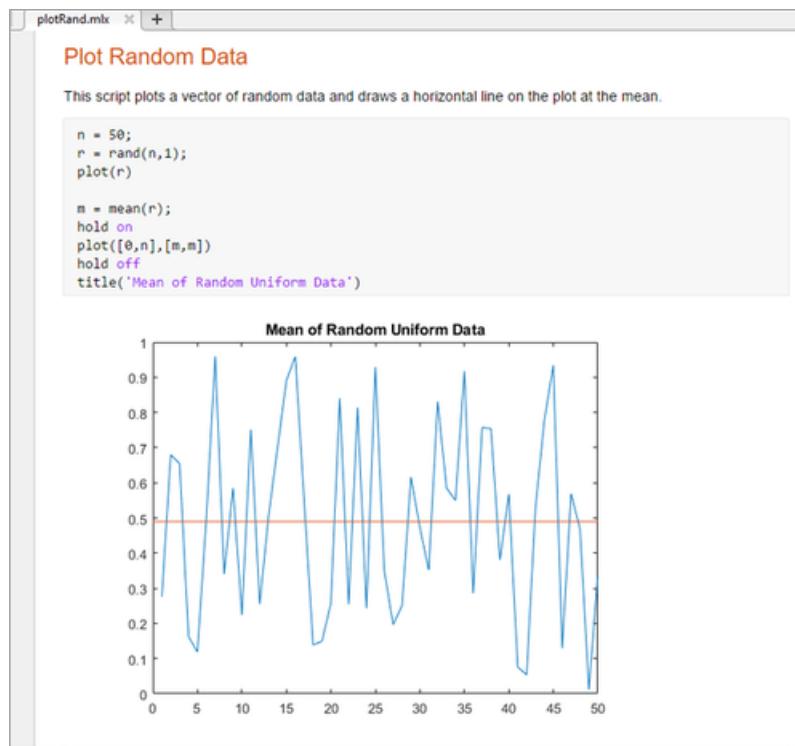
要分别在不同的窗口打开各个输出（例如变量和图窗），请点击输出右上角的 按钮。变量将在变量编辑器中打开，而图窗将在新图窗窗口中打开。在实时脚本之外对变量或图窗所做的更改不会应用于实时脚本中显示的输出。

## 设置文本格式

您可以将格式化文本、超链接、图像和方程添加到实时脚本中，以创建要与其他人共享的演示文档。例如，将标题和某些介绍性文本添加到 `plotRand.mlx`：

- 1** 将光标放在实时脚本的顶部，然后在**实时编辑器**选项卡中选择 **文本**。一个新的文本行将显示在代码上方。
- 2** 点击 并选择“标题”。
- 3** 添加文本 **Plot Random Data**。
- 4** 将光标置于行中，点击 按钮将文本居中。
- 5** 按 **Enter** 键移到下一行。
- 6** 键入文本 **This script plots a vector of random data and draws a horizontal line on the plot at the mean.**

有关详细信息（包括所有可用格式设置选项的列表），请参阅“在实时编辑器中格式化文件”（第 19-29 页）。



要在实时编辑器中调整显示的字体大小，请使用 **Ctrl+鼠标滚轮** 键盘快捷方式。将实时脚本导出为 PDF、Microsoft Word、HTML 或 LaTeX 时，显示字体大小的变化不会保留。

## 另请参阅

### 详细信息

- “在实时编辑器中格式化文件”（第 19-29 页）
- “在实时脚本中运行节”（第 19-11 页）
- “修改实时脚本中的图窗”（第 19-20 页）
- MATLAB 实时脚本库

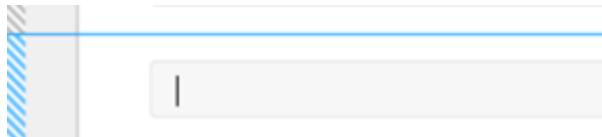
# 在实时脚本中运行节

## 将文件划分为不同节

实时脚本通常包含许多命令和文本行。您通常一次仅专注于程序中的某个部分，分块操作代码和相关文本。要更轻松地进行文档管理和导航，请将您的文件划分为多个节。代码、输出和相关文本可同时显示在单个节中。

要向实时脚本中插入分节符，请转到**实时编辑器**选项卡，然后在**节**部分中点击**分节符**按钮。新节以蓝色突出显示，指示它处于选定状态。该节左侧的斜纹竖条指示它是旧有的。旧有节是尚未运行的节，或自上次运行后尚未修改的节。

此图像演示了实时脚本中的一个新空节。

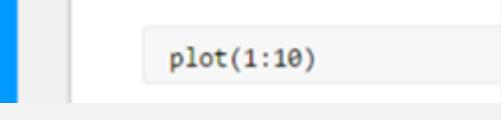


要删除分节符，请点击紧邻其后的行的开头，然后按**退格键**。您也可以点击紧邻分节符的上一行末尾并按**Delete 键**。

## 执行节

通过分别计算每个节或一次运行所有代码来运行实时脚本。要分别执行某节，该节必须包含它需要的所有值，或这些值必须存在于 MATLAB 工作区中。节计算会运行当前选定的节（以蓝色突出显示）。如果程序文件中只有一个节，则该节不会突出显示，因为它始终处于选中状态。

此表介绍运行代码的不同方法。

操作	说明
运行选定节中的代码。	点击该节左侧的蓝色竖条。  或 在 <b>实时编辑器</b> 选项卡上的 <b>节</b> 部分中，点击  <b>运行节</b> 。
运行选定节中的代码，然后移至下一节。	在 <b>实时编辑器</b> 选项卡上的 <b>节</b> 部分中，选择  <b>运行并前进</b> 。
运行选定节中的代码，然后运行选定节后的所有代码。	在 <b>实时编辑器</b> 选项卡上的 <b>节</b> 部分中，选择  <b>运行到结束</b> 。
运行文件中的所有代码。	在 <b>实时编辑器</b> 选项卡上的 <b>运行</b> 部分中，点击  <b>运行</b> 。

## 另请参阅

### 相关示例

- “修改实时脚本中的图窗” (第 19-20 页)
- “在实时编辑器中调试代码” (第 19-13 页)

## 在实时编辑器中调试代码

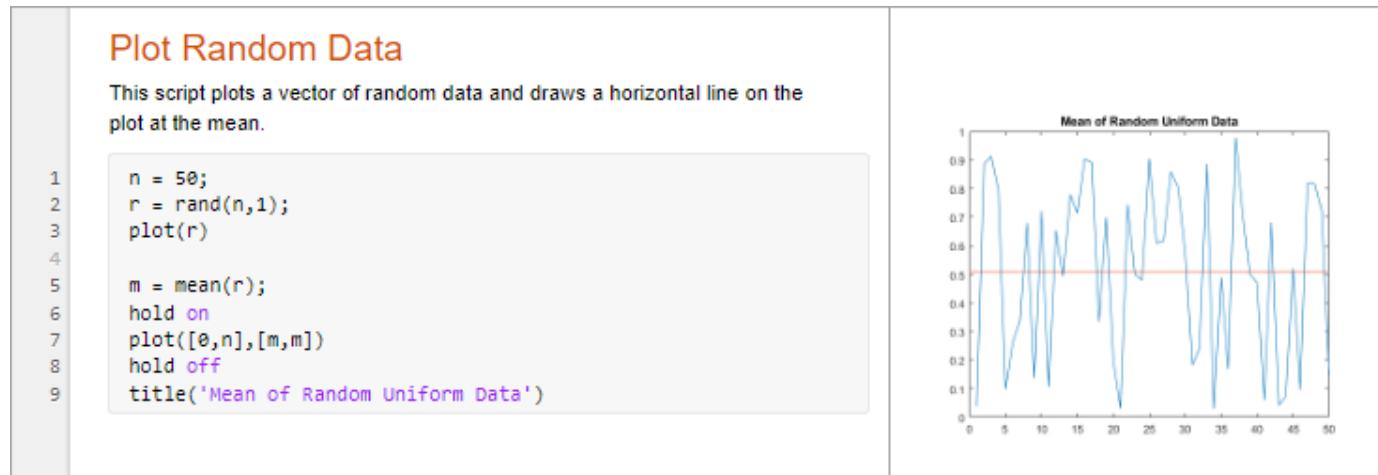
要诊断实时脚本或函数中的问题，请在实时编辑器中调试您的代码。在实时编辑器中有几种调试方法：

- 通过删除分号来显示输出。
- 使用 “运行到此行” 按钮运行到特定代码行并暂停。
- 使用 “步入” 按钮在暂停状态下步入函数和脚本。
- 将断点添加到文件中，以便在运行时在特定行暂停。

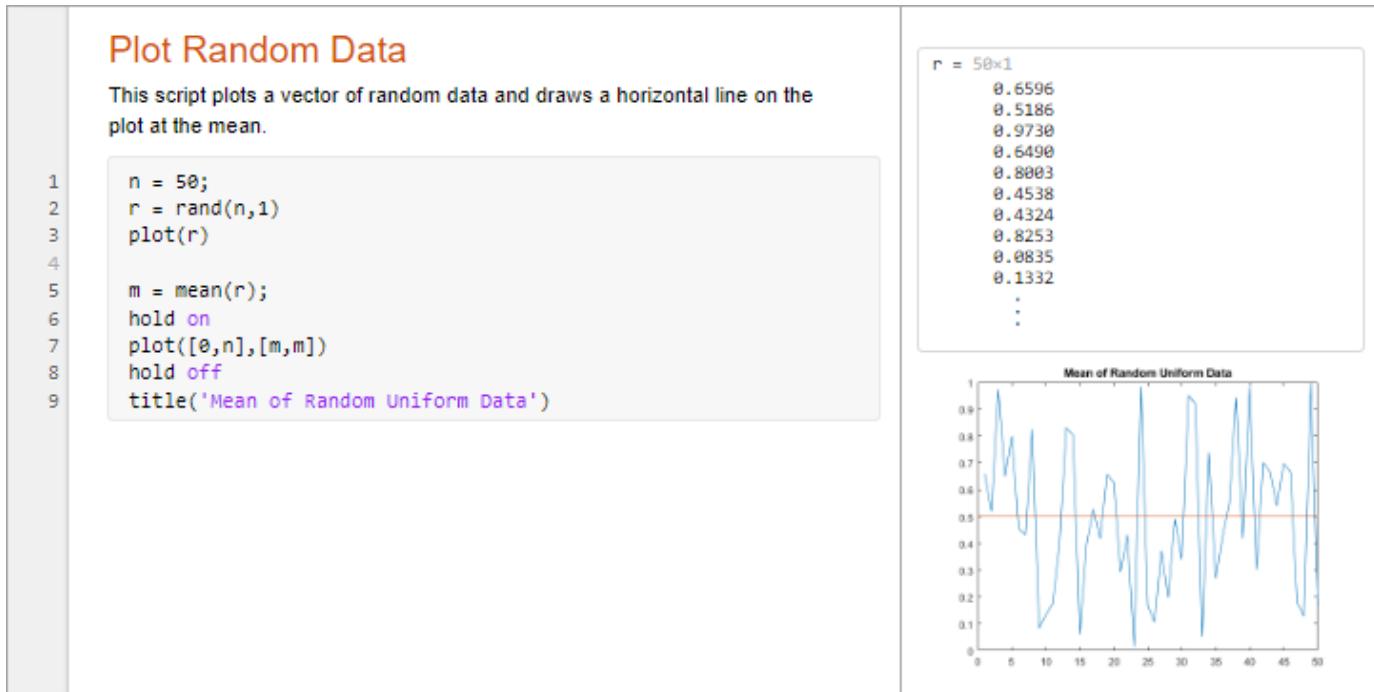
### 显示输出

要确定实时脚本或函数中发生问题的位置，一个简单的方法是显示输出。要显示某行的输出，请删除该行末尾的分号。实时编辑器会将输出与创建它的代码行并排显示，从而让您轻松确定发生问题的位置。

例如，假设您有名为 **plotRand.mlx** 的脚本，它绘制随机数据向量图，并在绘图中的均值处绘制一条水平线。



要在第 2 行显示 `rand` 函数的输出，请删除行尾的分号。要在实时编辑器中显示行号（如果它们不可见），请转到视图选项卡并按 行号 按钮。MATLAB 会显示 `r` 的值。

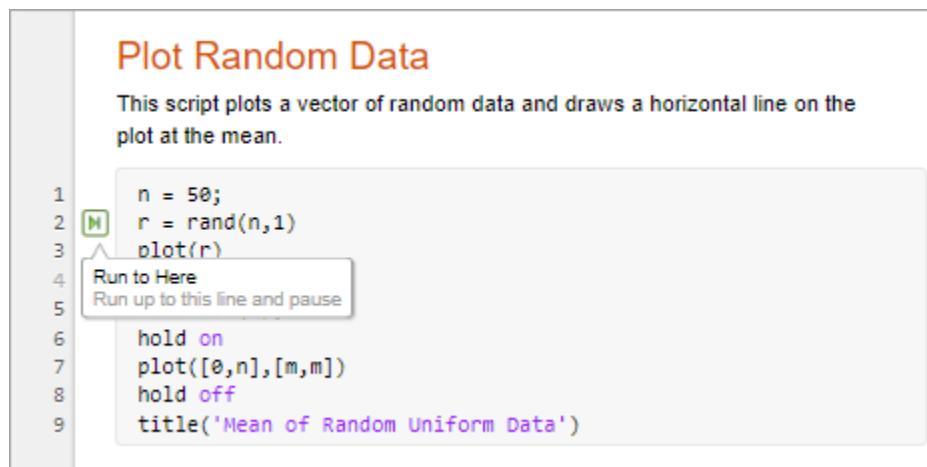


## 使用运行到此行进行调试

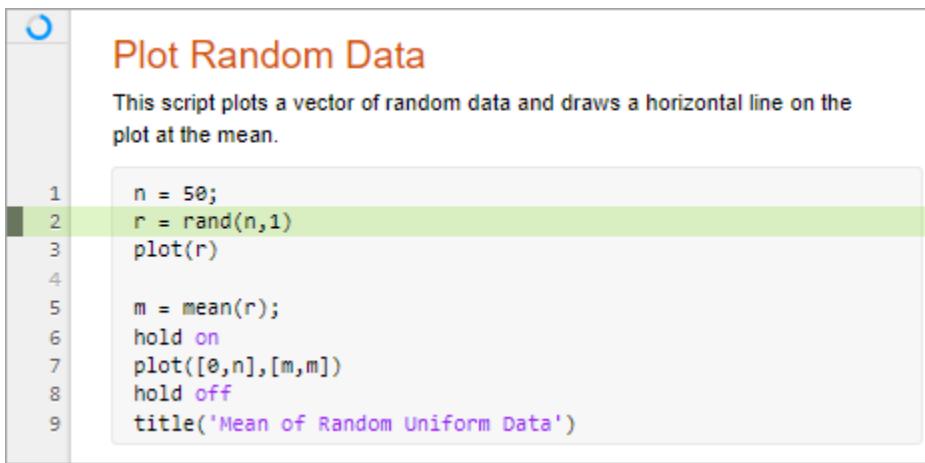
如果要显示单个变量的状态，则显示输出很有用。要了解工作区中所有变量的状态，则请运行您的实时脚本，然后在运行指定的代码行之前暂停。

要运行到指定的代码行，然后暂停，请点击该行左侧的“运行到此行”按钮。如果不能到达所选行，则 MATLAB 会继续运行，直到到达文件末尾或遇到断点。“运行到此行”按钮仅在调试实时函数时出现。

例如，点击 `plotRand mlx` 中第 2 行左侧的按钮。MATLAB 从第 1 行开始运行 `plotRand mlx` 并在运行第 2 行之前暂停。



当 MATLAB 暂停时，实时编辑器选项卡中的运行按钮变为继续按钮。实时编辑器会以绿色突出显示代码行，以此指示 MATLAB 在该行暂停。在 MATLAB 继续运行之前，突出显示的行不会运行。



```

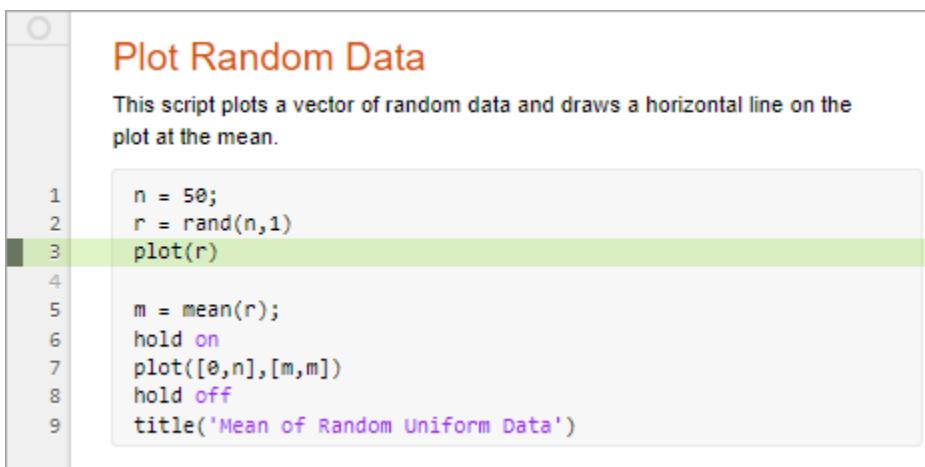
1 n = 50;
2 r = rand(n,1)
3 plot(r)
4
5 m = mean(r);
6 hold on
7 plot([0,n],[m,m])
8 hold off
9 title('Mean of Random Uniform Data')

```

**提示** 最好避免在 MATLAB 暂停时修改文件。在 MATLAB 暂停期间更改的代码不会运行，直至 MATLAB 完成代码运行并重新运行代码，这些更改后的代码才会运行。

要继续运行代码，请点击  继续 按钮。MATLAB 会继续运行该文件，直到到达文件末尾或断点。您也可以点击要继续运行到的代码行左侧的  按钮。

要继续逐行运行代码，请在实时编辑器选项卡上点击  步进。MATLAB 会执行当前所在的行，并在下一行暂停。



```

1 n = 50;
2 r = rand(n,1)
3 plot(r)
4
5 m = mean(r);
6 hold on
7 plot([0,n],[m,m])
8 hold off
9 title('Mean of Random Uniform Data')

```

您还可以运行到光标所在的行，方法是：转至实时编辑器选项卡，选择  步进，然后选择  运行到光标处。

## 在调试时查看变量值

要在 MATLAB 暂停期间查看某变量的值，请将鼠标指针悬停在该变量上方。该变量的当前值将显示在数据提示中。数据提示会一直在视图中，直到您移动指针。要禁用数据提示，请转到视图选项卡并按  数据提示按钮。

```

1 n = 50;
2 r = rand(n,1)
3 plot(r)
4 n = 50
5 m = mean(r);
6 hold on
7 plot([0,n],[m,m])
8 hold off
9 title('Mean of Random Uniform Data')

```

您也可以通过在命令行窗口中键入变量名称来查看变量的值。例如，要查看变量 `n` 的值，请键入 `n` 并按 **Enter** 键。命令行窗口会显示变量名及其值。要查看当前工作区中的所有变量，请使用工作区浏览器。

## 暂停运行文件

要暂停正在运行的程序，请转到**实时编辑器**选项卡并点击 **暂停** 按钮。MATLAB 会在下一个可执行代码行处暂停，并且 **暂停** 按钮会更改为 **继续**按钮。要继续运行，请按 **继续**按钮。

如果您想要检查长时间运行的程序的进度以确保它按预期运行，则暂停很有用。

---

**注意** 点击暂停按钮会使 MATLAB 在您自己的程序文件外的文件中暂停。按 **继续**按钮会继续运行而不更改文件结果。

---

## 结束调试会话

在发现问题后，通过转至**实时编辑器**选项卡并点击 **停止** 来结束调试会话。为避免混淆，请务必在每次完成调试时结束调试会话。当您保存时，实时编辑器会自动结束调试会话。

## 步入函数

调试时，您可以步入所调用的文件中，在要检查值的点暂停。要步入文件，请点击要步入的函数左侧的 按钮。您也可以使用 **F11** 键步入函数。实时编辑器仅在该行包含对另一个函数的调用时才显示该按钮。

默认情况下， 按钮仅在用户定义的函数和脚本中显示。要在 MathWorks 函数中也显示该按钮，请在**主页**选项卡的**环境**部分中，点击**预设**。然后，选择**MATLAB > 编辑器/调试器**，然后在**在实时编辑器中调试**部分清除**对用户定义的函数仅显示“步入”按钮**复选框。

步入后，点击文件顶部的 按钮以运行所调用函数的其余部分，离开所调用的函数，然后暂停。您也可以使用 **Shift+F11** 步出函数。

您还可以步入和步出函数，方法是：进入**实时编辑器**选项卡，选择 **步进**，然后选择 **步入**或 **步出**。这些按钮不遵循仅对用户定义的函数显示“步入”按钮预设项，并且始终可以步入和步出用户定义的函数和 MathWorks 函数。

## 在工作区中检查变量

当您步入所调用的函数或文件时，实时编辑器将显示 MATLAB 在当前行暂停前执行的函数列表。该列表显示在文件的顶部，按顺序显示各函数，最左侧是第一个调用的脚本或函数，最右侧是 MATLAB 暂停处的当前脚本或函数。该列表称为函数调用堆栈。



对于函数调用堆栈中的每个函数，都有一个相应的工作区。工作区包含您在 MATLAB 中创建的变量或从数据文件或其他程序导入的变量。通过命令行窗口分配或使用脚本创建的变量属于基础工作区。在函数中创建的变量属于其自己的函数工作区。

要在调试时检验变量，您必须首先选择其工作区。在函数调用堆栈中选定的函数指示当前工作区。要选择或更改工作区，请点击函数调用堆栈中的函数。MATLAB 在实时编辑器中打开该函数，并将当前工作区更改为该函数的工作区。

选择工作区后，您可以使用工作区浏览器查看其中的变量的值，或通过在实时编辑器中查看数据提示来查看变量值。

## 添加断点并运行

如果文件中存在您每次运行时都要暂停的代码行，请在这些行上添加断点。要在实时编辑器中添加断点，请点击要设置断点的可执行代码行左侧的灰色区域。例如，点击以下代码中第 3 行旁边的区域可在该行添加断点。您也可以使用 **F12** 键来设置断点。

```

1 n = 50;
2 r = rand(n,1);
3 plot(r)
4
5 m = mean(r);
6 hold on
7 plot([0,n],[m,m])
8 hold off
9 title('Mean of Random Uniform Data')

```

运行文件时，MATLAB 会在由断点指示的代码行处暂停。

## 清除断点

当您关闭并重新打开文件时，断点会保存。

要清除断点，请右键点击断点，然后从上下文菜单中选择清除断点。您也可以使用 F12 键清除断点。

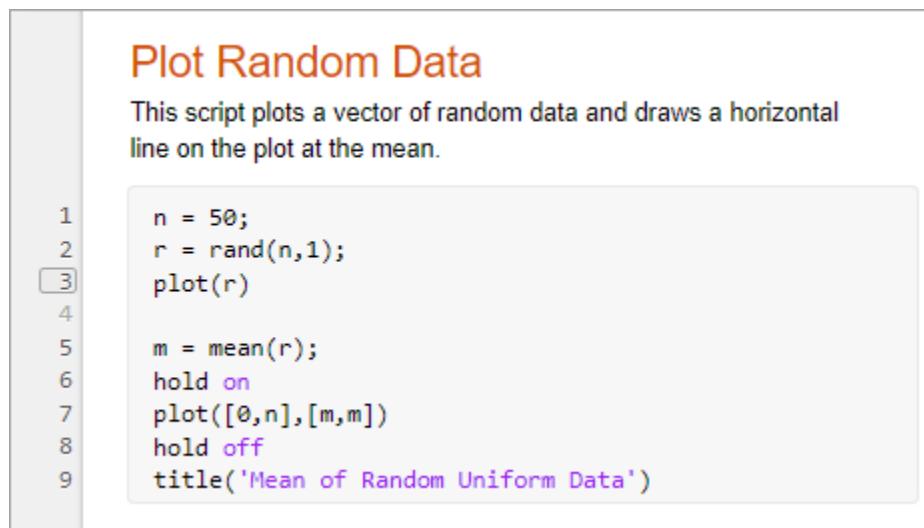
要清除文件中的所有断点，请选择**清除文件中的所有断点**。要清除所有文件中的全部断点，请选择**全部清除**。

在您终止 MATLAB 会话时，系统会自动清除断点。

### 禁用断点

您可以禁用所选断点以便让程序暂时忽略它们而不中断运行。例如，在确定并更正问题后，您可能会禁用断点。

要禁用断点，请右键点击它并从上下文菜单中选择**禁用断点**。断点变为灰色，表示它已禁用。



```

1 n = 50;
2 r = rand(n,1);
3 plot(r)
4
5 m = mean(r);
6 hold on
7 plot([0,n],[m,m])
8 hold off
9 title('Mean of Random Uniform Data')

```

要重新启用断点，请右键点击它并选择**启用断点**。要启用或禁用文件中的所有断点，请选择**启用文件中的所有断点**或**禁用文件中的所有断点**。只有当至少有一个断点可供启用或禁用时，才能使用这些选项。

### 添加条件断点

您可以将条件添加到断点以指示 MATLAB 在何种情况下在指定行暂停。要添加条件，请右键点击断点并选择“设置/修改条件”。当编辑器对话框打开时，请输入相应条件并点击**确定**。条件是返回逻辑标量值的任何有效的 MATLAB 表达式。MATLAB 在运行代码行之前对条件求值。

例如，假设您希望仅在随机生成的数据包含 0 时才在 **plotRand.mlx** 中暂停。

在第 3 行添加具有以下条件的断点：

```
any(r == 0)
```

该行显示一个黄色条件断点。

**Plot Random Data**

This script plots a vector of random data and draws a horizontal line on the plot at the mean.

```

1 n = 50;
2 r = rand(n,1);
3 plot(r)
4
5 m = mean(r);
6 hold on
7 plot([0,n],[m,m])
8 hold off
9 title('Mean of Random Uniform Data')

```

运行该文件时，MATLAB 在满足条件时才会在指定行处暂停。例如，在 **plotRand** 示例中，如果 **r** 中有任一值等于 0，MATLAB 会在运行第 3 行之前暂停。

### 在匿名函数中添加断点

您可以在包含匿名函数的 MATLAB 代码行中添加多个断点。您可以为行本身和行中的每个匿名函数设置一个断点。

要添加断点，请点击可执行代码行左侧的灰色区域，即可为该行添加断点。MATLAB 为该行添加一个断点，并为该行中的每个匿名函数添加一个处于禁用状态的断点。要为匿名函数启用断点，请右键点击它并选择**启用断点**。

要查看某行中所有断点的相关信息，请将光标悬停在断点图标上。随即会显示工具提示及可用信息。例如，在以下代码中，第 5 行包含两个匿名函数，每个匿名函数都有一个断点。

```

5 g = @(c) integral(@(x) (x.^2 + c*x + 1),0,1));
6 g(3);

```

Pause execution at line 5 in anonymous function 1

当您在匿名函数中设置断点时，MATLAB 会在调用匿名函数时暂停。以绿色突出显示的代码行是代码定义匿名函数的位置。以灰色突出显示的代码行是代码调用匿名函数的位置。例如，在以下代码中，MATLAB 在为匿名函数 **g** 设置的断点处暂停程序，该函数在第 5 行定义，并在第 6 行调用。

```

5 g = @(c) integral(@(x) (x.^2 + c*x + 1),0,1);
6 g(3);

```

## 另请参阅

### 详细信息

- “在实时脚本中运行节”（第 19-11 页）

## 修改实时脚本中的图窗

您可以在实时编辑器中以交互方式修改图窗。使用提供的工具探查数据并为图窗添加格式设置、注释或其他坐标区。然后，使用所生成的代码对您的代码进行更新以反映更改。

### 探查数据

当您将鼠标悬停在图窗上方时，您可以使用出现在图窗坐标区右上角的工具来平移、缩放和旋转脚本中的图窗。

- - 添加数据提示以显示数据值。
- - 旋转绘图（仅限三维绘图）。
- - 平移绘图。
- 、— - 放大和缩小绘图。
- - 撤消所有平移、缩放和旋转操作并还原绘图的原始视图。

要撤消或重做操作，请点击工具条右上角的 或 。

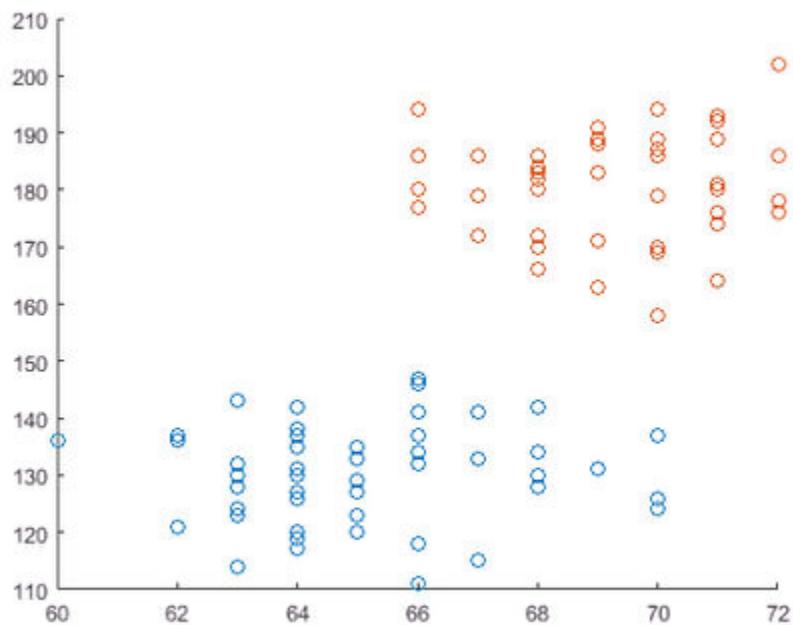
### 注意

- 当您打开所保存的实时脚本时，每个输出图窗的旁边会显示 ，指示交互式工具尚无法使用。要使这些工具可用，请运行实时脚本。
- 交互式工具无法用于不可见的坐标区。

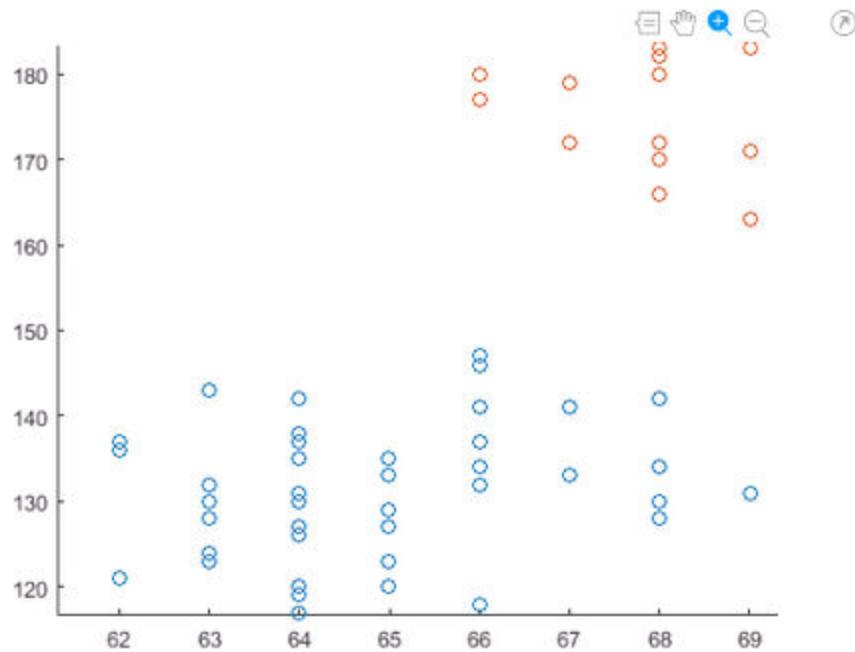
假定您要了解 100 位不同患者的健康信息。请创建一个名为 `patients mlx` 的实时脚本并添加代码，这些代码用于加载数据并添加一个散点图，以显示两组患者（女性和男性）的身高与体重对照情况。转至**实时编辑器**选项卡并点击 运行以运行该代码。

```
load patients

figure
Gender = categorical(Gender);
scatter(Height(Gender=='Female'),Weight(Gender=='Female'));
hold on
scatter(Height(Gender=='Male'),Weight(Gender=='Male'));
hold off
```



探查患者身高为 64 英寸的数据点。选择 按钮并点击身高为 64 的数据点之一。MATLAB 将放大图窗。



```
Code ▾
xlim([61.31 69.31])
ylim([116.7 183.3])
```

[Update Code](#)

[Copy](#)

## 更改图窗时更新代码

修改实时脚本中的输出图窗时，对图窗所做的更改不会自动添加到脚本中。在每次交互时 MATLAB 都会生成重现交互所需的代码，并将此代码显示在图窗的下方或右侧。使用**更新代码**按钮以将所生成的代码添加到脚本中。这样可确保在下次运行实时脚本时重现交互。

例如，在实时脚本 **patients mlx** 中，在放大身高为 64 的患者后，点击**更新代码**按钮。MATLAB 会将所生成的代码添加在包含用于创建绘图的代码的行后面。

```
xlim([61.31 69.31])
ylim([116.7 183.3])
```

如果 MATLAB 无法确定在何处放置所生成的代码，**更新代码**按钮将被禁用。例如，如果您修改代码而不重新运行脚本，则会出现这种情况。在这种情况下，请使用**复制**按钮将所生成的代码复制到剪贴板。然后，您就可以将代码粘贴到脚本中的相应位置。

## 添加格式设置和注释

除了探查数据之外，您还可以通过添加标题、标签、图例、网格线、箭头和线条以交互方式设置图窗格式并进行注释。要添加项目，请首先选择所需图窗。然后，转至**图窗**选项卡，并在**注释**部分中选择可用选项之一。使用该部分右侧的向下箭头以显示所有可用注释。要将格式设置或注释选项添加到收藏夹中，请点击所需注释按钮右上方的五角星。要撤消或重做格式设置或注释操作，请点击工具条右上角的 或 。

注释选项包括：

- **标题** - 向坐标区添加标题。要修改某个现有标题，请点击该标题并输入修改后的文本。
- **X 标签**, **Y 标签** - 向坐标区添加标签。要修改某个现有标签，请点击该标签并输入修改后的文本。
- **图例** - 向图窗添加图例。要修改现有图例说明，请点击该说明并输入修改后的文本。从**注释**部分选择**删除图例**可删除坐标区中的图例。
- **颜色栏** - 向图窗添加颜色栏图例。从**注释**部分选择**删除颜色栏**可删除坐标区中的颜色栏图例。
- **网格**, **X 网格**, **Y 网格** - 向图窗添加网格线。从**注释**部分选择**删除网格**可删除坐标区中的所有网格线。
- **线条**, **箭头**, **文本箭头**, **双箭头** - 向图窗添加线条或箭头注释。从尾部至头部绘制箭头。要移动现有注释，请点击该注释将其选中，然后拖动到所需位置。按 **Delete** 键可删除所选注释。

---

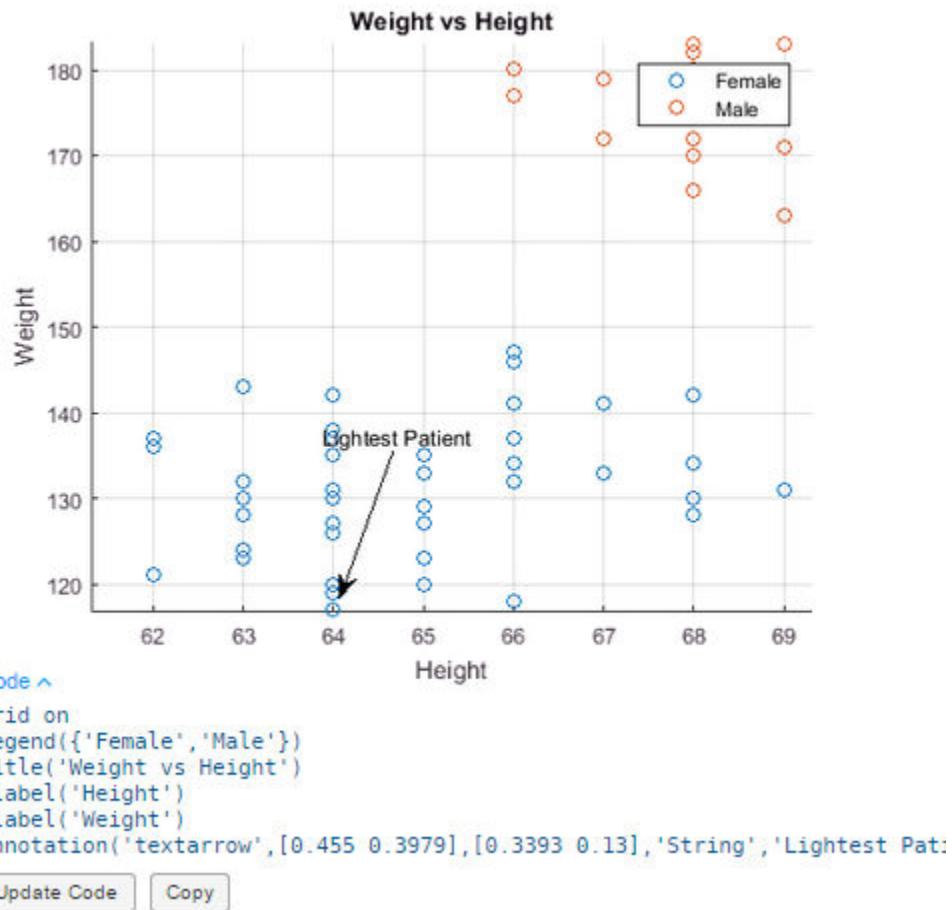
**注意** 不可见的坐标区不支持使用**图窗**选项卡来添加格式设置和注释。

---

例如，假定您要为 **patients mlx** 中的图窗添加格式设置和注释。

- 1 添加标题 - 在注释部分中选择  标题。此时将出现一个蓝色矩形，提示您输入文本 Weight vs. Height，然后按 Enter 键。
- 2 添加 X 标签和 Y 标签 - 在注释部分中选择  X 标签。此时将出现一个蓝色矩形，提示您输入文本。键入文本 Height，然后按 Enter。选择  Y 标签。此时将出现一个蓝色矩形，提示您输入文本。键入文本 Weight，然后按 Enter。
- 3 添加图例 - 在注释部分中选择  图例。坐标区的右上角将出现一个图例。点击该图例中的 data1 说明，然后将文本替换为 Female。点击该图例中的 data2 说明，然后将文本替换为 Male。按 Enter。
- 4 添加网格线 - 在注释部分中选择  网格。坐标区中将出现网格线。
- 5 添加箭头注释 - 在注释部分中选择  文本箭头。按从尾到头的顺序绘制箭头，将箭头指向散点图上体重最轻的患者。输入文本 Lightest Patient，然后按 Enter
- 6 更新代码 - 在选定的图窗中，点击更新代码按钮。实时脚本现在包含重现图窗更改所需的代码。

```
grid on
legend({'Female','Male'})
title('Weight vs Height')
xlabel('Height')
ylabel('Weight')
annotation('textarrow',[0.455 0.3979],[0.3393 0.13],'String','Lightest Patient');
```



## 添加和修改多个子图

您可以通过在图窗中创建子图来合并多个绘图。要在图窗中添加多个子图，请使用**子图**按钮将图窗划分为子图网格。首选，选择所需图窗。然后，转至**图窗**选项卡并使用**子图**  $\downarrow$  按钮选择子图布局。只有当图窗包含一个子图时，您才能为图窗添加其他子图。如果图窗包含多个子图，**子图**按钮将被禁用。

例如，假定您要比较吸烟患者和不吸烟患者的血压。创建一个称为 `patients_smoking mlx` 的实时脚本，并添加用于加载 100 位不同患者的健康信息的代码。

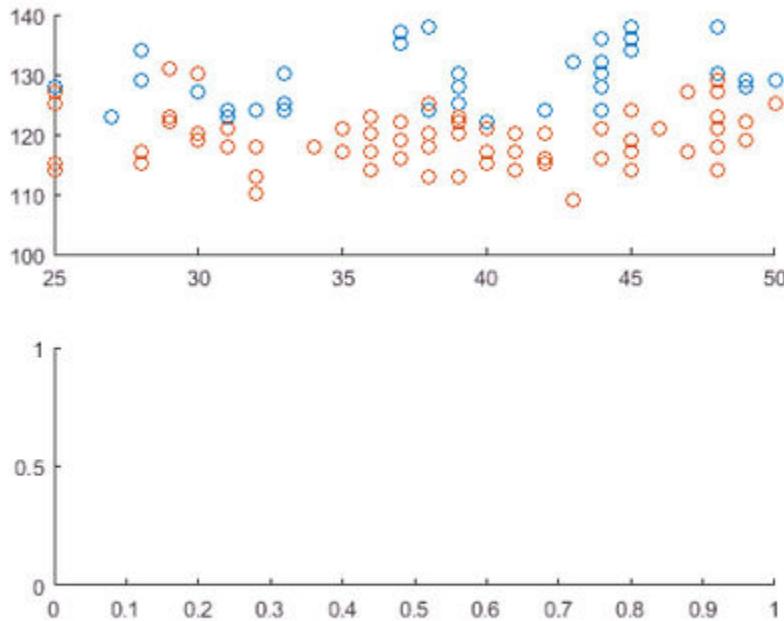
```
load patients
```

转至**实时编辑器**选项卡并点击 **运行**以运行该代码。

添加一个散点图，用于显示吸烟患者与不吸烟患者的收缩压对比情况。运行代码。

```
figure
scatter(Age(Smoker==1),Systolic(Smoker==1));
hold on
scatter(Age(Smoker==0),Systolic(Smoker==0));
hold off
```

在**图窗**选项卡中选择**子图**  $\downarrow$ ，然后为两个水平图选择布局。

[Code ^](#)

```
subplot(2,1,1,gca)
subplot(2,1,2)
```

[Update Code](#)[Copy](#)

在新创建的图窗中，点击[更新代码](#)按钮。实时脚本现在包含重现两个子图所需的代码。

```
subplot(2,1,1,gca)
subplot(2,1,2)
```

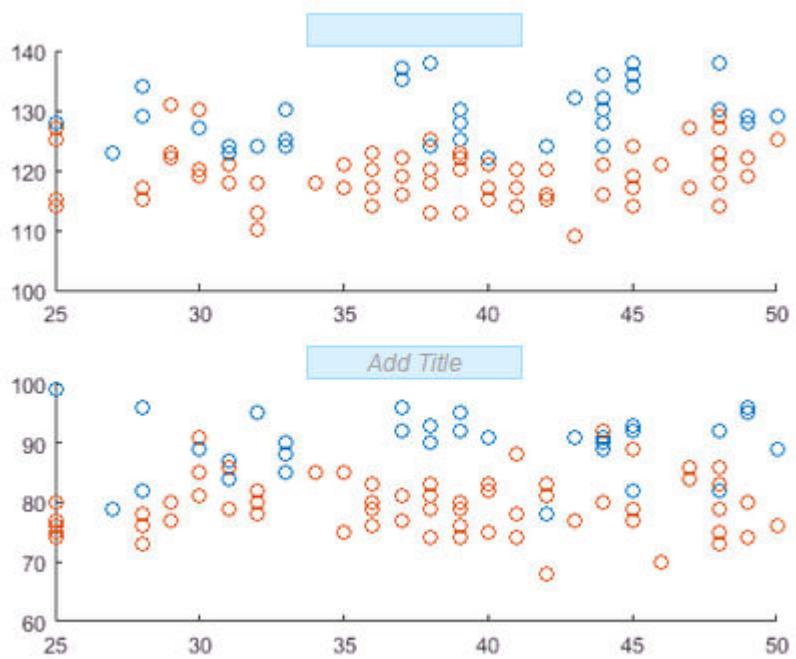
添加一个散点图，用于显示吸烟患者与不吸烟患者的舒张压对比情况。运行代码。

```
scatter(Age(Smoker==1),Diastolic(Smoker==1));
hold on
scatter(Age(Smoker==0),Diastolic(Smoker==0));
hold off
```

添加格式设置：

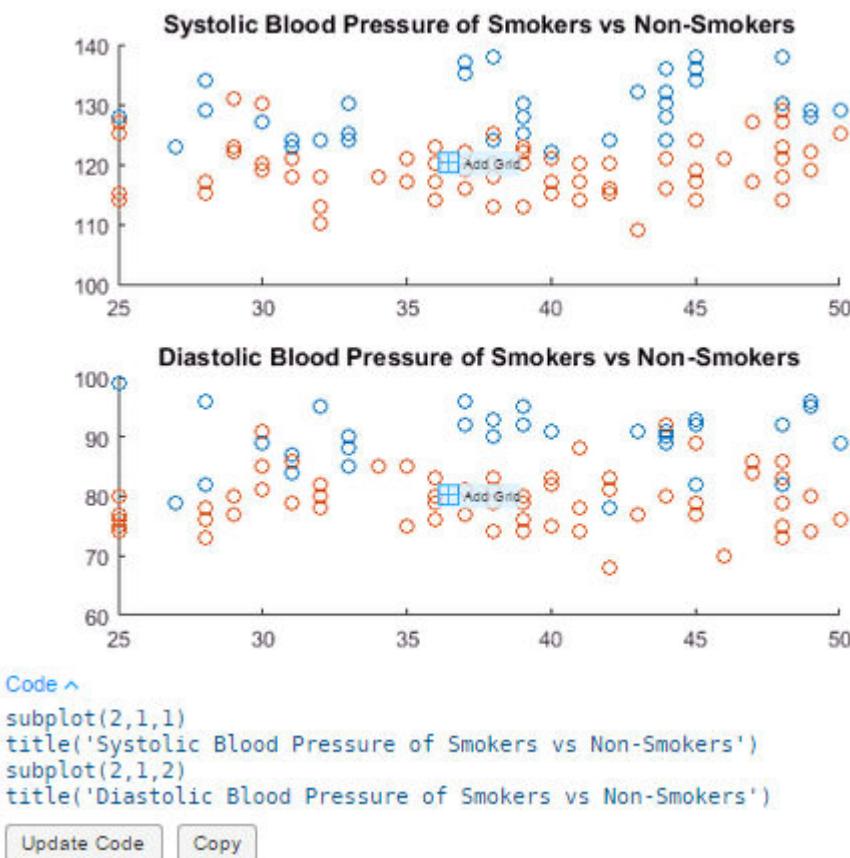
1

**向每个子图添加标题** - 在注释部分中选择 **标题**。每个子图中将出现一个蓝色矩形，提示您输入文本。在第一个子图中键入文本 **Systolic Blood Pressure of Smokers vs Non-Smokers**，在第二个子图中键入文本 **Diastolic Blood Pressure of Smokers vs Non-Smokers**，然后按 **Enter** 键。



2

向每个子图添加网格线 - 在注释部分中选择 网格。每个子图中都会出现一个添加网格按钮。点击每个子图中的添加网格按钮。两个子图中都会出现网格线。

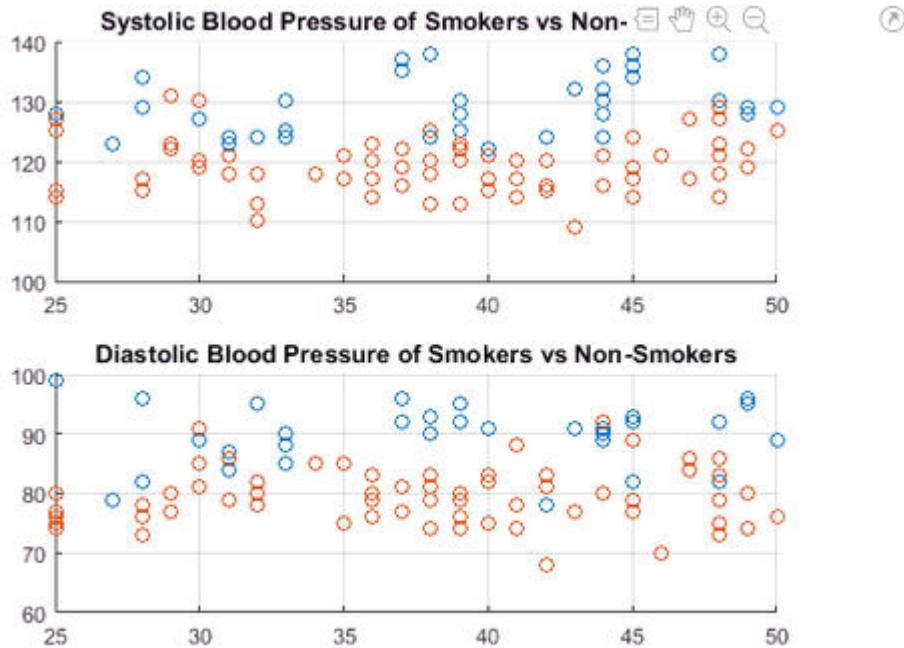


- 3 **更新代码** - 在选定的图窗中，点击**更新代码**按钮。实时脚本现在包含重现图窗更改所需的代码。

```

subplot(2,1,1)
grid on
title('Systolic Blood Pressure of Smokers vs Non-Smokers')
subplot(2,1,2)
grid on
title('Diastolic Blood Pressure of Smokers vs Non-Smokers')

```



## 保存和打印图窗

在图窗修改期间，您可以随时选择保存或打印图窗以供将来使用。

- 1 点击输出右上角的 按钮。这将在单独的图窗窗口中打开图窗。
- 2 a **保存图窗 - 选择文件 > 另存为**。有关保存图窗的详细信息，请参阅“将绘图保存为图像或向量图形文件”或“保存图窗以供以后在 MATLAB 中重新打开”。
   
b **打印图窗 - 选择文件 > 打印**。有关打印图窗的详细信息，请参阅“从“文件”菜单打印图窗”。

**注意** 实时脚本中将不会反映在该单独的图窗窗口中对图窗所做的任何更改。同样，该打开的图窗窗口中也不会反映在实时脚本中对图窗所做的任何更改。

## 另请参阅

### 相关示例

- “在实时编辑器中格式化文件”（第 19-29 页）
- “在实时脚本中运行节”（第 19-11 页）

## 在实时编辑器中格式化文件

您可以将格式化文本、超链接、图像和方程添加到实时脚本和函数中，以创建要与其他人共享的演示文档。

要插入新项目，请转到**插入**选项卡，然后从可用选项中选择：

选项	说明	其他详细信息
 <b>代码</b>	插入空的代码行。	您可以在文本行之前、之后或之间插入代码行。
 <b>分节符</b>	插入分节符。	您可以插入分节符，将实时脚本或函数分为易于管理的节，以便分别求值。节可以包括代码、文本和输出。有关详细信息，请参阅“在实时脚本中运行节”（第 19-11 页）。
 <b>文本</b>	插入空的文本行。	文本行可以包含格式化文本、超链接、图像或方程。您可以在代码行之前、之后或之间插入文本行。
 <b>目录</b>	插入目录。	目录包含文档中所有标题和题头的列表。只有目录的标题是可编辑的。 您只能在文本行中添加目录。如果您将目录插入到代码行中，MATLAB 会将其置于当前代码节的上一行。
 <b>代码示例</b>	插入格式化的代码示例。	代码示例是显示为缩进的等宽文本的示例代码。 选择 <b>纯文本</b> 可将示例代码作为非突出显示的文本插入。 选择 <b>MATLAB</b> 可根据 MATLAB 语法将示例代码作为突出显示的文本插入。
 <b>图像</b>	插入图像。	您只能在文本行中添加图像。如果您向代码行中插入图像，则 MATLAB 会将该图像置于选定代码行正下方的新文本行中。
 <b>超链接</b>	插入超链接。	选择 <b>外部 URL</b> 可插入外部 URL。 选择 <b>内部超链接</b> 可插入指向文档中现有位置的超链接。出现提示时，点击文档中的所需位置以将其选为目标。您还可以使用 <b>Alt + 向上箭头</b> 和 <b>Alt + 向下箭头</b> 键盘快捷方式。位置可以是代码节、文本段落、标题或题头。不支持链接到单行文本或代码。 您只能在文本行中添加超链接。如果您向代码行中插入超链接，则 MATLAB 会将该超链接置于选定代码行正下方的新文本行中。
 <b>方程</b>	插入方程。	您只能在文本行中添加方程。如果您向代码行中插入方程，则 MATLAB 会将该方程置于选定代码行正下方的新文本行中。有关详细信息，请参阅“将方程插入实时编辑器中”（第 19-33 页）。

要设置现有文本的格式，请使用**实时编辑器**选项卡的**文本**部分中包含的各个选项：

格式类型	选项
文本样式	<input type="checkbox"/> Aa 普通 <input type="checkbox"/> Aa 题头 1 <input type="checkbox"/> Aa 题头 2 <input type="checkbox"/> Aa 题头 3 <input checked="" type="checkbox"/> Aa 标题
文本对齐方式	<input type="checkbox"/> 左侧 <input type="checkbox"/> 居中 <input type="checkbox"/> 右侧
列表	<input type="checkbox"/> 一 编号列表 <input type="checkbox"/> 二 项目符号列表
标准格式设置	<input type="checkbox"/> B 加粗 <input type="checkbox"/> I 斜体 <input type="checkbox"/> U 下划线 <input type="checkbox"/> M 等宽

要将所选文本或代码全部由大写更改为小写（或者反之），请选择文本，点击鼠标右键，然后选择**更改大小写**。也可以按 **Ctrl+Shift+A**。如果文本中同时包含大写和小写文本，则 MATLAB 会将它们全部更改为大写。

要在实时编辑器中调整显示的字体大小，请使用 **Ctrl+鼠标滚轮** 键盘快捷方式。将实时脚本导出为 PDF、Microsoft Word、HTML 或 LaTeX 时，显示字体大小的变化不会保留。

## 自动格式设置

要在实时脚本和函数中快速进行格式设置，您可以将键盘快捷方式和字符序列结合使用。当您输入序列中的最后一个字符后，即会显示格式设置。

下表列出了格式设置样式及其可用的键盘快捷方式和自动格式设置序列。

格式设置样式	自动格式设置序列	键盘快捷方式
标题	# text + Enter	<b>Ctrl + Alt + L</b>

格式设置样式	自动格式设置序列	键盘快捷方式
题头 1	<b>## text + Enter</b>	<b>Ctrl + Shift + 1</b>
题头 2	<b>### text + Enter</b>	<b>Ctrl + Shift + 2</b>
题头 3	<b>#### text + Enter</b>	<b>Ctrl + Shift + 3</b>
分节符和题头 1	<b>%% text + Enter</b>	将光标置于带有 text 的行的开头: <b>Ctrl + Shift + 1, 然后按 Ctrl + Alt + Enter</b>
分节符	<b>%% + Enter</b> <b>-- + Enter</b> <b>*** + Enter</b>	<b>Ctrl + Alt + Enter</b>
项目符号列表	<b>* text</b> <b>- text</b> <b>+ text</b>	<b>Ctrl + Alt + U</b>
编号列表	<b>number. text</b>	<b>Ctrl + Alt + O</b>
斜体	<b>*text*</b> <b>_text_</b>	<b>Ctrl + I</b>
加粗	<b>**text**</b> <b>_text_</b>	<b>Ctrl + B</b>
加粗和斜体	<b>***text***</b> <b>_text_</b>	<b>Ctrl + B, 然后按 Ctrl + I</b>
等宽	<b>`text`</b> <b> text </b>	<b>Ctrl + M</b>
下划线	无	<b>Ctrl + U</b>
LaTeX 方程	<b>\$LaTeX\$</b>	<b>Ctrl + Shift + L</b>
超链接	<b>URL + 空格或 Enter</b> <b>&lt;URL&gt;</b> <b>[Label](URL)</b>	<b>Ctrl + K</b>
商标、服务标记和版权符号 (™、℠、® 和 ©)	<b>(TM)</b> <b>(SM)</b> <b>(R)</b> <b>(C)</b>	无

**注意** 标题、题头、分节符和列表序列必须在行开头输入。

有时，您可能希望自动格式设置序列（例如 \*\*\*）按字面显示。要显示序列中的字符，请按 **Backspace** 键或点击撤消  以退出自动格式设置。例如，如果您键入 ## text + **Enter**，则会显示题头 1 样式并带有单词 text 的题头。要撤消格式设置样式并只显示 ## text，请按 **Backspace** 键。您只有在完成序列后立即操作，才能从序列中退出。当您输入其他字符或移动光标之后，便无法再退出。

要恢复 LaTeX 方程和超链接的自动格式设置，请随时使用 **Backspace** 键。

要强制格式设置在退出序列后重新出现，请点击重做  按钮。您只能在退出操作后立即重做该操作。一旦您输入其他字符或移动光标，便无法执行重做。在这种情况下，要强制格式设置重新出现，请删除序列中的最后一个字符，然后再次键入该字符。

要禁用所有或部分自动格式设置序列，您可以调整“编辑器/调试器自动格式设置预设项”。

### 另请参阅

#### 详细信息

- “将方程插入实时编辑器中”（第 19-33 页）
- “共享实时脚本和函数”（第 19-57 页）

## 将方程插入实时编辑器中

要描述代码中使用的数学过程或方法，请将方程插入实时脚本或函数中。只有文本行才能包含方程。如果您向代码行中插入方程，则 MATLAB 会将该方程置于选定代码行正下方的新文本行中。

**Solar Elevation**

The sun's declination ( $\delta$ ) is the angle of the sun relative to the earth's equitorial plane. The solar declination is  $0^\circ$  at the vernal and autumnal equinox and rises to a maximum of  $23.45^\circ$  at the summer solstice. On any given day of the year ( $d$ ), declination can be calculated from the following formula

$$\delta = \sin^{-1} \left( \sin(23.45) \sin \left( \frac{360}{365} (d - 81) \right) \right)$$

From the declination ( $\delta$ ) and the latitude ( $\phi$ ) we can calculate the sun's elevation ( $\alpha$ ) at the current time.

$$\alpha = \sin^{-1} \left( \sin \delta \sin \phi + \cos \delta \cos \phi \cos \omega \right)$$

Here  $\omega$  is the hour angle which is the degrees of rotation of the earth between the current solar time and solar noon.

```
delta = asind(sind(23.45)*sind(360*(d - 81)/365)); % Declination
omega = 15*(solarTime.Hour + solarTime.Minute/60 - 12); % Hour angle
alpha = asind(sind(delta)*sind(phi) + ...
    cosd(delta)*cosd(phi)*cosd(omega));
fprintf('Solar Declination = %.2f\nSolar Elevation = %.2f\n', delta, alpha)
```

将方程插入实时脚本或函数中有两种方法。

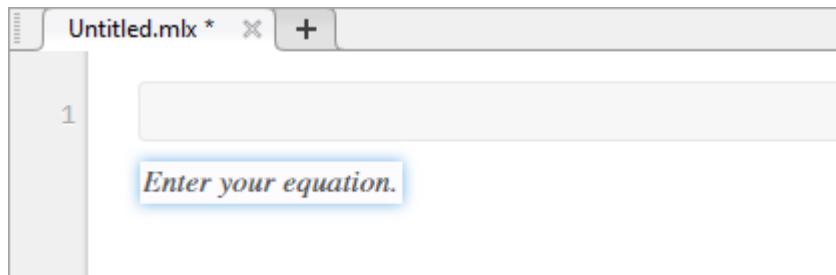
- 以交互方式插入方程 - 您可以通过从符号和结构体的图形显示中进行选择，从而以交互方式构建方程。
- 插入 LaTeX 方程 - 您可以输入 LaTeX 命令，实时编辑器就会插入对应的方程。

### 以交互方式插入方程

要以交互方式插入方程，请执行以下操作：

- 转到插入选项卡，然后点击 **Σ 方程**。

此时将会出现一个空白方程。



- 从方程选项卡显示的选项中选择符号、结构体和矩阵以构建方程。点击各部分右侧的 **⋮** 以查看其他选项。

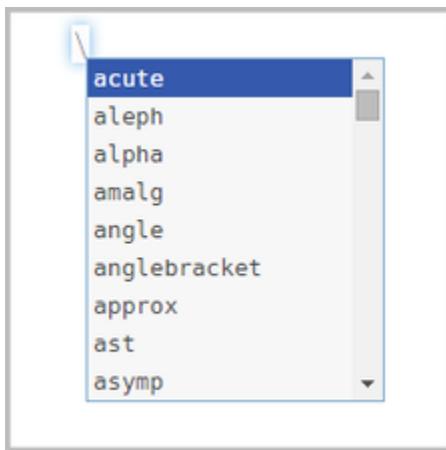
添加或编辑矩阵时，将会显示一个上下文菜单，您可以使用该菜单来删除和插入行与列。您也可以使用上下文菜单来更改或删除矩阵分隔符。

- 使用文本部分中提供的选项来设置方程格式。格式设置仅适用于方程中的文本。无法设置数值和符号的格式。除非将光标放在可设置格式的文本中，否则格式设置选项将处于禁用状态。

## 用于方程编辑的键盘快捷方式

方程编辑器提供了一些用于将元素添加到方程中的快捷方式：

- 要插入符号、结构体和矩阵，请键入一个反斜杠，后跟符号的名称。例如，键入 \pi 以在方程中插入  $\pi$  符号。要发现符号或结构体的名称，请悬停在**方程**选项卡中的对应按钮的上方。您也可以在方程编辑器中键入反斜杠，以显示所有支持名称的自动填充菜单。



**注意** 尽管 \name 语法与 LaTeX 命令语法非常相似，但在以交互方式插入方程时不支持输入完整的 LaTeX 表达式。

- 要插入下标、上标和分数，请使用符号 ‘\_’、‘^’ 或 ‘/’。例如：
  - 键入 x\_2 可将  $x_2$  插入到方程中。
  - 键入 x^2 可将  $x^2$  插入到方程中。
  - 键入 x/2 可将  $\frac{x}{2}$  插入到方程中。
- 要将新列插入矩阵，请在矩阵行中的最后一个元胞的末尾键入 ‘;’。要插入新行，请在矩阵列中的最后一个元胞的末尾键入分号 ‘;’。
- 要插入下表中列出的常用符号，请键入其他符号的组合。

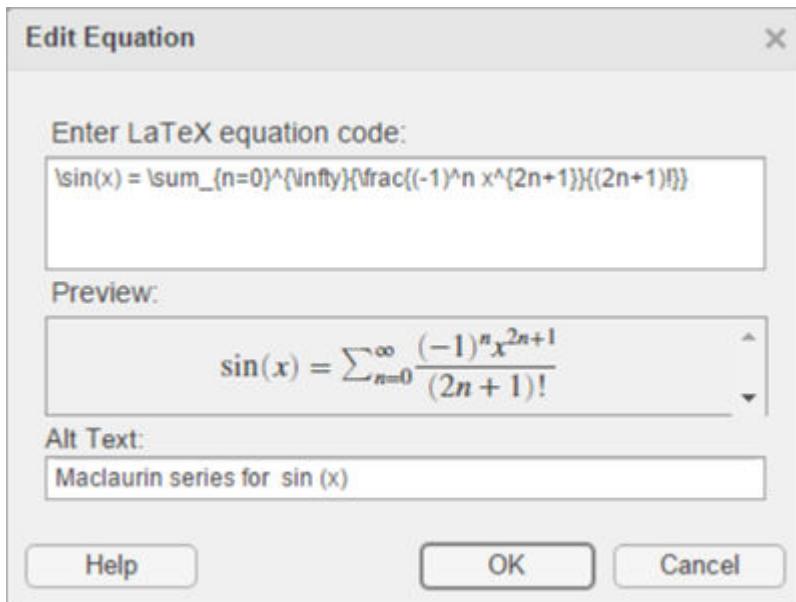
键盘输入	符号	键盘输入	符号	键盘输入	符号
		=>	$\Rightarrow$	!=	$\neq$
=	$\models$	<-->	$\leftrightarrow$	!<	$\prec$
-	$\vdash$	<->	$\Leftrightarrow$	!>	$\succ$
-	$\dashv$	<=	$\leq$	!<=	$\nleq$
->	$\rightarrow$	>=	$\geq$	!>=	$\ngeq$
<-	$\leftarrow$	<>	$\neq$		
<--	$\longleftarrow$	~ =	$\not\equiv$		

## 插入 LaTeX 方程

要插入 LaTeX 方程，请执行以下操作：

- 1 转至插入选项卡，点击方程 并选择 LaTeX 方程。
- 2 在显示的对话框中输入 LaTeX 表达式。例如，您可以输入  $\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$ 。

预览窗格显示实时脚本中显示的方程预览。



- 3 要在将实时脚本导出到 HTML 时包含 LaTeX 方程的说明，请将文本添加到替换文本字段中。例如，您可以输入文本 Maclaurin series for  $\sin(x)$ 。

该说明为方程指定替换文本，并作为 alt 属性保存在 HTML 文档中。例如，在用户使用屏幕阅读器时，它可以提供有关方程的附加信息。

- 4 按确定将方程插入实时脚本中。

LaTeX 表达式描述了各种方程。下表显示了多个 LaTeX 表达式示例及其在插入实时脚本中以后的外观。

LaTeX 表达式	实时脚本中的方程
$a^2 + b^2 = c^2$	$a^2 + b^2 = c^2$
$\int_0^2 x^2 \sin(x) dx$	$\int_0^2 x^2 \sin(x) dx$
$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$	$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$
$\{a,b,c\} \neq \{a,b,c\}$	$a, b, c \neq \{a, b, c\}$
$x^2 \geq 0 \quad \text{for all } x \in \mathbf{R}$	$x^2 \geq 0 \quad \text{for all } x \in \mathbf{R}$
$\begin{matrix} a & b \\ c & d \end{matrix}$	$a \ b$ $c \ d$

### 支持的 LaTeX 命令

MATLAB 支持大多数标准 LaTeX 数学模式命令。这些表显示了支持的 LaTeX 命令列表。

## 希腊/希伯来字母

符号	LaTeX 命令	符号	LaTeX 命令	符号	LaTeX 命令
$\alpha$	alpha	$\nu$	nu	$\xi$	xi
$\beta$	beta	$\omega$	omega	$\zeta$	zeta
$\chi$	chi	$\circ$	omicron	$\varepsilon$	varepsilon
$\delta$	delta	$\phi$	phi	$\varphi$	varphi
$\epsilon$	epsilon	$\pi$	pi	$\varpi$	varpi
$\eta$	eta	$\psi$	psi	$\varrho$	varrho
$\nu$	gamma	$\rho$	rho	$\varsigma$	varsigma
$\iota$	iota	$\sigma$	sigma	$\vartheta$	vartheta
$\kappa$	kappa	$\tau$	tau	$\aleph$	aleph
$\lambda$	lambda	$\theta$	theta		
$\mu$	mu	$\upsilon$	upsilon		
$\Delta$	Delta	$\Phi$	Phi	$\Theta$	The
$\Gamma$	Gamma	$\Pi$	Pi	$\Upsilon$	Ups
$\Lambda$	Lambda	$\Psi$	Psi	$\Xi$	Xi
$\Omega$	Omega	$\Sigma$	Sigma		

## 运算符符号

符号	LaTeX 命令	符号	LaTeX 命令	符号	LaTeX 命令
*	ast	$\pm$	pm	$\cap$	cap
$\star$	star	$\mp$	mp	$\cup$	cup
.	cdot	$\amalg$	amalg	$\uplus$	uplus
$\circ$	circ	$\odot$	odot	$\sqcap$	sqcap
$\bullet$	bullet	$\ominus$	ominus	$\sqcup$	sqcup
$\diamond$	diamond	$\oplus$	oplus	$\wedge$	wedge
$\setminus$	setminus	$\oslash$	oslash	$\vee$	vee,
$\times$	times	$\otimes$	otimes	$\triangleleft$	tria
$\div$	div	$\dagger$	dagger	$\triangleright$	tria
$\perp$	bot, perp	$\ddagger$	ddagger	$\triangle$	bigtriangleup
$\top$	top	$\wr$	wr	$\bigtriangledown$	bigtriangledown
$\sum$	sum	$\prod$	prod	$\int$	int,
$\uplus$	biguplus	$\oplus$	bigoplus	$\vee$	bigvee
$\cap$	bigcap	$\otimes$	bigotimes	$\wedge$	bigwedge
$\cup$	bigcup	$\odot$	bigodot	$\sqcup$	bigsqcup

## 关系符号

符号	LaTeX 命令	符号	LaTeX 命令	符号	LaTeX 命令
$\equiv$	<code>equiv</code>	$<$	<code>lt</code>	$>$	<code>gt</code>
$\cong$	<code>cong</code>	$\leq$	<code>le, leq</code>	$\geq$	<code>ge, geq</code>
$\neq$	<code>neq, ne, not=</code>	$\not{<}$	<code>not&lt;</code>	$\not{>}$	<code>not&gt;</code>
$\sim$	<code>sim</code>	$\curlyeqprec$	<code>prec</code>	$\succ$	<code>succ</code>
$\simeq$	<code>simeq</code>	$\curlyeqsucc$	<code>preceq</code>	$\curlyeqsucc$	<code>succ</code>
$\approx$	<code>approx</code>	$\ll$	<code>ll</code>	$\gg$	<code>gg</code>
$\asymp$	<code>asymptotic</code>	$\subset$	<code>subset</code>	$\supset$	<code>supset</code>
$\doteq$	<code>doteq</code>	$\subseteq$	<code>subsequeq</code>	$\supseteq$	<code>supseteq</code>
$\propto$	<code>proto</code>	$\sqsubseteq$	<code>sqsubsequeq</code>	$\sqsupseteq$	<code>sqsupseteq</code>
$\models$	<code>models</code>	$ $	<code>mid</code>	$\epsilon$	<code>in</code>
$\bowtie$	<code>bowtie</code>	$\parallel$	<code>parallel</code>	$\notin$	<code>notin</code>
$\vdash$	<code>vdash</code>	$\Leftrightarrow$	<code>iff</code>	$\ni$	<code>ni</code>
$\dashv$	<code>dashv</code>				

注意 `leq`、`geq`、`equiv`、`approx`、`cong`、`sim`、`simeq`、`models`、`ni`、`succ`、`succ`、`prec`、`preceq`、`parallel`、`subset`、`supset`、`subsequeq` 和 `supseteq` 命令可以与 `not` 命令组合使用，以创建符号的否定版本。例如，`\not\leq` 可创建符号  $\not{<}$ 。

## 箭头

符号	LaTeX 命令	符号	LaTeX 命令	符号	LaTeX 命令
$\leftarrow$	<code>leftarrow</code>	$\rightarrow$	<code>rightarrow</code>	$\uparrow$	<code>uparrow</code>
$\Leftarrow$	<code>Leftarrow</code>	$\Rightarrow$	<code>Rightarrow</code>	$\uparrow\uparrow$	<code>Upuparrow</code>
$\longleftarrow$	<code>longleftarrow</code>	$\longrightarrow$	<code>longrightarrow</code>	$\downarrow$	<code>downdownarrows</code>
$\Longleftarrow$	<code>Longleftarrow</code>	$\Longrightarrow$	<code>Longrightarrow</code>	$\Downarrow$	<code>Downdownarrows</code>
$\hookleftarrow$	<code>hookleftarrow</code>	$\hookrightarrow$	<code>hookrightarrow</code>	$\updownarrow$	<code>updownarrows</code>
$\leftharpoonup$	<code>leftharpoonup</code>	$\rightharpoonup$	<code>rightharpoonup</code>	$\Updownarrow$	<code>Updownarrows</code>
$\leftharpoonondown$	<code>leftharpoonondown</code>				
$\leftharpoononup$	<code>leftharpoononup</code>				
$\swarrow$	<code>swarrow</code>	$\nearrow$	<code>nearrow</code>	$\Leftrightarrow$	<code>Leftarrow</code>
$\nwarrow$	<code>nwarrow</code>	$\searrow$	<code>searrow</code>	$\leftrightarrow$	<code>longleftrightarrow</code>
$\mapsto$	<code>mapsto</code>	$\longmapsto$	<code>longmapsto</code>	$\Leftrightarrow$	<code>Longleftrightarrow</code>

## 括号

符号	LaTeX 命令	符号	LaTeX 命令	符号	LaTeX 命令
{	lbrace	}	rbrace		vert
[	lbrack	]	rbrack		Ver
<	langle	>	rangle	\	bac
[	lceil	]	rceil		
[	lfloor	]	rfloor		

样本	LaTeX 命令	样本	LaTeX 命令	
{	big, bigl, bigr, bigm	{abc}	brace	
{	Big, Bigl, Bigr, Bigm	[abc]	brack	
{	bigg, biggl, biggr, biggm	(abc)	choose	
{	Bigg, Biggl, Biggr, Biggm			

## 杂项符号

符号	LaTeX 命令	符号	LaTeX 命令	符号	LaTeX 命令
$\infty$	infty	$\forall$	forall	$\wp$	wp
$\nabla$	nabla	$\exists$	exists	$\angle$	ang
$\partial$	partial	$\emptyset$	emptyset	$\triangle$	tria
$\Im$	Im	$\imath$	i	$\hbar$	hbar
$\Re$	Re	$\jmath$	j	$'$	prime
$\ell$	ell	$\imath$	imath	$\neg$	lnot
$\dots$	dots, ldots	$\jmath$	jmath	$\sqrt{}$	sqrt
$\cdots$	cdots	:	colon	$\leftarrow$	gets
$\ddots$	ddots	.	cdotp	$\rightarrow$	to
$\vdots$	vdots	.	ldotp		

注意 `exists` 命令可以与 `not` 命令组合使用，以创建符号的否定版本。例如，`\not\exists` 可创建符号  $\nexists$ 。

## 变音

符号	LaTeX 命令	符号	LaTeX 命令	符号	LaTeX 命令
$\acute{a}$	acute	$\ddot{a}$	ddot	$\tilde{a}$	tild

符号	LaTeX 命令	符号	LaTeX 命令	符号	LaTeX 命令
$\bar{a}$	<code>bar</code>	$\dot{a}$	<code>dot</code>	$\vec{a}$	<code>vec</code>
$\breve{a}$	<code>breve</code>	$\grave{a}$	<code>grave</code>		
$\check{a}$	<code>check</code>	$\widehat{a}$	<code>hat</code>		

## 函数

样本	LaTeX 命令	样本	LaTeX 命令	样本	LaTeX 命令
$\arccos$	<code>arccos</code>	$\det$	<code>det</code>	$\ln$	<code>ln</code>
$\arcsin$	<code>arcsin</code>	$\dim$	<code>dim</code>	$\log$	<code>log</code>
$\arctan$	<code>arctan</code>	$\exp$	<code>exp</code>	$\max$	<code>max</code>
$\arg$	<code>arg</code>	$\gcd$	<code>gcd</code>	$\min$	<code>min</code>
$\cos$	<code>cos</code>	$\hom$	<code>hom</code>	$\Pr$	<code>Pr</code>
$\cosh$	<code>cosh</code>	$\ker$	<code>ker</code>	$\sec$	<code>sec</code>
$\cot$	<code>cot</code>	$\lg$	<code>lg</code>	$\sin$	<code>sin</code>
$\coth$	<code>coth</code>	$\lim$	<code>lim</code>	$\sinh$	<code>sinh</code>
$\csc$	<code>csc</code>	$\liminf$	<code>liminf</code>	$\sup$	<code>sup</code>
$\deg$	<code>deg</code>	$\limsup$	<code>limsup</code>	$\tan$	<code>tan</code>

## 数学构造

样本	LaTeX 命令	样本	LaTeX 命令	样本	LaTeX 命令
$\frac{abc}{xyz}$	<code>frac</code>	$\frac{a}{b}$	<code>over</code>	$\frac{a}{b}$	<code>stackrel</code>
$\sqrt{abc}$	<code>sqrt</code>	$\left[ \frac{a}{b} \right]$	<code>overwithdelims</code>	$\left. b \atop a \right.$	<code>underbrace</code>
$\mod a$	<code>bmod</code>	$\overleftarrow{abc}$	<code>overleftarrow</code>	$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$	<code>pmatrix</code>
$(\mod a)$	<code>pmod</code>	$\overrightarrow{abc}$	<code>overrightarrow</code>	$\begin{matrix} a & b \\ c & d \end{matrix}$	<code>matrix</code>
$\widehat{abc}$	<code>widehat</code>	$\overleftrightarrow{abc}$	<code>overleftrightarrow</code>	$\begin{matrix} a & b \\ c & d \end{matrix}$	<code>begin</code>
$\widetilde{abc}$	<code>widetilde</code>	$\int_a^b$	<code>limits</code>	$\begin{cases} a & b \\ c & d \end{cases}$	<code>begin</code>
	<code>left</code>		<code>right</code>	$\overline{ab}$	<code>hline</code>
				$\underline{cd}$	

**注意** 要通过 `matrix` 和 `pmatrix` 命令创建矩阵，请使用 `&` 符号来分隔列，使用 `\cr` 来分隔行。例如，要创建一个  $2 \times 2$  矩阵，请使用表达式 `\matrix{a & b \cr c & d}`。

**空白**

样本	LaTeX 命令	样本	LaTeX 命令	样本	LaTeX 命令
$ab$	<code>negthinspace</code>	$abc$	<code>mathord</code>	$a[b$	<code>mat</code>
$ab$	<code>thinspace</code>	$a\sum b$	<code>mathop</code>	$a]b$	<code>mat</code>
$a\ b$	<code>enspace</code>	$a + b$	<code>mathbin</code>	$a \mid b$	<code>mat</code>
$a\ b$	<code>quad</code>	$a = b$	<code>mathrel</code>	$a \ b$	<code>kern</code>
$a\ b$	<code>qquad</code>	$a, b$	<code>mathpunct</code>		

**文本样式**

样本	LaTeX 命令	样本	LaTeX 命令	样本	LaTeX 命令
$\Sigma$	<code>displaystyle</code>	ABCDE	<code>text, textnormal</code>	$ABCDE$	<code>text</code>
$\Sigma$	<code>textstyle</code>	ABCDE	<code>bf, textbf, mathbf</code>	$ABCDE$	<code>text</code>
$\Sigma$	<code>scriptstyle</code>	$ABCDE$	<code>it, textit, mathit</code>	$A\mathcal{B}CD\mathcal{E}$	<code>cal,</code>
$\Sigma$	<code>scriptscriptstyle</code>	ABCDE	<code>rm, textrm, mathrm</code>	$ABCDE$	<code>hbo</code>

**另请参阅****相关示例**

- “共享实时脚本和函数”（第 19-57 页）

**外部网站**

- <https://www.latex-project.org/>

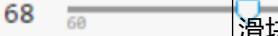
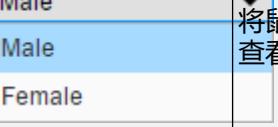
# 将交互式控件添加到实时脚本

您可以将滑块、下拉列表、复选框、编辑字段和按钮添加到实时脚本，以交互方式控制变量值。当您要与其他人共享脚本时，将交互式控件添加到脚本中非常有用。通过交互式控件，可使用熟悉的用户界面组件设置和更改实时脚本中变量的值。

## 插入控件

要将控件插入实时脚本中，请转至**实时编辑器**选项卡，在**代码**部分中，点击**控件**。然后，从可用选项中进行选择。要用一个控件替换现有值，请选择该值，然后插入该控件。**控件**菜单仅显示对所选值可用的选项。

要配置现有控件，请右键点击该控件，然后选择**配置控件**。按 **Tab** 或 **Enter** 键，或在控件配置菜单外部点击以返回实时脚本。

控件	说明	配置详细信息
<b>数值滑块</b>	使用数值滑块，可以通过将滑块移至所需的数值，以交互方式更改变量的值。  滑块左侧的值是其当前值。	在 <b>值</b> 部分中，指定 <b>最小值</b> 、 <b>最大值</b> 和 <b>步长</b> 值。
<b>下拉列表</b>	使用下拉列表，可以通过从值列表中进行选择，以交互方式更改变量的值。  将鼠标悬停在下拉列表中显示的文本上可查看其当前值。	在 <b>项目</b> 部分的 <b>项目标签</b> 字段中，指定要为下拉列表中的每个项显示的文本。
<b>复选框</b>	使用复选框以交互方式将变量值设置为逻辑值 1 (true) 或逻辑值 0 (false)。  复选框的显示状态（选中或未选中）决定其当前值。	不适用
<b>编辑字段</b>	使用编辑字段以交互方式将变量值设置为指定类型的输入。  编辑字段中显示的文本和选定的数据类型决定其当前值。	在 <b>类型</b> 部分的 <b>数据类型</b> 字段中，从可用选项中进行选择，以指定编辑字段中文本的数据类型。
<b>按钮</b>	使用按钮控件，通过点击按钮以交互方式运行代码。  使用按钮控件时，可以考虑将实时脚本中所有其他控件的 <b>运行</b> 字段设置为 <b>无</b> 。这样，代码仅在用户点击按钮控件时运行。当实时脚本要求在运行代码之前设置多个控件值时，这会很有用。	要更改按钮上显示的标签，请在 <b>标签</b> 部分中输入标签名称。

## 标签

要指定在隐藏代码时要显示在控件旁边的标签，请在**标签**部分中输入标签名称。这也是所有视图中按钮控件上显示的文本。

要隐藏代码并仅显示带标签的控件、输出和格式化文本，请点击实时脚本右侧的隐藏代码  按钮。您也可以转至视图选项卡，在视图部分中，点击  隐藏代码。要再次显示代码，请点击输出内嵌  按钮或右侧的  按钮上的输出。

## 执行

默认情况下，当控件的值发生变化时，实时编辑器会运行当前节中的代码。要配置此行为，请在控件配置菜单的**执行**部分指定以下字段：

- 运行位置**（仅滑块控件）- 选择**正在更改的值**以在滑块值更改时运行代码。选择**已经更改的值**，等待滑块值更改完成（用户释放了滑块）。
- 运行** - 从可用选项中进行选择，以确定控件值发生变化时运行的代码。例如，如果选择**当前节**，则当控件值发生更改时将仅运行包含该控件的代码节。如果选择**从当前节到结束**，则将运行包含控件的节以及随后的所有节。

**提示** 在实时脚本中使用按钮控件时，可以考虑将实时脚本中所有其他控件的**运行**字段设置为**无**。这样，代码仅在用户点击按钮控件时运行。当实时脚本要求在运行代码之前设置多个控件值时，这会很有用。

## 使用多个交互式控件创建实时脚本

以下示例说明如何使用交互式控件在 MATLAB® 中可视化患者数据并对这些数据进行研究。此示例绘制男性或女性患者的身高与体重对照图，并突出显示具有指定身高和体重的患者。

使用交互式控件指定要绘制的患者的性别以及阈值身高和体重。要查看控件并与之交互，请在浏览器或 MATLAB 中打开此示例。

```
load patients

thresholdHeight = ; % Slider with min=60, max=70, step=1
thresholdWeight = ; % Slider with min=111, max=202, step=1
selectedGender = ; % Drop down with options "Male", "Female"

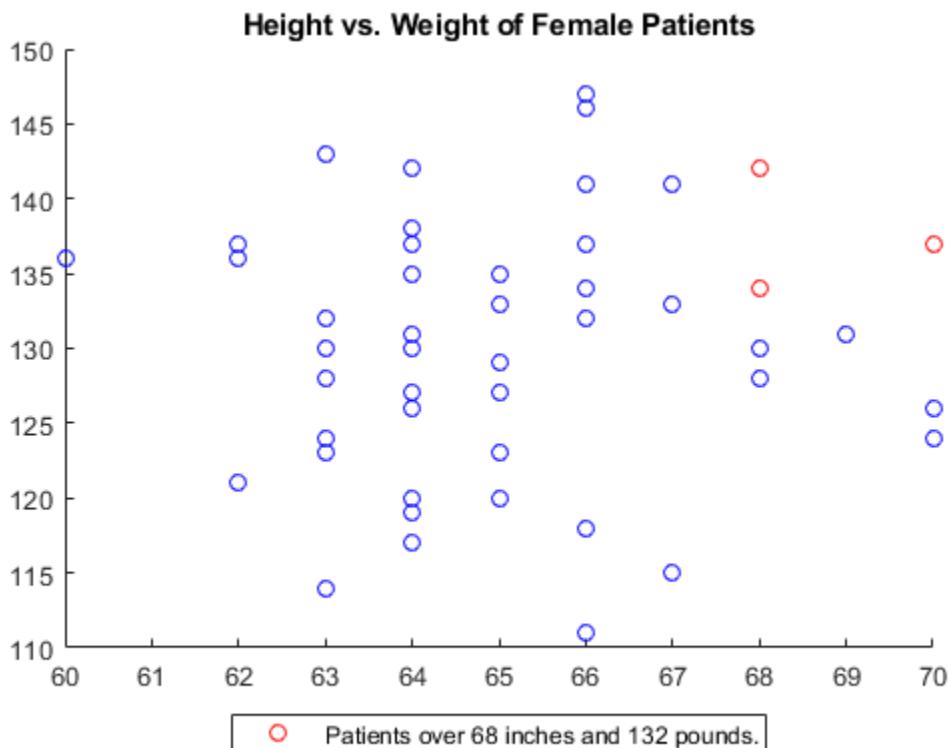
overThresholdWeights = Weight(Gender==selectedGender & Weight>=thresholdWeight & Height>=thresholdHeight);
overThresholdHeights = Height(Gender==selectedGender & Weight>=thresholdWeight & Height>=thresholdHeight);

sp1 = scatter(Height(Gender==selectedGender),Weight(Gender==selectedGender),'blue');
hold on

sp2 = scatter(overThresholdHeights, overThresholdWeights,'red');
hold off

title('Height vs. Weight of ' + selectedGender + ' Patients')

legendText = sprintf('Patients over %d inches and %d pounds.',thresholdHeight,thresholdWeight);
legend(sp2,legendText,'Location','southoutside')
```



## 共享实时脚本

实时脚本完成后，可与其他人共享。用户可以在 MATLAB 中打开实时脚本，并以交互方式使用控件来进行实验。

如果您将实时脚本本身作为交互式文档共享，请考虑在共享实时脚本之前隐藏其中的代码。隐藏代码后，实时编辑器仅显示带标签的控件、输出和格式化文本。要隐藏代码，请点击实时脚本右侧的隐藏代码 按钮。您也可以转至视图选项卡，在视图部分中，点击 隐藏代码。

如果您将实时脚本作为静态 PDF、Microsoft Word、HTML 或 LaTeX 文档共享，则实时编辑器会将控件保存为代码。例如，在此处显示的实时脚本中，实时编辑器用滑块的当前值（68 和 132）替换滑块控件，并用下拉列表的当前值（"Female"）替换下拉列表控件。

**Create Live Script with Multiple Interactive Controls**

This example shows how you can use interactive controls to visualize and investigate patient data in MATLAB®. The example plots the height versus the weight of either male or female patients, and highlights the patients over a specified height and weight.

Use the interactive controls to specify the gender of the patients to plot, as well as the threshold height and weight. To view and interact with the controls, open this example in your browser or in MATLAB.

```

load patients

thresholdHeight = 68;
thresholdWeight = 132;
selectedGender = "Female";

overThresholdWeights = Weight(Gender==selectedGender & Weight>=thresholdWeight & Height>=thresholdHeight);
overThresholdHeights = Height(Gender==selectedGender & Weight>=thresholdWeight & Height>=thresholdHeight);

sp1 = scatter(Height(Gender==selectedGender),Weight(Gender==selectedGender),'blue');
hold on

sp2 = scatter(overThresholdHeights, overThresholdWeights,'red');
hold off

title('Height vs. Weight of ' + selectedGender + ' Patients')

legendText = sprintf('Patients over %d inches and %d pounds.',thresholdHeight,thresholdWeight);
legend(sp2,legendText,'Location','southoutside')

```

## 另请参阅

### 详细信息

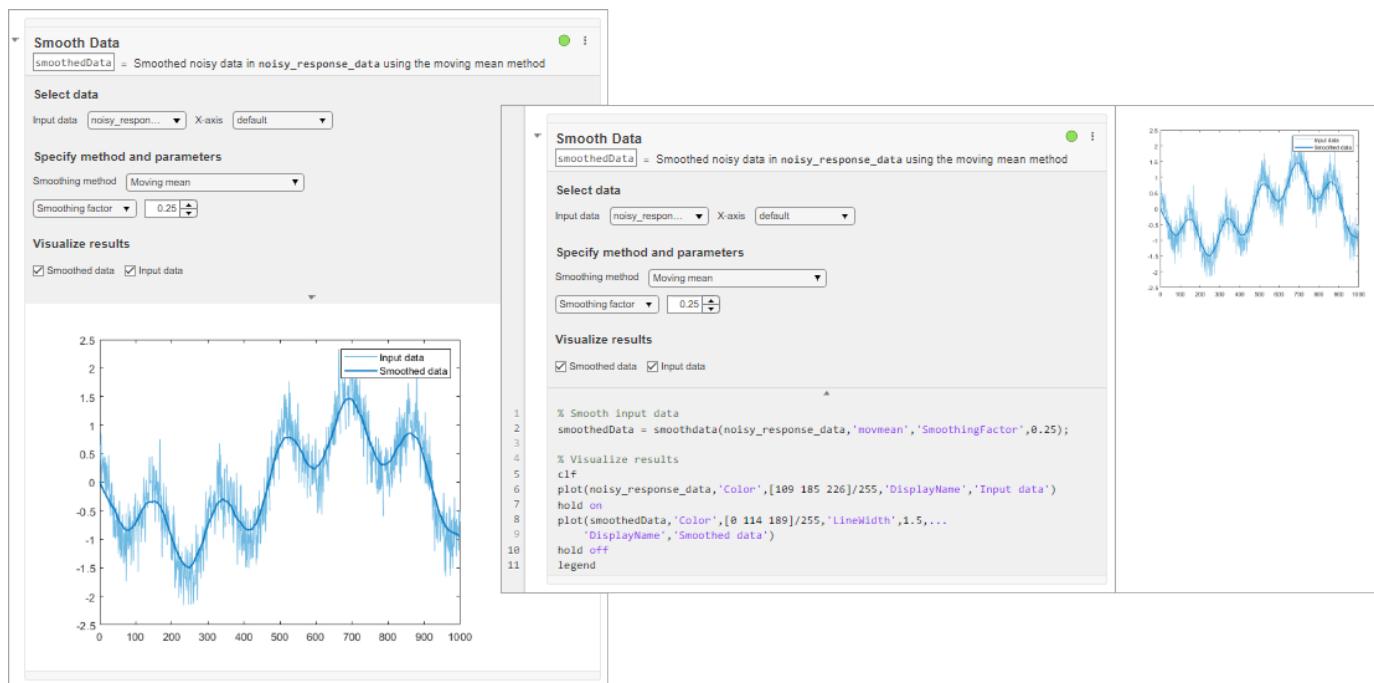
- “将交互式任务添加到实时脚本中” (第 19-45 页)
- “共享实时脚本和函数” (第 19-57 页)

# 将交互式任务添加到实时脚本中

## 什么是实时编辑器任务？

实时编辑器任务是可以添加到实时脚本中以执行一组特定操作的 App。您可以将任务添加到实时脚本中，以探查参数并自动生成代码。使用任务可缩短开发时间、减少错误并缩短在绘图上花费的时间。

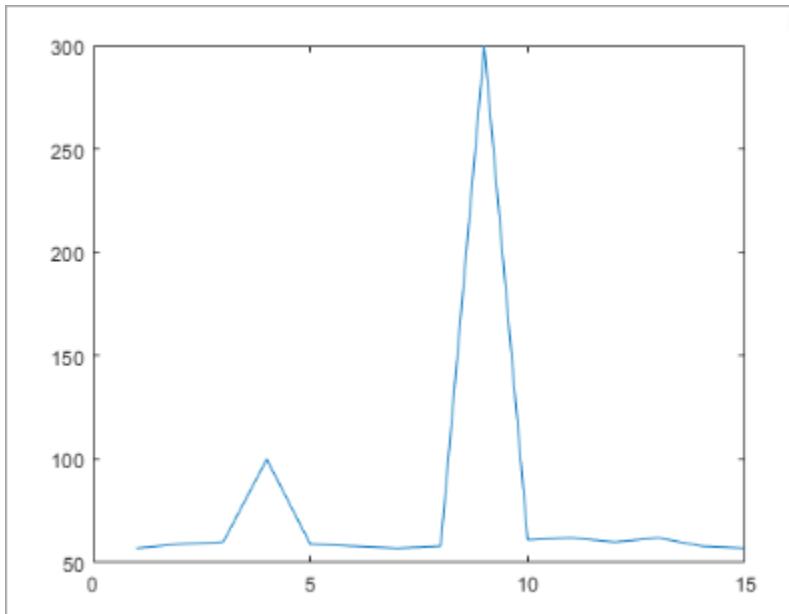
任务代表一系列 MATLAB 命令。您可以采用内嵌方式或在右侧显示其输出。要查看任务运行的 MATLAB 命令，请显示生成的代码。



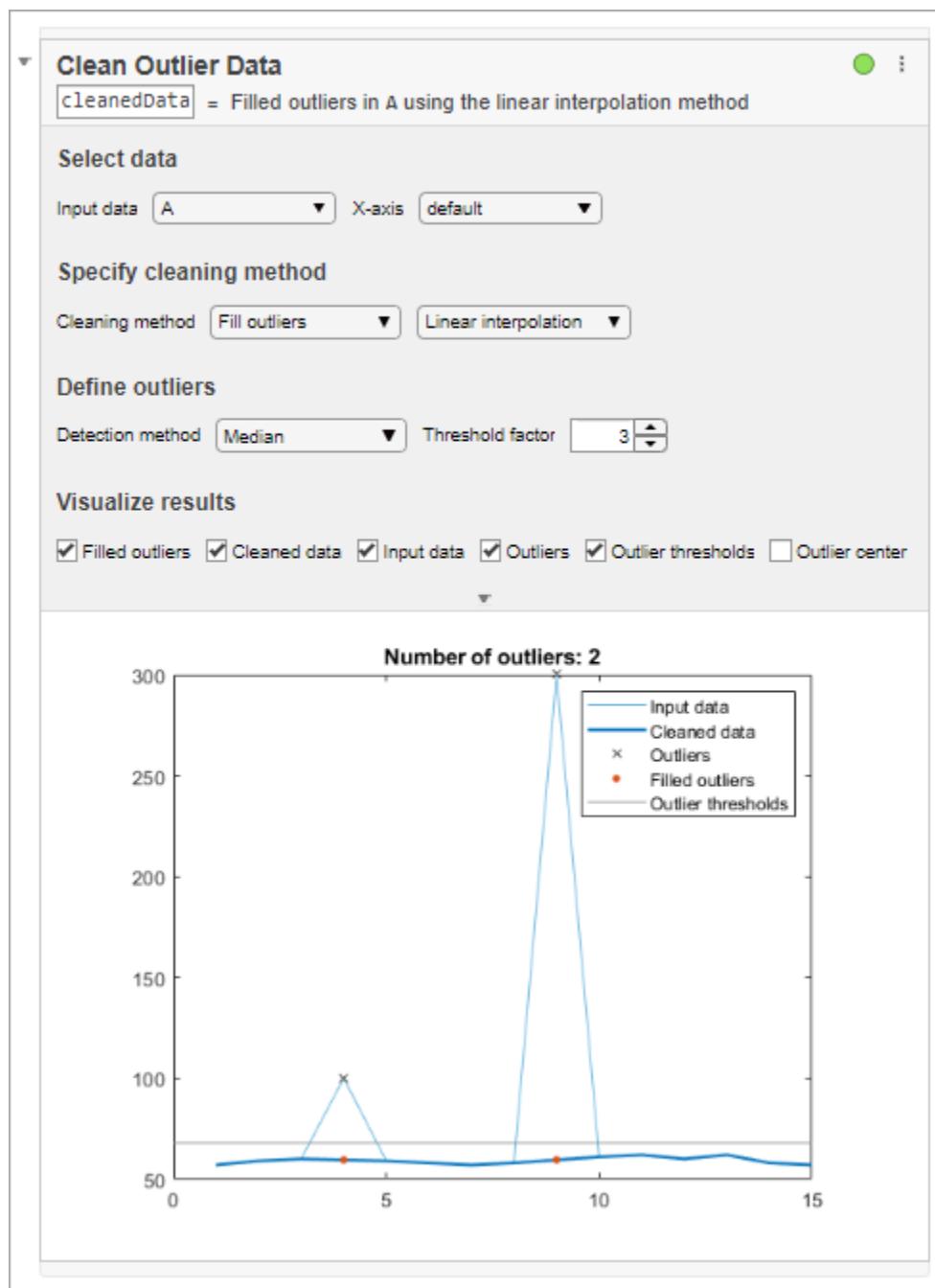
## 插入任务

要将任务添加到实时脚本中，请转至**实时编辑器**选项卡，点击 **任务** ▾，并从可用任务中进行选择。您也可以在实时脚本代码块中直接键入任务的名称。在您键入的过程中，实时编辑器会显示可能的匹配项，您可以选择并插入所需的任务。例如，创建一个实时脚本，该脚本创建一个包含离群值的数据向量。

```
A = [57 59 60 100 59 58 57 58 300 61 62 60 62 58 57];
plot(1:15,A)
```



将 **Clean Outlier Data** 任务添加到您的实时脚本中，以对含噪数据进行平滑处理，从而避免偏斜的结果。要添加任务，请在实时脚本中键入单词 `clean`，并从建议的命令自动填充项中选择 **清除离群数据**。在任务中，将 **输入数据** 设置为 `A`。该任务识别并填充数据中的两个离群值，并基于存储的结果在 MATLAB 工作区中创建变量 `cleanedData`。您还可以在任务输出的图中查看结果。继续修改其他参数，直到您对结果满意为止。

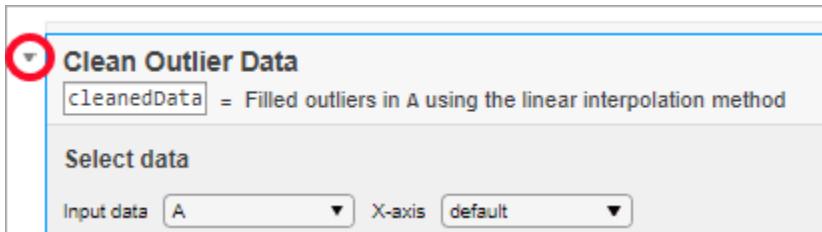


## 还原默认参数

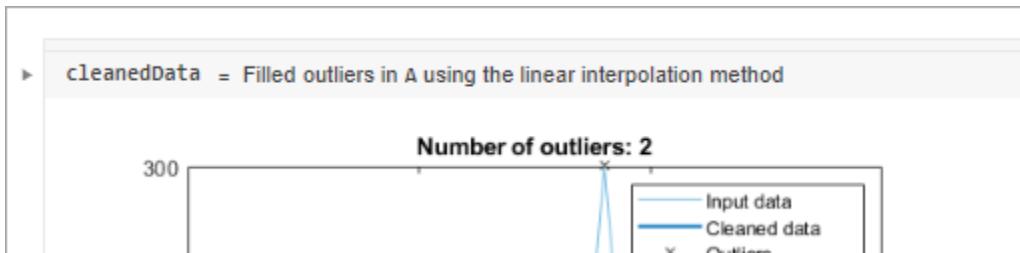
要将所有参数值还原为默认值，请点击任务右上角的选项按钮 , 然后选择还原默认值。

## 折叠任务以提高可读性

修改完参数后，您可以折叠任务以提高可读性。要折叠任务，请点击任务左上角的箭头。



该任务显示为一行用户可读的带输出伪代码。



### 删除任务

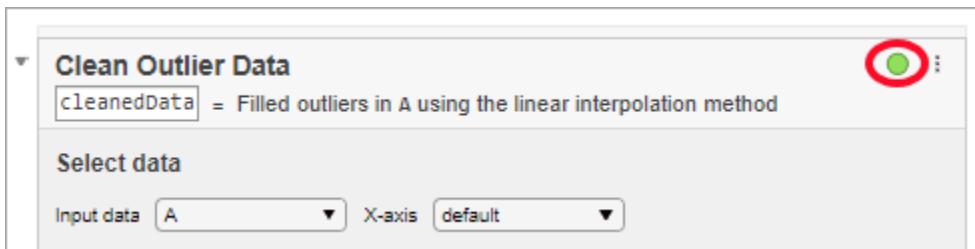
要删除任务，请点击该任务，然后按 **Delete** 或 **Backspace** 键。您也可以将光标放在紧挨任务之前或之后的位置，并分别使用 **Delete** 或 **Backspace** 键删除任务。

### 运行任务及其周围的代码

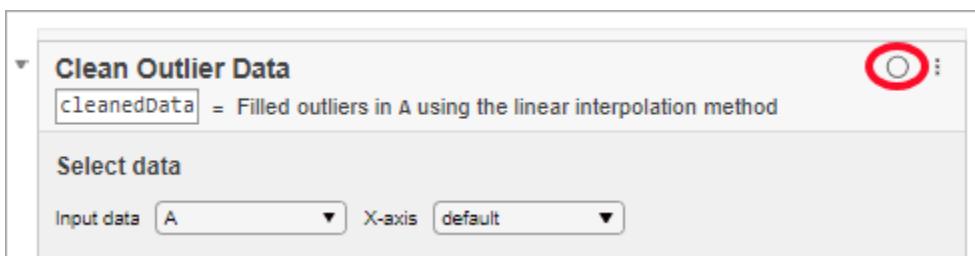
默认情况下，当您修改任务中的参数值时，任务和当前节（包括该节中的其他任务）会自动运行。这可以确保该节中的结果及其周围的代码保持最新。例如，在实时脚本 `cleanmydata mlx` 中，在您每次修改

“清除离群数据”任务中的参数值时，包括创建含噪数据向量的代码在内的整个节都会重新运行。要仅运行任务，请在任务前后添加分节符。有关节和如何添加分节符的详细信息，请参阅“在实时脚本中运行节”（第 19-11 页）。

任务右上角的绿色圆形图标表示当您修改任务参数时任务会自动运行。



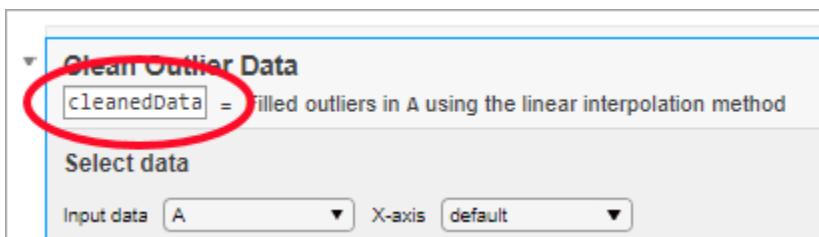
要禁用自动运行节，请点击自动运行 图标。该图标会更新以显示禁用状态。要运行任务和当前节，请在 **实时编辑器** 选项卡上，点击 运行节按钮。



有些任务默认不自动运行。此默认设置可确保这些任务实现最佳性能。

## 修改输出参数名称

要修改输出参数的名称，请点击包含参数名称的文本框并输入新名称。

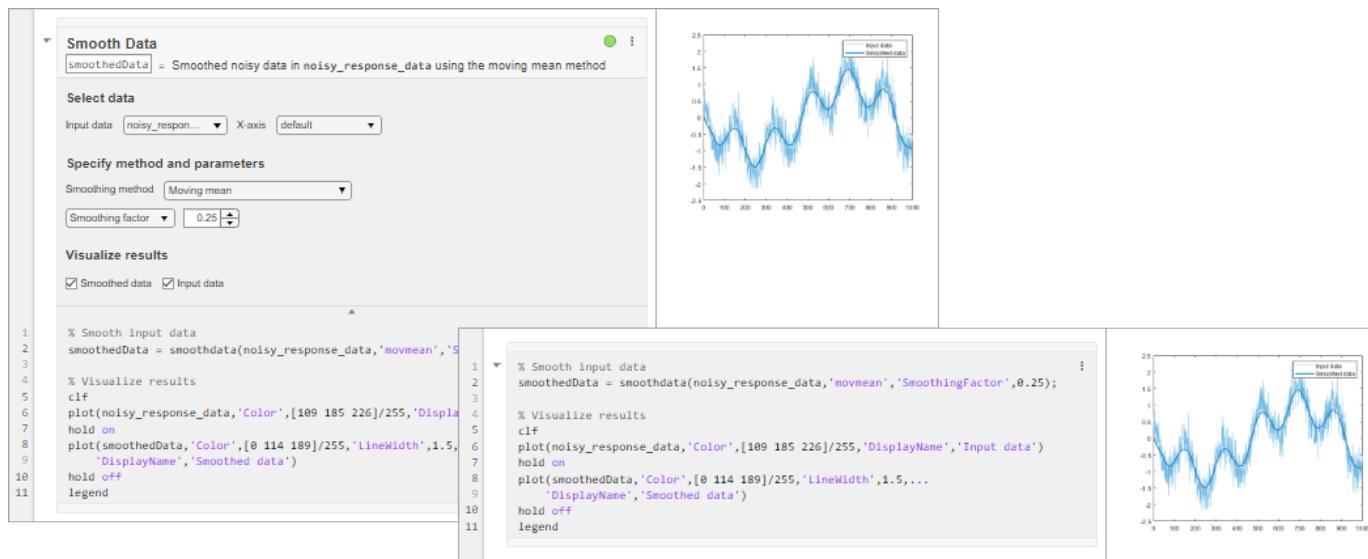


您可以在后续代码中使用结果输出参数，如将其作为其他实时编辑器任务的输入。

## 查看和编辑生成的代码

要查看任务运行的 MATLAB 命令，请点击任务右上角的选项按钮 ，然后选择**控件和代码或仅代码**。您也可以使用任务底部的向下箭头来显示和隐藏生成的代码。生成的代码为只读。

要编辑生成的代码，请点击选项按钮 ，然后选择**将任务转换为可编辑代码**。此选项会删除任务并用生成的代码替换它，然后您可以编辑这些代码。



## 另请参阅

### 详细信息

- “将交互式控件添加到实时脚本” (第 19-41 页)
- “使用实时编辑器任务清理杂乱数据并找到极值”
- MATLAB 实时脚本库

# 创建实时函数

实时函数是在一个称为实时编辑器的交互式环境中同时包含代码和格式化文本的程序文件。与实时脚本类似，实时函数允许您通过将命令序列存储在程序文件中来重用它们。不过，实时函数提供更大的灵活性，主要体现在您可以向其传递输入值并获得输出值。

## 创建实时函数

要创建实时函数，请转至[主页](#)选项卡并选择[新建 > 实时函数](#)。

### 将现有函数作为实时函数打开

如果您有现有函数，可以在实时编辑器中将其作为实时函数打开。以实时函数方式打开函数会创建一个文件副本，并保持原始文件不变。MATLAB 会将原始脚本中的发布标记转换为新实时函数中的格式化内容。

要从编辑器将现有函数 (.m) 作为实时函数 (.mlx) 打开，请右键点击文档选项卡并从上下文菜单中选择[以实时函数方式打开 functionName](#)。

您还可以转至[编辑器](#)选项卡，点击[保存](#)，然后选择[另存为](#)。然后，将[保存类型](#): 设置为“[MATLAB 实时代码文件 \(\\*.mlx\)](#)”并点击[保存](#)。

---

**注意** 您必须使用上述的转换方法之一将函数转换为实时函数。仅使用 .mlx 扩展名重命名该函数行不通，并可能损坏文件。

### 从选定代码创建实时函数

如果您有一个现有的大型实时脚本或函数，可以通过自动将选定代码区域转换为函数或局部函数，将它分成若干更小的部分。这称为代码重构。

要重构选定的代码区域，请选择一行或多行代码，并在[实时编辑器](#)选项卡上的[代码](#)部分中，点击 **重构**。然后，从可用选项中进行选择。MATLAB 会用选定的代码创建一个函数，并将原始代码替换为对新创建函数的调用。

## 添加代码

创建实时函数后，向该函数添加代码并保存。例如，添加以下代码并将其另存为名为 `mymean.mlx` 的函数。`mymean` 函数计算输入列表的平均值并返回结果。

```
function a = mymean(v,n)
    a = sum(v)/n;
end
```

## 添加帮助

要为该函数提供文字说明，请在函数定义上方添加格式化帮助文本。例如，添加标题和一些文字来描述功能。有关将帮助文本添加到函数的详细信息，请参阅“[为实时函数添加帮助](#)”（第 19-54 页）。

### Mean value for a set of values

Get the mean value for a set of values by calculating the sum of all the values and dividing by the total number of values.

```
1 function a = mymean(v,n)
2     a = sum(v)/n;
3 end
```

## 运行实时函数

您可以使用几种方法运行实时函数，包括从命令行窗口调用它们或从实时脚本调用。

要从命令行窗口运行实时函数，请在命令行窗口中输入该函数的名称。例如，使用 **mymean mlx** 来计算从 1 到 10 的 10 个连续数字的均值。

```
mymean(1:10, 10)
```

```
ans =
5.5000
```

您也可以从实时脚本中调用实时函数。例如，创建名为 **mystats mlx** 的实时脚本。添加以下代码来声明一个数组、确定该数组长度并将这两个值传递给函数 **mymean**。

```
x = 1:10;
n = length(x);
avg = mymean(x,n);
disp(['Average = ', num2str(avg)])
```

运行该实时脚本。实时编辑器显示输出。

<pre>1 x = 1:10; 2 n = length(x); 3 avg = mymean(x,n); 4 disp(['Average = ', num2str(avg)])</pre>	<pre>Average = 5.5</pre>
---	--------------------------

如果实时函数会显示文本或返回值，则实时编辑器会在调用函数的实时脚本中与实时函数调用并排显示输出。例如，在 **mymean** 中添加一行，以在返回值之前显示计算出的均值：

```
function a = mymean(v,n)
    a = sum(v)/n;
    disp(['a = ', num2str(a)])
end
```

运行 **mystats** 时，实时编辑器会将来自 **mymean** 的输出和来自 **mystats** 的输出显示在一起。

<pre>1 x = 1:10; 2 n = length(x); 3 avg = mymean(x,n); 4 disp(['Average = ', num2str(avg)])</pre>	<pre>a = 5.5 Average = 5.5</pre>
---	----------------------------------

## 另请参阅

### 详细信息

- “为实时函数添加帮助” (第 19-54 页)

## 为实时函数添加帮助

您可以为您编写的实时函数提供帮助文本。您使用 **help** 命令时帮助文本显示在命令行窗口中。您也可以使用 **doc** 命令在单独的浏览器中显示帮助文本。

要创建帮助文本，请在文件的开头、函数定义行（带有 **function** 关键字的行）的上一行插入文本。

例如，使用以下代码创建名为 **addme.mlx** 的实时函数：

```
function c = addme(a,b)

switch nargin
    case 2
        c = a + b;
    case 1
        c = a + a;
    otherwise
        c = 0;
end
```

添加帮助文本来描述该函数。

<pre>1 2 3 4 5 6 7 8 9 10</pre>	<p>Add two values together</p> <p>C = ADDME(A) adds A to itself.</p> <p>C = ADDME(A,B) adds A and B together.</p> <p>See also SUM, PLUS.</p> <pre>function c = addme(a,b)  switch nargin     case 2         c = a + b;     case 1         c = a + a;     otherwise         c = 0; end</pre>
---------------------------------	---

在命令行窗口中键入 **help addme** 时，将显示帮助文本。

<pre>&gt;&gt; help addme</pre>	<pre>addme Add two values together c = addme(a,b)  C = addme(A) adds A to itself.  C = addme(A,B) adds A and B together.  See also sum, plus.  Open documentation in Help browser</pre>
--------------------------------	---

第一行帮助文本（通常称为 H1 行）通常包含该函数的简要说明。显示函数帮助时，MATLAB 首先显示函数名称，后跟 H1 行。然后，MATLAB 显示该函数的语法。最后，MATLAB 显示其余帮助文本。

要添加“另请参阅”链接，请在帮助文本的末尾添加一个文本行，该文本行以 **See also** 字样开头，后跟函数名称列表。如果函数存在于搜索路径或当前文件夹中，**help** 命令会将其中的每个函数名称显示为指向其帮助的超链接。否则，**help** 将按帮助文本中函数名称的原样输出这些名称。

**注意** 如果多个程序的名称相同，**help** 命令通过应用“函数优先顺序”（第 20-34 页）中介绍的规则确定要显示的帮助文本。但是，如果程序的名称与某内置函数相同，上下文菜单中的**关于所选内容的帮助**选项始终都会显示有关该内置函数的文档。

要进一步增强“帮助”浏览器中显示的文档，您可以格式化文本并添加超链接、图像、方程和示例代码。例如，在 **addme** 函数中选择 H1 行，并在**实时编辑器**选项卡中将**普通**文本样式更改为**标题**。然后，将光标置于第二个语法描述的末尾，转至**插入**选项卡并选择 **方程**。输入方程  $c = a + b$  并按 **Esc** 键。最后，在**插入**选项卡中，选择**代码示例** > **MATLAB** 并添加两个示例。有关在**实时编辑器**中格式化文件的详细信息，请参阅“在**实时编辑器**中格式化文件”（第 19-29 页）。

The screenshot shows the MATLAB Help browser with the following content:

**Add two values together**

C = ADDME(A) adds A to itself.

C = ADDME(A,B) adds A and B together using the equation  $c = a + b$ .

**Examples**

Add a value to itself:

```
c = addme(10);
```

Add two values:

```
c = addme(10, 90);
```

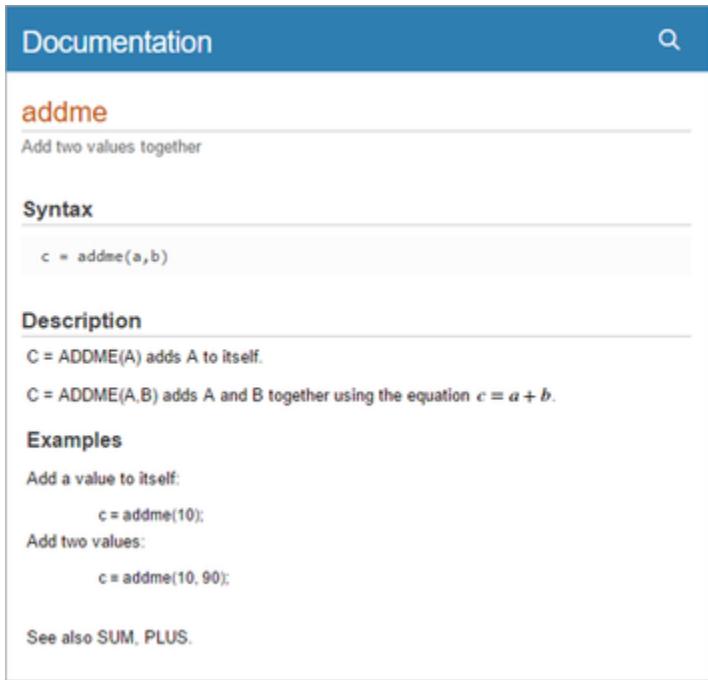
See also SUM, PLUS.

```

1 function c = addme(a,b)
2
3 switch nargin
4   case 2
5     c = a + b;
6   case 1
7     c = a + a;
8   otherwise
9     c = 0;
10 end

```

使用 **doc** 命令在单独的浏览器中显示帮助文本。



您还可以通过在文件开头插入注释来为实时函数添加帮助。当您使用 `help` 和 `doc` 命令时，文件开头的注释显示为帮助文本，类似于文件开头文本的显示方式。有关使用注释添加帮助的详细信息，请参阅“为程序添加帮助”（第 20-5 页）。

## 另请参阅

### 详细信息

- “创建实时函数”（第 19-51 页）
- “在实时编辑器中格式化文件”（第 19-29 页）
- “共享实时脚本和函数”（第 19-57 页）

## 共享实时脚本和函数

您可与其他人共享实时脚本和函数以供教学或演示之用，或提供代码的可读外部文档。您可以与其他 MATLAB 用户共享文件，也可以共享文件的静态 PDF、Microsoft Word、HTML 和 LaTeX 版本以在 MATLAB 外部查看。

下表显示了共享实时脚本和函数的不同方式。

共享方式	说明
作为交互式文档共享	<p>分发实时代码文件 (.mlx)。该文件的接收者可以在 MATLAB 中打开它，并以您上次保存它的状态查看它。这包括生成的输出。</p> <p>MATLAB 支持 R2016a 及更高版本中的实时脚本，以及 R2018a 及更高版本中的实时函数。</p>
作为全屏演示文稿	<p>打开实时脚本或函数，转至视图选项卡并点击全屏按钮。MATLAB 以全屏模式打开文件。</p> <p>要退出全屏模式，请将鼠标移至屏幕顶部以显示视图选项卡，然后再次点击全屏按钮。</p>
与 MATLAB 旧版本用户共享	<p>将实时脚本或函数另存为纯代码文件 (.m) 并分发它。该文件的接收者可以在 MATLAB 中打开和查看它。MATLAB 会将实时脚本或函数中的格式化内容转换为新脚本或函数中的发布标记。</p> <p>有关详细信息，请参阅“将实时脚本和函数另存为纯代码”（第 19-5 页）。</p>
作为能够在 MATLAB 之外查看的静态文档共享	<p>将脚本或函数导出为标准格式。可用的格式包括 PDF、Microsoft Word、HTML 和 LaTeX。</p> <p>要将实时脚本或函数导出为其中一种格式，请在实时编辑器选项卡上选择保存 &gt; 导出为 PDF、保存 &gt; 导出为 Word、保存 &gt; 导出为 HTML 或保存 &gt; 导出为 LaTeX。</p> <p>所保存文件的外观与实时脚本或函数在实时编辑器中以内嵌输出布局显示的外观非常相似。</p> <p>导出为 LaTeX 时，MATLAB 将在输出文档所在的文件夹中另行创建一个 matlab.sty 文件（如果尚不存在）。.sty 文件也称为 LaTeX 样式文档，可让您更好地控制输出文档的外观。</p> <p>要在导出前自定义图窗格式以及文档纸张大小、方向和边距，请使用设置。有关详细信息，请参阅 matlab.editor 设置。</p>

## 共享前隐藏代码

在将实时脚本作为交互式或静态文档共享之前，可以考虑隐藏其中的代码。隐藏代码后，实时编辑器仅显示带标签的控件、输出和格式化文本。

要隐藏代码，请点击实时脚本右侧的  隐藏代码按钮。您也可以转至视图选项卡，在视图部分中，点击  隐藏代码。要再次显示代码，请点击输出内嵌  按钮或右侧的  按钮上的输出。

### 另请参阅

#### 相关示例

- “在实时编辑器中创建实时脚本” (第 19-6 页)
- “在实时编辑器中格式化文件” (第 19-29 页)
- MATLAB 实时脚本库

# 实时代码文件格式 (.mlx)

MATLAB 在带有 .mlx 扩展名的文件中使用实时代码文件格式存储实时脚本和函数。实时代码文件格式使用 Open Packaging Conventions 技术，这是 zip 文件格式的扩展。代码和格式化内容使用 Office Open XML (ECMA-376) 格式存储在与输出不同的 XML 文档中。

## 实时代码文件格式的好处

- **可跨区域设置互操作** - 实时代码文件支持跨所有区域设置存储和显示字符，从而便于在国际范围内共享文件。例如，如果您使用日语区域设置创建一个实时脚本，然后使用俄语区域设置打开该实时脚本，则其中的字符会正确显示。
- **可扩展** - 实时代码文件可通过 ECMA-376 格式（支持 Microsoft Word 提供的各种格式选项）进行扩展。ECMA-376 格式还适应任意名称-值对组，以防需要将该格式扩展为除标准格式外的其他格式。
- **向前兼容** - 通过实施 ECMA-376 标准的向前兼容性策略，以后版本的实时代码文件可与先前版本的 MATLAB 兼容。
- **向后兼容** - 以后版本的 MATLAB 可支持由先前版本的 MATLAB 创建的实时代码文件。

## 源代码管理

要确定和显示不同实时脚本或函数间的代码差别，请使用 MATLAB 比较工具。

如果您使用源代码管理，请将 .mlx 扩展名注册为二进制文件。有关详细信息，请参阅“使用 SVN 注册二进制文件”（第 33-14 页）或“在 Git 中注册二进制文件”（第 33-24 页）。

## 另请参阅

### 详细信息

- “什么是实时脚本或实时函数？”（第 19-2 页）
- “在实时编辑器中创建实时脚本”（第 19-6 页）

### 外部网站

- Open Packaging Conventions Fundamentals
- Office Open XML File Formats (ECMA-376)

## 实时编辑器介绍

以下示例是对实时编辑器的介绍。在实时编辑器中，可以创建随代码一起显示代码输出的实时脚本。添加格式化文本、方程、图像和超链接用于增强您的记叙脚本，以及将实时脚本作为交互式文档与其他人共享。

在实时编辑器中创建实时脚本。要创建实时脚本，请在[主页](#)选项卡上，点击[新建实时脚本](#)。

### 添加人口统计数据

将实时脚本划分为多个节。每一节均可以包含文本、代码和输出。MATLAB 代码显示为灰色背景，输出显示为白色背景。要创建新的节，请转至[实时编辑器](#)选项卡，然后点击[分节符](#)按钮。

添加 1900 至 2000 年间美国的人口统计数据。

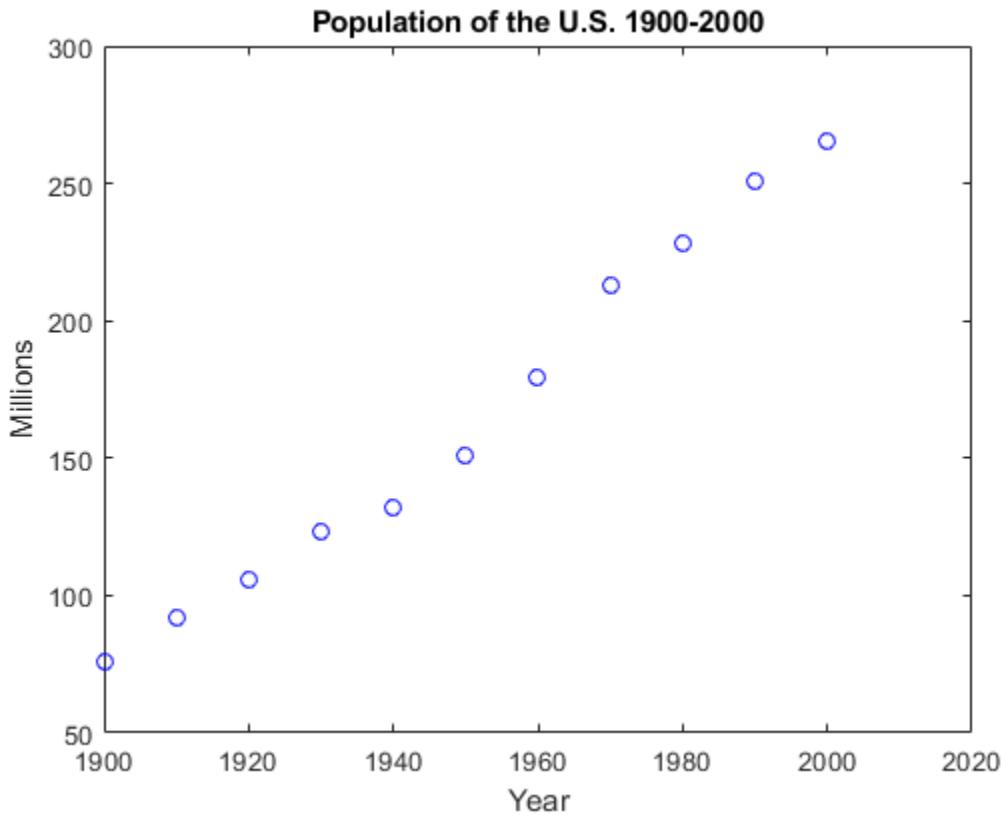
```
years = (1900:10:2000); % Time interval
pop = [75.995 91.972 105.711 123.203 131.669 ... % Population Data
    150.697 179.323 213.212 228.505 250.633 265.422]
pop = 1×11
75.9950 91.9720 105.7110 123.2030 131.6690 150.6970 179.3230 213.2120 228.5050 250.6330 265.4220
```

### 以可视方式呈现一段时间内的人口变化

各节可独立运行。要运行某节中的代码，请转至[实时编辑器](#)选项卡，然后点击[运行节](#)按钮。您也可以点击在将鼠标移至节左侧时显示的蓝条。运行节时，输出和图窗会随生成这些内容的代码一起显示。

绘制不同年份的人口数据图。

```
plot(years,pop,'bo'); % Plot the population data
axis([1900 2020 0 400]);
title('Population of the U.S. 1900-2000');
ylabel('Millions');
xlabel('Year')
ylim([50 300])
```



是否可以预测 2010 年的美国人口？

### 拟合数据

将支持信息添加到文本中，包括方程、图像和超链接。

下面我们尝试使用多项式拟合数据。我们将使用 MATLAB **polyfit** 函数获取系数。

拟合方程为：

$$\begin{array}{ll} y = ax + b & \text{linear} \\ y = ax^2 + bx + c & \text{quadratic} \\ y = ax^3 + bx^2 + cx + d. & \text{cubic} \end{array}$$

```
x = (years-1900)/50;
coef1 = polyfit(x,pop,1)
```

```
coef1 = 1×2
```

```
98.9924 66.1296
```

```
coef2 = polyfit(x,pop,2)
```

```
coef2 = 1×3
```

```
15.1014 68.7896 75.1904
```

```
coef3 = polyfit(x,pop,3)  
coef3 = 1×4  
-17.1908 66.6739 29.4569 80.1414
```

### 绘制曲线图

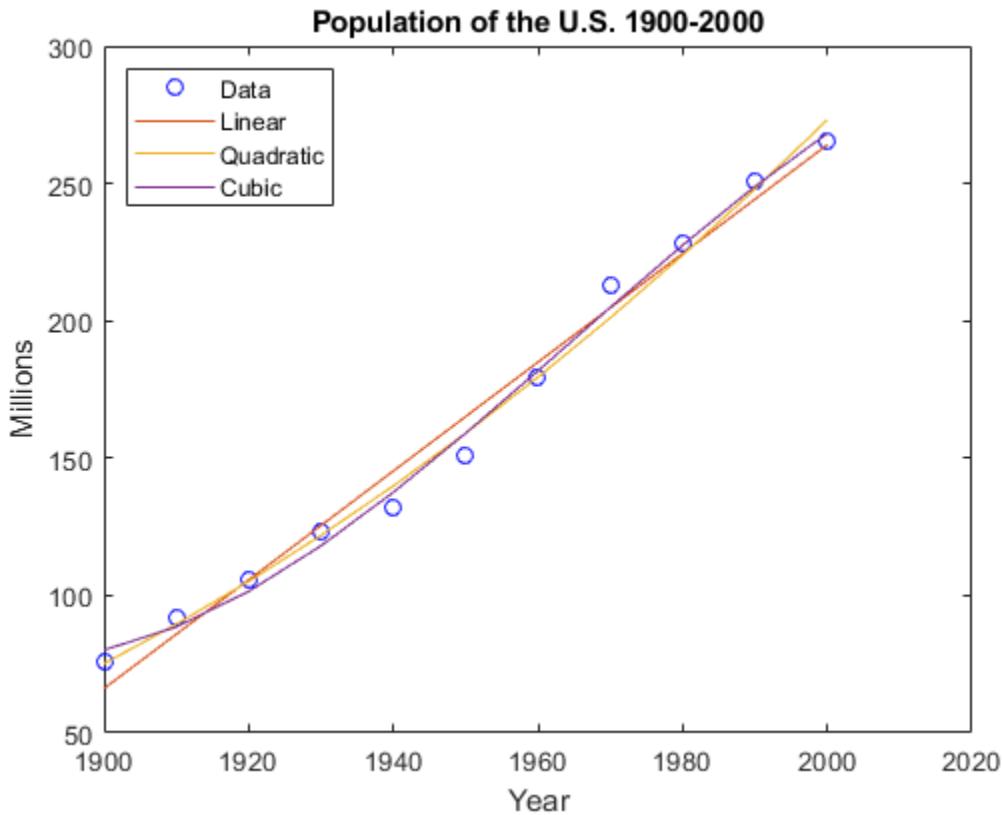
创建具有任意数量的文本和代码行的节。

我们可以绘制用于拟合数据的线性、二次和立方曲线。我们使用 `polyval` 函数来计算在点 `x` 处的拟合多项式。

```
pred1 = polyval(coef1,x);  
pred2 = polyval(coef2,x);  
pred3 = polyval(coef3,x);  
[pred1; pred2; pred3]  
  
ans = 3×11  
  
66.1296 85.9281 105.7266 125.5250 145.3235 165.1220 184.9205 204.7190 224.5174 244.3159 264.1144  
75.1904 89.5524 105.1225 121.9007 139.8870 159.0814 179.4840 201.0946 223.9134 247.9403 273.1753  
80.1414 88.5622 101.4918 118.1050 137.5766 159.0814 181.7944 204.8904 227.5441 248.9305 268.2243
```

下面我们绘制每个多项式的预测值。

```
hold on  
plot(years,pred1)  
plot(years,pred2)  
plot(years,pred3)  
ylim([50 300])  
legend({'Data' 'Linear' 'Quadratic' 'Cubic'},'Location','NorthWest')  
hold off
```



### 预测人口

您可以将您的实时脚本与其他 MATLAB 用户共享，这样他们可以重现您的结果。您也可以将结果发布为 PDF、Microsoft® Word 或 HTML 文档。在实时脚本中添加控件，可以向用户展示重要参数会对分析产生怎样的影响。要添加控件，请转至**实时编辑器**选项卡，点击**控件**按钮，然后从可用选项中进行选择。

我们现在可以使用三个方程计算预测的给定年份的人口。

```
year = ;
xyear = (year-1900)/50;
pred1 = polyval(coef1,xyear);
pred2 = polyval(coef2,xyear);
pred3 = polyval(coef3,xyear);
[pred1 pred2 pred3]

ans = 1×3
299.7517 321.6427 295.0462
```

以 2010 年为例，线性拟合和三次拟合的预测值相似，约为 2.84 亿人口，而二次拟合的预测值更高，约为 3 亿人口。

## 另请参阅

### 详细信息

- “在实时编辑器中创建实时脚本” (第 19-6 页)
- MATLAB 实时脚本库

# 使用实时编辑器加快探索编程速度

下面是如何使用实时编辑器加快探索编程速度的示例。以下示例演示如何使用实时编辑器：

- 随代码一起查看代码输出。
- 将您的程序划分为多个节以分别计算各个代码块。
- 纳入可视化内容。
- 使用控件对参数值进行试验。
- 汇总和共享您的发现。

## 加载高速公路死亡数据

实时编辑器随代码一起显示代码输出。要运行节，请转至**实时编辑器**选项卡，然后选择**运行节**按钮。您也可以点击在将鼠标移至节的左边缘时显示的蓝条。

在下面的示例中，我们将探索一些高速公路死亡数据。首先加载数据。变量显示为表的列标题。

```
load fatalities
fatalities(1:10,:)

ans=10×8 table
    longitude    latitude    deaths    drivers    vehicles    vehicleMiles    alcoholRelated    urbanPopulat
    _____    _____    _____    _____    _____    _____    _____    _____
Wyoming      -107.56    43.033    164    380.18    671.53    9261    54    65.226
District_of_Columbia -77.027    38.892    43    349.12    240.4    3742    12    100
Vermont       -72.556    44.043    98    550.46    551.52    7855    20    38.196
North_Dakota   -99.5     47.469    100    461.78    721.84    7594    35    55.807
South_Dakota   -99.679    44.272    197    563.3     882.77    8784    76    51.923
Delaware       -75.494    39.107    134    533.94    728.52    9301    48    80.021
Montana        -110.58    46.867    229    712.88    1056.7    11207    100    54.031
Rhode_Island   -71.434    41.589    83     741.84    834.5     8473    41    90.936
New_Hampshire   -71.559    43.908    171    985.77    1244.6    13216    51    59.181
Maine          -69.081    44.886    194    984.83    1106.8    14948    58    40.206
```

## 计算死亡率

使用实时编辑器可以将您的程序划分为包含文本、代码和输出的多个节。要创建新的节，请转至**实时编辑器**选项卡，然后点击**分节符**按钮。节中的代码可以独立运行，这使得可轻松了解您在编写程序时的想法。

计算每一百万行车里程的死亡率。从这些值中，我们可以发现具有最低和最高死亡率的州。

```
states = fatalities.Properties.RowNames;
rate = fatalities.deaths./fatalities.vehicleMiles;
[~, minIdx] = min(rate);           % Minimum accident rate
[~, maxIdx] = max(rate);           % Maximum accident rate
disp([states{minIdx} ' has the lowest fatality rate at ' num2str(rate(minIdx))])

Massachusetts has the lowest fatality rate at 0.0086907
```

```
disp([states{maxIdx} ' has the highest fatality rate at ' num2str(rate(maxIdx))])
```

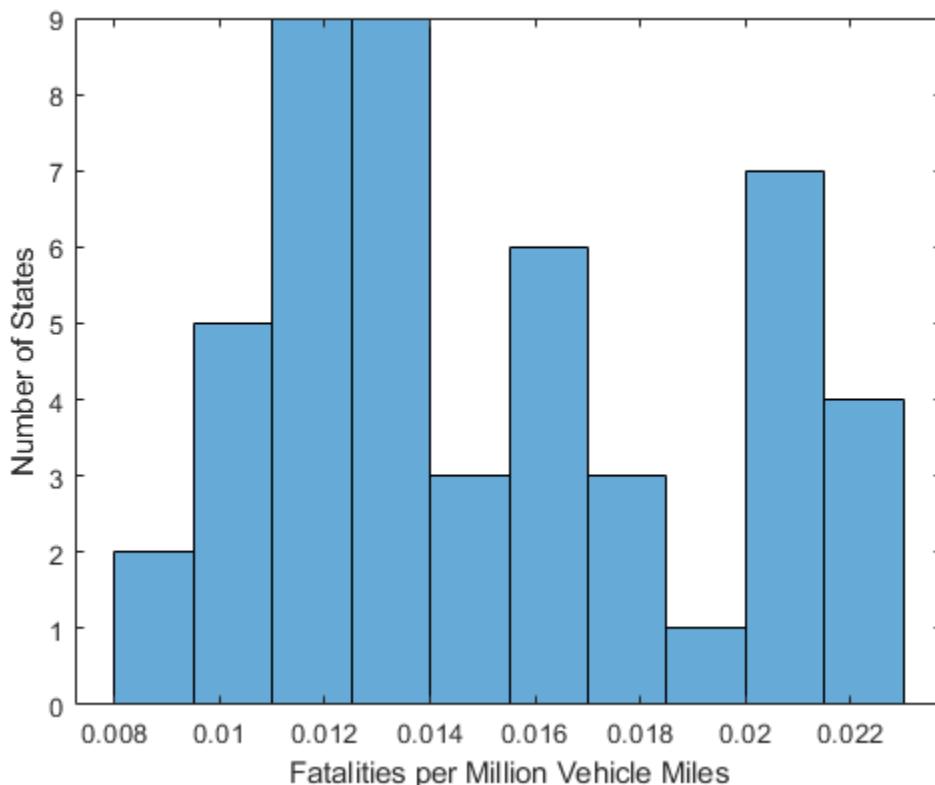
Mississippi has the highest fatality rate at 0.022825

## 死亡分布

您可以在程序中纳入可视化内容。与输出类似，绘图和图窗随生成这些内容的代码一起显示。

我们可以使用条形图查看各个州的死亡率分布。有 11 个州每百万行车里程的死亡率大于 0.02。

```
histogram(rate,10)
xlabel('Fatalities per Million Vehicle Miles')
ylabel('Number of States')
```



## 查找数据中的相关性

您可以通过在实时编辑器中用不同的参数值进行试验来查看结果的变化，从而快速探索您的数据。添加控件以交互方式更改参数值。要添加控件，请转至**实时编辑器**选项卡，点击**控件**按钮，然后从可用选项中进行选择。

我们可以用这些数据进行试验，以了解表中是否有任何变量与高速公路的死亡事故存在关联。例如，结果显示，城市人口百分比更高的州中的高速公路死亡率更低。

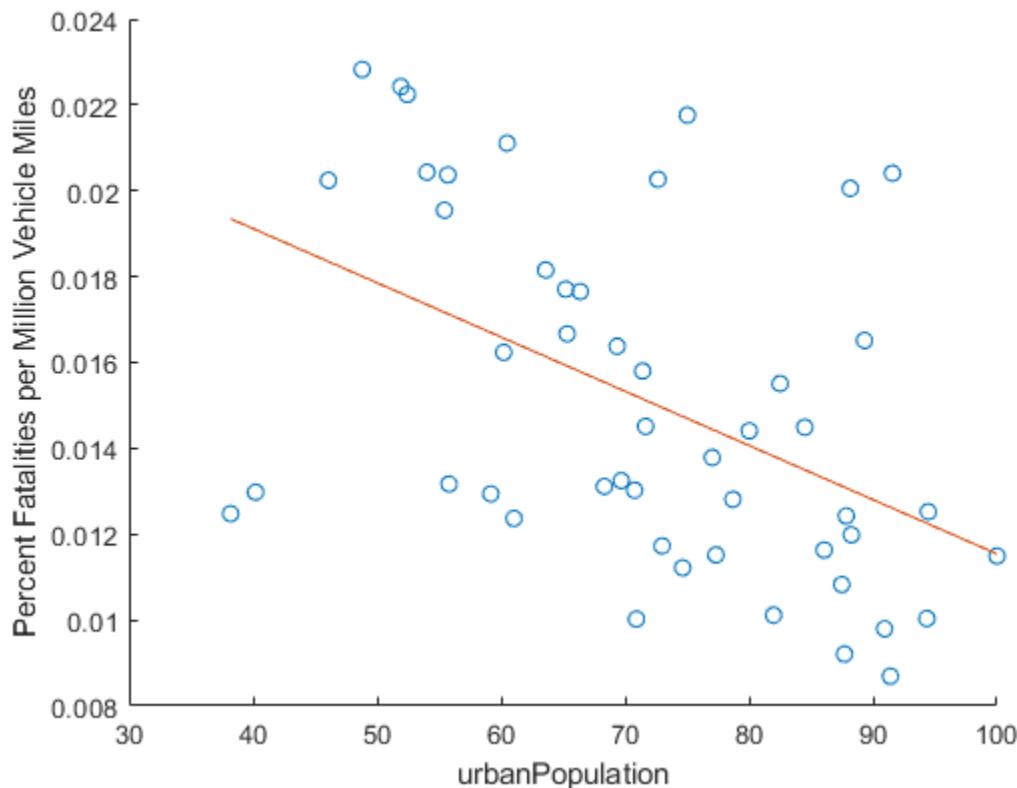
```
dataToPlot = ;
close % Close any open figures
scatter(fatalities.(dataToPlot),rate) % Plot fatalities vs. selected variable
xlabel(dataToPlot)
ylabel('Percent Fatalities per Million Vehicle Miles')

hold on
xmin = min(fatalities.(dataToPlot));
```

```

xmax = max(fatalities.(dataToPlot));
p = polyfit(fatalities.(dataToPlot),rate,1); % Calculate & plot least squares line
plot([xmin xmax], polyval(p,[xmin xmax]))

```



### 在美国地图上绘制死亡率与城市化率之间的关系图

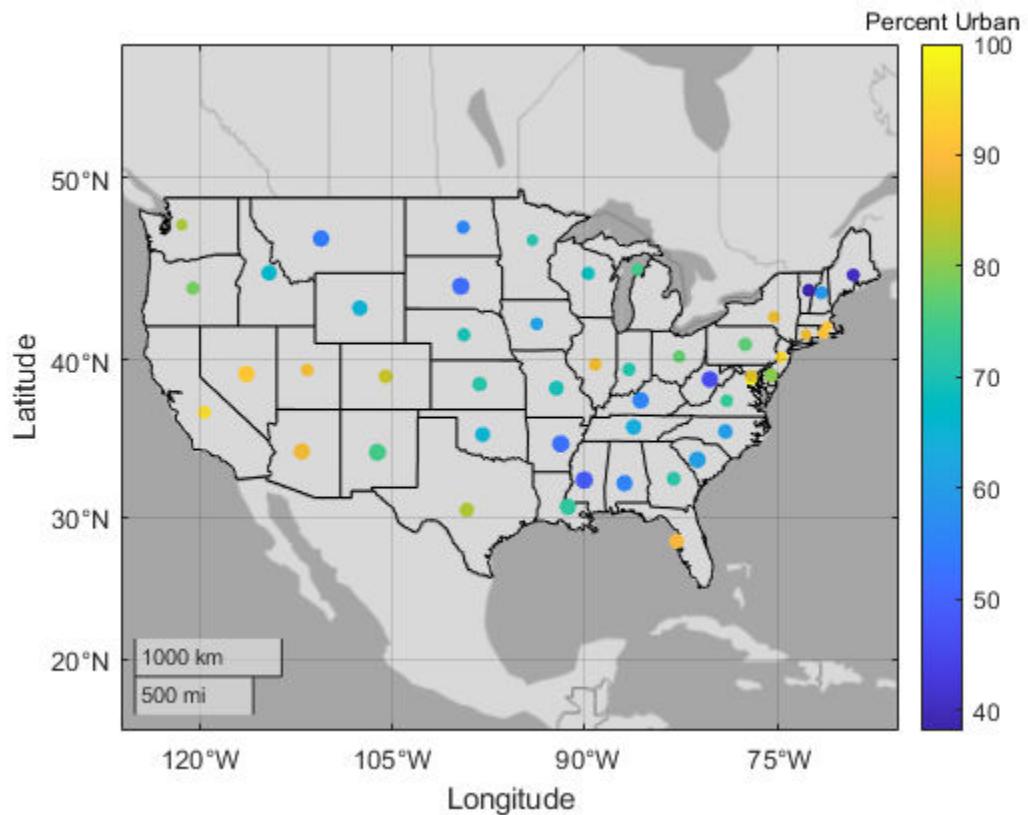
汇总您的结果并将您的实时脚本与您的同事共享。通过使用您的实时脚本，您的同事可以重新创建并扩展您的分析。您也可以将分析另存为 HTML、Microsoft® Word 或 PDF 文档以供发布。

基于此分析，我们可以在美国大陆地图上使用死亡率和城市人口图汇总我们的发现。

```

load usastates.mat
figure
geoplot([usastates.Lat], [usastates.Lon], 'black')
geobasemap darkwater
hold on
geosscatter(fatalities.latitude,fatalities.longitude,2000*rate,fatalities.urbanPopulation,'filled')
c = colorbar;
title(c,'Percent Urban')

```



## 另请参阅

### 详细信息

- “在实时编辑器中创建实时脚本”（第 19-6 页）
- MATLAB 实时脚本库

# 使用实时编辑器创建交互式记叙脚本

下面是如何在实时编辑器中创建交互式记叙脚本的示例。交互式记叙脚本将您求解问题所用的各项计算整合到一起。以下示例演示如何：

- 使用格式化文本描述您的方法。
- 同时显示输出与 MATLAB 代码。
- 用方程描述底层数学原理。
- 用图像阐释要点。
- 指向背景材料的链接。
- 使用控件修改参数并重新运行分析。
- 绘制数据以实现可视化。
- 邀请同事扩展您的分析。

## 方法概述

将格式化文本纳入到交互式记叙脚本。使用粗体、斜体和下划线文本突出显示重要的字词。使用项目符号或数字设置列表格式。

通过计算以下各项，估计一个典型太阳能电池板在特定日期、时间和位置的电力输出：

- 太阳时
- 太阳赤纬角和太阳高度角
- 到达地球表面的空气质量
- 太阳辐射
- 基于太阳能电池板上的辐射，可算出太阳的位置、倾角和效率
- 单日和全年的发电情况

使用这些计算的结果对示例日期和位置的太阳辐射和电池板辐射绘图。然后对电池板全年的预期发电情况绘图。为了简化分析，请使用为此示例创建的两个 MATLAB 函数：**solarCorrection** 和 **panelRadiation**。

## 太阳时

同时显示输出与产生该输出的代码。要运行代码节，请转至**实时编辑器**选项卡，然后点击**运行节**按钮。

太阳能电池板的发电量取决于到达电池板的太阳辐射量。这又取决于太阳在运动时相对于电池板的位置。例如，假设您要计算 6 月 1 日正午 12 点位于马萨诸塞州波士顿的太阳能电池板的电力输出。

```
lambda = ; % longitude
phi = ; % latitude
UTCOff = ; % UTC offset
january1 = datetime(2019,1,1); % January 1st
localTime = datetime(2019,6,1,12,0,0) % Noon on June 1

localTime = datetime
01-Jun-2019 12:00:00
```

要计算太阳在给定日期和时间的位置，请使用太阳时。太阳时正午 12 点是太阳在天空中位于最高处的时刻。要计算太阳时，需要对地方时间进行修正。该修正包含两个部分：

- 修正观察者位置与地方子午线之间差异的项。
- 与地球轨道偏心率和转轴倾角有关的轨道项。

使用 **solarCorrection** 函数计算太阳时。

```
d = caldays(between(january1,localTime,'Day')); % Day of year
solarCorr = solarCorrection(d,lambda,str2double(UTCOff)); % Correction to local time
solarTime = localTime + minutes(solarCorr)

solarTime = datetime
01-Jun-2019 12:18:15
```

### 太阳赤纬角和高度角

在脚本中纳入表示底层数学原理的方程。使用 **LaTeX** 命令创建方程。要添加新方程，请转至**实时编辑器**选项卡，然后点击**方程**按钮。双击方程可在方程编辑器中对其进行编辑。

太阳赤纬角 ( $\delta$ ) 是太阳相对于地球赤道平面的角度。春分和秋分平分点的太阳赤纬角为  $0^\circ$ ，并在夏至升至最大值  $23.45^\circ$ 。使用以下方程计算一年中某一天 (d) 的太阳赤纬角

$$\delta = \sin^{-1} \left( \sin(23.45) \sin \left( \frac{360}{365}(d - 81) \right) \right)$$

然后，使用赤纬角 ( $\delta$ )、纬度 ( $\phi$ ) 和时角 ( $\omega$ ) 计算当前时刻的太阳高度角 ( $\alpha$ )。时角是当前太阳时与太阳时正午之间的地球旋转角度数。

$$\alpha = \sin^{-1}(\sin\delta \sin\phi + \cos\delta \cos\phi \cos\omega)$$

```
delta = asind(sind(23.45)*sind(360*(d - 81)/365)); % Declination
omega = 15*(solarTime.Hour + solarTime.Minute/60 - 12); % Hour angle
alpha = asind(sind(delta)*sind(phi) + ... % Elevation
    cosd(delta)*cosd(phi)*cosd(omega));
disp(['Solar Declination = ' num2str(delta) ' Solar Elevation = ' num2str(alpha)])
```

Solar Declination = 21.8155 Solar Elevation = 69.113

使用太阳赤纬角和当地纬度计算以标准时间表示的日出和日落时间。

$$\text{sunrise} = 12 - \frac{\cos^{-1}(-\tan\phi\tan\delta)}{15^\circ} - \frac{TC}{60} \quad \text{sunset} = 12 + \frac{\cos^{-1}(-\tan\phi\tan\delta)}{15^\circ} - \frac{TC}{60}$$

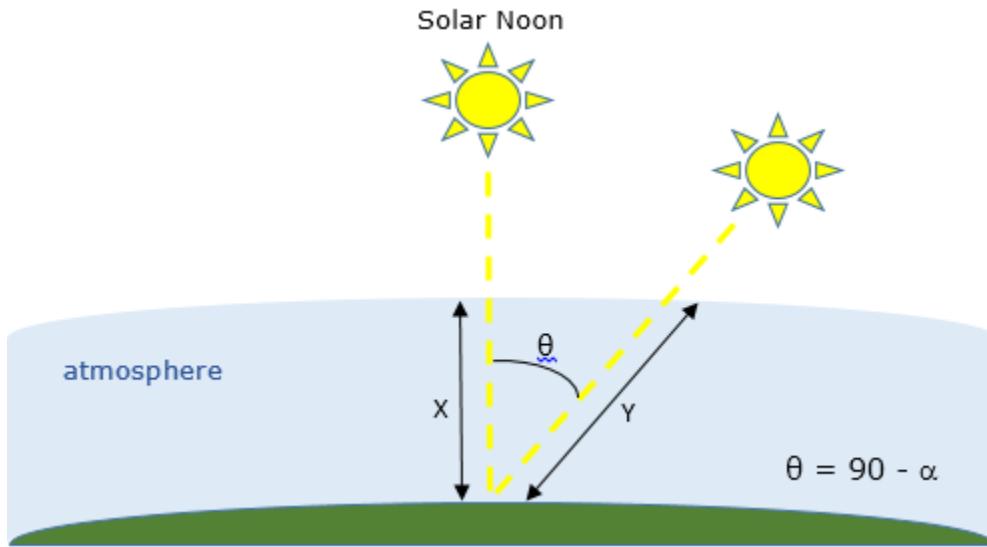
```
midnight = dateshift(localTime,'start','day');
sr = 12 - acosd(-tand(phi)*tand(delta))/15 - solarCorr/60;
sunrise = timeofday(midnight + hours(sr));
ss = 12 + acosd(-tand(phi)*tand(delta))/15 - solarCorr/60;
sunset = timeofday(midnight + hours(ss));
disp(['Sunrise = ', datestr(sunrise,'HH:MM:SS'), ' Sunset = ', datestr(sunset,'HH:MM:ss')])
```

Sunrise = 04:16:06 Sunset = 19:07:22

### 空气质量

纳入图像以阐释记叙脚本中的要点。要纳入某图像，请从其他来源复制并粘贴该图像，或者转至**实时编辑器**选项卡，然后点击**图像**按钮。

当太阳光穿过地球大气层时，部分太阳辐射被吸收。空气质量体现了当太阳高度角为 90° 时，阳光穿过大气层 (Y) 的路径长度相对于最短可能路径 (X) 的关系，如下图所示。它是太阳高度角的函数 ( $\alpha$ )。



空气质量越大，则辐射到达地面越少。使用以下方程计算空气质量

$$AM = \frac{1}{\cos(90 - \alpha) + 0.5057(6.0799 + \alpha)^{-1.6364}}.$$

然后，使用经验方程计算到达地面的太阳辐射（单位：千瓦/平方米）

$$sRad = 1.353 * 0.7^{AM^{0.678}}.$$

```
AM = 1/(cosd(90-alpha) + 0.50572*(6.07955+alpha)^-1.6354);
solarRad = 1.353*0.7^(AM^0.678); % kW/m^2
disp(['Air Mass = ' num2str(AM) ' Solar Radiation = ' num2str(solarRad) ' kW/m^2'])
```

Air Mass = 1.0698 Solar Radiation = 0.93141 kW/m<sup>2</sup>

### 固定电池板上的太阳辐射

使用超链接引用其他来源的支持信息。要添加超链接，请转至**实时编辑器**选项卡，然后点击**超链接**按钮。

安装了太阳跟踪器的太阳能电池板可以随太阳移动，可以随着太阳移动而接收 100% 的太阳辐射。但是，大多数安装的太阳能电池板采用固定的方位角和倾角安装。因此，到达电池板的实际辐射还取决于太阳方位角。 $\gamma$  是太阳在天空中的位置的指南针方向。在北半球上的太阳时正午，太阳方位角是 180°（对应于正南方向）。使用以下方程计算太阳方位角

$$\gamma = \begin{cases} \cos^{-1}\left(\frac{\sin\delta\cos\phi - \cos\delta\sin\phi\cos\omega}{\cos\alpha}\right) & \text{for solar time } \leq 12 \\ 360^\circ - \cos^{-1}\left(\frac{\sin\delta\cos\phi - \cos\delta\sin\phi\cos\omega}{\cos\alpha}\right) & \text{for solar time } > 12 \end{cases}$$

```
gamma = acosd((sind(delta)*cosd(phi) - cosd(delta)*sind(phi)*cosd(omega))/cosd(alpha));
if (hour(solarTime) >= 12) && (omega >= 0)
    gamma = 360 - gamma;
```

```
end
disp(['Solar Azimuth = ' num2str(gamma)])
```

Solar Azimuth = 191.7888

在北半球，典型的太阳能电池板安装方法是将电池板朝南，并且电池板方位角 ( $\beta$ ) 为  $180^\circ$ 。在北纬上，通常的倾角 ( $\tau$ ) 为  $35^\circ$ 。使用以下方程基于总的太阳辐射计算固定电池板的电池板辐射

$$pRad = sRad[\cos(\alpha)\sin(\tau)\cos(\beta - \gamma) + \sin(\alpha)\cos(\tau)].$$

```
beta = 180; % Panel azimuth
tau = 35; % Panel tilt
panelRad = solarRad*max(0,(cosd(alpha)*sind(tau)*cosd(beta-gamma) + sind(alpha)*cosd(tau)));
disp(['Panel Radiation = ' num2str(panelRad) ' kW/m^2'])
```

Panel Radiation = 0.89928 kW/m<sup>2</sup>

### 一天中的电池板辐射和发电情况

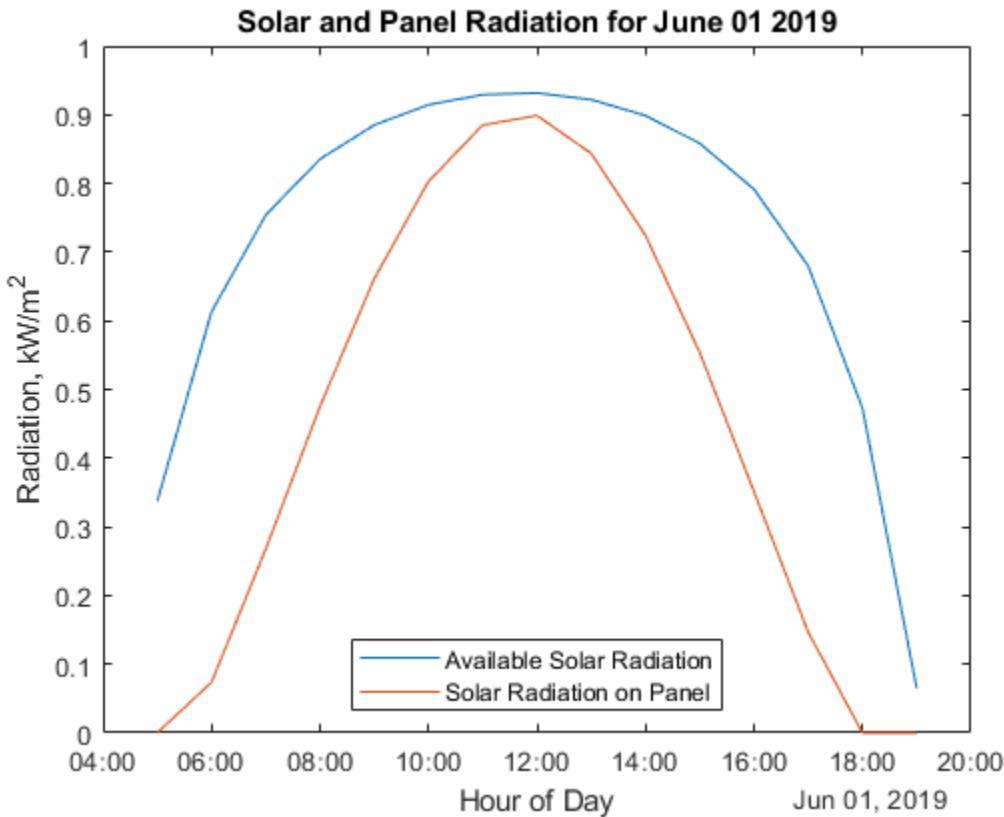
使用交互式控件修改参数。同时显示绘图及生成绘图的代码。

#### 电池板辐射

对于一年中的某一天，计算总的太阳辐射和电池板辐射。为了简化分析，请使用 `panelRadiation` 函数。在不同日期进行尝试，了解太阳能和电池板辐射在一年中不同时间的变化情况。

```
selectedMonth = ;
selectedDay = ;
selectedDate = datetime(2019,selectedMonth,selectedDay);
[times,solarRad,panelRad] = panelRadiation(selectedDate,lambda,phi,UTCoff,tau,beta);

plot(times,solarRad,times,panelRad)
title(['Solar and Panel Radiation for ' datestr(selectedDate,'mmmm dd yyyy')])
xlabel('Hour of Day');
ylabel('Radiation, kW/m^2')
legend('Available Solar Radiation','Solar Radiation on Panel','Location','South')
```



## 发电情况

到目前为止，计算假定所有到达电池板的辐射都可用于发电。但是，太阳能电池板不会将可用的太阳辐射 100% 转换为电能。太阳能电池板的效率是指到达电池板的辐射转换为电能的比率。太阳能电池板的效率取决于电池的设计和材料。

通常，住宅安装包括  $20\text{m}^2$  的太阳能电池板，效率为 25%。请在下面修改参数以了解效率和大小如何影响电池板产生的电能。

```
eff = ; % Panel efficiency
pSize = ; % Panel size in m^2
radiation = sum(panelRad(1:end-1)+panelRad(2:end))/2;
dayPower = eff*pSize*radiation; % Panel electric output in kW
disp(['Expected daily electrical output for ' datestr(selectedDate) '=' num2str(dayPower) ' kW-hrs'])
```

Expected daily electrical output for 01-Jun-2019 = 33.4223 kW-hrs

## 全年的发电情况

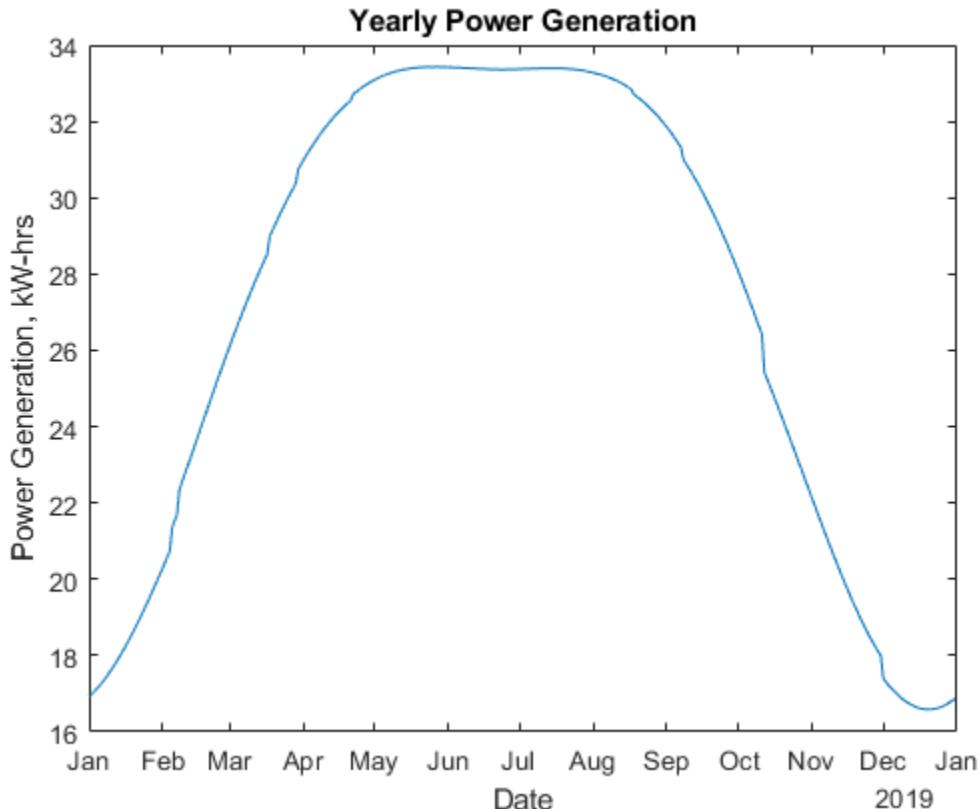
在绘图上悬停鼠标以进行交互。与实时编辑器中的绘图交互将生成代码，您可以随后将其添加到脚本中。

重复该计算来估计一年中的每一天产生的电能。

```
yearDates = datetime(2019,1,1:365); % Create a vector of days in the year
for i = 1:365
    [times,solarRad,panelRad] = panelRadiation(yearDates(i),lambda,phi,UTCoff,tau,beta);
    radiation = sum(panelRad(1:end-1)+panelRad(2:end))/2;
```

```
dailyPower(i) = eff*pSize*radiation;
end
```

```
plot(yearDates,dailyPower)
title('Yearly Power Generation')
xlabel('Date');
ylabel('Power Generation, kW-hrs')
```



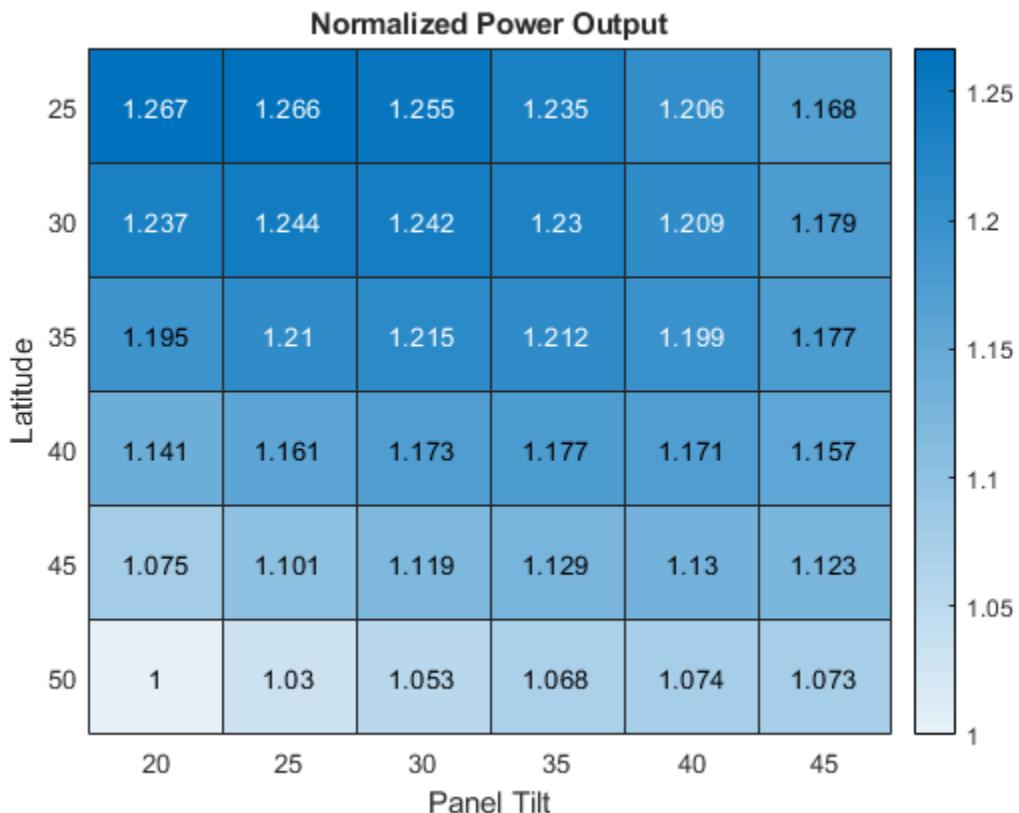
```
yearlyPower = sum(dailyPower);
disp(['Expected annual power output = ' num2str(yearlyPower) ' kW-hrs'])
```

Expected annual power output = 9954.3272 kW-hrs

### 电池板倾角和纬度

使用热图确定电池板倾斜如何影响发电情况。以下热图显示，任何位置的最佳电池板倾斜度都比纬度低大约 5°。

```
load LatitudeVsTilt.mat
heatmap(powerTbl,'Tilt','Latitude',...
    'ColorVariable','Power');
xlabel('Panel Tilt')
ylabel('Latitude')
title('Normalized Power Output')
```



## 扩展分析

与同事共享您的分析。邀请同事重现或扩展您的分析。使用实时编辑器协作工作。

实际上，太阳能电池板的实际电力输出受到当地气候条件的显著影响。关于此分析的一个有趣扩展是了解云层会对结果产生怎样的影响。在美国，您可以使用以下政府网站的数据。

- 使用美国国家气象局网站上的历史当地气象数据。
- 使用美国国家太阳能辐射数据库中测量的太阳辐射数据。

## 另请参阅

### 详细信息

- “在实时编辑器中创建实时脚本”（第 19-6 页）
- “在实时编辑器中格式化文件”（第 19-29 页）
- MATLAB 实时脚本库

## 使用实时编辑器创建交互式课程材料

下面是一个如何在课堂中使用实时脚本的示例。以下示例演示如何：

- 添加方程用于解释底层数学原理。
- 执行 MATLAB 代码的各个节。
- 纳入绘图以实现可视化。
- 使用链接和图像提供支持信息。
- 使用 MATLAB 代码进行交互式试验。
- 使用其他示例加深概念理解。
- 使用实时脚本进行赋值。

### 计算 1 的 n 次方根意味着什么？

为您要讲解的概念添加方程以解释底层数学原理。要添加方程，请转至**实时编辑器**选项卡并点击**方程**按钮。然后，从**方程**选项卡中选择符号和结构体。

现在我们探讨如何计算 1 的根。计算 1 的 n 次方根意味着什么？1 的 n 次方根是方程  $x^n - 1 = 0$  的解。

对于平方根，很容易计算。值为  $x = \pm \sqrt{1} = \pm 1$ 。对于高阶根，则要困难很多。要计算 1 的立方根，需要对方程  $x^3 - 1 = 0$  求解。我们可以分解此方程以获取

$$(x - 1)(x^2 + x + 1) = 0.$$

因此，第一个立方根是 1。现在，我们可以使用二次公式获取第二个和第三个立方根。

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

### 计算立方根

要执行 MATLAB 代码的各个节，请转至**实时编辑器**选项卡，然后点击**运行节**按钮。输出与创建它的代码显示在一起。使用**分节符**按钮创建节。

在示例中，a、b 和 c 都等于 1。其他两个根基于下列公式计算得到：

```
a = 1; b = 1; c = 1;
roots = [];
roots(1) = 1;
roots(2) = (-b + sqrt(b^2 - 4*a*c))/(2*a); % Use the quadratic formula
roots(3) = (-b - sqrt(b^2 - 4*a*c))/(2*a);
```

因此 1 的完整立方根集合为：

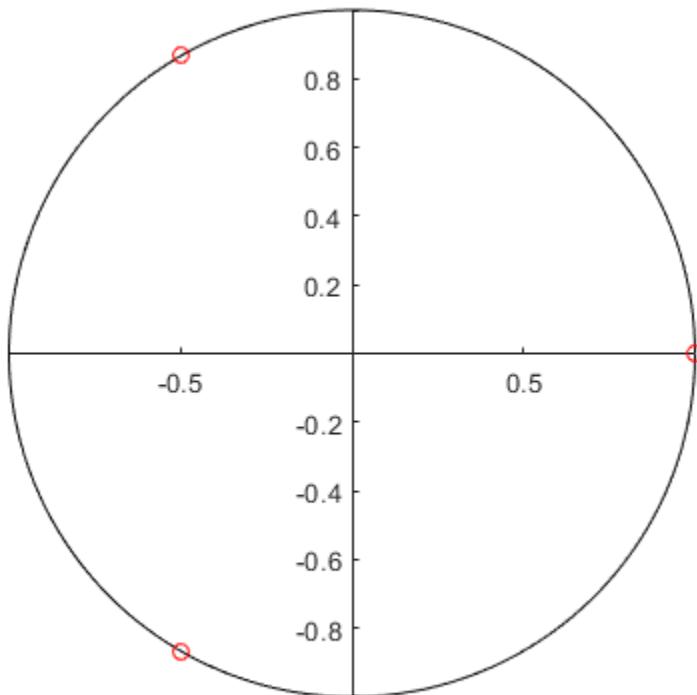
```
disp(roots')
1.0000 + 0.0000i
-0.5000 - 0.8660i
-0.5000 + 0.8660i
```

### 在复平面中显示根

在实时编辑器中纳入绘图，以便学生以可视方式了解重要概念。

我们可以在复平面中以可视方式呈现根以查看其位置。

```
range = 0:0.01:2*pi;
plot(cos(range),sin(range),'k') % Plot the unit circle
axis square; box off
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
hold on
plot(real(roots), imag(roots), 'ro') % Plot the roots
```



## 计算高阶根

要添加支持信息，请转至**实时编辑器**选项卡，然后点击**超链接和图像**按钮。学生可以在课堂外使用支持信息了解课程主题。

一旦超过  $n = 3$ ，情况会变得更加棘手。可以使用 Lodovico Ferrari 在 1540 年发现的四次公式来获取 4 次方根。但此公式较长且难以处理，对于计算 4 次以上的根没有帮助。幸运地是，17 世纪的法国数学家 Abraham de Moivre 给出了更好的解决方法。

Abraham de Moivre 于 1667 年 5 月 26 日出生于香巴尼地区的维特里。他与 Isaac Newton、Edmund Halley 和 James Stirling 是同代人，并且都是朋友。[https://en.wikipedia.org/wiki/Abraham\\_de\\_Moivre](https://en.wikipedia.org/wiki/Abraham_de_Moivre)



**Abraham de Moivre**

他因 de Moivre 定理而闻名于世，该定理在复数与三角学之间建立了关联，并且他在正态分布和概率论方面也做出了巨大贡献。De Moivre 撰写了一本关于概率论的书 *The Doctrine of Chances*，倍受赌博者的推崇。De Moivre 首先发现了 Binet 公式，它是 Fibonacci 数的闭型表达式，该公式将黄金分割率  $\phi$  的  $n$  次幂与第  $n$  个 Fibonacci 数建立关联。他也是第一个做出中心极限定理假设的人，这一定理奠定了概率论的基础。

de Moivre 定理阐释，对于任何实数  $x$  和任何整数  $n$ ,

$$(\cos x + i \sin x)^n = \cos(nx) + i \sin(nx).$$

这如何帮助我们解决我们的问题？我们还知道，对于任何整数  $k$ ,

$$1 = \cos(2k\pi) + i \sin(2k\pi).$$

因此，根据 De Moivre 定理，可以得到

$$1^{1/n} = (\cos(2k\pi) + i \sin(2k\pi))^{1/n} = \cos\left(\frac{2k\pi}{n}\right) + i \sin\left(\frac{2k\pi}{n}\right).$$

### 计算 1 的 $n$ 次方根

使用实时编辑器对 MATLAB 代码进行交互式试验。添加控件以向学生展示重要参数会对分析产生怎样的影响。要添加控件，请转至**实时编辑器**选项卡，点击**控件**按钮，然后从可用选项中进行选择。

我们可以使用此最后一个方程计算 1 的  $n$  次方根。例如，对于任何值  $n$ ，我们可以将值  $k = 0 \dots n - 1$  用于上述公式。我们可以使用此 MATLAB 代码对不同的  $n$  值进行试验：

```

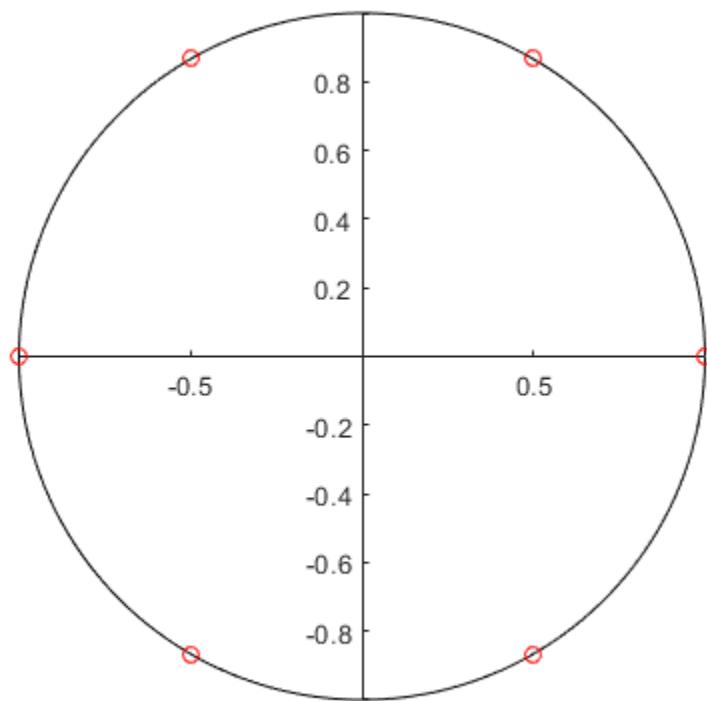
n = ;
roots = zeros(1, n);
for k = 0:n-1
    roots(k+1) = cos(2*k*pi/n) + 1i*sin(2*k*pi/n); % Calculate the roots
end
disp(roots')

1.0000 + 0.0000i
0.5000 - 0.8660i
-0.5000 - 0.8660i
-1.0000 - 0.0000i
-0.5000 + 0.8660i
0.5000 + 0.8660i

```

在复平面中绘制的根显示，这些根以  $2\pi/n$  的间隔均匀地分布在单位圆上。

```
cla
plot(cos(range),sin(range),'k')          % Plot the unit circle
hold on
plot(real(roots),imag(roots),'ro')        % Plot the roots
```

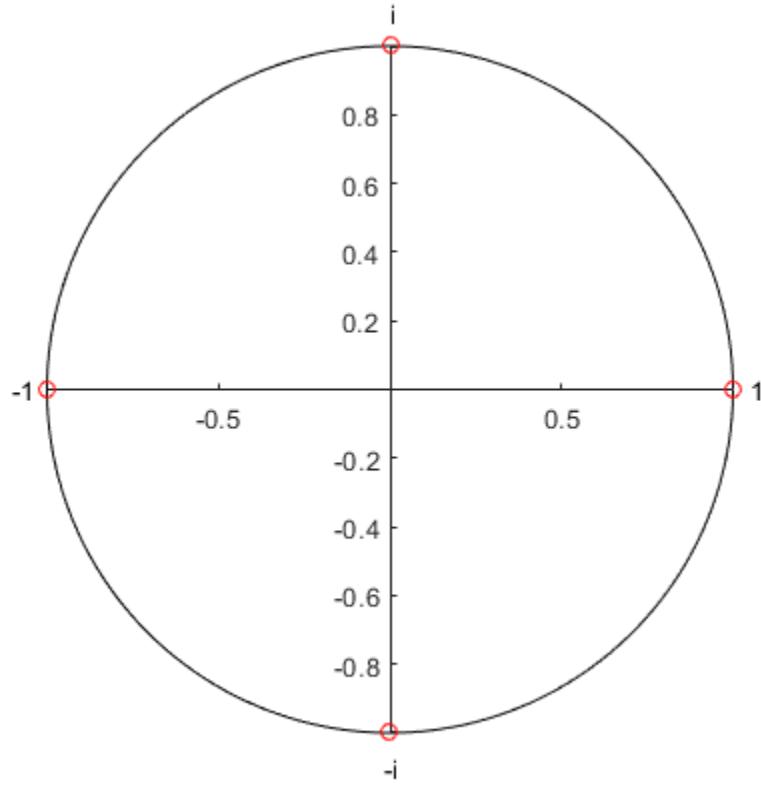


### 计算 $-1$ 、 $i$ 和 $-i$ 的 $n$ 次方根

使用其他示例加深对重要概念的理解。在讲课过程中修改代码以回答问题或者更深入地探讨想法。

只需对上述方法进行延伸，即可计算出  $-1$ 、 $i$  和  $-i$  的根。如果观察一下单位圆，可以看出值  $1$ 、 $i$ 、 $-1$ 、 $-i$  分别出现在角度  $0$ 、 $\pi/2$ 、 $\pi$  和  $3\pi/2$  位置。

```
r = ones(1,4);
theta = [0 pi/2 pi 3*pi/2];
[x,y] = pol2cart(theta,r);
cla
plot(cos(range),sin(range),'k')          % Plot the unit circle
hold on
plot(x, y, 'ro')                      % Plot the values of 1, i, -1, and -i
text(x(1)+0.05,y(1),'1')              % Add text labels
text(x(2),y(2)+0.1,'i')
text(x(3)-0.1,y(3),'-1')
text(x(4)-0.02,y(4)-0.1,'-i')
```



清楚这一点后，可以对  $i$  编写以下表达式：

$$i = \cos((2k + 1/2)\pi) + i \sin((2k + 1/2)\pi).$$

对等式的两端取  $n$  次方根，得到

$$i^{1/n} = (\cos((2k + 1/2)\pi) + i \sin((2k + 1/2)\pi))^{1/n}$$

通过 de Moivre 定理，得到

$$i^{1/n} = (\cos((2k + 1/2)\pi) + i \sin((2k + 1/2)\pi))^{1/n} = \cos\left(\frac{(2k + 1/2)\pi}{n}\right) + i \sin\left(\frac{(2k + 1/2)\pi}{n}\right).$$

### 家庭作业

以实时脚本为基础分配作业。为学生提供课堂中使用的实时脚本，并让学生完成练习，从而测试学生对课堂内容的理解程度。

使用上述方法完成以下练习：

**练习 1：**编写 MATLAB 代码，计算  $i$  的 3 个立方根。

% Put your code here

**练习 2：**编写 MATLAB 代码，计算  $-1$  的 5 个 5 次方根。

% Put your code here

**练习 3:** 描述计算任意复数的 n 次方根的数学方法。提供该方法所用的方程。

(在此处描述该方法)

## 另请参阅

### 详细信息

- “在实时编辑器中创建实时脚本” (第 19-6 页)
- MATLAB 实时脚本库

## 使用实时编辑器创建示例

以下示例说明如何使用实时编辑器来记录或创建您的代码示例。该示例使用格式化文本、方程和可运行代码来记录示例函数 `estimatePanelOutput`。它还使用滑块和文本字段来允许用户用不同函数输入进行试验。

要查看控件并与之交互，请在 MATLAB® 中打开此示例。

### estimatePanelOutput Function

#### Overview

The `estimatePanelOutput` function estimates the power output from a typical solar panel installation based on location panel size, and panel efficiency.

The function uses this formula to calculate the solar declination:

$$\alpha = \sin^{-1} \left( \sin(23.45) \sin\left(\frac{360}{365}(d - 81)\right) \right)$$

and this formula to calculate solar elevation:

$$\epsilon = \sin^{-1} (\sin \delta \sin \phi + \cos \delta \cos \phi \cos \omega)$$

The functions calls two additional MATLAB functions, `solarCorrection` and `hourlyPanelRadiation`.

#### Examples

##### Estimate Panel Output with Standard Efficiency

Call the `estimatePanelOutput` function with the panel size, specified in  $m^2$ . Since no efficiency value is specified, `calculatePanelOutput` uses the default efficiency value of 25%.

Specify the latitude, longitude, and UTC offset for the location of your panels, as well as the size of your panels in  $m^2$ .

```
longitude = -71.06;
latitude = 42.36;
UTCoffset = -5;
pSize = 10;
```

Calculate the expected electrical output of the panel.

```
panelOutput = estimatePanelOutput(longitude, latitude, UTCoffset, pSize);
disp(['Expected electrical output = ' num2str(panelOutput) ' kW'])
```

```
Expected electrical output = 2.2472 kW
```

## 另请参阅

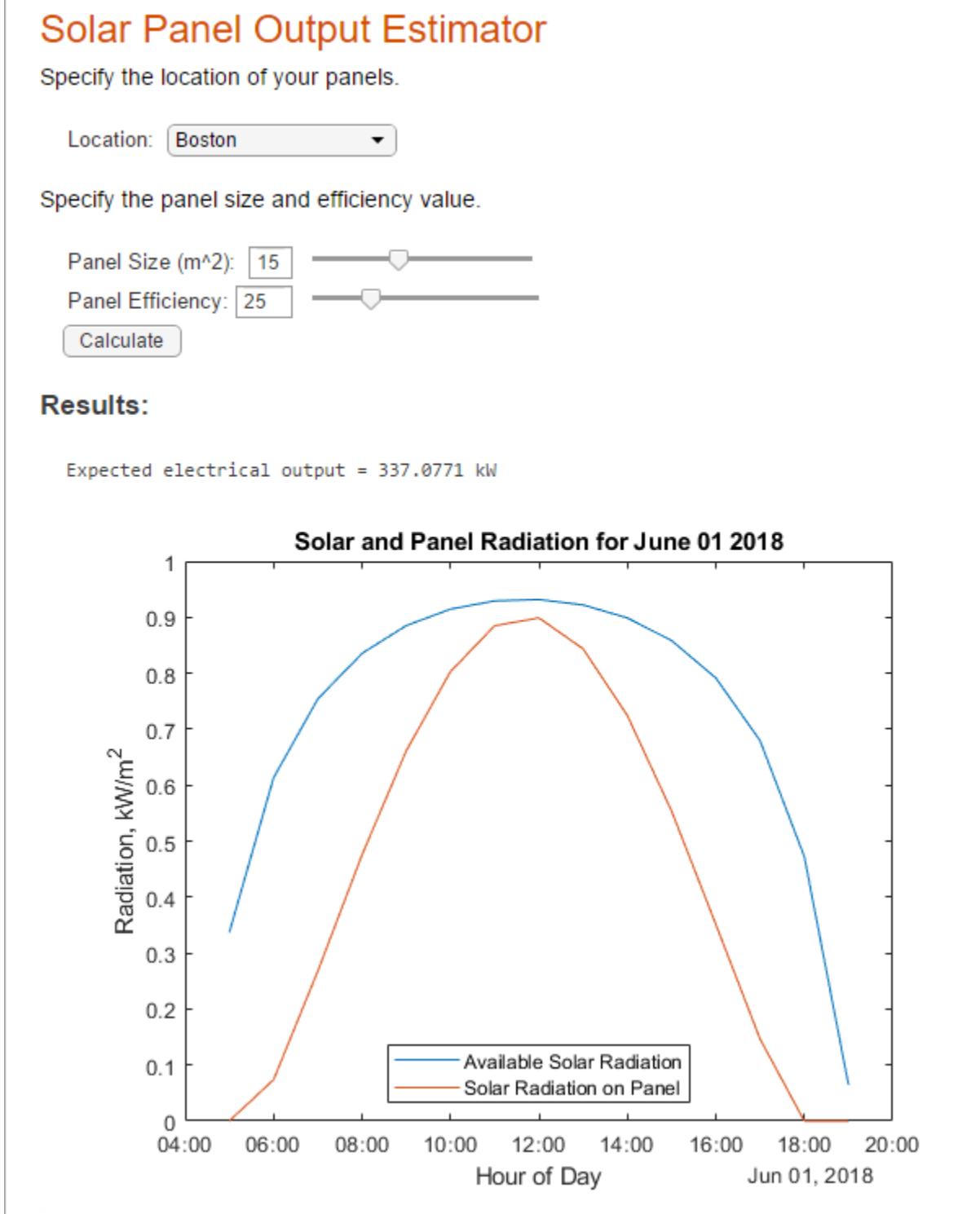
### 详细信息

- “在实时编辑器中创建实时脚本”（第 19-6 页）
- “为实时函数添加帮助”（第 19-54 页）
- “显示自定义文档”（第 31-20 页）
- MATLAB 实时脚本库

## 使用实时编辑器创建交互式表单

以下示例说明如何使用实时编辑器创建一个交互式表单，该表单根据用户提供的输入完成计算。该示例使用下拉菜单和滑块提示用户进行输入，然后使用按钮根据提供的输入运行计算。该示例隐藏了代码，只向用户显示控件和结果。

要查看示例并与之交互，请在 MATLAB® 中打开它。要在 MATLAB 中查看此示例的代码，请转至**视图**选项卡，然后点击**内嵌输出**或**右侧输出**。



另请参阅

详细信息

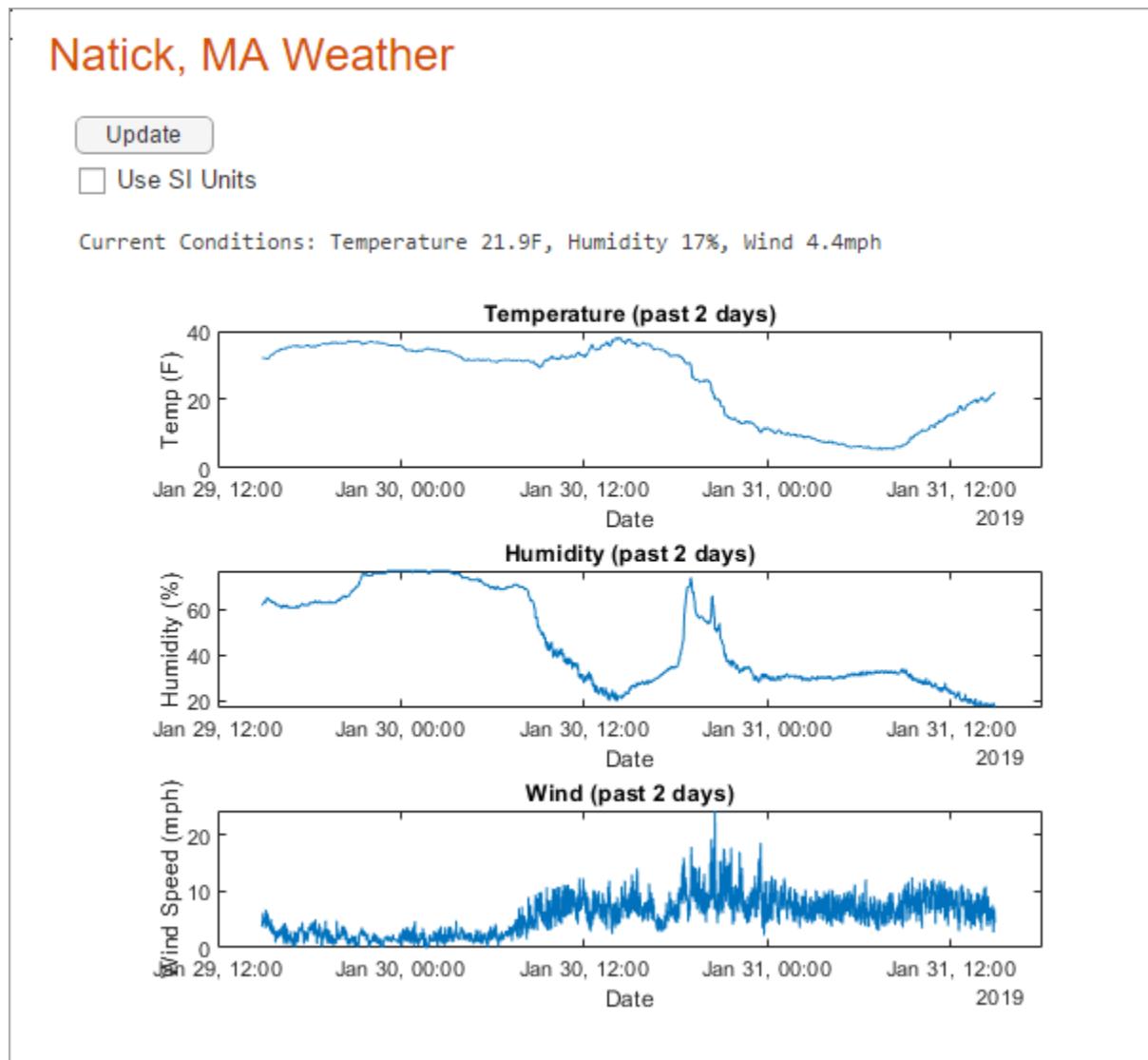
- “在实时编辑器中创建实时脚本” (第 19-6 页)

- MATLAB 实时脚本库

## 使用实时编辑器创建实时控制板

以下示例说明如何使用实时编辑器创建用于显示和分析实时数据的控制板。该示例使用一个按钮和一个复选框来按需获取和显示实时数据。该示例隐藏了代码，只向用户显示控件和结果。

要查看示例并与之交互，请在 MATLAB® 中打开它。要在 MATLAB 中查看此示例的代码，请转至**视图**选项卡，然后点击**内嵌输出**或**右侧输出**。



## 另请参阅

### 详细信息

- “在实时编辑器中创建实时脚本”（第 19-6 页）
- MATLAB 实时脚本库

# 函数基础知识

---

- “在文件中创建函数” (第 20-2 页)
- “为程序添加帮助” (第 20-5 页)
- “在编辑器中运行函数” (第 20-7 页)
- “基础和函数工作区” (第 20-8 页)
- “在工作区之间共享数据” (第 20-9 页)
- “在编辑器中检查变量作用域” (第 20-13 页)
- “函数类型” (第 20-16 页)
- “匿名函数” (第 20-19 页)
- “局部函数” (第 20-24 页)
- “嵌套函数” (第 20-26 页)
- “嵌套函数和匿名函数中的变量” (第 20-32 页)
- “私有函数” (第 20-33 页)
- “函数优先顺序” (第 20-34 页)
- “针对 R2019b 对函数优先顺序的更改更新代码” (第 20-36 页)
- “对函数调用结果进行索引” (第 20-41 页)

## 在文件中创建函数

脚本和函数都允许您通过将命令序列存储在程序文件中来重用它们。脚本是最简单的程序类型，因为它们存储命令的方式与您在命令行中键入命令完全相同。函数提供的灵活性更大，主要因为您可以传递输入值并返回输出值。例如，名为 `fact` 的以下函数用于计算某个数 (`n`) 的阶乘并返回结果 (`f`)。

```
function f = fact(n)
    f = prod(1:n);
end
```

此类型的函数必须在文件中而不是在命令行中定义。通常，您可以将函数存储在其自己的文件中。在这种情况下，最佳做法是对函数和文件使用相同的名称（此示例中为 `fact.m`），因为 MATLAB 将程序与文件名相关联。将文件保存在当前文件夹中，或者保存在 MATLAB 搜索路径上的某个文件夹中。

您可以使用应用于随 MATLAB 一起安装的函数的相同语法规则从命令行调用函数。例如，计算 5 的阶乘。

```
x = 5;
y = fact(5)

y =
120
```

从 R2016b 开始，用于存储函数的另一个选项是将函数包含在脚本文件的末尾。例如，创建一个名为 `mystats.m` 并包含一些命令和两个函数 `fact` 和 `perm` 的文件。该脚本用于计算  $(3,2)$  的置换。

```
x = 3;
y = 2;
z = perm(x,y)

function p = perm(n,r)
    p = fact(n)*fact(n-r);
end

function f = fact(n)
    f = prod(1:n);
end
```

从命令行调用该脚本。

```
mystats

z =
6
```

## 函数定义语法

每个函数的第一行为定义语句，其中包含以下元素。

<code>function</code> 关键字（必需）	对关键字使用小写字符。
-------------------------------	-------------

输出参数 (可选)	<p>如果您的函数返回一个输出，则您可以在 <code>function</code> 关键字后面指定输出名称。</p> <pre><code>function myOutput = myFunction(x)</code></pre> <p>如果您的函数返回多个输出，请将输出名称括在方括号中。</p> <pre><code>function [one,two,three] = myFunction(x)</code></pre> <p>如果没有输出，您可以将其忽略。</p> <pre><code>function myFunction(x)</code></pre> <p>您也可以使用空的方括号。</p> <pre><code>function [] = myFunction(x)</code></pre>
函数名称 (必需)	<p>有效的函数名字遵守与变量名称相同的规则。它们必须以字母开头，可以包含字母、数字或下划线。</p> <p><b>注意</b> 为避免混淆，对函数文件及函数文件内的第一个函数使用相同名称。MATLAB 将您的程序与文件名而不是函数名称相关联。脚本文件不能与文件中的函数具有相同的名字。</p>
输入参数 (可选)	<p>如果您的函数接受任何输入，请在函数名称之后将输入名称括在圆括号中。用逗号将各个输入隔开。</p> <pre><code>function y = myFunction(one,two,three)</code></pre> <p>如果没有输入，可以忽略圆括号。</p>

**提示** 定义带有多个输入或输出参数的函数时，首先列出任何必需的参数。该顺序允许您在不指定可选参数的情况下调用函数。

## 函数和文件的内容

函数主体可以包括有效的 MATLAB 表达式、控制流语句、注释、空白行和嵌套函数。您在函数内创建的任何变量都存储在特定于该函数的工作区内，该工作区独立于基础工作区。

程序文件可以包含多个函数。如果文件仅包含函数定义，则第一个函数是主函数，也是 MATLAB 与文件名关联的函数。主函数或脚本代码后面的函数称为局部函数。局部函数只能在文件内使用。

## End 语句

函数以 `end` 语句、文件末尾或局部函数的定义行结束，以先出现的为准。`end` 语句在以下情况下是必需的：

- 文件中的任何函数都包含嵌套函数（完全包含在其父级内的函数）。
- 该函数是函数文件中的局部函数，并且文件中有局部函数使用 `end` 关键字。
- 该函数是脚本文件内的局部函数。

虽然它有时是可选的，但使用 `end` 可提高代码可读性。

## 另请参阅

`function`

## 详细信息

- “MATLAB 可访问的文件和文件夹”
- “基础和函数工作区” (第 20-8 页)
- “函数类型” (第 20-16 页)
- “向脚本中添加函数” (第 18-13 页)

## 为程序添加帮助

此示例说明如何为您编写的程序提供帮助。您使用 **help** 函数时帮助文本显示在命令行窗口中。

通过在程序开始处插入注释来创建帮助文本。如果您的程序包含函数，请将帮助文本放在函数定义行（带有 **function** 关键字的行）的紧下方。

例如，在名称为 **addme.m** 并包含帮助文本的文件中创建函数：

```
function c = addme(a,b)
% ADDME Add two values together.
% C = ADDME(A) adds A to itself.
%
% C = ADDME(A,B) adds A and B together.
%
% See also SUM, PLUS.

switch nargin
    case 2
        c = a + b;
    case 1
        c = a + a;
    otherwise
        c = 0;
end
```

在命令行中键入 **help addme** 时，帮助文本显示在命令行窗口中：

```
addme Add two values together.
C = addme(A) adds A to itself.

C = addme(A,B) adds A and B together.

See also sum, plus.
```

第一个帮助文本行通常称为 H1 行，包括程序名称和简短说明。当前文件夹浏览器以及 **help** 和 **lookfor** 函数使用 H1 行显示有关程序的信息。

通过在帮助文本结尾处以 % See also 开始的行中纳入函数名称，可创建 See also 链接。如果函数存在于搜索路径或当前文件夹中，**help** 命令会将其中的每个函数名称显示为指向其帮助的超链接。否则，**help** 将按帮助文本中函数名称的原样输出这些名称。

您可以在帮助文本中包含指向网站的超链接（以 URL 的形式）。通过包含 HTML [<a></a>](#) 锚点元素创建超链接。在锚点内，使用 **matlab:** 语句执行 **web** 命令。例如：

```
% For more information, see <a href="matlab:
% web('https://www.mathworks.com')">the MathWorks Web site</a>.
```

以空白行（没有 %）结束您的帮助文本。帮助系统忽略出现在帮助文本块之后的任何注释行。

---

**注意** 如果多个程序的名称相同，**help** 命令通过应用“函数优先顺序”（第 20-34 页）中介绍的规则确定要显示的帮助文本。但是，如果程序的名称与 MathWorks 函数相同，上下文菜单中的**关于所选内容的帮助**选项始终都会显示有关 MathWorks 函数的文档。

## 另请参阅

`help` | `lookfor`

## 相关示例

- “向程序中添加注释” (第 18-3 页)
- “创建帮助摘要文件 - `Contents.m`” (第 31-10 页)
- “检查哪些程序具有帮助” (第 31-8 页)
- “显示自定义文档” (第 31-20 页)
- “`Use Help Files with MEX Functions`”

## 在编辑器中运行函数

本示例介绍在编辑器中执行操作时如何运行需要一些初始设置的函数，例如输入参数值。

- 1 在名称为 `myfunction.m` 的程序文件中创建函数。

```
function y = myfunction(x)
y = x.^2 + x;
```

该函数需要输入 `x`。

- 2 通过点击编辑器选项卡上的运行，可以查看用于运行函数的命令。当您点击运行图标时，列表顶端的命令是编辑器默认情况下使用的命令。



- 3 将键入要运行的代码替换为允许您运行函数的表达式。

```
y = myfunction(1:10)
```

您可以在同一行输入多条命令，例如

```
x = 1:10; y = myfunction(x)
```

对于更复杂的多行命令，可创建单独的脚本文件，然后运行脚本。

**注意** 运行使用基础工作区的命令。您在运行命令中定义的任何变量都可以覆盖同名基础工作区中的变量。

- 4 通过点击下拉列表中的运行或特定运行命令来运行函数。对于 `myfunction.m` 和 `1:10` 的输入，结果显示在命令行窗口中：

```
y =
     2    6   12   20   30   42   56   72   90  110
```

从列表中选择运行命令时，它变为运行按钮的默认设置。

要编辑或删除现有的运行命令，请选择命令，右键点击，然后选择编辑或删除。

## 基础和函数工作区

本主题介绍基础工作区与函数工作区的区别，包括局部函数、嵌套函数和脚本的工作区。

基础工作区存储您在命令行中创建的变量。这包括脚本创建的任何变量（假定您从命令行或编辑器中运行脚本）。在您清除基础工作区中的变量或结束您的 MATLAB 会话之前，这些变量一直存在。

函数不使用基础工作区。每个函数都有自己的函数工作区。每个函数工作区都与基础工作区和所有其他工作区分开以保护数据的完整性。即使普通文件中的局部函数也有它们自己的工作区。某函数工作区的特定变量称为局部变量。在一个函数调用转到下一个函数调用时，局部变量通常不保留在内存中。

从函数中调用脚本时，脚本使用函数工作区。

和局部函数一样，嵌套函数也有它们自己的工作区。但是，这些工作区从以下两个重要的独特之处：

- 嵌套函数可以访问和修改它们所在的函数工作区中的变量。
- 嵌套函数或包含嵌套函数的函数中的所有变量都必须显式定义。即，除非变量已存在于函数工作区中，否则无法调用函数或脚本向那些变量赋值。

### 另请参阅

#### 相关示例

- “在工作区之间共享数据”（第 20-9 页）

#### 详细信息

- “嵌套函数”（第 20-26 页）

# 在工作区之间共享数据

## 本节内容

- “简介” (第 20-9 页)
- “最佳做法：传递参数” (第 20-9 页)
- “嵌套函数” (第 20-9 页)
- “持久变量” (第 20-10 页)
- “全局变量” (第 20-10 页)
- “在另一工作区中计算” (第 20-11 页)

## 简介

本主题介绍如何在工作区之间共享变量或如何使它们在函数执行之间持久保留。

大多数情况下，在函数内创建的变量是仅可在该函数内识别的局部变量。局部变量不能用在命令行中，也不适用于任何其他函数。但是，可以通过多种方式在函数或工作区之间共享数据。

## 最佳做法：传递参数

扩大函数变量作用域的最安全的方式是使用函数输入和输出参数，这样您可以传递变量的值。

例如，创建两个函数 `update1` 和 `update2`，它们共享和修改输入值。`update2` 可以是文件 `update1.m` 中的局部函数，也可以是它自己的文件 `update2.m` 中的函数。

```
function y1 = update1(x1)
    y1 = 1 + update2(x1);

function y2 = update2(x2)
    y2 = 2 * x2;
```

从命令行调用 `update1` 函数并向基础工作区中的变量 `Y` 赋值：

```
X = [1,2,3];
Y = update1(X)
```

```
Y =
3 5 7
```

## 嵌套函数

嵌套函数可以访问其所在的所有函数的工作区。所以，例如嵌套函数可以使用在其父函数中定义的变量（在本例中为 `x`）：

```
function primaryFx
    x = 1;
    nestedFx

    function nestedFx
        x = x + 1;
    end
end
```

如果父函数不使用指定变量，变量保持为嵌套函数的局部变量。例如，在该版本的 **primaryFx** 中，以下两个嵌套函数拥有它们自己的不能彼此交互的 **x** 版本。

```
function primaryFx
    nestedFx1
    nestedFx2

    function nestedFx1
        x = 1;
    end

    function nestedFx2
        x = 2;
    end
end
```

有关详细信息，请参阅“[嵌套函数](#)”（第 20-26 页）。

## 持久变量

如果将函数内的变量声明为持久变量，则从一个函数调用转到下个函数调用时变量会保留其值。其他局部变量仅在当前函数执行期间保留它们的值。持久变量等效于其他编程语言中的静态变量。

要使用 **persistent** 关键字声明变量之后再使用它们。MATLAB 将持久变量初始化为空矩阵 []。

例如，在名为 **findSum.m** 的文件中定义一个函数，先将总和值初始化为 0，然后在每次迭代时与该值相加。

```
function findSum(inputvalue)
persistent SUM_X

if isempty(SUM_X)
    SUM_X = 0;
end
SUM_X = SUM_X + inputvalue;
```

调用该函数时，**SUM\_X** 的值在后续执行之间持久保留。

以下操作可清除函数的持久变量：

- **clear all**
- **clear functionname**
- 编辑函数文件

要避免清除持久变量，请使用 **mlock** 锁定函数文件。

## 全局变量

全局变量是您可以从函数或命令行中访问的变量。它们拥有自己的工作区，这些工作区与基础和函数工作区分开。

但是，全局变量具有显著风险。例如：

- 任何函数都可以访问和更新全局变量。使用此类变量的其他函数可能返回意外结果。

- 如果您无意间提供与现有全局变量同名的“新”全局变量，一个函数可能覆盖另一个函数预期的值。此类错误很难诊断。

请尽可能谨慎使用全局变量。

如果您使用全局变量，请使用 **global** 关键字声明它们，然后从任何特定位置（函数或命令行）访问它们。例如，在名为 **falling.m** 的文件中创建一个函数：

```
function h = falling(t)
    global GRAVITY
    h = 1/2*GRAVITY*t.^2;
```

然后在提示符下输入这些命令：

```
global GRAVITY
GRAVITY = 32;
y = falling((0:.1:5));
```

通过上述两条全局语句，可以在函数内使用在命令提示符下赋值给 **GRAVITY** 的值。但是，更为稳健的做法是，重新定义函数以接该值作为输入：

```
function h = falling(t,gravity)
    h = 1/2*gravity*t.^2;
```

然后在提示符下输入这些命令：

```
GRAVITY = 32;
y = falling((0:.1:5),GRAVITY);
```

## 在另一工作区中计算

通过 **evalin** 和 **assignin** 函数，您可以计算字符向量中的命令或变量名称并指定是否使用当前或基础工作区。

和全局变量一样，这些函数存在覆盖现有数据的风险。请谨慎使用它们。

**evalin** 和 **assignin** 有时可用于图形用户界面中的回调函数，以针对基础工作区计算。例如，从基础工作区创建变量名称的列表框：

```
function listBox
figure
lb = uicontrol('Style','listbox','Position',[10 10 100 100],...
    'Callback',@update_listBox);
update_listBox(lb)

function update_listBox(src,~)
vars = evalin('base','who');
src.String = vars;
```

对于其他编程应用程序，考虑使用“**eval** 函数的替代方法”（第 2-80 页）中介绍的参数传递和方法。

**另请参阅**

**详细信息**

- “基础和函数工作区” (第 20-8 页)

# 在编辑器中检查变量作用域

## 本节内容

[“使用自动函数和变量高亮显示功能” \(第 20-13 页\)](#)

[“使用自动函数和变量高亮显示功能的示例” \(第 20-13 页\)](#)

作用域问题可能是一些编码问题的来源。例如，如果您不知道嵌套函数共享特定变量，运行代码的结果可能和您预期的不同。同样，局部、全局和持久变量使用不当可能会导致意外结果。

代码分析器并不总会指出作用域问题，因为在函数间共享变量可能不是错误，而是您有意为之。使用 MATLAB 函数和变量高亮显示功能来确定您的代码使用函数和变量的时间和位置。如果可以连接到 Internet，可以观看“变量和函数突出显示”视频以了解主要功能的概要信息。

有关嵌套函数和各种类型的 MATLAB 变量的概念信息，请参阅“[在父函数与嵌套函数之间共享变量](#)”(第 20-26 页) 和“[在工作区之间共享数据](#)”(第 20-9 页)。

## 使用自动函数和变量高亮显示功能

默认情况下，编辑器以各种渐变蓝色显示函数、局部变量和带有共享作用域的变量。带有共享作用域的变量包括：全局变量 (第 20-10 页)、持久变量 (第 20-10 页) 以及嵌套函数内的变量。(有关详细信息，请参阅“[嵌套函数](#)”(第 20-9 页)。)

要启用和禁用突出显示功能或更改颜色，请点击  预设，然后选择 **MATLAB > 颜色 > 编程工具**。在 MATLAB Online 中，突出显示功能默认情况下处于启用状态，并且不能更改突出显示预设项。

默认情况下，编辑器的行为如下：

- 在您将光标放在函数或变量名称内时以天蓝色高亮显示特定函数或局部变量的所有实例。例如：

`collatz`

- 无论光标位置在哪儿，都以水鸭蓝显示带有共享作用域的变量。例如：

`x`

## 使用自动函数和变量高亮显示功能的示例

考虑对函数 `rowsum` 使用以下代码：

```
function rowTotals = rowsum
% Add the values in each row and
% store them in a new array

x = ones(2,10);
[n, m] = size(x);
rowTotals = zeros(1,n);
for i = 1:n
    rowTotals(i) = addToSum;
end

function colsum = addToSum
colsum = 0;
```

```

thisrow = x(i,:);
for i = 1:m
    colsum = colsum + thisrow(i);
end
end

end

```

运行该代码时，不会返回每行的值并显示以下结果：

```

ans =
10 10

MATLAB 会转而显示以下结果：

ans =
0 0 0 0 0 0 0 0 0 10

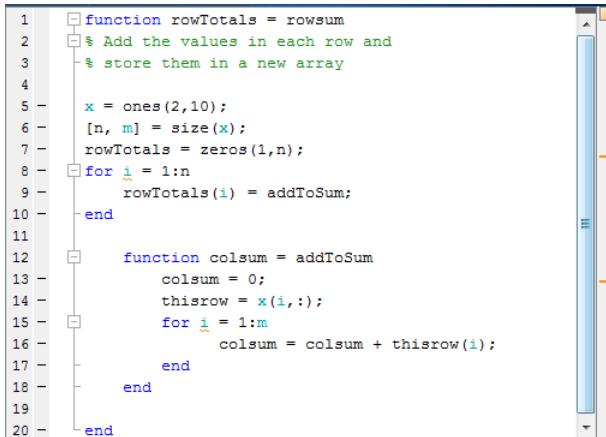
```

按照下列步骤检查该代码：

- 1 在主页选项卡上的环境部分中，点击  预设并选择 MATLAB > 颜色 > 编程工具。确保选择自动突出显示和具有共享范围的变量。
- 2 将 rowsum 代码复制到编辑器中。

请注意变量 `i` 以水鸭蓝显示，这表示 `i` 不是局部变量。`rowTotals` 函数和 `addToSum` 函数都设置和使用变量 `i`。

第 6 行的变量 `n` 显示为黑色，表示它未跨越多个函数。



```

1 function rowTotals = rowsum
2 % Add the values in each row and
3 % store them in a new array
4
5 x = ones(2,10);
6 [n, m] = size(x);
7 rowTotals = zeros(1,n);
8 for i = 1:n
9     rowTotals(i) = addToSum;
10 end
11
12 function colsum = addToSum
13     colsum = 0;
14     thisrow = x(i,:);
15     for i = 1:m
16         colsum = colsum + thisrow(i);
17     end
18 end
19
20 end

```

- 3 将鼠标指针悬停在变量 `i` 的实例上方。

随即出现工具提示：变量 “`i`” 的作用域跨越多个函数。

- 4 点击工具提示链接以了解其作用域跨越多个函数的变量相关信息。
- 5 点击 `i` 的实例。

对 `i` 的每次引用都会用天蓝色高亮显示，标记显示在编辑器右侧的指示标记条中。

```
1 function rowTotals = rowsum
2 % Add the values in each row and
3 % store them in a new array
4
5 x = ones(2,10);
6 [n, m] = size(x);
7 rowTotals = zeros(1,n);
8 for i = 1:n
9     rowTotals(i) = addToSum;
10 end
11
12 function colsum = addToSum
13     colsum = 0;
14     thisrow = x(i,:);
15     for j = 1:m
16         colsum = colsum + thisrow(j);
17     end
18 end
19
20 end
```

- 6 将鼠标光标悬停在某个指示标记条的标记上方。

随即出现工具提示，其中显示函数或变量名称以及用标记表示的代码行。

- 7 点击标记以导航至在该标记的工具提示中指定的行。

如果您的文件包含许多代码，您在编辑器中不能一次查看完，上述做法非常有用。

通过将第 15 行的 `i` 的实例更改为 `y` 来修复代码。

在函数引用上点击时，您可以看到类似的高亮显示效果。例如，在 `addToSum` 上点击。

## 函数类型

### 本节内容

- “文件中的局部和嵌套函数”（第 20-16 页）
- “子文件夹中的私有函数”（第 20-17 页）
- “无需文件的匿名函数”（第 20-17 页）

### 文件中的局部和嵌套函数

程序文件可以包含多个函数。局部和嵌套函数可用于将程序分为更小的任务，使读取和维护代码变得更容易。

局部函数是在同一文件中可用的子例程。局部函数是拆分编程任务的最常见方法。在仅包含函数定义的函数文件中，局部函数可以任意顺序出现在文件中主函数的后面。在包含命令和函数定义的脚本文件中，局部函数必须位于文件末尾。（R2016b 或更高版本支持脚本中的函数。）

例如，创建一个名为 `myfunction.m` 的函数文件，其中包含主函数 `myfunction` 以及两个局部函数 `squareMe` 和 `doubleMe`：

```
function b = myfunction(a)
    b = squareMe(a)+doubleMe(a);
end
function y = squareMe(x)
    y = x.^2;
end
function y = doubleMe(x)
    y = x.*2;
end
```

可以从命令行或另一程序文件中调用主函数，但局部函数仅适用于 `myfunction`：

```
myfunction(pi)
```

```
ans =
16.1528
```

嵌套函数完全包含在另一函数内。嵌套函数与局部函数的主要区别是，嵌套函数可以使用在父函数内定义的变量，无需将这些变量作为参数显式传递。

子例程共享数据（例如在组件间传递数据的应用程序）时，嵌套函数很有用。例如，创建一个函数，该函数允许您使用滑块或可编辑的文本框设置介于 0 与 1 之间的一个值。如果您将嵌套函数用于回调，滑块和文本框可以共享值和彼此的句柄，无需显式传递它们：

```
function myslider
value = 0;
f = figure;
s = uicontrol(f,'Style','slider','Callback',@slider);
e = uicontrol(f,'Style','edit','Callback',@edittext,...
    'Position',[100,20,100,20]);

function slider(obj,~)
    value = obj.Value;
    e.String = num2str(value);
end
```

```
function edittext(obj,~)
    value = str2double(obj.String);
    s.Value = value;
end

end
```

## 子文件夹中的私有函数

与局部或嵌套函数一样，私有函数仅供特定位置的函数访问。但是，私有函数与可以调用它们的函数不在同一个文件中。它们位于名称为 **private** 的子文件夹中。仅 **private** 文件夹紧邻的上一级文件夹内的函数可使用私有函数。使用私有函数将代码分割为不同的文件，或在多个相关函数间共享代码。

## 无需文件的匿名函数

只要函数包含一个语句，匿名函数即允许您定义该函数而不必创建程序文件。匿名函数通常用于定义数学表达式，然后使用 MATLAB® 功能函数（即接受函数句柄用作输入的函数）基于某个值范围计算该表达式。

例如，以下语句为匿名函数创建名称为 **s** 的函数句柄：

```
s = @(x) sin(1./x);
```

该函数具有一个输入 **x**。@ 运算符创建函数句柄。

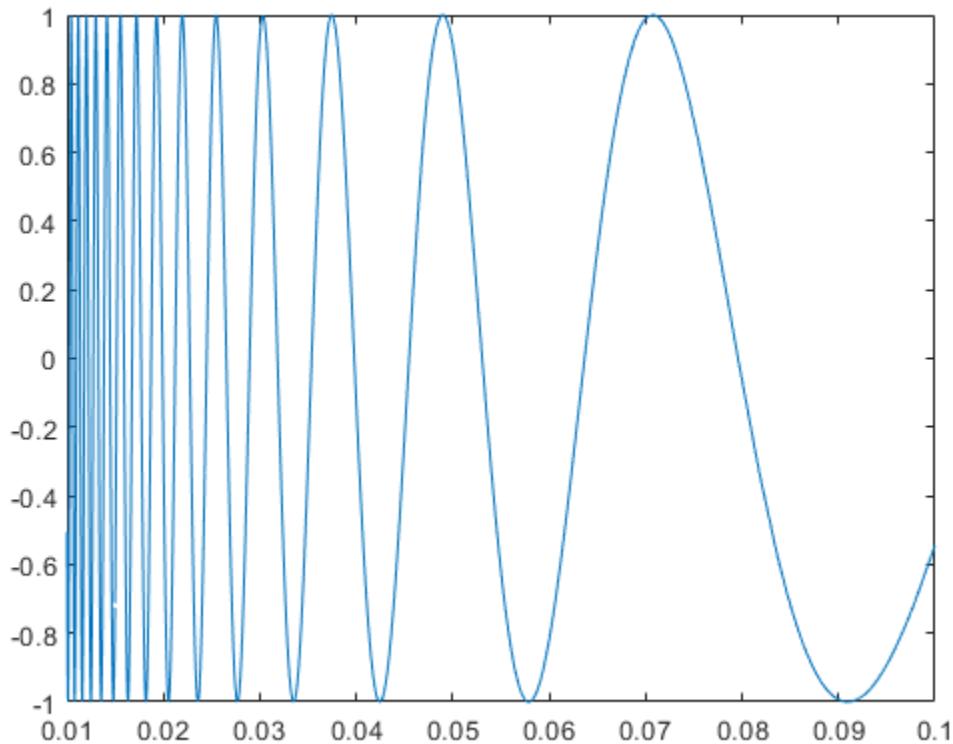
您可以使用该函数句柄针对特定值计算函数，例如

```
y = s(pi)
```

```
y = 0.3130
```

您也可以将该函数句柄传递给在某一值范围内计算的函数，例如 **fplot**：

```
range = [0.01,0.1];
fplot(s,range)
```



## 另请参阅

### 详细信息

- “局部函数” (第 20-24 页)
- “嵌套函数” (第 20-26 页)
- “私有函数” (第 20-33 页)
- “匿名函数” (第 20-19 页)

# 匿名函数

## 本节内容

- “什么是匿名函数？”（第 20-19 页）
- “表达式中的变量”（第 20-19 页）
- “多个匿名函数”（第 20-20 页）
- “不带输入的函数”（第 20-21 页）
- “带有多个输入或输出的函数”（第 20-21 页）
- “匿名函数的数组”（第 20-22 页）

## 什么是匿名函数？

匿名函数是不存储在程序文件中、但与数据类型是 **function\_handle** 的变量相关的函数。匿名函数可以接受输入并返回输出，就像标准函数一样。但是，它们可能只包含一个可执行语句。

例如，创建用于计算平方数的匿名函数的句柄：

```
sqr = @(x) x.^2;
```

变量 **sqr** 是一个函数句柄。**@** 运算符创建句柄，**@** 运算符后面的圆括号 () 包括函数的输入参数。该匿名函数接受单个输入 **x**，并显式返回单个输出，即大小与包含平方值的 **x** 相同的数组。

通过将特定值 (5) 传递到函数句柄来计算该值的平方，与您将输入参数传递到标准函数一样。

```
a = sqr(5)
```

```
a =
25
```

许多 MATLAB 函数接受将函数句柄用作输入，这样您可以在特定值范围内计算函数。您可以为匿名函数或程序文件中的函数创建句柄。使用匿名函数的好处是不必为仅需要简短定义的函数编辑和维护文件。

例如，通过将函数句柄传递到 **integral** 函数，计算 **sqr** 函数从 0 到 1 范围内的积分：

```
q = integral(sqr,0,1);
```

您无需在工作区中创建变量以存储匿名函数。可以在表达式内创建临时函数句柄，例如这次对 **integral** 函数的调用：

```
q = integral(@(x) x.^2,0,1);
```

## 表达式中的变量

函数句柄不仅可以存储表达式，还能存储表达式进行计算需要的变量。

例如，为需要系数 **a**、**b** 和 **c** 的匿名函数创建函数句柄。

```
a = 1.3;
b = .2;
c = 30;
parabola = @(x) a*x.^2 + b*x + c;
```

由于 **a**、**b** 和 **c** 在您创建 **parabola** 时可用，该函数句柄包含这些值。即使您清除变量，这些值仍持久保留在函数句柄内：

```
clear a b c
x = 1;
y = parabola(x)

y =
31.5000
```

要为这些系数提供不同值，您必须创建新的函数句柄：

```
a = -3.9;
b = 52;
c = 0;
parabola = @(x) a*x.^2 + b*x + c;

x = 1;
y = parabola(1)

y =
48.1000
```

可以将函数句柄及其相关值存储在 MAT 文件中，然后使用 **save** 和 **load** 函数在后续的 MATLAB 会话中加载它们，例如

```
save myfile.mat parabola
```

在构造匿名函数时仅使用显式变量。如果匿名函数访问未在参数列表或主体中显式引用的任何变量或嵌套函数，则 MATLAB 会在您调用该函数时引发错误。隐式变量和函数调用通常会在 **eval**、**evalin**、**assignin** 和 **load** 等函数中遇到。请避免在匿名函数主体中使用这些函数。

## 多个匿名函数

匿名函数中的表达式可以包含其他匿名函数。这可用于将不同的参数传递到在某一值范围内计算的函数。例如，您可以针对不同的

$$g(c) = \int_0^1 (x^2 + cx + 1) dx$$

**c** 值求解以下方程，方法是合并使用两个匿名函数：

```
g = @(c) (integral(@(x) (x.^2 + c*x + 1),0,1));
```

下面介绍得出该语句的步骤：

- 1 将被积函数编写为匿名函数，  
 $\text{@}(x) (x.^2 + c*x + 1)$
- 2 通过将函数句柄传递到 **integral** 在从 0 到 1 的范围内计算函数，  
 $\text{integral}(\text{@}(x) (x.^2 + c*x + 1),0,1)$
- 3 通过为整个方程构造匿名函数以提供 **c** 的值，  
 $\text{g} = \text{@}(c) (\text{integral}(\text{@}(x) (x.^2 + c*x + 1),0,1));$

最终的函数可以针对任何 `c` 值来求解方程。例如：

```
g(2)

ans =
2.3333
```

## 不带输入的函数

如果您的函数不需要任何输入，请在定义和调用匿名函数时输入空的圆括号。例如：

```
t = @( ) datestr(now);
d = t()

d =
26-Jan-2012 15:11:47
```

在赋值语句中省略圆括号会创建另一函数句柄，并且不执行函数：

```
d = t

d =
@( ) datestr(now)
```

## 带有多个输入或输出的函数

匿名函数需要您像对标准函数一样显式指定输入参数，用逗号隔开多个输入。例如，以下函数接受两个输入 `x` 和 `y`：

```
myfunction = @(x,y) (x^2 + y^2 + x*y);

x = 1;
y = 10;
z = myfunction(x,y)

z = 111
```

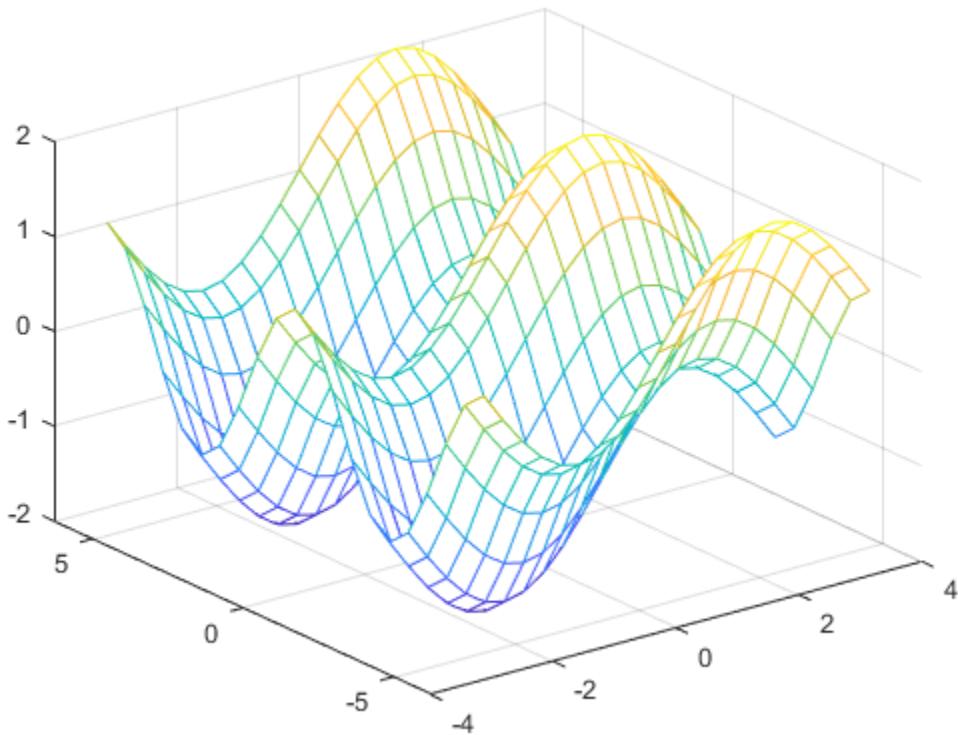
但是，您不用在创建匿名函数时显式定义输出参数。如果函数中的表达式返回多个输出，您可以在调用该函数时请求它们。将多个输出变量括在方括号中。

例如，`ndgrid` 函数可以返回与输入向量数量一样多的输出。调用 `ndgrid` 的这个匿名函数还可以返回多个输出：

```
c = 10;
mygrid = @(x,y) ndgrid((-x:x/c:x),(-y:y/c:y));
[x,y] = mygrid(pi,2*pi);
```

您可以使用来自 `mygrid` 的输出创建网格图或曲面图：

```
z = sin(x) + cos(y);
mesh(x,y,z)
```



## 匿名函数的数组

虽然大多数的 MATLAB 基本数据类型支持多维数组，但函数句柄必须是标量（单个元素）。但您可以使用元胞数组或结构体数组存储多个函数句柄。最常见的方式是使用元胞数组，例如

```
f = {@(x)x.^2;
 @(y)y+10;
 @(x,y)x.^2+y+10};
```

创建元胞数组时，记住 MATLAB 将空格解释为列分隔符。如上面的代码所示，省略表达式中的空格，或将表达式括在圆括号中，例如

```
f = {@(x) (x.^2);
 @(y) (y + 10);
 @(x,y) (x.^2 + y + 10)};
```

使用花括号访问元胞内容。例如，`f{1}` 返回第一个函数句柄。要执行该函数，请在花括号之后的圆括号中传递输入值：

```
x = 1;
y = 10;

f{1}(x)
f{2}(y)
f{3}(x,y)
```

```
ans =  
1
```

```
ans =  
20
```

```
ans =  
21
```

## 另请参阅

### 详细信息

- “创建函数句柄” (第 13-2 页)

## 局部函数

本主题介绍术语局部函数并说明如何创建和使用局部函数。

MATLAB 程序文件可以包含用于多个函数的代码。在函数文件中，第一个函数称为主函数。此函数对其他文件中的函数可见，或者您也可以从命令行调用它。文件中的其他函数称为局部函数，它们可以任意顺序出现在主函数后面。局部函数仅对同一文件中的其他函数可见。它们等效于其他编程语言的子例程，有时被称为子函数。

自 R2016b 起，您也可以在脚本文件中创建局部函数，只要这些函数都出现在脚本代码的最后一行的后面。有关详细信息，请参阅“向脚本中添加函数”（第 18-13 页）。

例如，创建一个名为 `mystats.m` 的函数文件，其中包含主函数 `mystats` 以及两个局部函数 `mymean` 和 `mymedian`。

```
function [avg, med] = mystats(x)
n = length(x);
avg = mymean(x,n);
med = mymedian(x,n);
end

function a = mymean(v,n)
% MYMEAN Example of a local function.

a = sum(v)/n;
end

function m = mymedian(v,n)
% MYMEDIAN Another example of a local function.

w = sort(v);
if rem(n,2) == 1
    m = w((n + 1)/2);
else
    m = (w(n/2) + w(n/2 + 1))/2;
end
end
```

局部函数 `mymean` 和 `mymedian` 计算输入列表的平均值和中位数。主函数 `mystats` 决定着列表 `n` 的长度并将其传递到局部函数。

虽然您不能从命令行或其他文件中的函数调用局部函数，但您可以使用 `help` 函数访问其帮助。同时指定文件和局部函数的名称，用 `>` 字符将它们隔开：

```
help mystats>mymean

mymean Example of a local function.
```

当前文件中的局部函数优先于其他文件中的函数。即，当您在程序文件内调用函数时，MATLAB 在查找其他主函数前检查该函数是否为局部函数。因此，您可以在创建特定函数的备用版本的同时，将原始版本保留在另一文件中。

包括局部函数在内的所有函数都有与基础工作区分开的专属工作区。局部函数不能访问其他函数使用的变量，除非您将这些变量作为参数传递。与此相反，嵌套函数（完全包含在另一函数内的函数）可以访问包含它们的函数使用的变量。

**另请参阅**  
localfunctions

### 详细信息

- “嵌套函数” (第 20-26 页)
- “函数优先顺序” (第 20-34 页)

## 嵌套函数

### 本节内容

- “什么是嵌套函数？”（第 20-26 页）
- “嵌套函数的要求”（第 20-26 页）
- “在父函数与嵌套函数之间共享变量”（第 20-26 页）
- “使用句柄存储函数参数”（第 20-27 页）
- “嵌套函数的可见性”（第 20-30 页）

### 什么是嵌套函数？

嵌套函数是完全包含在父函数内的函数。程序文件中的任何函数都可以包含嵌套函数。

例如，名称为 **parent** 的函数包含名称为 **nestedfx** 的嵌套函数：

```
function parent
    disp('This is the parent function')
    nestedfx

    function nestedfx
        disp('This is the nested function')
    end

end
```

嵌套函数与其他类型的函数的主要区别是，嵌套函数可以访问和修改在其父函数中定义的变量。因此：

- 嵌套函数可以使用不是以输入参数形式显式传递的变量。
- 在父函数中，您可以为嵌套函数创建包含运行嵌套函数所必需的数据的句柄。

### 嵌套函数的要求

- 嵌套函数通常不需要 **end** 语句。但是，要在程序文件中嵌套任何函数，该文件中的所有函数都必须使用 **end** 语句。
- 不能在任何 MATLAB 程序控制语句内定义嵌套函数，例如 **if/elseif/else**、**switch/case**、**for**、**while** 或 **try/catch**。
- 必须按名称直接调用嵌套函数，而不使用 **feval** 或使用您使用 @ 运算符创建的函数句柄（并不是 **str2func**）。
- 嵌套函数或包含嵌套函数的函数中的所有变量都必须显式定义。即，除非变量已存在于函数工作区中，否则无法调用函数或脚本向那些变量赋值。（有关详细信息，请参阅“[嵌套函数和匿名函数中的变量](#)”（第 20-32 页）。）

### 在父函数与嵌套函数之间共享变量

通常，一个函数工作区中的变量不可用于其他函数。但是，嵌套函数可以访问和修改它们所在的函数工作区中的变量。

这意味着嵌套函数及包含它的函数都可以修改同一变量，不必将该变量作为参数传递。例如，在 **main1** 和 **main2** 函数的每个函数中，主函数和嵌套函数都可以访问变量 **x**：

```

function main1
x = 5;
nestfun1

function nestfun1
    x = x + 1;
end

end

function main2
nestfun2

function nestfun2
    x = 5;
end

    x = x + 1;
end

```

如果父函数不使用指定变量，变量保持为嵌套函数的局部变量。例如，在这个名为 **main** 的函数中，以下两个嵌套函数拥有它们自己的不能彼此交互的 **x** 版本：

```

function main
nestedfun1
nestedfun2

function nestedfun1
    x = 1;
end

function nestedfun2
    x = 2;
end
end

```

返回输出参数的函数在其工作区中有这些输出对应的变量。但是，父函数只有在显示请求嵌套函数的输出对应的变量时，父函数才具有这些变量。例如，函数 **parentfun** 在其工作区中没有变量 **y**：

```

function parentfun
x = 5;
nestfun;

function y = nestfun
    y = x + 1;
end

end

```

如果您修改如下所示的代码，变量 **z** 位于 **parentfun** 的工作区中：

```

function parentfun
x = 5;
z = nestfun;

function y = nestfun
    y = x + 1;
end

end

```

## 使用句柄存储函数参数

嵌套函数可以使用以下三个来源的变量：

- 输入参数

- 在嵌套函数内定义的变量
- 在父函数中定义的变量，还称为外部作用域变量。

为嵌套函数创建函数句柄时，该句柄不仅存储函数名称，还存储外部作用域变量的值。

例如，在名为 `makeParabola.m` 的文件中创建一个函数：该函数接受几个多项式系数，并返回嵌套函数的句柄以计算该多项式的值。

```
function p = makeParabola(a,b,c)
p = @parabola;

function y = parabola(x)
y = a*x.^2 + b*x + c;
end

end
```

`makeParabola` 函数返回 `parabola` 函数的句柄，该函数包含系数 `a`、`b` 和 `c` 的值。

在命令行中，调用具有系数值 1.3、.2 和 30 的 `makeParabola` 函数。使用返回的函数句柄 `p` 计算多项式在特定点处的值：

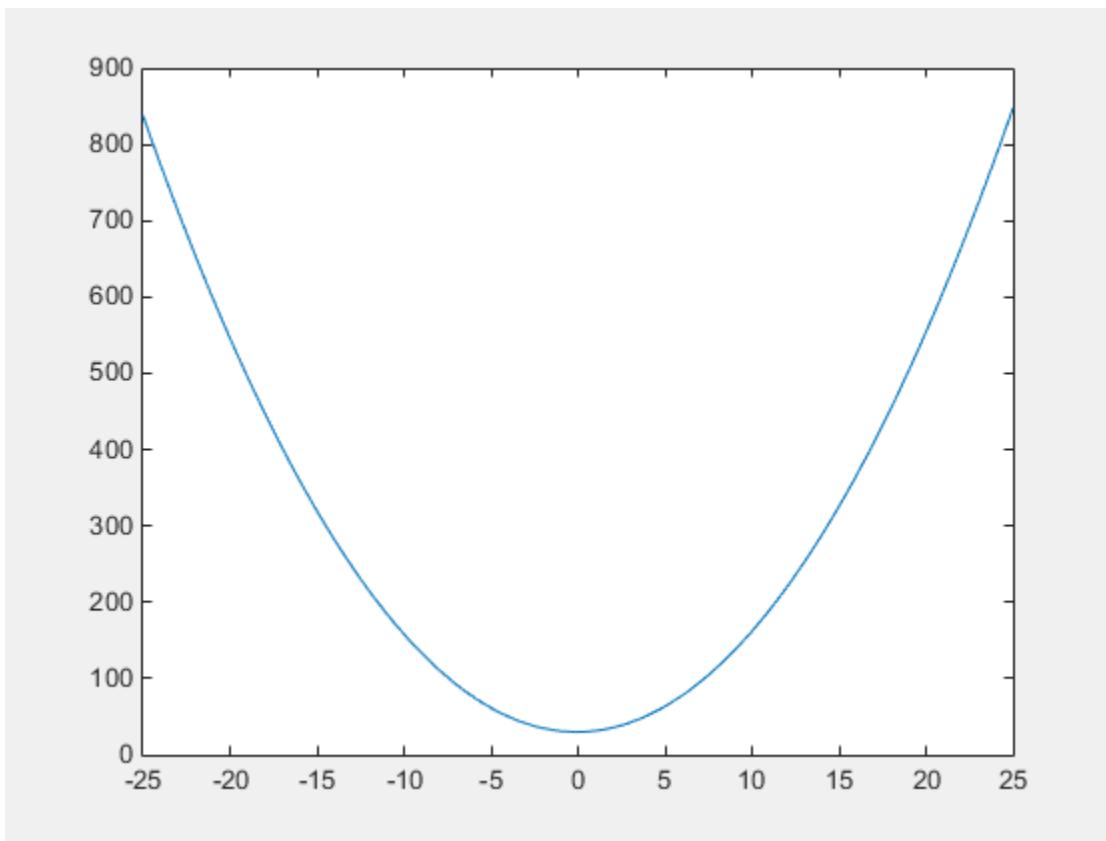
```
p = makeParabola(1.3,.2,30);

X = 25;
Y = p(X)

Y =
847.5000
```

许多 MATLAB 函数接受函数句柄输入以计算特定值范围内的函数。例如，绘制从 -25 到 +25 的抛物型方程：

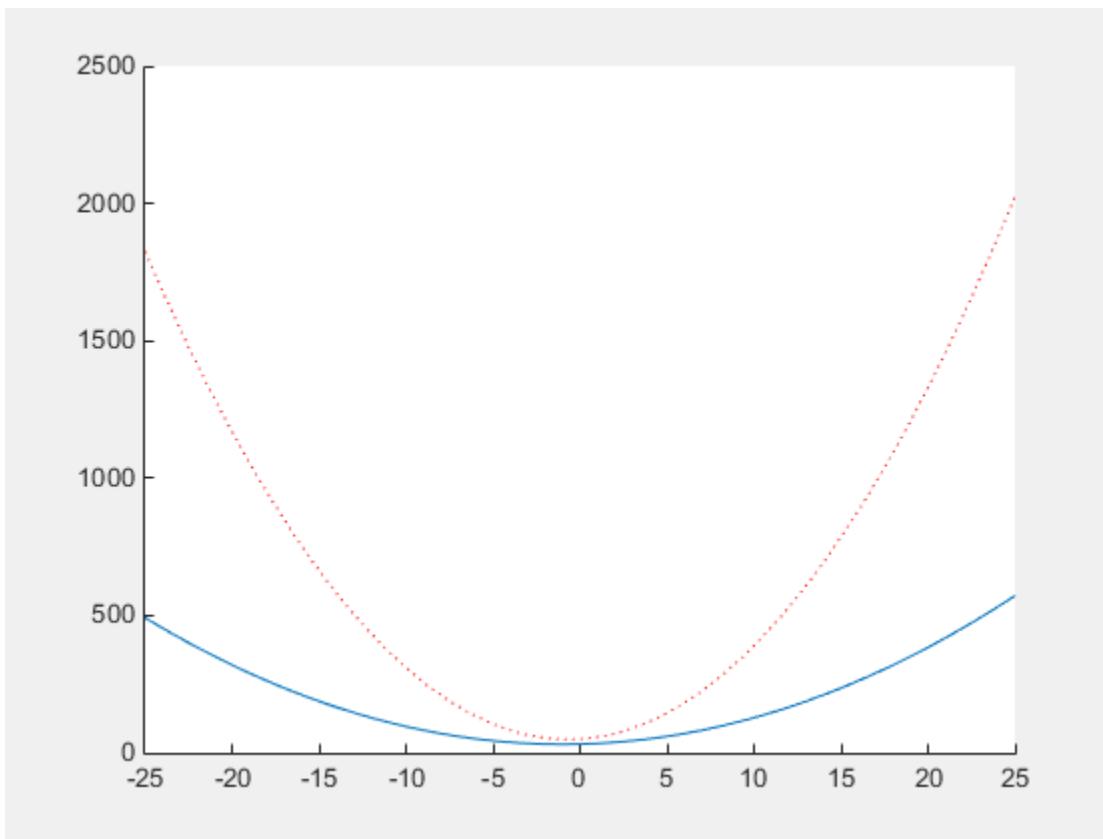
```
fplot(p,[-25,25])
```



可以为 `parabola` 函数创建多个句柄，其中每个句柄使用不同的多项式系数：

```
firstp = makeParabola(0.8,1.6,32);
secondp = makeParabola(3,4,50);
range = [-25,25];

figure
hold on
fplot(firstp,range)
fplot(secondp,range,'r:')
hold off
```



## 嵌套函数的可见性

每个函数都有特定的作用域，即可以看到该函数的一组其他函数。嵌套函数在以下位置可用：

- 紧邻的上一级。（在下面的代码中，函数 A 可以调用 B 或 D，但不能调用 C 或 E。）
- 在同一父函数内的同级别嵌套的函数中。（函数 B 可以调用 D，而 D 可以调用 B。）
- 任何更低级别的函数中。（函数 C 可以调用 B 或 D，但不能调用 E。）

```

function A(x, y)      % Main function
B(x,y)
D(y)

function B(x,y)      % Nested in A
C(x)
D(y)

function C(x)        % Nested in B
D(x)
end
end

function D(x)        % Nested in A
E(x)

function E(x)        % Nested in D
disp(x)

```

```
    end  
end  
end
```

扩大嵌套函数作用域的最简单的方式是创建函数句柄并将其作为输出参数返回，如“使用句柄存储函数参数”（第 20-27 页）中所示。只有能调用嵌套函数的函数才可以创建该嵌套函数的句柄。

## 另请参阅

### 详细信息

- “嵌套函数和匿名函数中的变量”（第 20-32 页）
- “创建函数句柄”（第 13-2 页）
- “嵌套函数中的参数检查”（第 21-8 页）

## 嵌套函数和匿名函数中的变量

嵌套函数和匿名函数的作用域规则要求在函数内使用的所有变量都存在于代码文本中。

如果您尝试向匿名函数、嵌套函数或包含嵌套函数的函数的工作区中动态添加变量，MATLAB 将引发以下形式的错误

**Attempt to add variable to a static workspace.**

下表介绍了尝试动态赋值的典型运算以及推荐的避免动态赋值的方法。

运算类型	避免动态赋值的最佳做法
<code>load</code>	指定变量名称作为 <code>load</code> 函数的输入。或者，对从 <code>load</code> 函数到结构体数组的输出赋值。
<code>eval</code> 、 <code>evalin</code> 或 <code>assignin</code>	如果可能，避免同时使用这些函数。请参阅“ <code>eval</code> 函数的替代方法”（第 2-80 页）。
调用 MATLAB 脚本创建变量	将脚本转换为函数并使用参数传递变量。这种方法还阐明了相应代码。
向 MATLAB 调试器中的变量赋值	在基础工作区中对变量赋值，例如 <code>K&gt;&gt; assignin('base','X',myvalue)</code>

避免动态赋值的另一种方法是显式声明函数内的变量。例如，假定名称为 `makeX.m` 的脚本向变量 `X` 赋值。使用函数调用 `makeX` 并显式声明 `X` 可避免动态赋值错误，因为 `X` 位于函数工作区内。常用的变量声明方法是将变量值初始化为一个空数组：

```
function noerror
X = [];
nestedfx

    function nestedfx
        makeX
    end
end
```

## 另请参阅

### 详细信息

- “基础和函数工作区”（第 20-8 页）

# 私有函数

本主题介绍术语私有函数并说明如何创建和使用私有函数。

私有函数在您希望限制函数的作用域时很有用。可以通过将函数存储在名称为 **private** 的子文件夹中，来将该函数指定为私有函数。这样，仅 **private** 子文件夹紧邻的上一级文件夹中的函数可使用该函数，也可由父文件夹中的函数调用的脚本使用。

例如，在位于 MATLAB 搜索路径下的文件夹内，创建名称为 **private** 的子文件夹。不要将 **private** 添加到该路径。在 **private** 文件夹内的名称为 **findme.m** 的文件中，创建一个函数：

```
function findme
% FINDME An example of a private function.

disp('You found the private function.')
```

更改为包含 **private** 文件夹的文件夹并创建一个名称为 **visible.m** 的文件。

```
function visible
findme
```

将当前文件夹更改为任意位置并调用 **visible** 函数。

```
visible

You found the private function.
```

虽然您不能从命令行或 **private** 文件夹父级外部的函数中调用私有函数，但您可以访问它的帮助：

```
help private/findme

findme An example of a private function.
```

私有函数优先于标准函数，因此 MATLAB 先查找名称为 **test.m** 的私有函数，再查找名称为 **test.m** 的非私有程序文件。这样您可以在创建特定函数的备用版本的同时，将原始版本保留在另一文件夹中。

## 另请参阅

### 详细信息

- “[函数优先顺序](#)”（第 20-34 页）

## 函数优先顺序

本主题介绍 MATLAB 在当前作用域内的多个函数具有相同名称时如何确定要调用的函数。当前作用域包括当前文件、相对于当前运行的函数的可选私有子文件夹、当前文件夹以及 MATLAB 路径。

MATLAB 使用下面的优先顺序：

### 1 变量

在认定名称与函数匹配之前，MATLAB 会先在当前工作区中检查具有该名称的变量。

**注意** 如果您创建与函数同名的变量，MATLAB 不能运行该函数，直到您从内存中清除该变量。

### 2 名称与显式导入的名称匹配的函数或类

对于具有复合名称（即名称包含多个部分、各部分以点相连）的函数，`import` 函数允许仅使用复合名称的最后一部分调用该函数。当函数名称与显式（非基于通配符）导入的函数匹配时，MATLAB 使用导入的复合名称，并使其优先于同名的所有其他函数。

### 3 当前函数内的嵌套函数

### 4 当前文件内的局部函数

### 5 名称与基于通配符导入的名称匹配的函数或类

当函数名称与基于通配符导入的函数匹配时，MATLAB 会使用导入的复合名称，并使其优先于同名的所有其他函数，但嵌套函数和局部函数除外。

### 6 私有函数

私有函数是名称为 `private` 的子文件夹（即当前运行的文件所在的文件夹正下方的文件夹）中的函数。

### 7 对象函数

对象函数以输入参数列表形式接受特定类的对象。如果存在多个同名的对象函数，MATLAB 检查输入参数的类以确定要使用的函数。

### 8 @ 文件夹中的类构造函数

MATLAB 使用类构造函数创建各种对象（例如 `timeseries` 或 `audioplayer`），您也可以使用面向对象的编程定义自己的类。例如，如果您创建类文件夹 `@polynom` 和构造函数 `@polynom/polynom.m`，构造函数优先于路径中任何位置的名为 `polynom.m` 的其他函数。

### 9 加载的 Simulink® 模型

### 10 当前文件夹中的函数

### 11 路径中其他位置的函数，按照显示顺序

在同一文件夹内确定函数优先级时，MATLAB 按以下顺序考虑文件类型：

#### 1 内置函数

#### 2 MEX 函数

#### 3 未加载的 Simulink 模型文件，文件类型的顺序如下：

a SLX 文件

b MDL 文件

- 4 使用 MATLAB App 设计工具创建的 App 文件 (.mlapp)
- 5 带有 .mlx 扩展名的程序文件
- 6 P 文件 (即带有 .p 扩展名的编码程序文件)
- 7 带有 .m 扩展名的程序文件

例如，如果 MATLAB 在同一文件夹中找到同名的 .m 文件和 P 文件，它使用 P 文件。因为 P 文件不会自动重新生成，所以确保您在编辑 P 文件时重新生成该文件。

要确定 MATLAB 对特定输入所调用的函数，请在对 `which` 函数的调用中包括函数名称和输入。

## 函数优先顺序规则的更改

从 R2019b 开始，MATLAB 更改了名称解析规则，这会影响变量、嵌套函数、局部函数和外部函数的优先顺序。有关这些更改和相应的代码更新提示的信息，请参阅“针对 R2019b 对函数优先顺序的更改更新代码”（第 20-36 页）。

- 标识符在一个函数内只能用于一个目的。
- 没有显式声明的标识符可能不被视为变量
- 变量无法在父函数和嵌套函数之间隐式共享
- 复合名称解析优先级的更改
- 匿名函数可以同时包含已解析和未解析的标识符

`import` 函数的行为已更改。

- 基于通配符导入的优先级发生更改
- 完全限定的导入函数不能与嵌套函数同名
- 完全限定的导入函数会遮蔽同名的外层作用域定义
- 未发现导入时的错误处理
- 嵌套函数从父函数继承 `import` 语句

## 另请参阅

`import`

## 详细信息

- “什么是 MATLAB 搜索路径？”
- 变量（第 1-5 页）
- “函数类型”（第 20-16 页）
- “类优先级和 MATLAB 路径”

## 针对 R2019b 对函数优先顺序的更改更新代码

从 R2019b 开始，MATLAB 更改了名称解析规则，对变量、嵌套函数、局部函数和外部函数的优先顺序有所影响。新规则对名称解析进行了简化和标准化。有关详细信息，请参阅“[函数优先顺序](#)”（第 20-34 页）。

这些更改会影响 `import` 函数的行为。您应分析并根据需要相应地更新您的代码。首先，请在您的代码中搜索 `import` 语句。例如，使用“[查找文件和文件夹](#)”搜索包含文本 `import` 的 .m 和 .mlx 文件。在评估以下更改的影响时，请参考这些搜索结果。

### 标识符在一个函数内只能用于一个目的。

从 R2019b 开始，如果将一个标识符先用作局部或导入函数，再用作变量，会导致错误。在以前的版本中，同一标识符可以在某个函数的作用域内用于多个不同目的，这会导致代码具有多义性。

如果此行为更改影响了您的代码，请重命名变量或函数，使它们具有不同名称。

从 R2019b 开始	更新后的代码	R2019a 和更早版本
名称 <code>local</code> 先用作 <code>local</code> 函数，再用作变量。这段代码出错。  <pre>function myfunc % local is an undefined variable local(1); % Errors local = 2; disp(local); end</pre> <pre>function local(x) disp(x) end</pre>	将函数 <code>local</code> 重命名为 <code>localFcn</code> 。  <pre>function myfunc localFcn(1); local = 2; disp(local); end</pre> <pre>function localFcn(x) disp(x) end</pre>	以下代码先显示 1，然后显示 2。  <pre>function myfunc local(1); % local is a function local = 2; disp(local); end</pre> <pre>function local(x) disp(x) end</pre>

### 没有显式声明的标识符可能不被视为变量

从 R2019b 开始，MATLAB 不使用索引运算符来标识程序中的变量。以前，没有显式声明的标识符在使用冒号、`end` 或花括号进行索引时被视为变量。例如，在 `x(a,b,:)`、`x(end)`、`x{a}` 中，`x` 被视为变量。

以如下代码为例。此前，由于使用了冒号索引，MATLAB 会将 `x` 视为变量。从 R2019b 开始，如果路径上存在同名函数，MATLAB 会将 `x` 视为函数。

```
function myfunc
load data.mat; % data.mat contains variable x
disp(x(:))
end
```

如果您打算使用 `x` 作为 `data.mat` 的变量而不是函数，请显式声明它。同样，要使用标识符 `x` 作为从脚本中获取的变量，请在调用脚本之前声明该变量。如果变量由以下函数隐式引入，则此新行为也适用：`sim`、`eval`、`evalc` 和 `assignin`。

下表给出了一些更新代码的示例。

更新前	更新后
<pre>function myfunc load data.mat; disp(x(:)) end</pre>	<pre>function myfunc load data.mat x; disp(x(:)) end</pre>
<pre>function myfunc2 myscript; % Contains variable x disp(x(:)) end</pre>	<pre>function myfunc2 x = []; myscript; disp(x(:)) end</pre>

## 变量无法在父函数和嵌套函数之间隐式共享

从 R2019b 开始，仅当标识符在嵌套函数（第 20-26 页）的父函数中显式声明为变量时，才能在该嵌套函数及其父函数之间共享该标识符作为变量。

例如，在以下代码中，`myfunc` 中的标识符 `x` 不同于嵌套函数中的变量 `x`。如果 `x` 是路径上的函数，MATLAB 会将 `myfunc` 中的 `x` 视为函数，并继续运行代码。否则，MATLAB 将引发错误。

```
function myfunc
nested;
x(3) % x is not a shared variable
    function nested
        x = [1 2 3];
    end
end
```

在以前的版本中，如果 `x` 是路径上的函数，MATLAB 在 `myfunc` 中将其视为函数，在 `nested` 中将其视为变量。如果 `x` 不是路径上的函数，MATLAB 会将其视为在 `myfunc` 和 `nested` 之间共享的变量。这导致了代码的输出取决于路径的状态。

要将标识符用作父函数和嵌套函数之间共享的变量，您可能需要更新代码。例如，您可以将标识符初始化为父函数中的空数组。

更新前	更新后
<pre>function myfunc nested; x(3)     function nested         x = [1 2 3];     end end</pre>	<pre>function myfunc x = []; nested; x(3)     function nested         x = [1 2 3];     end end</pre>

## 基于通配符导入的优先级发生更改

从 R2019b 开始，基于通配符导入的导入函数的优先级低于变量、嵌套函数和局部函数。在 R2019a 及更早版本中，函数中的导入会遮蔽局部函数和嵌套函数。

例如，在以下代码中，语句 `local()` 在基于通配符的导入中调用 `myfunc/local` 而不是 `pkg1.local`。语句 `nest()` 调用 `myfunc/nest` 而不是 `pkg1.nest`。

从 R2019b 开始	R2019a 和更早版本
<pre>function myfunc % Import includes functions local and nest import pkg1.* local() % Calls myfunc/local      function nest     end  nest(); % Calls myfunc/nest end  function local end</pre>	<pre>function myfunc % Import includes functions local and nest import pkg1.* local() % Calls pkg1.local and         % displays warning since R2018a      function nest     end  nest(); % Calls pkg1.nest end  function local end</pre>

在 `import` 的搜索结果中，查找包含通配符 (\*) 的语句。

## 完全限定的导入函数不能与嵌套函数同名

从 R2019b 开始，如果完全限定的导入函数与同一作用域内的嵌套函数共享名称，则会引发错误。

从 R2019b 开始	更新后的代码	R2019a 和更早版本
<p>此函数出错，因为它与同一作用域内的嵌套函数共享名称。</p> <pre>function myfunc import pkg.nest % Errors nest();      function nest     end end</pre>	<p>要从 <code>import</code> 语句调用函数 <code>nest</code>，请重命名局部函数 <code>myfunc/nest</code>。</p> <pre>function myfunc import pkg.nest nest();      function newNest     end end</pre>	<p>此函数从 <code>import</code> 语句中调用函数 <code>nest</code>。</p> <pre>function myfunc import pkg.nest nest(); % Calls pkg.nest      function nest     end end</pre>
<p>此函数出错，因为不支持声明与导入函数 <code>nest</code> 同名的变量。</p> <pre>function myvarfunc import pkg.nest % Errors nest = 1 end</pre>	<p>重命名变量 <code>nest</code>。</p> <pre>function myvarfunc import pkg.nest % Errors thisNest = 1 end</pre>	<p>此函数修改变量 <code>nest</code>。</p> <pre>function myvarfunc import pkg.nest nest = 1 % Modifies variable nest and         % displays warning since R2018a end</pre>

## 完全限定的导入函数会遮蔽同名的外层作用域定义

从 R2019b 开始，完全限定的导入函数会始终遮蔽同名的外层作用域定义。在 R2019a 及更早版本中，完全限定的导入函数如果遮蔽了外层作用域中的标识符，则它将被忽略。

从 R2019b 开始	更新后的代码	R2019a 和更早版本
局部函数 nest 从导入的包中调用函数 x。  <pre>function myfunc x = 1;  function nest     % Import function x     import pkg1.x     % Calls pkg1.x     x() end end</pre>	要在局部函数 nest 中使用变量 x，请将该变量作为参数传递。  <pre>function myfunc x = 1; nest(x)  function nest(x1)     % Import function x     import pkg1.x     % Calls pkg1.x with     % variable x1     x(x1) end end</pre>	在以下代码中，函数 nest 忽略导入的函数 x。  <pre>function myfunc x = 1;  function nest     % Import function x     import pkg1.x     % x is a variable     x() end end</pre>

## 未发现导入时的错误处理

从 R2019b 开始，无论是否有 Java，无法解析的完全限定导入函数都会引发错误。在 R2019a 及更早版本中，MATLAB 的行为有所不同，具体取决于您是否使用 -nojvm 选项启动 MATLAB。不要使用 javachk 和 usejava 等函数自定义错误消息。

从 R2019b 开始	更新后的代码	R2019a 和更早版本
使用 -nojvm 选项启动 MATLAB 时，以下代码会引发错误。  <pre>function myfunc import java.lang.String % Errors if ~usejava('jvm')     % Statement never executes     disp('This function requires Java'); else     % Do something with Java String class end end</pre>	取消对 usejava 的调用。  <pre>function myfunc import java.lang.String % Errors % Do something with java String end</pre>	使用 -nojvm 选项启动 MATLAB 时，以下代码会显示一条消息。  <pre>function myfunc import java.lang.String if ~usejava('jvm')     % Display message     disp('This function requires Java'); else     % Do something with Java String class end end</pre>

## 嵌套函数从父函数继承 import 语句

从 R2019b 开始，嵌套函数从父函数继承 import 语句。在 R2019a 及更早版本中，嵌套函数不从其父函数继承 import 语句。

从 R2019b 开始	R2019a 和更早版本
<pre>function myfunc % Package p1 has functions plot and bar import p1.plot import p1.* nest  function nest     plot % Calls p1.plot     bar % Calls p1.bar end end</pre>	<pre>function myfunc % Package p1 has functions plot and bar import p1.plot import p1.* nest  function nest     plot % Calls plot function on path     bar % Calls bar function on path end end</pre>

## 复合名称解析优先级的更改

从 R2019b 开始，MATLAB 以另一种方式解析复合名称。复合名称包含以点相连的多个部分（例如，`a.b.c`），可用于引用包成员。在 R2019b 中，MATLAB 在解析复合名称时，将最长匹配前缀视为优先。在以前的版本中，优先顺序遵循一组更复杂的规则。

例如，假设包 `pkg` 包含具有静态方法 `bar` 的 `foo` 类，以及具有函数 `bar` 的 `foo` 子包。

```
+pkg/@foo/bar.m % bar is a static method of class foo
+pkg/+foo/bar.m % bar is a function in subpackage foo
```

在 R2019b 中，调用 `which pkg.foo.bar` 将返回包函数的路径。

```
which pkg.foo.bar
```

```
+pkg/+foo/bar.m
```

以前，在包和类同名的情况下，静态方法优先于包函数。

## 匿名函数可以同时包含已解析和未解析的标识符

从 R2019b 开始，匿名函数可以同时包含已解析和未解析的标识符。在以前的版本中，如果匿名函数中有任何标识符在创建时未解析，则该匿名函数中的所有标识符均为未解析。

从 R2019b 开始	R2019a 和更早版本
<p>为了计算匿名函数，MATLAB 使用在 <code>myscript</code> 中定义的 <code>x</code> 调用局部函数 <code>lf</code>，因为匿名函数中的 <code>lf</code> 解析为局部函数。</p> <pre>function myfun myscript; % Includes x = 1 and lf = 10 f = @(lf(x)); f()    % Displays 'Inside lf' end  % Local function to myfun function lf(y) disp('Inside lf'); end</pre>	<p>MATLAB 将 <code>lf</code> 和 <code>x</code> 视为未解析的标识符，并使用 <code>x</code> 对来自 <code>myscript</code> 的变量 <code>lf</code> 进行索引。</p> <pre>function myfun myscript; % Includes x = 1 and lf = 10 f = @(lf(x)); f()    % Displays 10 end  % Local function to myfun function lf(y) disp('Inside lf'); end</pre>

## 另请参阅

`import`

## 详细信息

- “导入类”

# 对函数调用结果进行索引

此主题说明如何对函数调用创建的临时变量进行点索引。当函数调用的结果被用作更大的表达式中的中间变量时，会创建临时变量。表达式中函数调用的结果是暂时性的，因为它创建的变量只短暂存在，并且在执行后不会存储在 MATLAB 工作区中。例如，表达式 `myFunction(x).prop` 使用参数 `x` 调用 `myFunction`，然后返回结果的 `prop` 属性。您可以用这种方式调用任何类型的函数（匿名函数、局部函数、嵌套函数或私有函数）。

## 示例

以下面的函数为例：

```
function y = myStruct(x)
    y = struct("Afield",x);
end
```

此函数创建包含一个字段（名为 `Afield`）的结构体，并为该字段赋值。您可以使用以下命令调用该函数并创建一个结构体，该结构体具有一个包含值 1 的字段：

```
myStruct(1)
```

```
ans =
```

```
struct with fields:
```

```
    Afield: 1
```

但是，如果您要直接返回字段值，则可以使用以下命令对函数调用结果进行索引：

```
myStruct(1).Afield
```

```
ans =
```

```
1
```

此命令执行后，由命令 `myStruct(1)` 创建的临时结构体不再存在，MATLAB 仅返回字段值。从概念上讲，这种用法与创建结构体、对其进行索引然后删除它是一样的：

```
S = struct("Afield",1);
S.Afield
clear S
```

## 支持的语法

MATLAB 支持对函数调用结果进行点索引，如 `foo(arg).prop` 中所示。不支持以其他形式对函数调用结果进行索引（例如使用圆括号，形如 `foo(arg)(2)`，或使用花括号，形如 `foo(arg){2}`）。成功的命令必须符合以下条件：

- 用圆括号调用函数，形如 `foo(arg1,arg2,...)`。
- 函数返回具有点索引定义的变量，如结构体、表或对象。
- 点索引下标有效。

MATLAB 总会尝试对临时变量应用点索引操作，即使该函数返回的变量没有点索引定义。例如，如果您尝试对由 `magic(3)` 创建的矩阵进行索引，就会出现错误。

```
magic(3).field
```

Dot indexing is not supported for variables of this type.

只要可以继续对临时变量进行索引，您就可以在表达式的末尾添加更多索引命令。以如下表达式为例：

```
table(rand(10,2)).Var1(3,:)
```

在此表达式中，您对一个表进行索引以获取它包含的矩阵，然后对该矩阵进行索引以获取第三行：

- `table(rand(10,2))` 创建一个表，该表包含一个名为 `Var1` 的变量。该变量包含一个  $10 \times 2$  矩阵。
- `table(rand(10,2)).Var1` 返回 `Var1` 所包含的  $10 \times 2$  矩阵。
- `table(rand(10,2)).Var1(3,:)` 返回 `Var1` 所包含的矩阵的第三行。

### 另请参阅

[function](#) | [subsref](#)

### 详细信息

- “[函数类型](#)” (第 20-16 页)
- “[数组索引](#)”
- “[访问表中的数据](#)” (第 9-33 页)

# 函数参数

---

- “查找函数参数的数量” (第 21-2 页)
- “支持可变数量的输入” (第 21-4 页)
- “支持可变数量的输出” (第 21-5 页)
- “验证函数参数的数量” (第 21-6 页)
- “嵌套函数中的参数检查” (第 21-8 页)
- “忽略函数输入” (第 21-10 页)
- “通过 validateattributes 检查函数输入” (第 21-11 页)
- “解析函数输入” (第 21-13 页)
- “输入解析器验证函数” (第 21-16 页)

## 查找函数参数的数量

本示例介绍如何使用 `nargin` 和 `nargout` 确定您的函数收到的输入或输出参数数量。

### 输入参数

在名称为 `addme.m` 的文件中创建最多可接受两个输入的函数。通过 `nargin` 确定输入数量。

```
function c = addme(a,b)

switch nargin
    case 2
        c = a + b;
    case 1
        c = a + a;
    otherwise
        c = 0;
end
```

调用具有一个、两个或零个输入参数的 `addme`。

```
addme(42)
```

```
ans =
 84
```

```
addme(2,4000)
```

```
ans =
 4002
```

```
addme
```

```
ans =
 0
```

### 输出参数

在名为 `addme2.m` 的文件中创建一个可以返回一个或两个输出（一个结果及其绝对值）的新函数。通过 `nargout` 确定请求的输出数量。

```
function [result,absResult] = addme2(a,b)

switch nargin
    case 2
        result = a + b;
    case 1
        result = a + a;
    otherwise
        result = 0;
end

if nargout > 1
    absResult = abs(result);
end
```

调用具有一个或两个输出参数的 `addme2`。

```
value = addme2(11,-22)
```

```
value =  
-11  
[value,absValue] = addme2(11,-22)
```

```
value =  
-11  
  
absValue =  
11
```

函数按照函数定义中声明的顺序返回输出。

## 另请参阅

[nargin](#) | [narginchk](#) | [nargout](#) | [nargoutchk](#)

## 支持可变数量的输入

本示例介绍如何使用 `varargin` 定义接受可变数量的输入参数的函数。`varargin` 参数是包含函数输入的元胞数组，其中每个输入都位于它自己的元胞中。

在名为 `plotWithTitle.m` 的文件中创建一个函数，为 `plot` 函数接受可变数量的成对输入 ( $x,y$ ) 和可选的标题。如果该函数接收的输入数为奇数，它将最后一个输入视为标题。

```
function plotWithTitle(varargin)
if rem(nargin,2) ~= 0
    myTitle = varargin{nargin};
    numPlotInputs = nargin - 1;
else
    myTitle = 'Default Title';
    numPlotInputs = nargin;
end

plot(varargin{1:numPlotInputs})
title(myTitle)
```

由于 `varargin` 是元胞数组，您要使用花括号 {} 来访问每个元胞的内容。语法 `varargin{1:numPlotInputs}` 创建 `plot` 函数的逗号分隔输入列表。

调用两组 ( $x,y$ ) 输入和一个标题的 `plotWithTitle`。

```
x = [1:1:10];
y1 = sin(x);
y2 = cos(x);
plotWithTitle(x,y1,x,y2,'Sine and Cosine')
```

您可以在输入参数列表中或输入列表末尾单独使用 `varargin`，例如

```
function myfunction(a,b,varargin)
```

在这种情况下，`varargin{1}` 对应于传递到该函数的第三个输入，并且 `nargin` 返回 `length(varargin) + 2`。

### 另请参阅

`nargin` | `varargin`

### 相关示例

- “访问元胞数组中的数据”（第 12-5 页）

### 详细信息

- “嵌套函数中的参数检查”（第 21-8 页）
- “逗号分隔的列表”（第 2-73 页）

## 支持可变数量的输出

本示例介绍如何使用 `varargout` 定义返回可变数量的输出参数的函数。输出 `varargout` 是包含函数输出的元胞数组，其中每个输出都位于它自己的元胞中。

在名为 `magicfill.m` 的文件中创建一个函数，该函数向每个请求的输出指定一个幻方矩阵。

```
function varargout = magicfill
    nOutputs = nargin;
    varargout = cell(1,nOutputs);

    for k = 1:nOutputs
        varargout{k} = magic(k);
    end
```

用花括号 {} 创建的索引会更新元胞的内容。

调用 `magicfill` 并请求三个输出。

```
[first,second,third] = magicfill

first =
    1

second =
    1   3
    4   2

third =
    8   1   6
    3   5   7
    4   9   2
```

MATLAB 根据输出在 `varargout` 数组中的顺序向它们赋值。例如，`first == varargout{1}`。

您可以在输出参数列表中或输出列表末尾单独使用 `varargout`，例如

```
function [x,y,varargout] = myfunction(a,b)
```

在这种情况下，`varargout{1}` 对应于函数返回的第三个输出，并且 `nargout` 返回 `length(varargout) + 2`。

## 另请参阅

[nargout | varargout](#)

## 相关示例

- “访问元胞数组中的数据”（第 12-5 页）

## 详细信息

- “嵌套函数中的参数检查”（第 21-8 页）

## 验证函数参数的数量

本例介绍如何检查您的自定义函数是否接收有效数量的输入或输出参数。MATLAB 自动执行一些参数检查。对于其他情况，您可以使用 `narginchk` 或 `nargoutchk`。

### 自动参数检查

MATLAB 在它可以确定来自函数定义的数量时，检查您的函数接收的参数是否多于预期。例如，该函数最多接受两个输出和三个输入：

```
function [x,y] = myFunction(a,b,c)
```

如果您向 `myFunction` 传递太多的输入，MATLAB 将发出错误。您无需调用 `narginchk` 即可检查这种情况。

```
[X,Y] = myFunction(1,2,3,4)
```

```
Error using myFunction  
Too many input arguments.
```

使用 `narginchk` 和 `nargoutchk` 函数验证您的函数接收的参数数量是否满足以下条件：

- 达到必需参数的最小数量。
- 当您的函数使用 `varargin` 或 `varargout` 时，不超出参数的最大数量。

### 通过 `narginchk` 进行输入检查

在名为 `testValues.m` 的文件中定义至少需要两个输入的函数。第一个输入是与其他输入对比的阈值。

```
function testValues(threshold,varargin)  
minInputs = 2;  
maxInputs = Inf;  
narginchk(minInputs,maxInputs)  
  
for k = 1:(nargin-1)  
    if (varargin{k} > threshold)  
        fprintf('Test value %d exceeds %d\n',k,threshold);  
    end  
end
```

调用带有太少输入的 `testValues`。

```
testValues(10)
```

```
Error using testValues (line 4)  
Not enough input arguments.
```

调用带有足够输入的 `testValues`。

```
testValues(10,1,11,111)
```

```
Test value 2 exceeds 10  
Test value 3 exceeds 10
```

## 通过 nargoutchk 进行输出检查

在名为 `mysize.m` 的文件中定义一个函数，该函数在向量（来自 `size` 函数）中返回输入数组的维度，并选择性地返回与每个维度大小对应的标量值。使用 `nargoutchk` 验证请求的各个维度大小的数量未超出可用维度的数量。

```
function [sizeVector,varargout] = mysize(x)
minOutputs = 0;
maxOutputs = ndims(x) + 1;
nargoutchk(minOutputs,maxOutputs)

sizeVector = size(x);

varargout = cell(1,nargout-1);
for k = 1:length(varargout)
    varargout{k} = sizeVector(k);
end
```

调用带有有效数量的输出的 `mysize`。

```
A = rand(3,4,2);
[fullsize,nrows,ncols,npages] = mysize(A)

fullsize =
3   4   2

nrows =
3

ncols =
4

npages =
2
```

调用带有太多输出的 `mysize`。

```
A = 1;
[fullsize,nrows,ncols,npages] = mysize(A)

Error using mysize (line 4)
Too many output arguments.
```

## 另请参阅

`narginchk` | `nargoutchk`

## 相关示例

- “支持可变数量的输入” (第 21-4 页)
- “支持可变数量的输出” (第 21-5 页)

## 嵌套函数中的参数检查

本主题介绍将 `varargin`、`varargout`、`nargin`、`nargout` 与嵌套函数结合使用时的特殊注意事项。

通过 `varargin` 和 `varargout`，您可以创建接受可变数量的输入或输出参数的函数。虽然 `varargin` 和 `varargout` 看上去像函数名称，但它们都是变量，不是函数。这一点很重要，因为嵌套函数与包含它们的函数共享工作区。

如果您在声明嵌套函数时未使用 `varargin` 或 `varargout`，则嵌套函数内的 `varargin` 或 `varargout` 引用外部函数的参数。

例如，在名为 `showArgs.m` 的文件中创建一个函数，该函数使用 `varargin` 并包含两个嵌套函数，一个使用 `varargin`，另一个不使用。

```
function showArgs(varargin)
nested1(3,4)
nested2(5,6,7)

function nested1(a,b)
    disp('nested1: Contents of varargin{1}')
    disp(varargin{1})
end

function nested2(varargin)
    disp('nested2: Contents of varargin{1}')
    disp(varargin{1})
end
```

调用该函数并比较这两个嵌套函数中的 `varargin{1}` 内容。

```
showArgs(0,1,2)

nested1: Contents of varargin{1}
0

nested2: Contents of varargin{1}
5
```

另一方面，`nargin` 和 `nargout` 都是函数。在包含嵌套函数的任何函数内，调用 `nargin` 或 `nargout` 返回该函数的参数数量。如果嵌套函数需要外部函数中的 `nargin` 或 `nargout` 的值，将该值传递到嵌套函数。

例如，在名为 `showNumArgs.m` 的文件中创建一个函数，该函数将主（父）函数的输入参数数量传递到嵌套函数。

```
function showNumArgs(varargin)

disp(['Number of inputs to showNumArgs: ',int2str(nargin)]);
nestedFx(nargin,2,3,4)

function nestedFx(n,varargin)
    disp(['Number of inputs to nestedFx: ',int2str(nargin)]);
    disp(['Number of inputs to its parent: ',int2str(n)]);
end
```

```
end
```

调用 `showNumArgs` 并比较父函数和嵌套函数中的 `nargin` 的输出。

```
showNumArgs(0,1)
```

```
Number of inputs to showNumArgs: 2
```

```
Number of inputs to nestedFx: 4
```

```
Number of inputs to its parent: 2
```

## 另请参阅

`nargin` | `nargout` | `varargin` | `varargout`

## 忽略函数输入

此示例说明如何使用波浪号 (~) 运算符忽略函数定义中的输入。

当您的函数必须接受预定义的一组输入但不使用所有输入时，请使用该运算符。常见的应用包括定义回调函数，如下所示。

在名为 **colorButton.m** 的文件中为不使用  **eventdata** 输入的普通按钮定义回调。使用波浪号忽略该输入。

```
function colorButton
figure;
uicontrol('Style','pushbutton','String','Click me','Callback',@btnCallback)

function btnCallback(h,~)
set(h,'BackgroundColor',rand(3,1))
```

**btnCallback** 的函数声明实际上与以下函数相同。

```
function btnCallback(h eventdata)
```

但是，使用波浪号可避免向函数工作区中添加  **eventdata** 并清楚地表明该函数不使用  **eventdata**。

您可以忽略函数定义中的任意多个输入，无论它们位于参数列表上的哪个位置。用逗号分隔连续的波浪号。例如：

```
function myFunction(myInput,~,~)
```

## 另请参阅

### 详细信息

- “忽略函数输出”（第 1-4 页）

## 通过 validateattributes 检查函数输入

使用 `validateattributes` 函数验证您的函数的输入是否符合一组要求。

`validateattributes` 要求您传递要检查的变量以及该变量支持的数据类型。或者，传递描述有效维度或值的一组属性。

### 检查数据类型和其他属性

在名为 `checkme.m` 的文件中定义最多可接受以下三个输入的函数：`a`、`b` 和 `c`。检查以下条件：

- `a` 是否由双精度正值组成的二维数组。
- `b` 是否在一个有 10 列的数组中包含 100 个数值。
- `c` 是否为非空字符向量或元胞数组。

```
function checkme(a,b,c)

validateattributes(a,{['double']},{['positive'],'2d'})
validateattributes(b,{['numeric']},{['numel',100,'ncols',10]}
validateattributes(c,{['char','cell']},{['nonempty']})

disp('All inputs are ok.')
```

花括号 {} 指示数据类型集和其他属性集都位于元胞数组中。元胞数组允许您将文本与数值数据的组合或不同长度的字符向量存储在单个变量中。

调用带有有效输入的 `checkme`。

```
checkme(pi,rand(5,10,2),'text')
```

All inputs are ok.

标量值 `pi` 是二维值，因为 `size(pi) = [1,1]`。

调用带有无效输入的 `checkme`。`validateattributes` 函数对首个验证失败的输入报错，`checkme` 停止处理。

```
checkme(-4)
```

Error using checkme (line 3)  
Expected input to be positive.

```
checkme(pi,rand(3,4,2))
```

Error using checkme (line 4)  
Expected input to be an array with number of elements equal to 100.

```
checkme(pi,rand(5,10,2),struct)
```

Error using checkme (line 5)  
Expected input to be one of these types:

char, cell

Instead its type was struct.

默认错误消息使用通用词 `input` 来表示验证失败的参数。使用默认错误消息时，确定失败输入的唯一方法是查看 `checkme` 中的指定代码行。

### 在错误中添加输入名称和位置

在名称为 `checkdetails.m` 的文件中定义一个函数，该函数执行与 `checkme` 相同的验证，但在错误消息中添加有关输入名称和位置的详细信息。

```
function checkdetails(a,b,c)
validateattributes(a,{<b>'double'</b>},{<b>'positive'</b>,'2d'},',[<b>'First'</b>,1]
validateattributes(b,{<b>'numeric'</b>},{<b>'numel'</b>,100,<b>'ncols'</b>,10},',[<b>'Second'</b>,2)
validateattributes(c,{<b>'char'</b>},{<b>'nonempty'</b>},',[<b>'Third'</b>,3)

disp('All inputs are ok.')
```

`validateattributes` 的第四个输入的空字符串向量 "`"` 是可选函数名称的占位符。您无需指定函数名称，因为它已显示在错误消息中。当您需要将函数名称包含在错误标识符中以进行更多的错误处理时，请指定函数名称。

调用带有无效输入的 `checkdetails`。

```
checkdetails(-4)
```

```
Error using checkdetails (line 3)
Expected input number 1, First, to be positive.
```

```
checkdetails(pi,rand(3,4,2))
```

```
Error using checkdetails (line 4)
Expected input number 2, Second, to be an array with
number of elements equal to 100.
```

### 另请参阅

[validateattributes](#) | [validatestring](#)

# 解析函数输入

此示例说明如何定义必需和可选的输入、如何指定可选输入的默认值以及如何使用输入解析器验证自定义函数的所有输入。

输入解析器提供一致的方式来验证输入和指定输入的默认值，改进您的代码的稳定性和可维护性。要验证输入，您可以利用现有的 MATLAB 函数或编写您自己的验证例程。

## 步骤 1. 定义您的函数。

在名为 `printPhoto.m` 的文件中创建函数。`printPhoto` 函数具有文件名必需的一个输入，以及抛光（有光泽或无光泽）、颜色空间（RGB 或 CMYK）、宽度和高度的可选输入。

```
function printPhoto(filename,varargin)
```

在您的函数声明语句中，先指定必需的输入。使用 `varargin` 支持可选输入。

## 步骤 2. 创建一个 `InputParser` 对象。

在您的函数内，调用 `inputParser` 以创建解析器对象。

```
p = inputParser;
```

## 步骤 3. 将输入添加到方案中。

使用 `addRequired`、`addOptional` 或 `addParameter` 将输入添加到您的函数中的解析方案中。为可选输入指定默认值。

对于每个输入，您可以指定验证函数的句柄，用于检查输入并返回标量逻辑值（`true` 或 `false`）或错误。验证函数可以是现有的 MATLAB 函数（例如 `ischar` 或 `isnumeric`）或您创建的函数（例如匿名函数或局部函数）。

在 `printPhoto` 函数中，`filename` 是必需的输入。将 `finish` 和 `color` 定义为可选输入，将 `width` 和 `height` 定义为可选的参数值对组。

```
defaultFinish = 'glossy';
validFinishes = {'glossy','matte'};
checkFinish = @(x) any(validatestring(x,validFinishes));

defaultColor = 'RGB';
validColors = {'RGB','CMYK'};
checkColor = @(x) any(validatestring(x,validColors));

defaultWidth = 6;
defaultHeight = 4;

addRequired(p,'filename',@ischar);
addOptional(p,'finish',defaultFinish,checkFinish)
addOptional(p,'color',defaultColor,checkColor)
addParameter(p,'width',defaultWidth,@isnumeric)
addParameter(p,'height',defaultHeight,@isnumeric)
```

您通过 `addRequired` 或 `addOptional` 添加的输入是位置性参数。调用具有位置性输入的函数时，按将它们添加到解析方案中的顺序来指定这些值。

通过 `addParameter` 添加的输入是非位置性的，因此您可以在 `width` 的值之前或之后传递 `height` 的值。但是，参数值输入要求您传递输入名称（'height' 或 'width'）以及输入值。

如果您的函数接受可选的输入字符串或字符向量以及参数名称-值对组，请为可选输入指定验证函数。否则，输入解析器将可选字符串或字符向量解释为参数名称。例如，`checkFinish` 验证函数确保 `printPhoto` 将 '`glossy`' 解释为 `finish` 的值，而不是无效的参数名称。

#### 步骤 4.设置属性以调整解析（可选）。

默认情况下，输入解析器会假设是否大小写区分、函数名称、结构体数组输入以及是否允许使用方案中没有的其他参数名称和值。属性允许您显式定义这些行为。使用圆点表示法置属性，类似于向结构体数组赋值。

通过设置输入解析器的 `KeepUnmatched` 属性来使 `printPhoto` 接受与输入方案不匹配的其他参数值输入。

```
p.KeepUnmatched = true;
```

如果 `KeepUnmatched` 是 `false`（默认值），输入解析器在输入与方案不匹配时发出错误。

#### 步骤 5.解析输入。

在您的函数内，调用 `parse` 方法。传递所有函数输入的值。

```
parse(p,filename,varargin{3})
```

#### 步骤 6.在您的函数中使用输入。

使用 `inputParser` 对象的以下属性访问解析的输入：

- `Results` - 包含方案中所有输入的名称和值的结构体数组。
- `Unmatched` - 包含不在方案中而是传递到函数中的参数名称和值的结构体数组（当 `KeepUnmatched` 是 `true` 时）。
- `UsingDefaults` - 具有特定的可选输入名称的元胞数组，由于这些可选输入名称未传递到函数，为它们指定了各自的默认值。

在 `printPhoto` 函数内，显示一些输入的值：

```
disp(['File name:',p.Results.filename])
disp(['Finish:', p.Results.finish])

if ~isempty(fieldnames(p.Unmatched))
    disp('Extra inputs:')
    disp(p.Unmatched)
end
if ~isempty(p.UsingDefaults)
    disp('Using defaults:')
    disp(p.UsingDefaults)
end
```

#### 步骤 7.调用您的函数。

输入解析器预期接收如下所示的输入：

- 首先是必需的输入，按照通过 `addRequired` 将它们添加到解析方案的顺序。
- 可选输入，按照通过 `addOptional` 将它们添加到方案的顺序。
- 参数名称-值对组输入之前的位置性输入。
- `Name1,Value1,...,NameN,ValueN` 形式的参数名称和值。

将多个输入组合传递到 `printPhoto`, 其中一些为有效组合, 一些为无效组合:

```
printPhoto('myfile.jpg')
```

File name: myfile.jpg  
Finish: glossy  
Using defaults:  
'finish' 'color' 'width' 'height'

```
printPhoto(100)
```

Error using printPhoto (line 23)  
The value of 'filename' is invalid. It must satisfy the function: ischar.

```
printPhoto('myfile.jpg','satin')
```

Error using printPhoto (line 23)  
The value of 'finish' is invalid. Expected input to match one of these strings:

'glossy', 'matte'

The input, 'satin', did not match any of the valid strings.

```
printPhoto('myfile.jpg','height',10,'width',8)
```

File name: myfile.jpg  
Finish: glossy  
Using defaults:  
'finish' 'color'

要传递第  $n$  个位置性输入的值, 请指定前一个  $(n - 1)$  输入的值或以参数名称-值对组形式传递输入。例如, 这些函数调用对 `finish` (默认为 'glossy') 和 `color` 赋予相同值:

```
printPhoto('myfile.gif','glossy','CMYK') % positional
```

```
printPhoto('myfile.gif','color','CMYK') % name and value
```

## 另请参阅

[inputParser](#) | [varargin](#)

## 详细信息

- “[输入解析器验证函数](#)” (第 21-16 页)

## 输入解析器验证函数

本主题介绍定义验证函数的方式，该验证函数要传递到输入解析器以检查自定义函数输入。

输入解析器方法 `addRequired`、`addOptional` 和 `addParameter` 都可接受验证函数的句柄。通过 @ 符号指定函数句柄。

验证函数必须接受单个输入参数，还必须返回标量逻辑值（`true` 或 `false`）或错误。如果验证函数返回 `false`，输入解析器将发出错误并且您的函数停止处理。

可以通过以下多种方式定义验证函数：

- 使用现有的 MATLAB 函数，例如 `ischar` 或 `isnumeric`。例如，检查名称为 `num` 的必需输入是否为数值：

```
p = inputParser;
checknum = @(x) isnumeric(x);
addRequired(p,'num',checknum)

parse(p,'text')
```

The value of 'num' is invalid. It must satisfy the function: isnumeric.

- 创建匿名函数。例如，检查输入 `num` 是否为大于零的数值标量：

```
p = inputParser;
checknum = @(x) isnumeric(x) && isscalar(x) && (x > 0);
addRequired(p,'num',checknum)

parse(p,rand(3))
```

The value of 'num' is invalid. It must satisfy the function: @(x) isnumeric(x) && isscalar(x) && (x>0).

- 定义您自己的函数，一般为位于和主函数相同的文件中的局部函数。例如，在名为 `usenum.m` 的文件中，定义名称为 `checknum` 的局部函数，用于在 `usenum` 的输入 `num` 不是大于零的数值标量时发出自定义的错误消息：

```
function usenum(num)
p = inputParser;
addRequired(p,'num',@checknum);
parse(p,num);

function TF = checknum(x)
TF = false;
if ~isscalar(x)
    error('Input is not scalar');
elseif ~isnumeric(x)
    error('Input is not numeric');
elseif (x <= 0)
    error('Input must be > 0');
else
    TF = true;
end
```

调用带有无效输入的函数：

```
usenum(-1)
```

Error using usenum (line 4)  
The value of 'num' is invalid. Input must be > 0

## 另请参阅

`inputParser | is* | validateattributes`

## 相关示例

- “解析函数输入” (第 21-13 页)
- “创建函数句柄” (第 13-2 页)

## 详细信息

- “匿名函数” (第 20-19 页)



# 调试 MATLAB 代码

---

- “调试 MATLAB 程序” (第 22-2 页)
- “设置断点” (第 22-6 页)
- “调试时检验值” (第 22-11 页)

## 调试 MATLAB 程序

要以图形方式调试 MATLAB 程序，请使用编辑器/调试器。您也可以在命令行窗口中使用调试函数。两种方法可互换。

开始调试之前，请确保您的程序已保存且该程序及其调用的任何文件位于您的搜索路径或当前文件夹中。

- 如果在未保存更改的情况下从编辑器内运行文件，该文件在运行前会自动保存。
- 如果您从命令行窗口运行未保存更改的文件，则 MATLAB 软件会运行已保存的文件版本。因此您看不到更改结果。

### 设置断点

设置断点可暂停执行 MATLAB 文件，以便检查您认为可能存在问题的值或变量。您可以使用编辑器、命令行窗口中的函数或同时使用这两种方法设置断点。

有三种不同类型的断点：标准、条件和错误。要在编辑器中添加标准断点，请在要设置断点的可执行代码行的断点列处点击。断点列是编辑器左侧、行号右侧的窄列。您也可以使用 F12 键来设置断点。

断点列中的可执行代码行以虚线（-）指示。例如，点击以下代码中第 2 行旁边的断点列，向该行添加断点。

```

myprogram.m
1 %Create an array of 10 ones.
2 x = ones(1,10);
3
4 %Perform a calculation on items 2-6 in the array
5 for n = 2:6
6     x(n) = 2 * x(n-1);
7 end

```

如果可执行语句跨多个行，您可以在该语句中的每一行均设置断点，即使其他行在断点列中没有 - (虚线) 也可以。例如，在此代码中，您可以在所有四个行中均设置断点：

```

1 if a ...
2     && b
3     c = 1;
4 end

```

有关不同类型断点的详细信息，请参阅“设置断点”（第 22-6 页）。

### 运行文件

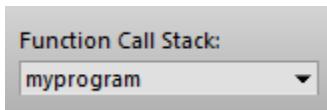
在设置断点后，从命令行窗口或编辑器运行该文件。运行该文件会产生以下结果：

- 运行 ➤ 按钮更改为暂停 ⏸ 按钮。
- 命令行窗口中的提示符将更改为 K>>，指示 MATLAB 处于调试模式且键盘受控制。

- MATLAB 在该程序的第一个断点处暂停。在编辑器中，断点右侧的绿色箭头表示暂停。在发生暂停的行恢复运行之前，程序不会执行该行。例如，在以下示例中，调试器会在程序执行 `x = ones(1,10);` 之前暂停。

```
1 % Create an array of 10 ones.
2 ●▶ x = ones(1,10);
```

- MATLAB 会在**函数调用堆栈**（位于**调试**部分中的**编辑器**选项卡上）中显示当前工作区。



如果您从命令行窗口使用调试函数，可使用 `dbstack` 查看函数调用堆栈。

有关使用函数调用堆栈的详细信息，请参阅“选择工作区”（第 22-11 页）

## 暂停运行文件

要暂停正在运行的程序，请转到**编辑器**选项卡并点击**暂停** 按钮。MATLAB 会在下一个可执行代码行处暂停执行，并且**暂停** 按钮会更改为**继续** 按钮。要继续执行，请按**继续** 按钮。

如果您想要检查长时间运行的程序的进度以确保它按预期运行，则暂停很有用。

---

**注意** 点击暂停按钮会使 MATLAB 在您自己的程序文件外的文件中暂停。按**继续** 按钮会继续正常执行而不更改文件结果。

---

## 查找并解决问题

当您的代码暂停时，您可以查看或更改变量的值，或修改代码。

### 调试时查看或更改变量

在调试时查看变量的值，以查看某行代码是否生成了预期的结果。要执行此操作，请将鼠标指针放到变量的左侧。该变量的当前值将显示在数据提示中。

```
- for n = 2:6
  end
      n: 1x1 double =
          6
```

数据提示会一直在视图中，直到您移动指针。如果在显示数据提示时遇到问题，请点击包含该变量的行，然后将指针移至靠近该变量。有关详细信息，请参阅“调试时检验值”（第 22-11 页）。

在调试时，您可以更改变量的值，以查看新值是否生成预期的结果。在程序暂停状态下，在命令行窗口、工作区浏览器或变量编辑器中向变量分配新值。然后继续运行或分步执行该程序。

例如，在以下例中，MATLAB 在 `for` 循环（其中 `n = 2`）内暂停：

```
%Create an array of 10 ones.
x = ones(1,10);

%Perform a calculation on items 2-6 in the array
for n = 2:6
    x(n) = 2 * x(n-1);
end
```

- 在命令行中键入 `n = 7;` 以将 `n` 的当前值从 2 更改为 7。
- 按继续 运行下一行代码。

MATLAB 会运行代码行 `x(n) = 2 * x(n-1);`, 其中 `n = 7`。

### 调试时修改代码节

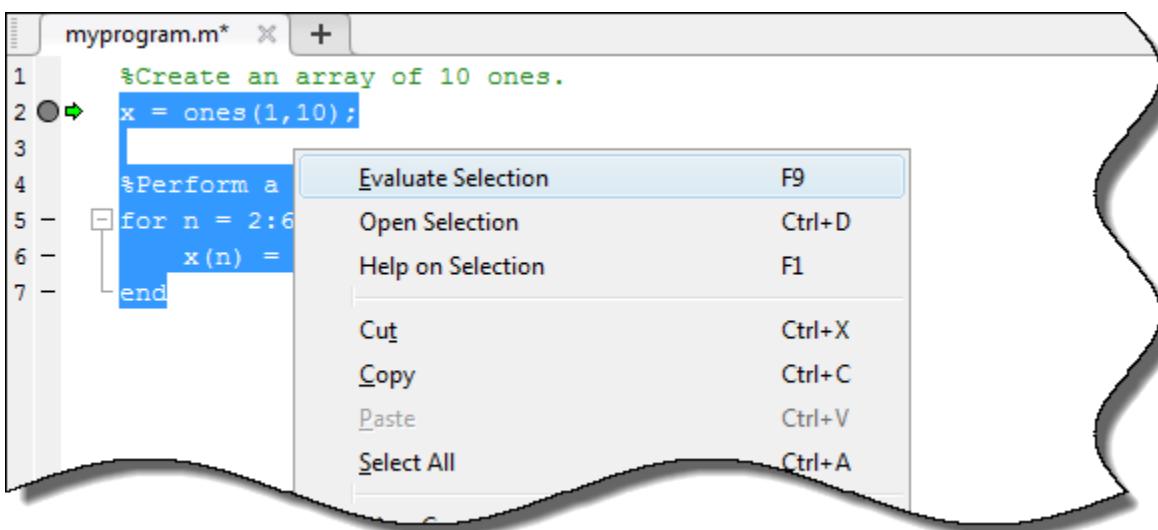
您可以在调试时修改代码段，以在不保存更改的情况下测试可能的修复。通常，最好是在退出调试后修改 MATLAB 文件，然后保存修改并运行该文件。否则，您可能获得意外结果。但是，在有些情况下，您希望在调试过程中进行试验。

要在调试时修改程序，请执行以下操作：

- 1 您的代码暂停后，修改尚未运行的文件部分。

断点变为灰色，指示它们无效。

- 2 选择 MATLAB 暂停所在行后的所有代码，右键点击，然后从上下文菜单中选择**执行所选内容**。



代码计算完成后，停止调试，保存或撤消所做的任何更改，然后再继续调试过程。

## 逐步执行文件

调试时，您可以在您要检查值的点暂停，逐步执行 MATLAB 文件。

此表介绍了可用调试操作以及可用于执行这些操作的不同方法。

说明	工具栏按钮	备用函数
继续执行文件，直到光标所在行。也可从上下文菜单中获得。	 运行到光标处	无
执行当前文件行。	 步长	<code>dbstep</code>
执行当前文件行，如果该行调用另一个函数，则步入该函数。	 步入	<code>dbstep in</code>
继续执行文件，直到完成或遇到另一断点为止。	 继续	<code>dbcont</code>
步入后，运行被调用函数或局部函数的其余部分，离开被调用函数并暂停。	 步出	<code>dbstep out</code>
暂停调试模式。	 暂停	无
退出调试模式。	 退出调试	<code>dbquit</code>

## 结束调试会话

在发现问题后，通过转至编辑器选项卡并点击 **退出调试** 来结束调试会话。如果您要更改并保存文件，或者要在 MATLAB 中运行其他程序，您必须结束调试会话。

退出调试后，编辑器屏幕中的暂停指示符不再显示，并且命令行窗口中会重新显示正常提示符 `>>` 而非 `K>>`。您无法再访问该调用堆栈。

如果 MATLAB 软件在断点位置暂停时变得无法响应，请按 **Ctrl+c** 返回到 MATLAB 提示符。

## 另请参阅

### 相关示例

- “设置断点”（第 22-6 页）
- “调试时检验值”（第 22-11 页）

## 设置断点

### 本节内容

- “标准断点”（第 22-6 页）
- “条件断点”（第 22-7 页）
- “错误断点”（第 22-8 页）
- “匿名函数中的断点”（第 22-8 页）
- “无效断点”（第 22-9 页）
- “禁用断点”（第 22-9 页）
- “清除断点”（第 22-9 页）

设置断点会暂停执行您的 MATLAB 程序，以便您检查您认为可能有问题的值。您可以使用编辑器或命令行窗口中的函数来设置断点。

有三种类型的断点：

- 标准断点
- 条件断点
- 错误断点

您可以只选择那些保存在特定位置（当前文件夹或搜索路径上的文件夹）的文件，在这些文件的可执行代码行处设置断点。您可以随时设置断点，无论 MATLAB 是处于闲置状态还是在忙于运行文件。

默认情况下，MATLAB 会在到达断点时自动打开文件。要禁用此选项，请执行以下操作：

- 1 从主页选项卡上，点击环境部分中的  预设。
- “预设项”对话框随即打开。
- 2 选择 MATLAB > 编辑器/调试器。
- 3 清除当 MATLAB 到达断点时自动打开文件选项，然后点击确定。

## 标准断点

标准断点在文件的指定行暂停。

要设置标准断点，请在您要设置断点的可执行代码行处点击断点列。断点列是编辑器左侧、行号右侧的窄列。您也可以使用 F12 键来设置断点。

断点列中的可执行代码行以 - (虚线) 指示。如果可执行语句跨多个行，您可以在该语句中的每一行均设置断点，即使其他行在断点列中没有 - (虚线) 也可以。例如，在此代码中，您可以在所有四个行中均设置断点：

```

1 - if a ...
2           && b
3 -     c = 1;
4 - end

```

如果您尝试在可执行代码行以外的行处设置断点，例如注释或空白行，则 MATLAB 会在下一个可执行代码处行设置断点。

要以编程方式设置标准断点，请使用 **dbstop** 函数。例如，要在名为 **myprogram.m** 的文件的第 2 行添加断点，请键入：

```
dbstop in myprogram at 2
```

MATLAB 会在函数 **myprogram** 的第 2 行添加断点。

```
myprogram.m
1 %Create an array of 10 ones.
2 x = ones(1,10);
3
4 %Perform a calculation on items 2-6 in the array
5 for n = 2:6
6 x(n) = 2 * x(n-1);
7 end
```

要按 **for** 循环中的增量检验值，请在循环中而非循环开头设置断点。如果在 **for** 循环的开头设置断点，然后逐步执行文件，则 MATLAB 只会在 **for** 语句处暂停一次。不过，如果您将断点放在循环中，则 MATLAB 每次通过循环时均会暂停。

```
myprogram.m
4 % Perform a calculation on items 2 - 6 in the array
5 for n = 2:6
6 x(n) = 2 * x(n - 1);
7 end
```

## 条件断点

如果使用条件断点，MATLAB 只在满足指定条件时才在文件的指定行处暂停。如果您想在循环中进行一些迭代后再检查结果，则可以使用条件断点。

要设置条件断点，请在您要设置断点的可执行代码行处右键点击断点列，并选择“设置/修改条件”。

当编辑器对话框打开时，请输入相应条件并点击**确定**。条件是任何返回逻辑标量值的有效 MATLAB 表达式。

如对话框中说明的那样，MATLAB 会在运行该行之前计算该条件。例如，假设您有一个名为 **myprogram.m** 的文件。

```
myprogram.m
1 %Create an array of 10 ones.
2 x = ones(1,10);
3
4 %Perform a calculation on items 2-6 in the array
5 for n = 2:6
6 x(n) = 2 * x(n-1);
7 end
```

在第 6 行添加具有以下条件的断点：

```
n >= 4
```

该行的断点列中会显示一个黄色的条件断点图标。

您也可以使用 **dbstop** 函数以编程方式设置条件断点。例如，要在 **myprogram.m** 的第 6 行添加条件断点，请键入：

```
dbstop in myprogram at 6 if n>=4
```

当您运行文件时，MATLAB 会进入调试模式，并在满足条件时在该行暂停。在 **myprogram** 示例中，MATLAB 会运行两次 **for** 循环，并在 **n** 等于 4 进行第三次迭代时在第 6 行暂停。如果继续执行，MATLAB 会在 **n** 等于 5 进行第四次迭代时在第 6 行再暂停一次。

## 错误断点

如果使用错误断点，MATLAB 会在遇到问题时暂停执行程序并进入调试模式。<sup>⑧</sup>与标准断点和条件断点不同的是，您不用在具体文件中的具体行上设置这些断点。设置错误断点后，只要符合所指定的错误条件，MATLAB 便会在任意文件的任意行处暂停。MATLAB 随后会进入调试模式，打开包含错误的文件，并且在包含错误的行上会出现执行箭头。

要设置错误断点，请点击编辑器选项卡上的 运行，并从以下选项中进行选择：

- **出现错误时暂停**，即一遇到错误就暂停。
- **出现警告时暂停**，即一遇到警告就暂停。
- **出现 NaN 或 Inf 时暂停**，即在遇到 NaN (非数字) 或 Inf (无限) 值时暂停。

您也可以将 **dbstop** 函数与指定的 **condition** 结合使用，以编程方式设置断点。例如，要一遇到错误就暂停执行，请键入

```
dbstop if error
```

要在 **try/catch** 块的 **try** 部分内第一次发生运行时错误且消息 ID 为 **MATLAB:ls:InputsMustBeStrings** 时暂停执行，请键入

```
dbstop if caught error MATLAB:ls:InputsMustBeStrings
```

## 匿名函数中的断点

您可以在包含匿名函数的 MATLAB 代码行中设置多个断点。例如，您可以为行本身设置断点，MATLAB 软件将在该行开始处暂停。您也可以为该行中的每个匿名函数设置断点。

当您向包含匿名函数的行中添加断点时，编辑器会询问您要在该行中的哪个位置添加断点。如果某行中有多个断点，则无论该行中的任何断点的状态如何，断点图标都为蓝色。

要查看某行中所有断点的相关信息，请将光标悬停在断点图标上。随即会显示工具提示及可用信息。例如，在以下代码中，第 5 行包含两个匿名函数，每个匿名函数都有一个断点。工具提示会告诉我们两个断点均处于启用状态。

```

5 g = @(c) (integral(@(x) (x.^2 + c*x + 1), 0, 1));
6 % (2)
7 Line: 5, anonymous function 1. Status: enabled.
    Line: 5, anonymous function 2. Status: enabled.

```

当您在匿名函数中设置断点时，MATLAB 会在调用匿名函数时暂停。绿色箭头显示代码定义匿名函数的位置。白色箭头显示代码调用匿名函数的位置。例如，在以下代码中，MATLAB 在为匿名函数 `sqr` 设置的断点（位于名为 `myanonymous.m` 的文件中的第 2 行）处暂停该程序。白色箭头指示 `sqr` 函数是从第 3 行调用的。

```
1 function myanonymous
2 %> sqr = @(x) x.^2;
3 -> sqr(5);
4 end
```

## 无效断点

灰色断点表示无效断点。

```
1 % Create an array of 10 ones.
2 %> x = ones(1,10);
```

出现以下情况时，断点无效：

- 文件中有未保存的更改。保存该文件以使断点有效。灰色断点变为红色，表示它们现在变为有效了。
- 文件中存在语法错误。当您设置断点时，会显示一则错误消息，指示语法错误的位置。修复语法错误并保存文件以使断点有效。

## 禁用断点

您可以禁用所选断点以便让程序暂时忽略它们而不中断运行。例如，您可以在您认为发现并更正了问题后或使用条件断点时禁用断点。

要禁用断点，请右键点击断点图标，从上下文菜单中选择禁用断点。

在断点图标中会显示一个 X，以指示该断点已禁用。

```
6 %> | x(n) = 2 * x(n - 1);
```

要重新启用断点，请右键点击断点图标，从上下文菜单中选择启用断点。

X 不再显示在断点图标上，程序执行将在该行暂停。

要启用或禁用文件中的所有断点，请选择启用文件中的所有断点或禁用文件中的所有断点。只有当至少有一个断点可供启用或禁用时，才能使用这些选项。

## 清除断点

所有断点都保留在文件中，直到您清除（删除）它们为止，或直到它们在您的 MATLAB 会话结束时自动被清除为止。

要清除断点，请右键点击断点图标，然后从上下文菜单中选择清除断点。您也可以使用 F12 键清除断点。

要以编程方式清除断点，请使用 `dbclear` 函数。例如，要清除名为 `myprogram.m` 的文件中第 6 行处的断点，请键入

**dbclear in myprogram at 6**

要清除文件中的所有断点，请右键点击断点列，然后选择**清除文件中的所有断点**。您也可以使用 **dbclear all** 命令。例如，要在名为 **myprogram.m** 的文件中清除所有断点，请键入

**dbclear all in myprogram**

要清除所有文件中的全部断点（包括错误断点），请右键点击断点列，然后选择**全部清除**。您也可以使用 **dbclear all** 命令。

在您终止 MATLAB 会话时，系统会自动清除断点。要保存断点以用于后续会话，请参阅 **dbstatus** 函数。

## 另请参阅

### 相关示例

- “调试 MATLAB 程序” (第 22-2 页)
- “调试时检验值” (第 22-11 页)

## 调试时检验值

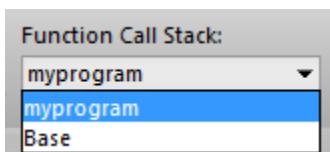
在程序暂停时，您可以查看当前在工作区中的任何变量的值。当您想查看某行代码是否产生了预期结果时，可以检查值。如果结果符合预期，请继续运行或进入下一行。如果结果不符合预期，则该行或上一行可能包含错误。

### 选择工作区

要在调试时检验变量，您必须首先选择其工作区。通过命令行窗口分配或使用脚本创建的变量属于基础工作区。在函数中创建的变量属于其自己的函数工作区。要查看当前工作区，请选择**编辑器**选项卡。**函数调用堆栈**字段将显示当前工作区。您也可以在命令行窗口中使用 **dbstack** 函数。

要选择或更改您要查看的变量的工作区，请使用以下方法之一：

- 在调试部分的**编辑器**选项卡中，从**函数调用堆栈**菜单列表中选择一个工作区。



- 在命令行窗口中，使用 **dbup** 和 **dbdown** 函数可以选择函数调用堆栈中的上一或下一工作区。

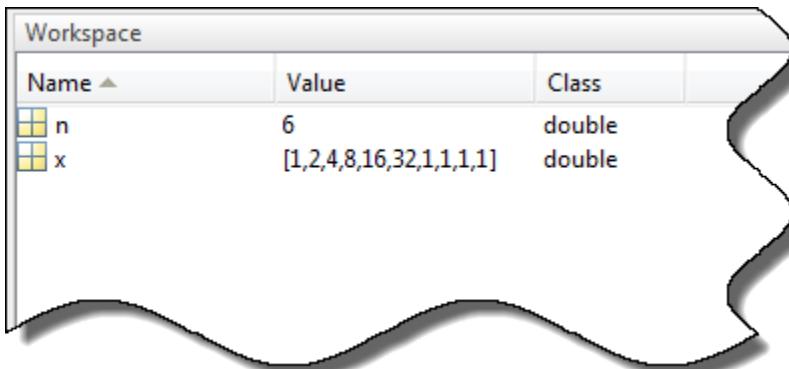
要列出当前工作区中的变量，请使用 **who** 或 **whos**。

### 查看变量值

在调试程序时有多种方法可查看变量的值：

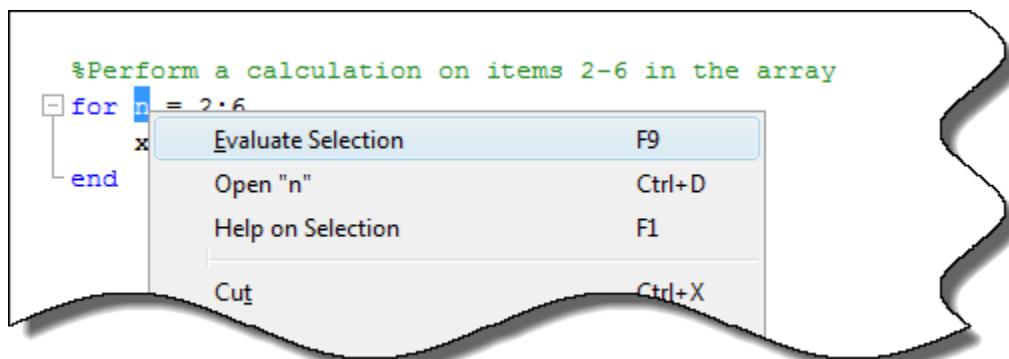
- 在工作区浏览器和变量编辑器中查看变量值。

工作区浏览器在当前工作区中显示所有变量。工作区浏览器的**值**列中显示变量的当前值。要查看更多详细信息，请双击该变量。变量编辑器随即打开，并显示该变量的内容。您也可以使用 **openvar** 函数在变量编辑器中打开变量。



- 在 MATLAB 编辑器中查看变量值。

使用鼠标选择变量或等式。右键点击并从上下文菜单中选择**执行所选内容**。命令行窗口中将显示该变量或方程的值。



**注意** 您不能在 MATLAB 忙碌时（例如在运行文件时）执行所选内容。

- 在 MATLAB 编辑器中以数据提示形式查看变量值。

要执行此操作，请将鼠标指针悬停在该变量上方。该变量的当前值将显示在数据提示中。数据提示会一直在视图中，直到您移动指针。如果在显示数据提示时遇到问题，请点击包含该变量的行，然后将指针移至靠近该变量。

```

for n = 2:6
    n: 1x1 double =
        6
end

```

在编辑器中调试文件时始终启用数据提示。在编辑器中编辑文件时，要查看数据提示，请在 MATLAB 预设中启用数据提示。

- 在主页选项卡上的环境部分中，点击 预设。然后，选择 MATLAB > 编辑器/调试器 > 显示。
- 在常规显示选项下，选择**在编辑模式下启用数据提示**。
- 在命令行窗口中查看变量值。

要查看当前位于工作区中的所有变量，请调用 **who** 函数。要查看变量的当前值，请在命令行窗口中键入该变量的名称。例如，要查看变量 **n** 的值，请键入 **n**，然后按 **Enter**。命令行窗口会显示变量名及其值。

如果您在函数中设置了断点，然后尝试在父工作区中查看变量的值，则该变量的值可能不可用。如果您尝试在 MATLAB 正覆盖变量时访问该变量，则会发生此错误。在此类情况下，MATLAB 会返回以下消息，其中 **x** 表示您要尝试检验其值的变量。

```
K>> x
Reference to a called function result under construction x.
```

无论您是使用 **dbup** 命令还是使用编辑器选项卡中调试部分的**函数调用堆栈**字段选择父工作区，都会发生此错误。

## 另请参阅

### 相关示例

- “调试 MATLAB 程序” (第 22-2 页)
- “设置断点” (第 22-6 页)



# 显示 MATLAB 代码

---

使用 MATLAB 软件可以通过多种方式显示您的 MATLAB 代码。您可以与他人共享您的代码和结果，即使他们没有 MATLAB 软件也如此。您可以将 MATLAB 输出保存为多种格式，包括 HTML、XML 和 LaTeX。如果您的 Microsoft Windows 系统上安装了 Microsoft Word 或 Microsoft PowerPoint 应用程序，也可以发布为这些格式。

- “发布和共享 MATLAB 代码”（第 23-2 页）
- “发布标记”（第 23-5 页）
- “用于发布的输出预设”（第 23-20 页）

## 发布和共享 MATLAB 代码

MATLAB 提供多种向其他人展示代码的选项，包括使用发布功能，以及在实时编辑器中创建实时脚本和函数。

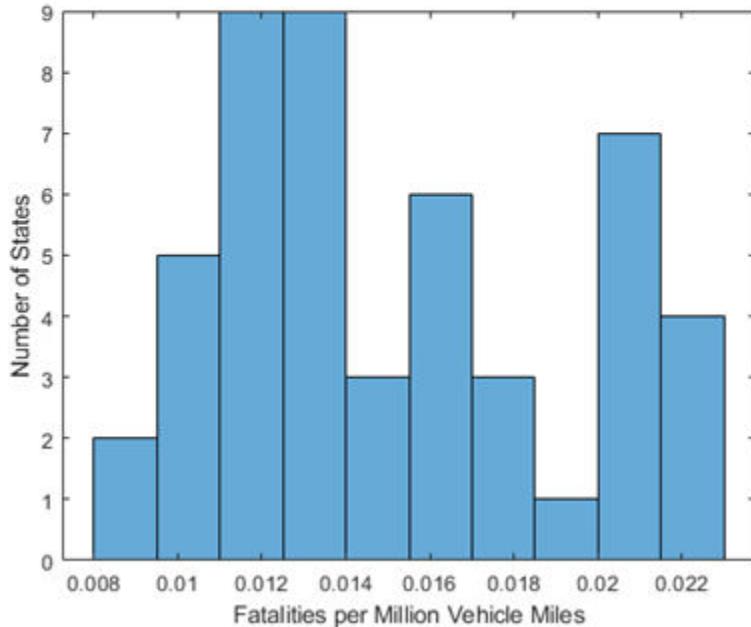
### 在实时编辑器中创建和共享实时脚本

要创建包含可执行 MATLAB 代码、嵌入式输出和格式化文本的可共享综合文档，最简单的方法是使用实时编辑器。支持的输出格式包括：MLX、PDF、Microsoft Word、HTML 和 LaTeX。有关详细信息，请参阅“在实时编辑器中创建实时脚本”（第 19-6 页）。

#### Distribution of Fatalities

We can use a bar chart to see the distribution of fatality rates among the states. There are 11 states that have a fatality rate greater than 0.02 per million vehicle miles.

```
histogram(rate,10)
xlabel('Fatalities per Million Vehicle Miles')
ylabel('Number of States')
```



## 发布 MATLAB 代码

发布 MATLAB 代码文件 (.m) 可创建包括您的代码、注释和输出的格式化文档。发布代码的常见原因是与其他人共享文档以用于教学或演示，或者生成您代码的可读外部文档。

此代码演示了方波的傅里叶级数展开式。

包含标记的 MATLAB 代码	发布的文档
<pre>%% Square Waves from Sine Waves % The Fourier series expansion for a square-wave is % made up of a sum of odd harmonics, as shown here % using MATLAB(R).  %% Add an Odd Harmonic and Plot It t = 0:1*pi/4; y = sin(t); plot(t,y);  %%  % In each iteration of the for loop add an odd % harmonic to y. As _k_ increases, the output % approximates a square wave with increasing accuracy.  for k = 3:2:9     %     % Perform the following mathematical operation     % at each iteration:     %     % \$\$ y = y + \frac{\sin(kt)}{k} \$\$     %     % Display every other plot:     %     y = y + sin(k*t)/k;     if mod(k,4)==1         display(sprintf('When k = %.lf',k));         display('Then the plot is:');         cla         plot(t,y)     end end  %% Note About Gibbs Phenomenon % Even though the approximations are constantly % improving, they will never be exact because of the % Gibbs phenomenon, or ringing.</pre>	<p><b>Square Waves from Sine Waves</b></p> <p>The Fourier series expansion for a square-wave is made up of a sum of odd harmonics, as shown here using MATLAB.</p> <p><b>Contents</b></p> <ul style="list-style-type: none"> <li>Add an Odd Harmonic and Plot It</li> <li>Note About Gibbs Phenomenon</li> </ul> <p><b>Add an Odd Harmonic and Plot It</b></p> <pre>t = 0:1*pi/4; y = sin(t); plot(t,y);</pre> <p>In each iteration of the for loop add an odd harmonic to y. As k increases, the output approximates a square wave with increasing accuracy.</p> <p><b>for k = 3:2:9</b></p> <p>Perform the following mathematical operation at each iteration:</p> $y = y + \frac{\sin(kt)}{k}$

要发布您的代码，请执行以下操作：

- 创建一个 MATLAB 脚本或函数。通过在每个部分的开头插入两个百分比符号 (%%) 来将代码划分为多个步骤或节。
- 通过在文件开头及每节中添加说明性注释来记录代码。

在各节顶部的注释中，您可以添加标记来增强输出的可读性。例如，上表中的代码包括以下标记。

标题	<b>%% Square Waves from Sine Waves</b> <b>%% Add an Odd Harmonic and Plot It</b> <b>%% Note About Gibbs Phenomenon</b>
斜体格式的变量名称	<b>% As <i>k</i> increases, ...</b>
LaTeX 方程	<b>% \$\$ y = y + \frac{\sin(kt)}{k} \$\$</b>

**注意** 如果您文件中的文本所包含的字符编码与您的平台编码不同，那么在您保存或发布文件时，MATLAB 会将这些字符显示为乱码。

- 发布代码。在**发布**选项卡上，点击**发布**。

默认情况下，MATLAB 为您的代码创建的每个图形创建一个名为 **html** 的子文件夹，其中包含一个或多个 HTML 文件。HTML 文件包括代码、格式化注释和输出。您也可以发布为其他格式，例如 PDF 文件或 Microsoft PowerPoint 演示文稿。有关发布为其他格式的详细信息，请参阅“指定输出文件”（第 23-21 页）。

在 MATLAB Online 中，MATLAB 发布为 HTML 或 PDF 格式，并将输出和支持文件存储在您的 **Published** 文件夹中。要允许 MATLAB 自动打开输出，请在 Web 浏览器中启用弹出窗口。结果可以使用 <https://matlab.mathworks.com/users/userid/Published/filename/index.html> 形式的 URL 公开访问。

显示在前面的图窗中的示例代码是已安装文档的一部分。您可以通过运行此命令在编辑器中查看该代码：

```
edit(fullfile(matlabroot,'help','techdoc','matlab_env',...
    'examples','fourier_demo2.m'))
```

### 添加帮助和创建文档

您可以通过在 MATLAB 代码文件的开头插入注释来为代码添加帮助。当您在命令行窗口中键入 `help file_name` 时，MATLAB 会显示帮助注释。有关详细信息，请参阅“为程序添加帮助”（第 20-5 页）。

您也可以创建自己的 MATLAB 文档主题，以便从 MATLAB 帮助浏览器或 Web 查看。有关详细信息，请参阅“显示自定义文档”（第 31-20 页）。

### 另请参阅

`publish`

### 详细信息

- “在实时编辑器中创建实时脚本”（第 19-6 页）
- “发布标记”（第 23-5 页）
- “用于发布的输出预设”（第 23-20 页）

### 外部网站

- 从编辑器发布 MATLAB 代码（视频）

# 发布标记

本节内容
“标记概述”（第 23-5 页）
“节和节标题”（第 23-7 页）
“文本格式设置”（第 23-8 页）
“项目符号和编号列表”（第 23-9 页）
“文本和代码块”（第 23-9 页）
“外部文件内容”（第 23-10 页）
“外部图形”（第 23-11 页）
“图像快照”（第 23-12 页）
“LaTeX 方程”（第 23-13 页）
“超链接”（第 23-14 页）
“HTML 标记”（第 23-17 页）
“LaTeX 标记”（第 23-17 页）

## 标记概述

要插入标记，您可以：

- 使用**发布**选项卡上的格式设置按钮和下拉菜单设置文件格式。此方法可自动为您插入文本标记。
- 从右键点击菜单的**插入文本标记**列表中选择标记。
- 直接在注释中键入标记。

下表提供文本标记选项的摘要说明。如果您未使用 MATLAB 编辑器，或者不希望使用**发布**选项卡应用标记，可参考此表。

---

**注意** 处理标记时：

- 注释符号 (%) 后面的空格通常确定随后的文本的格式。
  - 开始新标记通常需要前面的空注释行，如示例中所示。
  - 标记仅在紧跟有分节符的注释中有效。
- 

输出结果	对应文件标记的示例
“节和节标题”（第 23-7 页）	<pre>%&gt; SECTION TITLE % DESCRIPTIVE TEXT  %%% SECTION TITLE WITHOUT SECTION BREAK % DESCRIPTIVE TEXT</pre>

输出结果	对应文件标记的示例
“文本格式设置” (第 23-8 页)	<pre>%_ITALIC TEXT_ %*BOLD TEXT*   MONOSPACED TEXT    % Trademarks: % TEXT(TM)  % TEXT(R)</pre>
“项目符号和编号列表” (第 23-9 页)	<pre>%% Bulleted List % %* BULLETED ITEM 1 %* BULLETED ITEM 2 %  %% Numbered List % %# NUMBERED ITEM 1 %# NUMBERED ITEM 2 %</pre>
“文本和代码块” (第 23-9 页)	<pre>%% % % PREFORMATTED % TEXT %  %% MATLAB(R) Code % % for i = 1:10 % disp x % end %</pre>
“外部文件内容” (第 23-10 页)	<pre>%&lt;include&gt;filename.m&lt;/include&gt; %</pre>
“外部图形” (第 23-11 页)	<pre>%&lt;&gt;FILENAME.PNG&gt;&gt; %</pre>
“图像快照” (第 23-12 页)	<pre>snapshot;</pre>
“LaTeX 方程” (第 23-13 页)	<pre>%% Inline Expression % \$x^2+e^{\pi i}\$ % %% Block Equation % % \$\$e^{\pi i} + 1 = 0\$\$ %</pre>
“超链接” (第 23-14 页)	<pre>% &lt;https://www.mathworks.com MathWorks&gt; % &lt;matlab:FUNCTION DISPLAYED_TEXT&gt;</pre>

输出结果	对应文件标记的示例
"HTML 标记" (第 23-17 页)	<pre>%&lt;html&gt; %&lt;table border=1&gt;&lt;tr&gt; %&lt;td&gt;one&lt;/td&gt; %&lt;td&gt;two&lt;/td&gt;&lt;/tr&gt;&lt;/table&gt; %&lt;/html&gt; %</pre>
"LaTeX 标记" (第 23-17 页)	<pre>%% LaTeX Markup Example %&lt;latex&gt; % \begin{tabular}{  r   r  } % \hline \$n\$ &amp; \$n!\$ \\ % \hline 1&amp;1\ 2&amp;2\ 3&amp;6\ \\ % \hline % \end{tabular} %&lt;/latex&gt; %</pre>

## 节和节标题

通过代码节可以整理、添加注释以及执行部分代码。代码节以双百分号 (%%) 开头，后可跟节标题。节标题使用较大的加粗字体显示为顶层标题 (HTML 中为 h1)。

---

**注意** 您可以在紧随标题后的行中添加注释。但是，如果您需要整体文档的标题，则不能在下一节开头 (以 %% 开头的一行) 之前添加任何 MATLAB 代码。

---

例如，此代码在发布时生成美观的结果。

```
%% Vector Operations
% You can perform a number of binary operations on vectors.
%%
A = 1:3;
B = 4:6;
%% Dot Product
% A dot product of two vectors yields a scalar.
% MATLAB has a simple command for dot products.
s = dot(A,B);
%% Cross Product
% A cross product of two vectors yields a third
% vector perpendicular to both original vectors.
% Again, MATLAB has a simple command for cross products.
v = cross(A,B);
```

通过在编辑器中保存代码并点击**发布**选项卡上的**发布**按钮，MATLAB 可生成此图窗所示的输出。请注意，MATLAB 可从 MATLAB 文件中的节标题自动插入“内容”菜单。

### Vector Operations

You can perform a number of binary operations on vectors.

#### Contents

- Dot Product
- Cross Product

```
A = 1:3;
B = 4:6;
```

#### Dot Product

A dot product of two vectors yields a scalar. MATLAB has a simple command for dot products.

```
s = dot(A,B);
```

#### Cross Product

A cross product of two vectors yields a third vector perpendicular to both original vectors. Again, MATLAB has a simple command for cross products.

```
v = cross(A,B);
```

## 文本格式设置

您可以标记 MATLAB 注释中的选定文本，以便在您发布文件时使其显示为斜体、粗体或等宽文本。只需用 \_、**\*** 或 **|** 括起文本即可使其分别显示为斜体、粗体或等宽文本。

例如，以下行在发布后显示每个文本格式设置语法。

```
%% Calculate and Plot Sine Wave
% Define the *range* for |x|
```

### Calculate and Plot Sine Wave

*Define the range for x*

## 商标符号

如果您的 MATLAB 文件中的注释包括带商标的项，您可以包括文本以在输出中生成商标符号 (™) 或注册商标符号 (®)。只需紧随相关术语后添加 (R) 或 (TM)，之间不需要任何空格。

例如，假定您在一个文件中输入以下行。

```
%% Basic Matrix Operations in MATLAB(R)
% This is a demonstration of some aspects of MATLAB(R)
% and the Symbolic Math Toolbox(TM).
```

如果您将文件发布为 HTML，它会显示在 MATLAB Web 浏览器中。

## Basic Matrix Operations in MATLAB®

This is a demonstration of some aspects of MATLAB® and the Symbolic Math Toolbox™.

## 项目符号和编号列表

MATLAB 允许在注释中使用项目符号列表和编号列表。您可以使用此语法生成项目符号列表和编号列表。

```
%% Two Lists  
%  
% * ITEM1  
% * ITEM2  
%  
% # ITEM1  
% # ITEM2  
%
```

发布该示例代码将生成以下输出。

### Two Lists

- ITEM1
  - ITEM2
- 
1. ITEM1
  2. ITEM2

## 文本和代码块

### 预设格式的文本

预设格式的文本显示为等宽字体，保留空白，且长行不换行。预设格式文本第一行的文本与注释符号之间必须留有两个空格。

发布此代码将生成预设格式的段落。

```
%%  
% Many people find monospaced texts easier to read:  
%  
% A dot product of two vectors yields a scalar.  
% MATLAB has a simple command for dot products.
```

Many people find monospaced texts easier to read:

A dot product of two vectors yields a scalar.  
MATLAB has a simple command for dot products.

### 语法高亮显示的示例代码

可执行代码显示在已发布文档中时语法会高亮显示。也可以高亮显示示例代码。示例代码是显示在注释中的代码。

要指示示例代码，您必须在注释符号与第一行代码的开头之间放置三个空格。例如，点击[发布](#)选项卡上的[代码](#)按钮可在您的编辑器中插入以下示例代码。

```
%%  
%  
% for i = 1:10  
%     disp(x)  
% end  
%
```

将此代码发布为 HTML 可在 MATLAB Web 浏览器中生成输出。

```
for i = 1:10  
    disp(x)  
end
```

### 外部文件内容

要将外部文件内容添加到 MATLAB 发布的代码中，请使用 `<include>` 标记。指定外部文件路径（相对于所发布文件的位置）。所包含的 MATLAB 代码文件将发布为语法高亮显示代码。其他所有文件则发布为纯文本。

例如，此代码会将 `sine_wave.m` 的内容插入到发布的输出中：

```
%% External File Content Example  
% This example includes the file contents of sine_wave.m into published  
% output.  
%  
% <include>sine_wave.m</include>  
%  
% The file content above is properly syntax highlighted
```

将文件发布为 HTML。

### External File Content Example

This example includes the file contents of sine\_wave.m into published output.

```
% Define the range for x.
% Calculate and plot y = sin(x).
x = 0:1/6*pi;
y = sin(x);
plot(x,y)
title('Sine Wave')
xlabel('x')
ylabel('sin(x)')
fig = gcf;
fig.MenuBar = 'none';
```

The file content above is properly syntax highlighted

## 外部图形

要发布 MATLAB 代码无法生成的图像，请使用文本标记。默认情况下，MATLAB 已包括代码生成的图形。

此代码将在发布的输出中插入名为 FILENAME.PNG 的普通图像。

```
%%%
%
% <<FILENAME.PNG>>
%
```

MATLAB 要求 FILENAME.PNG 是从输出位置到您的外部图像的相对路径，或者是完全限定 URL。最好将您的图像保存在 MATLAB 发布其输出的同一文件夹中。例如，MATLAB 将 HTML 文档发布到子文件夹 html。将您的图像文件保存在同一子文件夹中。您可以通过更改发布配置设置来更改输出文件夹。在 MATLAB Online 中，将您的图像文件保存到位于根文件夹下的 Published 文件夹中。

### 使用 surf(peaks) 的外部图形示例

此示例演示如何将 surfpeaks.jpg 插入到 MATLAB 文件中以便发布。

要创建 surfpeaks.jpg，请在命令行窗口中运行以下代码。

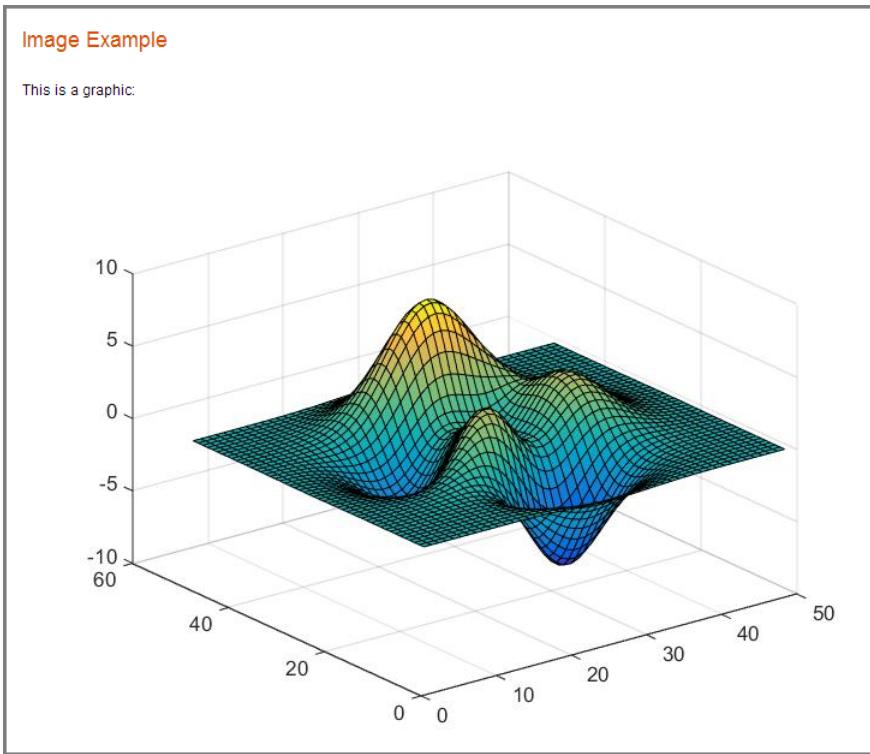
```
saveas(surf(peaks),'surfpeaks.jpg');
```

要从 MATLAB 文件生成包含 surfpeaks.jpg 的 HTML 文件，请执行以下操作：

- 1 在当前文件夹中创建一个名为 html 的子文件夹。
- 2 通过在命令行窗口中运行此代码创建 surfpeaks.jpg。

```
1 saveas(surf(peaks),'html/surfpeaks.jpg');
2 将此 MATLAB 代码发布为 HTML。
```

```
%%% Image Example
% This is a graphic:
%
% <<surfpeaks.jpg>>
%
```



### 输出文件格式的有效图像类型

您在发布文档时可以纳入的图像类型取决于下表中注明的文档输出类型。为获得最佳兼容性，最好为每种输出类型使用默认图像格式。

输出文件格式	默认图像格式	您可以纳入的图像类型
<b>doc</b>	<b>png</b>	所安装的 Microsoft Office 版本支持的任何格式。
<b>html</b>	<b>png</b>	可成功发布任何格式。确保您用于查看和处理输出文件的工具可以显示指定的输出格式。
<b>latex</b>	<b>png 或 epsc2</b>	可成功发布任何格式。确保您用于查看和处理输出文件的工具可以显示指定的输出格式。
<b>pdf</b>	<b>bmp</b>	<b>bmp 和 jpg。</b>
<b>ppt</b>	<b>png</b>	所安装的 Microsoft Office 版本支持的任何格式。
<b>xml</b>	<b>png</b>	可成功发布任何格式。确保您用于查看和处理输出文件的工具可以显示指定的输出格式。

### 图像快照

您可以插入捕捉您的 MATLAB 输出的快照的代码。例如，如果您有一个修改您要在每次迭代后捕获的图窗的 **for** 循环，这很有用。

下面的代码运行 **for** 循环三次并在每次迭代后生成输出。**snapnow** 命令捕获该代码生成的所有三个图像。

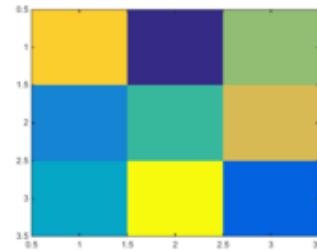
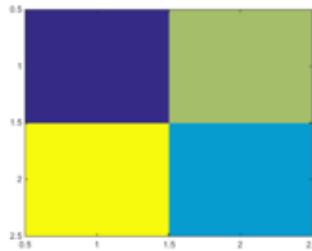
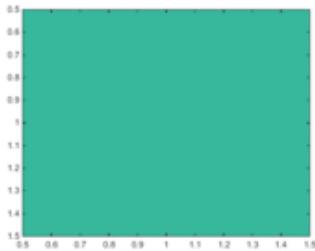
```
%% Scale magic Data and Display as Image
```

```
for i=1:3
    imagesc(magic(i))
    snapnow;
end
```

如果您将文件发布为 HTML，它将类似于以下输出。默认情况下，HTML 中的图像大于图窗中显示的图像。要调整 MATLAB 代码生成的图像的大小，请使用[发布设置](#)窗格中的**最大图像宽度**和**最大图像高度**字段，如“[用于发布的输出预设](#)”（第 23-20 页）中所述。

## Scale magic Data and Display as Image

```
for i=1:3
    imagesc(magic(i))
    snapnow;
end
```



## LaTeX 方程

### 行内 LaTeX 表达式

利用 MATLAB，您可以在想要发布的任何代码中包含行内 LaTeX 表达式。要插入内联表达式，请用美元符号字符 (\$) 括起您的 LaTeX 标记。\$ 必须紧靠在行内表达式第一个词的前面，以及紧跟在内联表达式最后一个词的后面，中间没有任何空格。

#### 注意

- 所有发布输出类型都支持 LaTeX 表达式，但 Microsoft PowerPoint 除外。
- MATLAB 发布支持标准 LaTeX 数学模式的指令。不支持文本模式的指令或需要额外包的指令。

此代码包含 LaTeX 表达式：

```
%% LaTeX Inline Expression Example
%
% This is an equation: $x^2+e^{\pi i}$. It is
% inline with the text.
```

如果将示例文本标记发布为 HTML，结果输出如下。

### LaTeX Inline Expression Example

This is an equation:  $x^2 + e^{\pi i}$ . It is inline with the text.

### LaTeX 行间方程

在 MATLAB 中，您可以在与主注释文本相离的文本块中插入 LaTeX 符号。方程每侧添加两个美元符号 (\$) 表示 LaTeX 方程块。在单独的块中发布方程要求块之间存在空行。

此代码是示例文本标记。

```
%% LaTeX Equation Example
%
% This is an equation:
%
% $$e^{\pi i} + 1 = 0$$
%
% It is not in line with the text.
```

如果您发布为 HTML，表达式如下所示。

### LaTeX Equation Example

This is an equation:

$$e^{\pi i} + 1 = 0$$

It is not in line with the text.

## 超链接

### 静态超链接

您可以在 MATLAB 注释中插入静态超链接，然后将文件发布为 HTML、XML 或 Microsoft Word 文件。在指定指向 Web 位置的静态超链接时，请在代码中包括完整的 URL。当您希望指引读者访问 Web 位置时，这很有用。您可以在已发布文本中显示或隐藏 URL。如果您确信读者是在线查看您的输出，并可以点击该超链接，则可考虑不包括 URL。

将 URL 和任何替代文本包括在尖括号中。

```
%%
% For more information, see our web site:
% <https://www.mathworks.com MathWorks>
```

将该代码发布为 HTML 将生成以下输出。

For more information, see our Web site: [MathWorks](#)

消除 URL 后面的文本 **MathWorks** 可生成以下经过修改的输出。

For more information, see our Web site: <http://www.mathworks.com>

**注意** 如果您的代码在 MATLAB 命令行窗口中生成带超链接的文本，则输出显示 HTML 代码而非超链接。

### 动态超链接

您可以插入动态超链接，MATLAB 在读者点击该链接时执行它。通过动态超链接，您可指引读者访问 MATLAB 代码或文档，或使读者能够运行代码。您可以使用 **matlab:** 语法实现这些链接。如果紧随 **matlab:** 声明后的代码包含空格，则将其替换为 %20。

**注意** 仅当在 MATLAB Web 浏览器中查看 HTML 时，动态链接才有效。

动态链接的不同用法包括：

- “用于运行代码的动态链接”（第 23-15 页）
- “指向文件的动态链接”（第 23-16 页）
- “指向 MATLAB 函数参考页的动态链接”（第 23-16 页）

#### 用于运行代码的动态链接

您可以指定动态超链接以在用户点击超链接时运行代码。例如，下面的 **matlab:** 语法在输出中创建超链接，点击该超链接时可启用或禁用回收：

```
%% Recycling Preference  
% Click the preference you want:  
%  
% <matlab.recycle('off') Disable recycling>  
%  
% <matlab.recycle('on') Enable recycling>
```

发布的结果如以下 HTML 输出所示。

### Recycling Preference

Click the preference you want:

[Disable recycling](#)

[Enable recycling](#)

当您点击其中一个超链接时，MATLAB 会相应地设置 `recycle` 命令。点击超链接后，在命令行窗口中运行 `recycle` 以确认设置是否符合您的预期。

### 指向文件的动态链接

您可以指定一个文件链接（您知道该文件位于您读者的 `matlabroot` 中）。您不需要知道每个读者安装 MATLAB 的位置。例如，指向 `publish` 函数代码的链接。

```
%%  
% See the  
% <matlab:edit(fullfile(matlabroot,'toolbox','matlab','codetools','publish.m)) code>  
% for the publish function.
```

接下来，将文件发布为 HTML。

See the [code](#) for the publish function.

当您点击 `code` 链接时，MATLAB 编辑器随即会打开并显示 `publish` 函数的代码。在读者的系统上，MATLAB 发出该命令（但该命令不显示在读者的命令行窗口中）。

### 指向 MATLAB 函数参考页的动态链接

您可以使用 `matlab:` 语法指定指向 MATLAB 函数参考页的链接。例如，假设您的读者已安装并正在运行 MATLAB。提供指向 `publish` 参考页的链接。

```
%%  
% See the help for the <matlab:doc('publish') publish> function.
```

将文件发布为 HTML。

See the help for the [publish](#) function.

当您点击 `publish` 超链接时，MATLAB 帮助浏览器随即会打开并显示 `publish` 函数的参考页。在读者的系统上，MATLAB 发出该命令（但该命令不显示在命令行窗口中）。

## HTML 标记

您可以将 HTML 标记插入到您的 MATLAB 文件中。您必须键入 HTML 标记，因为**发布**选项卡上没有可生成该标记的按钮。

**注意** 当您插入 HTML 代码的文本标记时，HTML 代码仅在指定的输出文件格式为 HTML 时才发布。

以下代码包括 HTML 标记。

```
%% HTML Markup Example
% This is a table:
%
% <html>
% <table border=1><tr><td>one</td><td>two</td></tr>
% <tr><td>three</td><td>four</td></tr></table>
% </html>
%
```

如果您将代码发布为 HTML，MATLAB 会创建一个包含两列的单行表。该表包含值 **one**、**two**、**three** 和 **four**。

HTML Markup Example

This is a table:

one	two
three	four

如果某节生成以 `<html>` 开头并以 `</html>` 结尾的命令行窗口输出，则 MATLAB 在发布的输出中包括 HTML 源代码。例如，MATLAB 显示 `disp` 命令并根据 HTML 代码（如果您发布此代码）生成表：

```
disp('<html><table><tr><td>1</td><td>2</td></tr></table></html>')
```

```
disp('<html><table><tr><td>1</td><td>2</td></tr></table></html>')
```

1	2
---	---

## LaTeX 标记

您可以将 LaTeX 标记插入到您的 MATLAB 文件中。您必须键入所有 LaTeX 标记，因为**发布**选项卡上没有可生成该标记的按钮。

**注意** 当您插入 LaTeX 代码的文本标记时，该代码仅在指定的输出文件格式为 LaTeX 时才发布。

---

以下代码是 LaTeX 标记的一个示例。

```
%% LaTeX Markup Example
% This is a table:
%
% <latex>
% \begin{tabular}{|c|c|} \hline
% $n$ & $n!$ \\ \hline
% 1 & 1 \\
% 2 & 2 \\
% 3 & 6 \\ \hline
% \end{tabular}
% </latex>
```

如果您将文件发布为 LaTeX，则编辑器会打开一个新的包含 LaTeX 标记的 .tex 文件。

```
% This LaTeX was auto-generated from MATLAB code.
% To make changes, update the MATLAB code and republish this document.
```

```
\documentclass{article}
\usepackage{graphicx}
\usepackage{color}

\sloppy
\definecolor{lightgray}{gray}{0.5}
\setlength{\parindent}{0pt}

\begin{document}

\section*{LaTeX Markup Example}

\begin{par}
This is a table:
\end{par} \vspace{1em}
\begin{par}

\begin{tabular}{|c|c|} \hline
$n$ & $n!$ \\ \hline
1 & 1 \\
2 & 2 \\
3 & 6 \\ \hline
\end{tabular}
\end{par} \vspace{1em}

\end{document}
```

MATLAB 包括使用 LaTeX 程序编译此文件所需的任何其他标记。

## 另请参阅

### 详细信息

- “发布和共享 MATLAB 代码”（第 23-2 页）
- “用于发布的输出预设”（第 23-20 页）

## 用于发布的输出预设

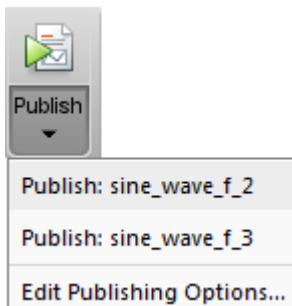
### 本节内容

- “如何编辑发布选项”（第 23-20 页）
- “指定输出文件”（第 23-21 页）
- “在发布过程中运行代码”（第 23-21 页）
- “在发布输出时操作图形”（第 23-23 页）
- “保存发布设置”（第 23-25 页）
- “管理发布配置”（第 23-26 页）

### 如何编辑发布选项

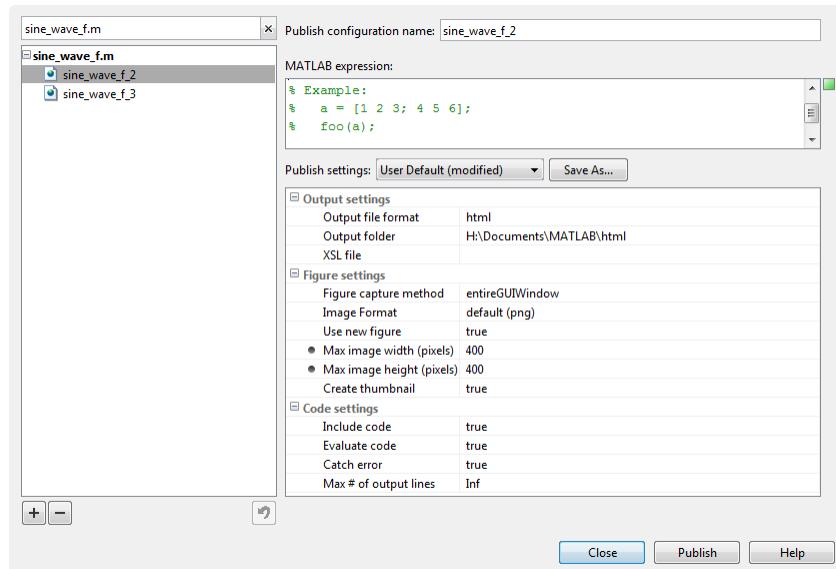
如果您的代码不需要任何输入参数并且您希望发布为 HTML，则使用默认发布预设。但是，如果您的代码需要输入参数，或者如果您希望指定输出设置、代码执行或图窗格式，则指定自定义配置。

- 1 找到发布选项卡并点击发布按钮箭头 。



- 2 选择编辑发布选项。

“编辑配置”对话框随即打开。指定输出预设。



**MATLAB 表达式**窗格指定在发布期间执行的代码。发布设置窗格包含输出、图窗和代码执行选项。MATLAB 将所有这些称为发布配置。MATLAB 将每个发布配置与 .m 文件相关联。发布配置的名称显示在左上窗格中。

## 指定输出文件

您在**发布设置**窗格上指定输出格式和位置。

MATLAB 发布为这些格式。

格式	注释
html	发布为 HTML 文档。您可以使用可扩展样式表语言 (XSL) 文件。
xml	发布为 XML 文档。您可以使用可扩展样式表语言 (XSL) 文件。
latex	发布为 LaTeX 文档。不保留语法高亮。您可以使用可扩展样式表语言 (XSL) 文件。
doc	发布至 Microsoft Word 文档。不保留语法高亮。此格式仅在 Windows 平台上提供。
ppt	发布至 Microsoft PowerPoint 文档。不保留语法高亮。此格式仅在 Windows 平台上提供。
pdf	发布为 PDF 文档。

---

**注意** XSL 文件允许您对输出文档的外观进行更多控制。有关详细信息，请参阅 <http://docbook.sourceforge.net/release/xsl/current/doc/>。

---

## 在发布过程中运行代码

- “指定代码” (第 23-21 页)
- “评估代码” (第 23-21 页)
- “包括代码” (第 23-22 页)
- “捕获错误” (第 23-22 页)
- “限制输出量” (第 23-22 页)

### 指定代码

默认情况下，MATLAB 执行您要发布的 .m 文件。但是，您可以在 **MATLAB 表达式**窗格中指定任何有效 MATLAB 代码。例如，如果您要发布的函数需要输入，则运行命令 `function(input)`。对于其他您要发布其输出的代码，它们会出现在函数调用后。如果您清除 **MATLAB 表达式**区域，则 MATLAB 发布文件而不执行任何代码。

---

**注意** 发布配置使用基础 MATLAB 工作区。因此，**MATLAB 表达式**窗格中的变量会覆盖基础工作区中现有变量的值。

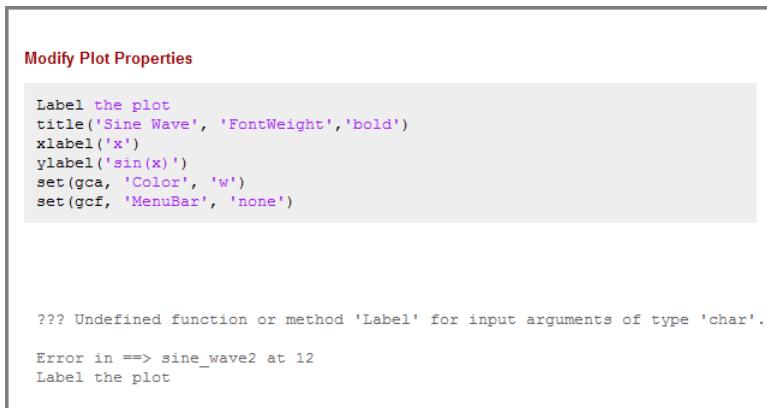
---

### 评估代码

影响 MATLAB 在发布期间执行的内容的另一种方式是设置**发布设置**窗格中的**评估代码**选项。该选项指示 MATLAB 是否评估要发布的 .m 文件中的代码。如果设置为 `true`，MATLAB 会执行该代码并将结果包括在输出文档中。

因为当您将**评估代码**选项设置为 `false` 时，MATLAB 既不评估代码也不包括代码结果，所以文件中可能有无效代码。因此，请首先考虑运行该文件并将该选项设置为 `true`。

例如，假设您在文件中包括注释文本 `Label the plot`，但忘记在其之前添加注释字符。如果您将文档发布为 HTML 并将**计算代码**选项设置为 `true`，则输出包括错误。



The screenshot shows the 'Modify Plot Properties' dialog box. In the code editor area, there is a MATLAB script:

```
Label the plot
title('Sine Wave', 'FontWeight','bold')
xlabel('x')
ylabel('sin(x)')
set(gca, 'Color', 'w')
set(gcf, 'MenuBar', 'none')
```

Below the code, an error message is displayed:

```
??? Undefined function or method 'Label' for input arguments of type 'char'.
Error in ==> sine_wave2 at 12
Label the plot
```

使用 `false` 选项发布包含 `publish` 函数的文件。否则，MATLAB 会尝试以递归方式发布该文件。

### 包括代码

您可以指定是否在最终输出中显示 MATLAB 代码。如果您将**包括代码**选项设置为 `true`，则 MATLAB 将该代码包括在发布的输出文档中。如果设置为 `false`，则 MATLAB 在除 HTML 之外的所有输出文件格式中排除该代码。

如果输出文件格式为 HTML，则 MATLAB 将代码作为 Web 浏览器中不可见的 HTML 注释插入。如果您想从输出 HTML 文件中提取该代码，请使用 MATLAB `grabcode` 函数。

例如，假设您使用**包含代码**选项设置为 `false` 的发布配置将 `H:/my_matlabfiles/my_mfiles/sine_wave.m` 发布为 HTML。如果您与同事共享输出，他们可以在 Web 浏览器中查看它。要查看生成输出的 MATLAB 代码，他们可以从包含 `sine_wave.html` 的文件夹发出以下命令：

```
grabcode('sine_wave.html')
```

MATLAB 在编辑器中打开创建了 `sine_wave.html` 的文件。

### 捕获错误

您可以捕获和发布在发布期间出现的任何错误。将**捕获错误**选项设置为 `true` 可在输出文档中包括任何错误消息。如果您将**捕获错误**设置为 `false`，MATLAB 会在代码评估期间出现错误时终止发布操作。但是，如果您将**评估代码**属性设置为 `false`，则此选项没有任何影响。

### 限制输出量

您可以限制包括在输出文档中的代码输出的行数，方法是指定**发布设置**窗格中的**最大输出行数量**选项。如果一个较小的代表性代码输出示例就能满足需求，则设置此选项很有用。

例如，以下循环在发布的输出中生成 100 行，除非**最大输出行数量**设置为较小的值。

```
for n = 1:100
    disp(x)
end;
```

## 在发布输出时操作图形

- “选择图像格式”（第 23-23 页）
- “设置图像大小”（第 23-23 页）
- “捕获图窗”（第 23-23 页）
- “指定自定义图窗窗口”（第 23-24 页）
- “创建缩略图”（第 23-25 页）

### 选择图像格式

发布时，您可以选择 MATLAB 用于存储代码执行期间生成的任何图形的图像格式。下拉列表中可用的图像格式取决于**图窗捕获方法**选项的设置。为获得最大兼容性，请选择此表中指定的默认值。

输出文件格式	默认图像格式	您可以纳入的图像类型
doc	png	所安装的 Microsoft Office 版本支持的任何格式。
html	png	可成功发布任何格式。确保您用于查看和处理输出文件的工具可以显示指定的输出格式。
latex	png 或 epsc2	可成功发布任何格式。确保您用于查看和处理输出文件的工具可以显示指定的输出格式。
pdf	bmp	bmp 和 jpg。
ppt	png	所安装的 Microsoft Office 版本支持的任何格式。
xml	png	可成功发布任何格式。确保您用于查看和处理输出文件的工具可以显示指定的输出格式。

### 设置图像大小

您可在“编辑配置”对话框窗口的**发布设置**窗格中设置 MATLAB 生成的图像的大小。您可指定图像大小（以像素为单位）以在输出中限制图像的宽度和高度。像素值充当最大大小，这是因为 MATLAB 保持图像的纵横比。在以下情况下，MATLAB 忽略大小设置：

- 当按“外部图形”（第 23-11 页）中所述使用外部图形时
- 当使用向量格式（例如 .eps）时
- 当发布为 .pdf 时

### 捕获图窗

您可以通过设置**图窗捕获方法**选项捕获图窗窗口的不同方面。此选项确定窗口修饰（标题栏、工具栏、菜单栏和窗口边框）和图窗窗口的绘图背景。

此表汇总了各种图窗捕获方法的效果。

使用此图窗捕获方法	获取包含这些外观详细信息的图窗捕获	
	窗口修饰	绘图背景
entireGUIWindow	对话框包括；图窗不包括	对于图窗设置为白色；对于对话框与屏幕一致
print	对话框和图窗不包括	设置为白色
getframe	对话框和图窗不包括	与屏幕绘图背景一致

使用此图窗捕获方法	获取包含这些外观详细信息的图窗捕获	
	窗口修饰	绘图背景
entireFigureWindow	对话框和图窗包括	与屏幕绘图背景一致

**注意** 通常，MATLAB 图窗将 `HandleVisibility` 属性设置为 `on`。对话框是其 `HandleVisibility` 属性设置为 `off` 或 `callback` 的图窗。如果您的结果不同于上表中列出的结果，则您的图窗或对话框的 `HandleVisibility` 属性可能是非典型的。有关详细信息，请参阅 [HandleVisibility](#)。

### 指定自定义图窗窗口

MATLAB 允许您指定其创建的图窗的自定义外观。如果发布设置窗格中的使用新图窗选项设置为 `true`，则在发布的输出中，MATLAB 使用默认大小且具有白色背景的图窗窗口。如果使用新图窗选项设置为 `false`，则 MATLAB 使用所打开图窗窗口中的属性确定代码生成的图窗的外观。此预设项不适用于使用“外部图形”（第 23-11 页）中的语法包括的图窗。

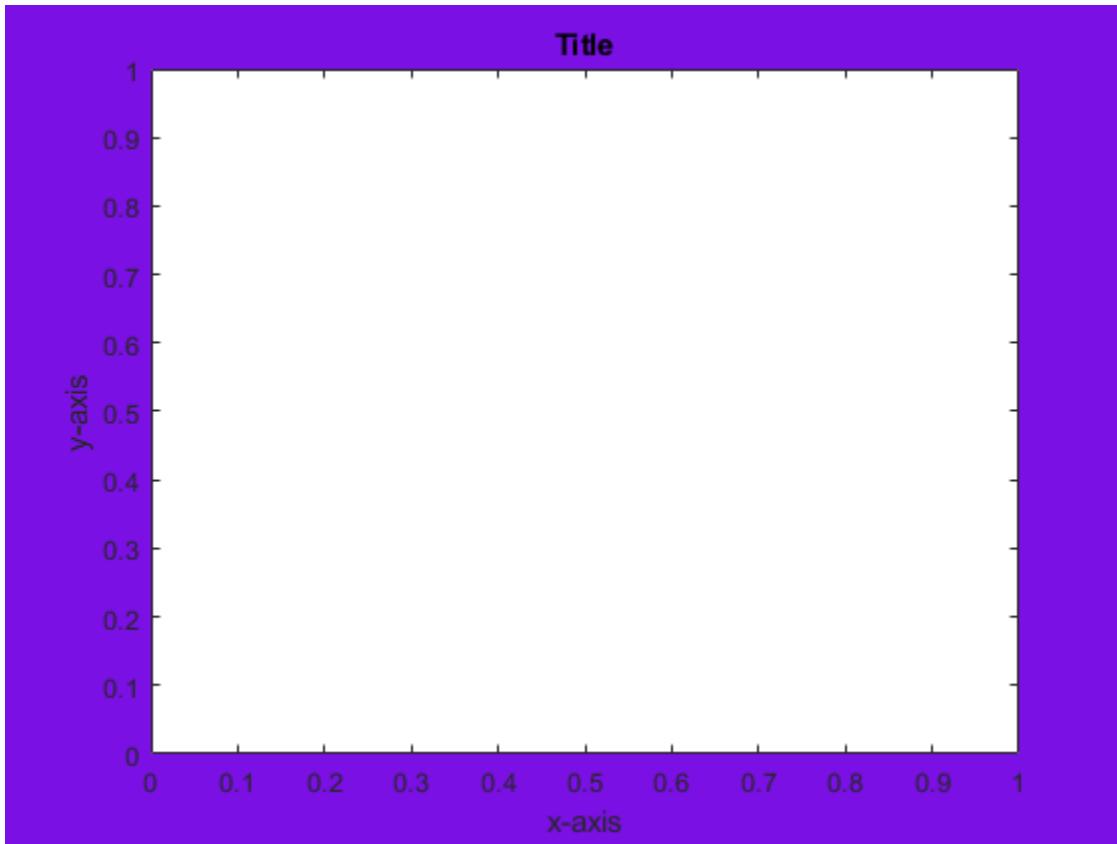
使用以下代码作为模板生成满足您的需求的图窗窗口。

```
% Create figure
figure1 = figure('Name','purple_background',...
'Color',[0.4784 0.06275 0.8941]);
colormap('hsv');

% Create subplot
subplot(1,1,1,'Parent',figure1);
box('on');

% Create axis labels
xlabel('x-axis');
ylabel({'y-axis'});

% Create title
title({'Title'});
```



```
% Enable printed output to match colors on screen
set(figure1,'InvertHardcopy','off')
```

通过在此窗口处于打开状态且**使用新图窗**选项设置为 `false` 时发布您的文件，任何代码生成的图窗都采用打开的图窗窗口的属性。

---

**注意** 您必须将**图窗捕获方法**选项设置为 `entireFigureWindow` 才能使最终发布的图窗显示打开的图窗窗口的所有属性。

---

### 创建缩略图

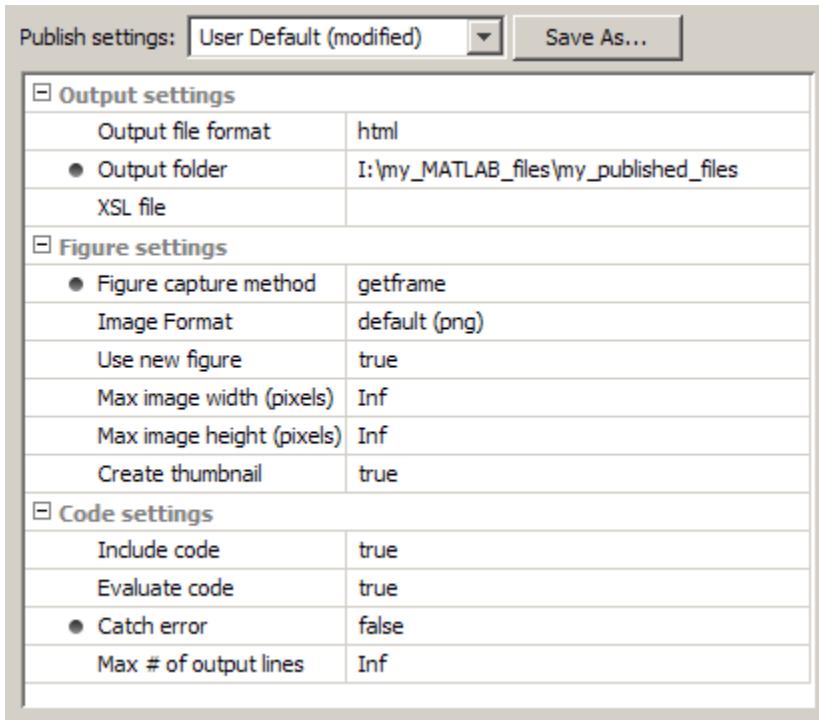
您可以将第一个代码生成的图形另存为缩略图图像。可以使用该缩略图在 HTML 页面中表示文件。要创建缩略图，请执行下列步骤：

- 1 在**发布**选项卡上，点击“发布”按钮下拉箭头，并选择**编辑发布选项**。“编辑配置”对话框随即打开。
- 2 将**图像格式**选项设置为位图格式，例如 `.png` 或 `.jpg`。MATLAB 以位图格式创建缩略图图像。
- 3 将**创建缩略图**选项设置为 `true`。

MATLAB 将缩略图图像保存在**发布设置**窗格中的**输出文件夹**选项指定的文件夹中。

### 保存发布设置

您可以保存发布设置，这样您可以便捷地重新生成输出。保存您的常用发布设置很有用。



设置**发布设置**选项后，您可以执行以下步骤来保存这些设置：

- 按您所需的方式设置这些选项后，点击**另存为**。

**将发布设置另存为**对话框随即打开，并显示当前定义的所有发布设置的名称。默认情况下，以下发布设置随 MATLAB 一起安装：

- **Factory Default**

您无法覆盖 **Factory Default**，可通过从**发布设置**列表中选择 **Factory Default** 来还原它们。

- **User Default**

最初，**User Default** 与 **Factory Default** 完全相同。您可以覆盖 **User Default** 设置。

- 在**设置名称**字段中，为这些设置输入有意义的名称。然后点击**保存**。

您现在可以将发布设置用于其他 MATLAB 文件。

您还可以覆盖以现有名称保存的发布属性。从**发布设置**列表中选择相应名称，然后点击**覆盖**。

## 管理发布配置

- “运行现有发布配置”（第 23-27 页）
- “为文件创建多个发布配置”（第 23-27 页）
- “重新关联和重新命名发布配置”（第 23-27 页）
- “跨不同系统使用发布配置”（第 23-28 页）

**MATLAB 表达式窗格**中的代码和**发布设置**窗格中的设置共同构成与一个文件关联的发布配置。使用这些配置可非常简便地引用单个文件的发布预设。

要创建发布配置，请在**发布**选项卡上点击**发布**按钮下拉箭头 ，然后选择**编辑发布选项**。“编辑配置”对话框随即打开，其中包含默认发布预设。在**发布配置名称**字段中，键入发布配置的名称，或接受默认名称。发布配置会自动保存。

## 运行现有发布配置

保存发布配置后，您可以运行它而不打开“编辑配置”对话框：

- 1 点击**发布**按钮下拉箭头 。如果您将鼠标指针放在发布配置名称上，MATLAB 会显示工具提示，其中显示与特定配置关联的 MATLAB 表达式。
- 2 选择用于发布配置的配置名称。MATLAB 使用与该配置关联的代码和发布设置发布文件。

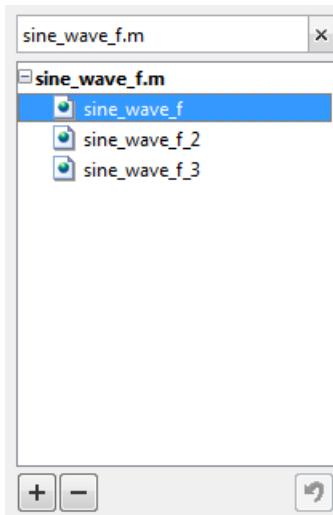
## 为文件创建多个发布配置

您可以为给定文件创建多个发布配置。您可以执行此操作来使用不同输入参数值和/或不同发布设置属性值发布该文件。为每种用途创建一个给定配置，所有配置均与同一文件关联。稍后您可以运行所需的任何特定发布配置。

使用以下步骤来创建新的发布配置。

- 1 在编辑器中打开文件。
- 2 点击**发布**按钮下拉箭头，并选择**编辑发布选项**。“编辑配置”对话框随即打开。
- 3 点击位于左窗格上的**添加按钮** .

配置列表上会显示一个新名称 `filename_n`，其中 `n` 的值取决于现有配置名称。



- 4 如果您修改 **MATLAB 表达式**或**发布设置**窗格中的设置，MATLAB 会自动保存更改。

## 重新关联和重新命名发布配置

每个发布配置都与一个特定文件关联。如果您移动或重命名某文件，请重新定义其关联。如果您删除某文件，请考虑删除关联的配置，或将其与其他文件关联。

当 MATLAB 无法将配置与文件关联时，“编辑配置”对话框会以红色显示文件名并显示**找不到文件**消息。要将配置与另一文件重新关联，请执行以下步骤。

- 1 点击“编辑配置”对话框左窗格上的“清除搜索”按钮 .
- 2 选择您要重新关联其发布配置的文件。
- 3 在“编辑配置”对话框的右窗格中，点击**选择...**。在“打开”对话框中，导航到并选择您要将配置与其重新关联的文件。

您随时都可以通过从左窗格的列表中选择一种配置来重命名配置。在右侧窗格中，编辑**发布配置名称**的值。

**注意** 要在文件名更改后正确运行，您可能需要在 **MATLAB 表达式**窗格中更改代码语句。例如，更改函数调用以反映该函数的新文件名。

---

### 跨不同系统使用发布配置

每当您使用“编辑配置”对话框创建或保存发布配置时，编辑器就会更新预设文件夹中的 **publish\_configurations.m** 文件。（这是 MATLAB 在您运行 MATLAB **prefdir** 函数时返回的文件夹。）

虽然您可以将此文件从一个系统上的预设文件夹移到另一个系统上，但一个系统上只能存在一个 **publish\_configurations.m** 文件。因此，仅当您尚未在另一个系统上创建任何发布配置时才将此文件移到另一个系统上。另外，由于 **publish\_configurations.m** 文件可能包含对文件路径的引用，因此请确保指定的文件和路径存在于另一个系统上。

MathWorks 建议您不要在 MATLAB 编辑器或文本编辑器中更新 **publish\_configurations.m**。使用这些工具而非“编辑配置”对话框进行的更改可能在以后被覆盖。

### 另请参阅

#### 详细信息

- “发布和共享 MATLAB 代码”（第 23-2 页）
- “发布标记”（第 23-5 页）

# 编码及工作效率的提示

---

- “在编辑器中打开和保存文件” (第 24-2 页)
- “检查代码中的错误和警告” (第 24-5 页)
- “提高代码可读性” (第 24-14 页)
- “在文件中查找并替换文本” (第 24-19 页)
- “在文件中添加提醒” (第 24-26 页)
- “MATLAB 代码分析器报告” (第 24-28 页)
- “MATLAB 代码兼容性报告” (第 24-30 页)

## 在编辑器中打开和保存文件

### 本节内容

“打开现有文件” (第 24-2 页)

“保存文件” (第 24-3 页)

### 打开现有文件

要在编辑器中打开现有的一个文件或多个文件，请按下表中所述选择可实现您的目的的方式。

目的	步骤	其他信息
<b>用关联工具打开</b> 使用适合文件类型的 MATLAB 工具打开文件。	在编辑器、实时编辑器或主页选项卡上的文件部分中，点击 。您还可以在当前文件夹浏览器中双击文件。	例如，这一选项会在编辑器中打开一个扩展名为 .m 或 .mlx 的文件，并将 MAT 文件加载到工作区浏览器中。
<b>作为文本文件打开</b> 在编辑器中将文件作为文本文件打开，即使文件类型是与其他应用程序或工具关联。	在编辑器选项卡上的文件部分中，点击 <b>打开</b> 并选择以文本方式打开。	例如，如果您已将制表符分隔的数据文件 (.dat) 导入到工作区中并且您发现需要添加数据点，则可使用这一方法。在编辑器中将文件作为文本打开，添加相应内容，然后保存文件。
<b>从文件中打开函数</b> 在编辑器中打开某个文件中的局部函数或打开一个函数文件。	将光标置于打开的文件中的名称上，然后右键点击并从上下文菜单中选择 <b>打开 file-name</b> 。	您也可以使用此方法打开变量或 Simulink 模型。 有关详细信息，请参阅“从文件中打开文件或变量” (第 24-24 页)。
<b>重新打开文件</b> 重新打开最近使用的文件。	在 <b>打开</b> 下拉列表底部，在 <b>最近使用的文件</b> 下选择一个文件。	要更改列表中的文件数，请点击  预设，然后选择 MATLAB 和 编辑器/调试器。在 <b>最近使用的文件</b> 列表下，更改条目数的值。
<b>在启动时重新打开文件</b> 在启动时，自动打开在先前 MATLAB 会话结束时处于打开状态的文件。	在主页选项卡上的环境部分中，点击  预设并选择 MATLAB 和 编辑器/调试器。然后，选择 <b>在重新启动时重新打开先前 MATLAB 会话中的文件</b> 。	
<b>打开显示在另一工具中的文件</b> 打开显示在另一 MATLAB 桌面工具或 Microsoft 工具中的文件。	将文件从其他工具拖到编辑器中。	例如，从当前文件夹浏览器或 Windows 资源管理器拖动文件。

目的	步骤	其他信息
使用函数打开文件	使用 <code>edit</code> 或 <code>open</code> 函数。	例如，键入以下内容可打开 <code>collatz.m</code> ：  <code>edit collatz.m</code>  如果 <code>collatz.m</code> 不在搜索路径或当前文件夹中，请使用文件的相对或绝对路径。

有关 Macintosh 平台的特殊注意事项，请参阅“在 macOS 平台上的 MATLAB 根文件夹中导航”。

用于打开文件的某些选项在 MATLAB Online 中不可用。

## 保存文件

在编辑器中修改某文件后，文件名后会显示一个星号 (\*)。此星号指示文件有未保存的更改。

您可以执行四种不同类型的保存操作，它们具有不同效果，如下表中所述。

保存类型	步骤
将文件保存到磁盘并保持文件在编辑器中处于打开状态。	在 <b>编辑器或实时编辑器</b> 选项卡上的 <b>文件</b> 部分中，点击  。
重命名文件，将其保存到磁盘，并使其成为活动的编辑器文档。原始文件在磁盘上保持不变。	<ol style="list-style-type: none"> <li>在<b>编辑器或实时编辑器</b>选项卡上的<b>文件</b>部分中，点击<b>保存</b>并选择<b>另存为</b>。</li> <li>为该文件指定新的名称和/或类型，然后点击<b>保存</b>。</li> </ol>
以新名称将文件保存到磁盘。原始文件保持打开和未保存状态。	<ol style="list-style-type: none"> <li>在<b>编辑器</b>选项卡上的<b>文件</b>部分中，点击<b>保存</b>并选择<b>将副本另存为</b>。 MATLAB 会打开“选择要备份的文件”对话框。</li> <li>指定备份文件的名称和类型，然后点击<b>保存</b>。</li> </ol>
使用当前文件名保存对所有打开的文件的更改。所有文件保持打开状态。	在 <b>编辑器</b> 选项卡上的 <b>文件</b> 部分中，点击 <b>保存</b> 并选择 <b>全部保存</b> 。

用于保存文件的某些选项在 MATLAB Online 中不可用。

### 有关保存文件的建议

MathWorks 建议您将所创建的文件和 MathWorks 中所编辑的文件保存在 `matlabroot` 文件夹树之外的文件夹中，其中 `matlabroot` 是当您在命令行窗口中键入 `matlabroot` 时返回的文件夹。如果将文件保留在 `matlabroot` 文件夹中，当安装新版 MATLAB 软件时，可能会覆盖这些文件。

在每个 MATLAB 会话开始时，MATLAB 会将 `matlabroot` 文件夹树中的文件位置加载并缓存在内存中。因此，需根据情况采取相应的操作：

- 如果使用外部编辑器将文件保存至 `matlabroot` 文件夹，需要先运行 `rehash toolbox`，然后再将这些文件用在当前会话中。

- 如果使用文件系统操作在 `matlabroot` 文件夹中添加或删除文件，需要先运行 `rehash toolbox`，然后再将这些文件用在当前会话中。
- 如果使用外部编辑器修改 `matlabroot` 文件夹中的现有文件，需要先运行 `clear function-name`，然后再将这些文件用在当前会话中。

有关详细信息，请参阅 `rehash` 或“MATLAB 中的工具箱路径缓存”。

### 备份文件

当您在编辑器中修改某文件时，编辑器会使用同一文件名但使用 `.asv` 扩展名每 5 分钟保存一次该文件的副本。如果您遇到系统问题并丢失对文件所做的更改，备份版本会很有用。在这种情况下，您可以打开备份版本 `filename.asv`，然后将其另存为 `filename.m` 以使用 `filename` 的上一未损坏版本。

---

**注意** 实时编辑器和 MATLAB Online 不会自动保存文件副本。

---

要选择预设，请点击  **预设**，然后在**主页**选项卡上的**环境**部分中选择 **MATLAB > 编辑器/调试器 > 备份文件**。您随后可以执行以下操作：

- 打开或关闭备份功能。
- 当您关闭相应的源文件时自动删除备份文件。

默认情况下，MATLAB 在您关闭编辑器时自动删除备份文件。最好确保备份版本与对应文件的关系明确且保持最新。因此，当您重命名或删除文件时，考虑删除或重命名相应的备份文件。

- 指定备份文件保存操作之间间隔的分钟数。
- 指定备份文件的文件扩展名。
- 指定备份文件的位置。

如果您编辑只读文件夹中的文件并将备份**位置**预设为**源文件目录**，则编辑器不会创建该文件的备份副本。

# 检查代码中的错误和警告

MATLAB 代码分析器可自动检查您的代码有无编码问题。

## 在编辑器中自动检查代码 - 代码分析器

您可以查看有关您代码的警告和错误消息，并基于这些消息修改您的文件。这些消息会自动持续更新，因此您可以看到您的更改是否解决了消息中指明的问题。某些消息提供其他信息和/或自动代码更正。

### 启用持续代码检查

要在编辑器中的 MATLAB 代码文件中启用持续代码检查，请执行以下操作：

- 1 在主页选项卡上的环境部分中，点击  预设。
- 2 选择 MATLAB > 代码分析器，然后选中启用集成警告和错误消息复选框。
- 3 将下划线选项设置为“为警告和错误添加下划线”，然后点击确定。

---

**注意** MATLAB Online 不支持更改持续代码检查预设项。始终启用持续代码检查。

---

### 使用持续代码检查

您可以在编辑器中的 MATLAB 代码文件中使用持续代码检查：

- 1 在编辑器中打开一个 MATLAB 代码文件。此示例使用 MATLAB 软件随附的示例文件 `lengthofline.m`：
  - a 打开该示例文件：

```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...
    'examples','lengthofline.m'))
```
  - b 将该示例文件保存至您对其有写访问权限的文件夹。例如，将 `lengthofline.m` 保存至 `C:\my_MATLAB_files`。
- 2 检查消息栏顶部的消息指示标记，以查看报告的有关该文件的代码分析器消息：
  - 红色指示检测到语法错误或其他重大问题。
  - 橙色指示检测到警告或改进之处，但未检测到错误。
  - 绿色指示未检测到任何错误、警告或改进机会。

在此示例中，指示条是红色，意味着文件中至少有一个错误。

```

1 function [len,dims] = lengthofline(hline)
2 % LENGTHOFLINE Calculates the length of a line object
3 % LEN = LENGTHOFLINE(HLINE) takes the handle to a line
4 % input, and returns its length. The accuracy of the
5 % dependent on the number of distinct points used to
6 %
7 % [LEN,DIM] = LENGTHOFLINE(HLINE) additionally tells w
8 % 2D or 3D by returning either a numeric 2 or 3 in DIM
9 % plane parallel to a coordinate plane is considered :
10 %
11 % If HLINE is a matrix of line handles, LEN and DIM w
12 %
13 % Example:

```

check\_results.m x lengthofline.m x Untitled2\* x

lengthofline Ln 12 Col 2 OVR ::

- 3 点击消息指示标记以转到包含消息的下一代码片段。下一段代码是相对于当前光标位置来定位的，可在状态栏中查看。

在 `lengthofline` 示例中，第一条消息是在第 21 行。光标移至第 21 行的开头。

其中存在消息的代码片段以红色（表示错误）或橙色（表示警告和改进机会）下划线标出。

- 4 可通过在带有下划线的代码片段内移动鼠标指针来查看该消息。

该消息以工具提示形式打开并包含一个**详细信息**按钮，通过该按钮可展开消息，获取更多信息。并非所有消息都有附加信息。

```

16 % [len,dim] = lengthofline([h1 h2])
17
18 % Copyright 1984-2004 The MathWorks, Inc.
19
20 % Find input indices that are not line objects
21 - nohandle = ~ishandle(hline);
22 - ⚠ The value assigned to variable 'nohandle' might be unused. Details ▾
23 - not |line(nh) = ~ishandle(hline(nh)) || ~strcmp('line',
24 - end

```

- 5 点击**详细信息**按钮。

窗口将扩大以显示说明和用户操作。

- 6 根据需要修改代码。

消息指示标记和下划线会自动更新，以反映您所做的更改，即使您不保存文件也如此。

- 7 在第 27 行上，将指针悬停在 `prod` 上。

该代码带有下划线，这是因为有一个警告消息，并且它是高亮显示的，这是因为可进行自动修复。当您查看该消息时，它提供一个按钮来应用自动修复。

```
25
26 -     len = zeros(size(hline));
27 -     for nl = 1:prod(size(hline))
28 -         if ~isline(nl)
29 -             flds = get(hline(nl));
30 -             fdata = {'XData','YData','ZData'};
31 -             for nd = 1:length(fdata)
32 -                 data(nd) = getfield(flds,fdata(nd));
33 -             end
```

## **8 通过执行以下操作之一修复该问题:**

- 如果您知道修复的内容（根据以前的经验），请点击**修复**。
  - 如果您不熟悉修复，可查看修复内容，然后按如下所示进行应用：
    - a 右键点击高亮显示的代码（对于单键鼠标，按 **Ctrl** 并点击），然后查看上下文菜单中的第一项。
    - b 点击**修复**。

MATLAB 会自动更正代码。

在此示例中，MATLAB 将 `prod(size(hline))` 替换为 `numel(hline)`。

**9** 通过执行以下操作之一转到其他消息：

- 要转到下一消息，请点击消息指示标记或下一个带下划线的代码片段。
  - 要转到标记表示的行，请在指示标记条中点击红色或橙色的行。

要查看 `lengthofline` 中的第一个错误，请点击消息栏中的第一个红色标记。光标移至第 47 行中的第一个可疑代码片段。**详细信息和修复**按钮灰显（如果是在 MATLAB Online 中，则不可见），指示不存在有关此消息的更多信息且没有自动修复。

```
21 -     nohandle = ~ishandle(hline);
22 -     for nh = 1:prod(size(hline))
23 -         notline(nh) = ~ishandle(hline(nh)) || ~strcmp('line',
24 -     end
25 -
26 -     %% Line 47: Invalid syntax at '.': Possibly, a ')' is missing.
27 -     %% Line 47: Invalid syntax at ')'. Possibly, a ')' is missing.
28 -     %% Line 47: Parse error at ']': usage might be invalid MATLAB syntax.
29 -     flds = get(hline(nl));
30 -     %-----
```

多条消息可能代表存在一个问题或多个问题。解决一个问题或许能解决所有问题，或者在解决一个问题后，其他消息可能有变化，或者您需要采取的操作可能变得更清晰。

## 10 修改代码以解决消息中指明的问题 - 消息指示标记会自动更新。

该消息指示在第 47 行存在分隔符不平衡问题。要研究此消息，请在编辑器中将箭头键移到各个分隔符上，查看 MATLAB 是否指示不匹配。有关如何在使用箭头键时启用分隔符匹配的说明，请参阅“设置键盘预设项”。

代码看起来没有任何不匹配的分隔符。但是，代码分析检测到圆括号中有分号：`data{3}();`，并将其解释为语句结束符。该消息报告第 47 行上的两个语句都存在分隔符不平衡问题。

要解决此问题，请在第 47 行中将 `data{3}(:)` 更改为 `data{3}(::)`。现在，下划线不再显示在第 47 行中。在第 47 行中进行一次更改即可解决两条消息中的问题。因为通过更改移除了文件中的唯一错误，所以栏顶部的消息指示标记从红色变为橙色，指示只存在警告和潜在改进机会。

在修改代码以解决所有消息中的问题或禁用指定消息后，消息指示标记变为绿色。所有消息均得到解决的示例文件已另存为 `lengthofline2.m`。使用以下命令打开更正后的示例文件：

```
open(fullfile(matlabroot,'help','techdoc',...
    'matlab_env','examples','lengthofline2.m'))
```

## 创建代码分析器消息报告

您可以使用以下方法之一为单个文件创建一个消息报告，或为文件夹中的所有文件创建一个消息报告：

- 为单个 MATLAB 代码文件运行一个报告：

- 1 在编辑器窗口中，点击 并选择 **显示代码分析器报告**。

代码分析器报告随即显示在 MATLAB Web 浏览器中。

- 2 基于报告中的消息修改您的文件。

- 3 保存文件。

- 4 重新运行该报告，以查看您的更改是否解决了消息中指明的问题。

- 对文件夹中的所有文件运行报告：

- 1 在当前文件夹浏览器上，点击 .

- 2 选择 **报告 > 代码分析器报告**。

- 3 基于报告中的消息修改您的文件。

有关详细信息，请参阅“**MATLAB 代码分析器报告**”（第 24-28 页）。

- 4 保存修改后的文件。

- 5 重新运行该报告，以查看您的更改是否解决了消息中指明的问题。

## 调整代码分析器消息指示标记和消息

根据您在完成 MATLAB 文件时所处的阶段，您可能需要限制代码下划线标记的使用。您可以使用步骤 1 的“**检查代码中的错误和警告**”（第 24-5 页）中引用的代码分析器预设执行此操作。例如，首次编码时，您可能希望只对错误使用下划线，这是因为警告会带来纷扰。

代码分析并不提供关于每一情形的完美信息，有时，您可能不需要根据消息更改代码。如果您不想更改代码，并且不希望看到该行的标记和消息，可隐藏它们。对于 `lengthofline` 示例，在第 49 行中，第一条消息是 **Terminate statement with semicolon to suppress output (in functions)**。在语句末尾添加分号隐藏输出，这是一种常见做法。针对生成输出但缺少终止分号的行，代码分析可向您发出警报。如果您想查看第 49 行后的输出，请不要按消息指示添加分号。

可通过一些不同方法隐藏（关闭）警告和错误消息指示标记：

- “**在当前文件中隐藏消息实例**”（第 24-9 页）
- “**在当前文件中隐藏所有消息实例**”（第 24-9 页）
- “**在所有文件中隐藏所有消息实例**”（第 24-9 页）

- “保存并重新使用代码分析器消息设置”（第 24-10 页）

您不能隐藏诸如语法错误之类的错误消息。因此，有关隐藏消息的说明不适用于这些类型的消息。

### 在当前文件中隐藏消息实例

您可以在当前文件中隐藏代码分析器消息的特定实例。例如，使用“检查代码中的错误和警告”（第 24-5 页）中显示的代码，执行下列步骤：

- 1 在第 49 行，在第一个下划线处右键点击（对于单键鼠标，按 **Ctrl** 并点击）。
- 2 从上下文菜单中，选择取消‘使用分号终止语句...’>在此行中。

在此行末尾显示注释 `%#ok<NOPRT>`，指示 MATLAB 不在此行检查终止分号。该消息的下划线和指示标记条中对应的标记随即消失。

- 3 如果某行上有两条您不希望显示的消息，请在每个下划线处分别右键点击，并从上下文菜单中选择适当的条目。

`%#ok` 语法将扩展。例如，在“检查代码中的错误和警告”（第 24-5 页）中显示的代码中，忽略第 49 行的两条消息会添加 `%#ok<NBRAK,NOPRT>`。

即使代码分析器预设设置为启用此消息，按此方式隐藏的消息的特定实例也不会显示，这是因为 `%#ok` 的优先级高于预设设置。如果您以后决定要在该行检测终止分号，请从该行中删除 `%#ok<NOPRT>`。

### 在当前文件中隐藏所有消息实例

您可以在当前文件中隐藏特定代码分析器消息的所有实例。例如，使用“检查代码中的错误和警告”（第 24-5 页）中显示的代码，执行下列步骤：

- 1 在第 49 行，在第一个下划线处右键点击（对于单键鼠标，按 **Ctrl** 并点击）。
- 2 从上下文菜单中，选择取消‘使用分号终止语句...’>在此文件中。

在此行末尾显示注释 `%#ok<*NOPRT>`，指示 MATLAB 不在此文件中检查终止分号。此消息的所有下划线以及消息指示标记条中对应的标记都将消失。

如果某行上有两条您不希望显示在当前文件中的任何位置的消息，请在每个下划线处分别右键点击，并从上下文菜单中选择适当的条目。`%#ok` 语法将扩展。例如，在“检查代码中的错误和警告”（第 24-5 页）中显示的代码中，忽略第 49 行的两条消息会添加 `%#ok<*NBRAK,*NOPRT>`。

即使代码分析器预设设置为启用此消息，此消息也不会显示，这是因为 `%#ok` 的优先级高于预设设置。如果您以后决定要在此文件中检测终止分号，请从该行中删除 `%#ok<*NOPRT>`。

### 在所有文件中隐藏所有消息实例

您可以在所有文件中禁用代码分析器消息的所有实例。例如，使用“检查代码中的错误和警告”（第 24-5 页）中显示的代码，执行下列步骤：

- 1 在第 49 行，在第一个下划线处右键点击（对于单键鼠标，按 **Ctrl** 并点击）。
- 2 选择取消‘使用分号终止语句...’>在所有文件中。

这会修改代码分析器预设设置。

如果您知道要隐藏哪一条或哪几条消息，可以直接使用代码分析器预设禁用它们，如下所示：

- 1 在主页选项卡上的环境部分中，点击  预设。
- 2 选择 MATLAB > 代码分析器。
- 3 搜索消息以查找要隐藏的消息。
- 4 清除与您要在所有文件中隐藏的每条消息关联的复选框。
- 5 点击确定。

实时编辑器不支持通过右键点击下划线来隐藏所有文件中代码分析器消息的所有实例。要在实时编辑器中隐藏代码分析器消息的所有实例，请使用代码分析器预设项。MATLAB Online 不支持在所有文件中隐藏代码分析器消息的所有实例。

### 保存并重新使用代码分析器消息设置

您可以指定您要启用或禁用某些代码分析器消息，然后将这些设置保存至文件。当您希望将设置文件用于某特定文件时，可从代码分析器预设窗格选择它。该设置文件会一直有效，直到您选择另一设置文件为止。通常，当您希望将某特定设置文件用于一部分文件时，要更改设置文件。

执行下列步骤：

- 1 在主页选项卡上的环境部分中，点击  预设。  
“预设项”对话框随即打开。
- 2 选择 MATLAB > 代码分析器。
- 3 启用或禁用特定消息或消息类别。
- 4 点击“操作”按钮 ，选择另存为，然后将设置保存为 txt 文件。
- 5 点击确定。

您可以对任何 MATLAB 文件重复使用这些设置，或向另一用户提供该设置文件。

要使用保存的设置，请执行以下操作：

- 1 在主页选项卡上的环境部分中，点击  预设。  
“预设项”对话框随即打开。
- 2 选择 MATLAB > 代码分析器。
- 3 使用当前设置下拉列表选择浏览....。  
随即显示“打开”对话框。
- 4 从任何设置文件中进行选择。

您选择的设置对于所有 MATLAB 文件都有效，直到您选择另一组代码分析器设置为止。

### 了解包含隐藏消息的代码

如果您收到包含隐藏消息的代码，您可能需要查看这些消息，而无需首先取消隐藏它们。由于以下任何原因，消息可能处于隐藏状态：

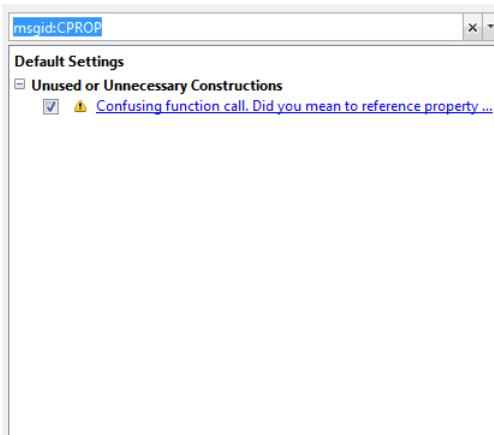
- 一个或多个 %#ok<message-ID> 指令所在的代码行引发了 <message-ID> 指定的消息。

- 一个或多个 %#ok<\*message-ID> 指令所在的文件引发了 <message-ID> 指定的消息。
- 它在代码分析器预设窗格中被清除。
- 它在默认情况下被禁用。

要确定某些消息被隐藏的原因，请执行以下操作：

- 1 搜索文件中的 %#ok 指令并创建与该指令关联的所有消息 ID 列表。
- 2 在主页选项卡上的环境部分中，点击  预设。
- “预设项”对话框随即打开。
- 3 选择 MATLAB > 代码分析器。
- 4 在搜索字段中，键入 msgid: 并且后跟您在步骤 1 中找到的其中一个消息 ID（如果有）。

消息列表现在仅包含与该 ID 对应的消息。如果该消息是超链接，点击它可查看该消息的说明和建议的操作。这可以深入了解该消息被隐藏或禁用的原因。下面的图像显示当您在搜索字段中输入 msgid:CPROP 时“预设项”对话框的显示情况。



- 5 点击  按钮以清除搜索字段，然后对您在步骤 1 中找到的每个消息 ID 重复步骤 4。
- 6 显示默认被禁用以及在“预设”窗格中被禁用的消息，方法是点击搜索字段右侧的向下箭头。然后，**点击显示已禁用的消息**。
- 7 查看与每个消息 ID 关联的消息以了解它为何在代码中被隐藏或在“预设”中被禁用。

## 了解代码分析的局限

代码分析是一个非常有用的工具，但有一些局限：

- 有时候，它无法生成您预期的代码分析器消息。

按照设计，代码分析会尝试最大限度地减少其返回的不正确的消息数，即使此行为导致某些问题不被发现也如此。

- 有时，它生成的消息不适用于您的情形。

当提供消息时，**点击详细信息按钮**可了解详细信息，这可帮助您做出此判断。错误消息几乎总说明存在问题。但是，许多警告是一些建议，指示您查看代码中的某些异常因此可疑的内容，但在您特定的情况下可能正确。

如果您确信某警告消息不适用于您的情形，可隐藏该消息。如果您隐藏消息的理由不明显或模糊，请包含注释加以说明。这样，阅读您的代码的用户就可以知道这种情况。

有关详细信息，请参阅“调整代码分析器消息指示标记和消息”（第 24-8 页）。

这些节介绍了关于以下方面的代码分析限制：

- “区分函数名称与变量名称”（第 24-12 页）
- “区分结构体与句柄对象”（第 24-12 页）
- “区分内置函数与重载函数”（第 24-13 页）
- “确定变量的大小或形状”（第 24-13 页）
- “分析具有超类的类定义”（第 24-13 页）
- “分析类方法”（第 24-13 页）

### 区分函数名称与变量名称

代码分析无法始终将函数名称与变量名称区分开。对于以下代码，如果启用了代码分析器消息，则代码分析会返回消息 **Code Analyzer cannot determine whether xyz is a variable or a function, and assumes it is a function.** 代码分析无法做出判断，这是因为 **xyz** 未向其分配明显的值。但是，程序可能已将该值放入工作区中，只是代码分析无法检测到。

```
function y=foo(x)
%
%
%
y = xyz(x);
end
```

例如，在以下代码中，**xyz** 可以为函数，或者可以为从 MAT 文件加载的变量。代码分析无法做出判断。

```
function y=foo(x)
    load abc.mat
    y = xyz(x);
end
```

当您使用 **eval**、**evalc**、**evalin** 或 **assignin** 函数时，变量也有可能不被代码分析检测到。

如果代码分析将变量误认为是函数，请执行以下操作之一：

- 初始化该变量，以便代码分析不将其视为函数。
- 对于 **load** 函数，请在 **load** 命令行中显式指定变量名称。例如：

```
function y=foo(x)
    load abc.mat xyz
    y = xyz(x);
end
```

### 区分结构体与句柄对象

代码分析无法始终将结构体与句柄对象区分开。在以下代码中，如果 **x** 是一个结构体，您可能收到代码分析器消息，指示代码从未使用该结构体的更新值。但如果 **x** 是一个句柄对象，则该代码可能正确。

```
function foo(x)
    x.a = 3;
end
```

代码分析无法确定 `x` 是结构体还是句柄对象。为了最大限度减少不正确的消息数，代码分析对以上代码不返回任何消息，即使它可能包含细微和严重的问题也如此。

### 区分内置函数与重载函数

如果在类中或在路径上重载某些内置函数，则代码分析器消息可能适用于该内置函数，但不适用于您所调用的重载函数。本例中，在消息显示的行上隐藏该消息或者在整个文件隐藏它。

有关隐藏消息的信息，请参阅“调整代码分析器消息指示标记和消息”（第 24-8 页）。

### 确定变量的大小或形状

代码分析在确定变量的类型和矩阵的形状时的能力有限。代码分析生成的消息可能适合多数常见情况（例如向量）。但是，这些消息可能不适用于不太常见的情况（例如矩阵）。

### 分析具有超类的类定义

代码分析器仅提供有限的功能检查具有超类的类定义。例如，代码分析器不总能确定类是否为句柄类，但它有时能验证类中使用的自定义属性（如果这些属性是从某个超类继承的）。在分析类定义时，代码分析器会尽量使用来自超类的信息，但它往往无法获得足够的信息做出肯定的决定。

### 分析类方法

多数类方法都必须至少包含一个参数，该参数是与方法属于相同类的对象。但它并非必须总是第一个参数。当它是第一个参数时，代码分析可确定参数是否属于您所定义类的对象，并且是否可执行各种检查。例如，它可以检查属性和方法名称是否存在并且拼写是否正确。但是，当代码分析无法确定对象是否属于您所定义类的参数时，它无法提供这些检查。

## 启用 MATLAB 编译器部署消息

通过更改此消息类别的代码分析器预设项，可以在处理文件时选择显示或隐藏编译器部署消息。您的选择可能取决于您是否在处理要部署的文件。当您更改该预设时，会同时在编辑器中更改该设置。反之亦然 - 当您从编辑器中更改该设置时，也会有效更改该预设。但是，如果您在编辑器中修改该设置时该对话框处于打开状态，则您将看不到反映在“预设项”对话框中的更改。无论您是从编辑器还是从“预设项”对话框中更改该设置，它都应用于编辑器和代码分析器报告。

要启用 MATLAB Compiler™ 部署消息，请执行以下操作：

- 1 在主页选项卡上的环境部分中，点击  预设。
- 2 “预设项”对话框随即打开。
- 3 选择 MATLAB > 代码分析器。
- 4 点击启用类别按钮。
- 5 清除不想为您的代码显示的消息（如果有）。
- 6 决定您是否要保存这些设置，以便下次您处理要部署的文件时可重复使用它们。

您可按“保存并重新使用代码分析器消息设置”（第 24-10 页）中所述创建的设置 txt 文件包括该设置的状态。

## 提高代码可读性

### 本节内容

- “缩进代码”（第 24-14 页）
- “右侧文本限制指示器”（第 24-15 页）
- “代码折叠 - 展开和折叠代码构造”（第 24-16 页）
- “代码重构 - 自动将选定的代码转换为函数”（第 24-18 页）

### 缩进代码

缩进代码使阅读 `while` 循环之类的语句更容易。要在编辑器中设置缩进预设项并将其应用于代码，请执行以下操作：

- 1 在主页选项卡上的环境部分中，点击  预设。
- 2 选择 MATLAB > 编辑器/调试器 > 语言。
- 3 从语言下拉列表中选择一种计算机语言。
- 4 在缩进部分，选中或清除键入时应用智能缩进，具体取决于您是否希望在键入时自动应用缩进。在实时编辑器中，智能缩进始终会自动应用。  
如果清除此选项，可以手动应用缩进，方法是在编辑器中选择要缩进的行，右键点击，然后从上下文菜单中选择智能缩进。
- 5 执行以下操作之一：
  - 如果您在步骤 2 中选择了 MATLAB 之外的任何其他语言，请点击确定。
  - 如果您在步骤 2 中选择了 MATLAB，请选择一种函数缩进格式，然后点击确定。函数缩进格式有：
    - “经典” - 编辑器将函数代码与函数声明对齐。
    - “缩进嵌套函数” - 编辑器缩进嵌套函数中的函数代码。
    - “缩进所有函数” - 编辑器缩进主函数和嵌套函数的函数代码。

此图像展示了函数缩进格式。

```

1 % Indenting Preferences
2
3 % Classic
4 function classic_one
5 - disp('Main function code')
6     function classic_two
7         disp('Nested function code')
8     end
9 end
10
11 % Indent Nested Functions
12 function nested_one
13 - disp('Main function code')
14     function nested_two
15         disp('Nested function code')
16     end
17 end
18
19 % Indent All Functions
20 function all_one
21 - disp('Main function code')
22     function all_two
23         disp('Nested function code')
24     end
25 end
26 |

```

**注意** TLC、VHDL 或 Verilog 不支持缩进预设项。

不管您是自动还是手动应用缩进，都可以通过执行下列操作之一将选定行继续向左或向右移动：

- 在编辑器选项卡上的**编辑**部分中，点击 、 或 。对于实时脚本和函数，此功能在**实时编辑器**选项卡上的**代码**部分中提供。
- 分别按 **Tab** 键或 **Shift+Tab** 键。

如果您选择的编辑器/调试器 **Tab** 预设为 **Emac 样式的 Tab 键智能缩进**，则情况会有所不同 - 当您将光标置于任一行中或选择一组行并按 **Tab** 时，这些行会根据智能缩进方法缩进。

## 右侧文本限制指示器

默认情况下，编辑器中的第 75 列会显示淡灰色的竖直线（规则），指示超过 75 个字符的行的位置。您可以将此文本限制指示标记设置为其他值，如果您想在具有不同行宽限制的另一文本编辑器中查看代码，这很有用。实时编辑器和 MATLAB Online 不支持右侧文本限制指示符。

要隐藏或更改竖直线的外观，请执行以下操作：

- 1 在主页选项卡上的**环境**部分中，点击 预设。“预设”对话框随即打开。
- 2 选择 **MATLAB > 编辑器/调试器 > 显示**。
- 3 调整**右侧文本限制**部分中的设置。

**注意** 这一限制只是可视的提示，并不能防止文本超出限制。要自动在指定列号处使注释文本换行，请调整“预设”对话框的 **MATLAB > 编辑器/调试器 > 语言**下的**注释格式设置**部分中的设置。

## 代码折叠 - 展开和折叠代码构造

代码折叠功能可以展开和折叠某些 MATLAB 程序构造块。当某文件包含许多函数或其他代码块（您希望在当前未处理这部分文件时隐藏它们）时，这可以提高可读性。MATLAB 程序构造块包括：

- 用于运行和发布代码的代码节
- 类代码
- **for** 和 **parfor** 模块
- 函数和类帮助
- 函数代码

要查看整个构造块列表，请选择“预设项”对话框中的**编辑器/调试器 > 代码折叠**。

要展开或折叠代码，请在编辑器中点击显示在构造块左侧的加号 或减号 。您还可以使用 **Ctrl+Shift+. (句点)** 和 **Ctrl+. (句点)** 键盘快捷方式，或使用视图选项卡中的代码折叠按钮。

要展开或折叠文件中的所有代码，请将光标置于该文件中的任意位置，右键点击，然后从上下文菜单中选择**代码折叠 > 全部展开**或者**代码折叠 > 全部折叠**。您也可以使用 **Ctrl+Shift+, (逗号)** 和 **Ctrl+, (逗号)** 键盘快捷方式，或视图选项卡中的代码折叠按钮。

**注意** 实时编辑器不支持代码折叠

## 在工具提示中查看折叠代码

您可以通过将指针置于代码的省略号 上来查看当前折叠的代码。代码显示在工具提示中。MATLAB Online 不支持在工具提示中查看折叠代码。

下面的图像显示了在 **for** 循环处于折叠状态时，将指针置于 **lengthofline.m** 的第 23 行上的省略号上方时显示的工具提示。

```

20
21 % Find input indices that are not line objects
22 - nothandle = ~ishandle(hline);
23 - for nh = 1:prod(size(hline)) 
24 -     for nh = 1:prod(size(hline))
25 -         len = zeros(size(hline));
26 -         notline(nh) = ~ishandle(hline(nh)) || ~strcmp('line',lower(get(hline(nh),'type')));
27 -     for nl = 1:prod(size(hline)) end
28 -
29 - % If some indices are not lines, fill the results with NaNs.
30 - if any(notline())
31 -     warning('lengthofline:FillWithNaNs', ...
32 -         '\n% of non-line objects are being filled with %s.', ...
33 -         'Lengths','NaNs','Dimensions','NaNs')
34 -     len(notline) = NaN;
35 - 
```

## 打印包含折叠代码的文件

如果您打印的文件包含一个或多个折叠构造块，这些构造块将在该文件的打印版本中展开。

### 没有显式 End 语句的函数的代码折叠行为

如果您为函数启用代码折叠并且您代码中的函数不以显式 **end** 语句结尾，您会看到以下行为：

- 如果显示在此类函数末尾的行只包含注释，则编辑器不会在折叠函数时包括该行。这是因为 MATLAB 不会在没有任何显式 **end** 语句的函数定义中包括尾随空白和注释。在 MATLAB Online 中，编辑器在折叠函数时会包括该行。

“仅对函数代码启用代码折叠”（第 24-17 页）描述了此行为。foo 函数的折叠不包括第 13 行。

- 如果某代码节的折叠与函数代码重叠，则编辑器不显示重叠节的折叠。

随后的三张图描述了此行为。前两张图“仅对函数代码启用代码折叠”（第 24-17 页）和“仅对节启用代码折叠”（第 24-17 页）描述了在您分别只对函数代码和只对节启用代码折叠时代码折叠的外观。最后一张图“同时对函数和节启用代码折叠”（第 24-18 页）描述了同时对二者启用代码折叠时的效果。因为第 3 段（第 11–13 行）的折叠与函数 foo 的折叠重叠，所以编辑器不显示第 3 节的折叠。

```

1  function main
2   disp function main % end of function main
3
4  function foo
5   %% section 1
6   disp section 1
7   % end of cell 1
8   %% section 2
9   disp section2
10  % end of cell 2
11  %% section 3
12  disp section 3      % end of function foo
13  % end of cell 3
14
15  function bar
16  disp 'function bar' % end of function bar
17

```

### 仅对函数代码启用代码折叠

```

1  function main
2   disp function main % end of function main
3
4  function foo
5   %% section 1
6   disp section 1
7   % end of cell 1
8   %% section 2
9   disp section2
10  % end of cell 2
11  %% section 3
12  disp section 3      % end of function foo
13  % end of cell 3
14
15  function bar
16  disp 'function bar' % end of function bar
17

```

### 仅对节启用代码折叠

```
1 function main
2 disp function main % end of function main
3
4 function foo
5 %% section 1
6 disp section 1
7 % end of cell 1
8 %% section 2
9 disp section2
10 % end of cell 2
11 %% section 3
12 disp section 3      % end of function foo
13 % end of cell 3
14
15 function bar
16 disp 'function bar' % end of function bar
17 |
```

同时对函数和节启用代码折叠

### 代码重构 - 自动将选定的代码转换为函数

在实时编辑器中，您可以通过自动将选定的代码区域转换为函数或局部函数，将大型实时脚本或函数分成较小的部分。这称为代码重构。

要重构选定的代码区域，请执行以下操作：

- 1 选中一行或多行代码。
- 2 在实时编辑器选项卡的**代码**部分中，点击  **重构**，然后从可用选项中进行选择。
- 3 输入新函数的名称。MATLAB 会用选定的代码创建函数，并将原始代码替换为对新创建函数的调用。

也可以在 MATLAB Online 的 Editor 中使用重构。要在 MATLAB Online 的 Editor 中重构选定的代码区域，请在 **Editor** 选项卡的 **Edit** 部分中，点击代码重构  按钮。然后，从可用选项中进行选择。

# 在文件中查找并替换文本

## 本节内容

- “在当前文件中查找任何文本”（第 24-19 页）
- “在当前文件中查找和替换函数或变量”（第 24-19 页）
- “自动对文件中的所有函数或变量进行重命名”（第 24-20 页）
- “查找和替换任何文本”（第 24-21 页）
- “查找多个文件名或文件中的文本”（第 24-21 页）
- “用于查找文本的备选函数”（第 24-22 页）
- “在编辑器中执行增量搜索”（第 24-22 页）
- “转到文件中的位置”（第 24-22 页）

## 在当前文件中查找任何文本

您可以使用“查找和替换”工具在文件中搜索文本。

- 1 在当前文件中，选择要查找的文本。
- 2 在编辑器或实时编辑器选项卡上的导航部分中，点击  查找，然后选择查找...。
- 3 点击查找下一处以继续查找该文本的更多实例。

要在当前文件中查找选定文本的上一个实例，请点击“查找和替换”对话框上的查找上一处。

## 在当前文件中查找和替换函数或变量

要搜索对特定函数或变量的引用，请对变量和函数使用自动高亮显示功能。此功能比使用文本查找工具更高效。高亮显示函数和变量的方式仅指明对特定函数或变量而非出现的其他函数或变量的引用。例如，它不在注释中查找函数或变量名称的实例。而且，高亮显示变量的方式仅包括对同一变量的引用。即，如果两个变量使用同一名称但在不同范围（第 20-9 页）中，则高亮显示一个变量不会使另一个变量也高亮显示。

要使用自动突出显示功能查找对函数或变量的引用，请按照以下步骤操作：

- 1 在编辑器中打开的文件中，点击要在整个文件中查找的变量实例。MATLAB 将通过以下方式指示该变量在该文件中出现的所有实例：
  - 在整个文件中以蓝绿色（默认）高亮显示它们
  - 在指示标记条中为每个实例添加一个标记
 如果代码分析器指示标记和变量指示标记显示在文件中的同一行上，则变量的标记优先。
- 2 将光标悬停在指示标记条中的某个标记上可查看该标记所代表的行。
- 3 点击指示标记条的某个标记可导航定位到该变量所在之处。
- 4 通过对出现在已导航定位到某行中的函数或变量进行编辑来替换该函数或变量的实例。

下图通过示例说明了启用变量突出显示功能后编辑器的外观。在下图中，变量 *i* 以天蓝色突出显示，并且指示标记条中包含三个变量标记。

```

1  function rowTotals = rowsum
2  % Add the values in each row and
3  % store them in a new array.
4
5  x = ones(2,10);
6  [n, m] = size(x);
7  rowTotals = zeros(1,n);
8  for i = 1:n
9      rowTotals(i) = addToSum;
10 end
11
12 function colsum = addToSum
13     colsum = 0;
14     thisrow = x(i,:); Line14:thisrow = x(i,:);
15     for i = 1:m
16         colsum = colsum + thisrow(i);
17     end
18 end
19
20 end

```

要禁用自动突出显示，请转到主页选项卡，然后在环境部分中，点击  预设项。在 MATLAB > 颜色 > 编程工具中，清除自动突出显示选项。

## 自动对文件中的所有函数或变量进行重命名

为了有助于防止录入错误，MATLAB 提供了一个功能，以帮助您在手动更改以下任何内容时重命名对文件中的函数或变量的引用：

重命名的函数或变量	示例
函数声明中的函数名称	对以下函数声明中的 <b>foo</b> 函数名称进行重命名： <b>function foo(m)</b>
函数声明中的输入或输出变量名称	对以下函数声明中的 <b>y</b> 或 <b>m</b> 输入或输出变量名称进行重命名： <b>function y = foo(m)</b>
赋值语句左侧的变量名称	对以下赋值语句中的 <b>y</b> 变量名称进行重命名： <b>y = 1</b>

在对此类函数或变量进行重命名时，如果文件中存在多处对该变量或函数的引用，则会打开一个工具提示。该工具提示指明 MATLAB 将在您按下 **Shift + Enter** 后对文件中该函数或变量的所有实例进行重命名。

```

1 function rowTotals = rowsum
2 % Add the values in each row and
3 % store them in a new array.
4
5 x = ones(2,10);
6 [n, m] = size(x);
7 rowTotals = zeros(1,n);
8 for j = 1:n
9 Press Shift+Enter to rename 5 instances of "i" to "j"
10 end
11
12 function colsum = addToSum
13 colsum = 0;
14 thisrow = x(:,1);
15 for i = 1:m
16 colsum = colsum + thisrow(i);
17 end
18 end
19
20 end

```

通常情况下，在您使用嵌套函数或局部函数时，会出现对函数的多个调用。

**注意** 在您更改以下内容时，MATLAB 不会提示您：

- 全局变量的名称。
- 函数的输入和输出参数 **varargin** 和 **varargout**。

要撤消对名称所做的自动更改，请点击  一次。

默认情况下，自动重命名变量和函数的功能处于启用状态。要禁用此功能，请执行以下操作：

- 1 在**主页**选项卡上的**环境**部分中，点击  **预设**。
- 2 选择 **MATLAB > 编辑器/调试器 > 语言**。
- 3 在**语言**字段中，选择 **MATLAB**。
- 4 清除**启用变量和函数自动重命名**。

## 查找和替换任何文本

您可以搜索和（可选）替换文件中的指定文本。在**编辑器**或**实时编辑器**选项卡上的**导航**部分中，点击  **查找**以打开并使用“查找和替换”对话框。

### 查找多个文件名或文件中的文本

您可以查找包括指定文本或其内容包含指定文本的文件夹和文件名。在**编辑器**或**实时编辑器**选项卡上的**文件**部分中，点击  **查找文件**以打开“查找文件”对话框。有关详细信息，请参阅“[查找文件和文件夹](#)”。

## 用于查找文本的备选函数

使用 `lookfor` 可在搜索路径上搜索扩展名为 .m 的所有文件中第一行帮助内容里的指定文本。

### 在编辑器中执行增量搜索

当您执行增量搜索时，光标会移至当前文件中指定文本所在的下一处或上一处。它类似于 Emacs 搜索功能。增量搜索在编辑器中使用的控件与在命令行窗口中相同。有关详细信息，请参阅“使用键盘快捷方式进行搜索”。

### 转到文件中的位置

您可以转到文件中的特定位置（例如行号或函数定义）、设置书签、在文件中向后和向前导航，以及从文件中打开文件或变量。

#### 导航到指定位置

下表汇总了在编辑器和实时编辑器中，在打开的文件中导航到特定位置的步骤。

转至位置	步骤	注释
行号	<ol style="list-style-type: none"> <li>1 在编辑器或实时编辑器选项卡上的<b>导航部</b>分中，点击  转至。</li> <li>2 选择<b>转至行...</b></li> <li>3 指定您要导航到的行。</li> </ol>	无
函数定义	<ol style="list-style-type: none"> <li>1 在编辑器或实时编辑器选项卡上的<b>导航部</b>分中，点击  转至。</li> <li>2 在“函数”标题下，选择您要导航到的局部函数或嵌套函数。</li> </ol> <ol style="list-style-type: none"> <li>1 在当前文件夹浏览器中，点击已在编辑器中打开的文件的名称。</li> <li>2 点击当前文件夹浏览器底部的向上箭头  以打开详细信息面板。</li> <li>3 在详细信息面板中，双击与您要导航到的函数或局部函数的标题对应的函数图标 。</li> </ol>	包括局部函数和嵌套函数。  对于类和函数文件，这些函数按字母顺序列出 - 只不过在函数文件中，主函数的名称始终显示在列表的最上方。
代码节	<ol style="list-style-type: none"> <li>1 在编辑器或实时编辑器选项卡上的<b>导航部</b>分中，点击  转至。</li> <li>2 在<b>节</b>下，选择您要导航到的代码节的标题。</li> </ol>	有关详细信息，请参阅“将您的文件分为多个代码节”（第 18-5 页）。

转至位置	步骤	注释
	<p>1 在当前文件夹浏览器中，点击已在编辑器中打开的文件的名称。</p> <p>2 点击当前文件夹浏览器底部的向上箭头  以打开详细信息面板。</p> <p>3 在详细信息面板中，双击与您要导航到的节标题对应的节图标 。</p>	
属性	<p>1 在当前文件夹浏览器中，点击已在编辑器中打开的文件的名称。</p> <p>2 点击当前文件夹浏览器底部的向上箭头  以打开详细信息面板。</p> <p>3 在详细信息面板中，双击与您要导航到的属性的名称对应的属性图标 。</p>	有关详细信息，请参阅“使用属性的方式”。
方法	<p>1 在当前文件夹浏览器中，点击已在编辑器中打开的文件的名称。</p> <p>2 点击当前文件夹浏览器底部的向上箭头  以打开详细信息面板。</p> <p>3 在详细信息面板中，双击与您要导航到的方法的名称对应的图标 。</p>	有关详细信息，请参阅“类设计中的方法”。
书签	<p>1 在编辑器选项卡上的<b>导航</b>部分中，点击  转至 。</p> <p>2 在<b>书签</b>下，选择您要导航到的书签。</p>	有关设置和清除书签的信息，请参阅“设置书签”（第 24-23 页）。

**注意** 详细信息面板和书签在实时编辑器和 MATLAB Online 中不可用。

## 设置书签

您可以在已在编辑器中打开的文件内的任一行上设置书签，以便快速导航到带书签的行。这对于长文件特别有用。例如，假设在处理某行时，您希望查看该文件的另一部分，然后返回。在当前行上设置书签，转到该文件的另一部分，然后使用书签返回。

书签在实时编辑器和 MATLAB Online 中不可用。

要设置书签，请执行以下操作：

- 1 将光标置于该行上的任意位置。
- 2 在编辑器选项卡上的**导航**部分中，点击 转至 .
- 3 在**书签**下，选择**设置/清除**

该行左侧会显示一个书签图标 .

要清除书签，请将光标置于该行上的任意位置。点击 转至 并选择**书签**下的**设置/清除**。

在您关闭文件后，MATLAB 不保留书签。

### 在文件中向后和向前导航

要按您之前导航或编辑文件中的行的顺序访问这些行，请使用 和 。

实时编辑器和 MATLAB Online 不支持向后和向前导航。

如果您执行以下操作，会中断后退和前进顺序：

- 1 点击 .
- 2 点击 .
- 3 使用“导航到指定位置”（第 24-22 页）中所述的功能列表编辑行或导航至另一行。

您仍可以按顺序转到中断点之前的行，但您无法转到该点后的任何行。而您在中断该顺序后所编辑或导航至的行将按顺序添加到中断点之后。

例如：

- 1 打开一个文件。
- 2 编辑第 2 行、第 4 行和第 6 行。
- 3 点击 以返回到第 4 行，然后返回到第 2 行。
- 4 点击 以返回到第 4 和第 6 行。
- 5 点击 以返回到第 1 行。
- 6 在第 3 行中编辑。

这会中断该顺序。您无法再使用 返回到第 4 行和第 6 行。但是，您可以点击 返回到第 1 行。

### 从文件中打开文件或变量

您可以从已在编辑器中打开的文件内打开函数、文件、变量或 Simulink 模型。将光标置于名称上，然后右键点击并从上下文菜单中选择**打开所选内容**。基于所选内容，编辑器会执行不同操作，如此表中所述。

项目	操作
局部函数	导航到当前文件中的局部函数（如果该文件是 MATLAB 代码文件）。如果当前文件中不存在具有该名称的函数，编辑器会对所选内容运行 <code>open</code> 函数，从而用适当的工具来打开所选内容。
文本文件	在编辑器中打开。
图窗文件 (.fig)	在图窗窗口中打开。
位于当前工作区中的 MATLAB 变量	在变量编辑器中打开。
模型	在 Simulink 中打开。 在 MATLAB Online 中不可用。

项目	操作
其他	如果所选内容属于其他某种类型，则 <b>打开所选内容</b> 会在当前文件夹中查找私有文件夹中的匹配文件并执行适当的操作。

## 在文件中添加提醒

通过为文件添加注释可以更轻松地查找您的代码中您打算在日后改进、完成或更新的区域。要为文件添加注释，请使用文本 **TODO**、**FIXME** 或所选的任何文本添加注释。在为若干个文件添加注释后，运行 **TODO/FIXME** 报告，以确定给定文件夹中您已添加注释的所有 MATLAB 代码文件。

此 **TODO/FIXME** 报告示例显示了一个包含文本 **TODO**、**FIXME** 和 **NOTE** 的文件。搜索不区分大小写。

The screenshot shows the MATLAB Web browser displaying a 'TODO/FIXME Report'. The report lists files in the folder H:\Documents\MATLAB. One file, area.m, is shown with its code. Several lines in the code are annotated with markers: a blue 'area' marker above a line of code, a red 'fixme' marker above another line, a green 'fixme' marker above yet another line, and a blue 'note' marker above a final line. The browser interface includes buttons for 'Rerun This Report' and 'Run Report on Current Folder', and a checkbox for selecting 'TODO', 'FIXME', or 'NOTE' types.

## 使用 **TODO/FIXME** 报告

- 1 使用当前文件夹浏览器导航到其中包含了要生成 **TODO/FIXME** 报告的文件的文件夹。

**注意** 您不能在路径为 UNC (通用命名约定) 路径时运行报告；即路径以 \\ 开头。应使用您系统上的实际硬盘驱动器，或映射的网络驱动器。

- 2 在当前文件夹浏览器上，点击 ，然后选择**报告 > TODO/FIXME 报告**。

**TODO/FIXME** 报告在 MATLAB Web 浏览器中打开。

- 3 在 **TODO/FIXME** 报告窗口中，选择以下一项或多项以指定您希望在该报告中包括的行：

- **TODO**
- **FIXME**
- 文本字段复选框

您随后可以在此字段中输入任何文本，包括正则表达式（第 2-48 页）。例如，您可以输入 **NOTE**、**tbd** 或 **re.\*check**。

MATLAB Online 不支持指定自定义文本或要搜索的行。

- 4 通过点击**重新运行此报告**对当前文件夹中的文件运行报告。

窗口将刷新并列出指定文件夹中包含您在步骤 1 中选择的文本的 MATLAB 文件中的所有行。匹配不区分大小写。

如果您想对报告窗口中当前所指定的文件夹以外的文件夹运行报告，请更改当前文件夹。然后，点击**对当前文件夹运行报告**。

要在编辑器中将某个文件打开到某一特定行处，请在报告中点击相应的行号。然后您可以根据需要更改该文件。

假设当前文件夹中有一个文件 **area.m**。**area.m** 的代码显示在随后的图中。

```

1  function output = area(flag,radius)
2  % This function calculates the area of the entity
3  % flag = 1 for calculating the area of a
4  % flag = 2 for calculating the surface area of a sphere
5  % radius = radius of the entity
6
7  switch flag
8      % Modify the function to include the area of square
9      % and rectangle. (todo)
10     case 1
11         output = pi *radius^2;
12     case 2
13         output = 4 * pi * radius^2;
14         % Fixme: Is the area of hemisphere as below?
15         % case 3
16         % output = 2 * pi * radius^2;
17         % FIXME
18     otherwise
19         disp('Incorrect flag')
20         output = NaN;
21         % NOTE: Find out from the manager if we need to include
22         % the area of a cone
23 end

```

当您对包含 `area.m` 的文件夹运行 TODO/FIXME 报告并且选择了文本 TODO 和 FIXME，且指定并选择了文本 NOTE 时，该报告会列出：

`9 and rectangle. (todo)`

`14 Fixme: Is the area of hemisphere as below?`

`17 FIXME`

`21 NOTE: Find out from the manager if we need to include`

<u>area</u>	<u>9</u> and rectangle. (todo) <u>14</u> Fixme: Is the area of hemisphere as below? <u>17</u> FIXME <u>21</u> NOTE: Find out from the manager if we need to include
-------------	--

注意该报告包括以下内容：

- 第 9 行，作为文本 TODO 的匹配项。该报告包括具有所选文本的行，而与该文本在注释中的位置无关。
- 第 14 和 17 行，作为文本 FIXME 的匹配项。该报告会与文件中的所选文本相匹配，而与大小写无关。
- 第 21 行，作为文本 NOTE 的匹配项。该报告包括具有文本字段中指定的文本的行，前提条件是您选择了该文本字段。

# MATLAB 代码分析器报告

## 本节内容

- “运行代码分析器报告”（第 24-28 页）
- “根据代码分析器消息更改代码”（第 24-29 页）
- “访问代码分析器消息的其他方法”（第 24-29 页）

## 运行代码分析器报告

代码分析器报告通过消息显示代码中的潜在错误和问题以及改进机会。例如，一则常见消息会指明可能未使用变量 `foo`。

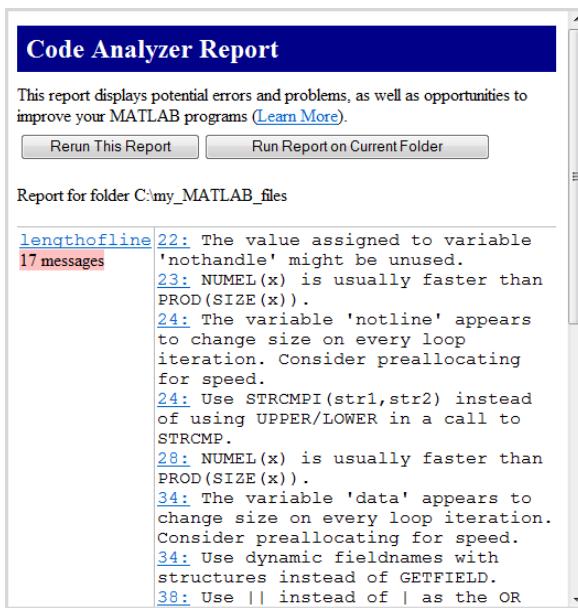
要运行代码分析器报告，请执行以下操作：

- 1 在当前文件夹浏览器中，导航到包含您要检查的文件的文件夹。

要使用本文档中显示的 `lengthofline.m` 示例，请将文件保存到当前文件夹或您具有写访问权限的文件夹中。以下示例将文件保存到当前文件夹 `C:\my_MATLAB_files`。

- 2 `copyfile(fullfile(matlabroot,'help','techdoc','matlab_env','examples','lengthofline.m'))`

报告显示在 MATLAB Web 浏览器中，其中显示了已确定存在潜在问题或改进机会的文件。



- 3 针对报告中的每则消息，查看相关建议及对应的代码。点击行号，即可在编辑器中将文件打开到该行，然后根据消息来更改文件。请遵循以下一般性建议：

- 如果您不确定某则消息的含义或要在代码中更改的内容，请点击该消息中的链接（如果显示有链接）。有关详细信息，请参阅“检查代码中的错误和警告”（第 24-5 页）。
- 如果消息中不含链接，并且您不确定某则消息的含义或需要采取的操作，请在帮助浏览器中搜索相关主题。有关消息以及如何处理它们的示例（包括对示例 `lengthofline.m` 所做的特定更改），请参阅“根据代码分析器消息更改代码”（第 24-29 页）。

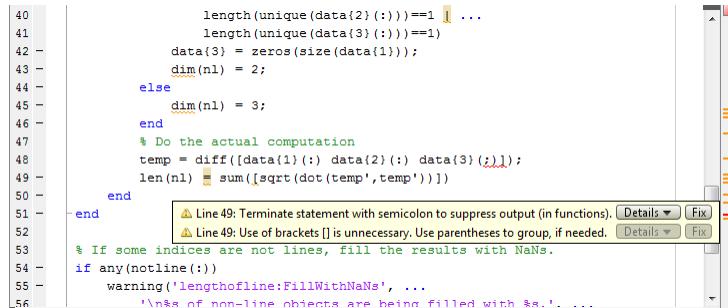
- 这些消息并不提供适合每种情形的信息，在某些情况下，您可能不需要根据消息更改任何内容。有关详细信息，请参阅“了解代码分析的局限”（第 24-11 页）。
  - 如果您不希望看到某些消息或某些类型的消息，可以将其隐蔽起来。有关详细信息，请参阅“调整代码分析器消息指示标记和消息”（第 24-8 页）。
- 4** 修改该文件后，请将它保存。如果您进行了大量更改，可能会带来错误，请考虑以不同的名称来保存该文件。随后，您可以根据需要参考原始文件来解决已更新文件中的问题。使用**编辑器或实时编辑器**选项卡上的 **比较**按钮有助于标识您对文件所做的更改。有关详细信息，请参阅“比较和合并文本”。
- 5** 再次运行并调试文件以确保您没有不经意地引入任何错误。
- 6** 如果正在显示报告，点击**重新运行此报告**可根据您对文件所做的更改来更新报告。根据您对文件所做的更改，确保这些消息不复存在。要在 MATLAB Online 中返回报告，请在当前文件夹浏览器中点击，然后选择**报告 > 代码分析器报告**。

## 根据代码分析器消息更改代码

要了解有关如何更正代码分析器消息中指出的潜在问题的信息，请使用以下资源：

- 在编辑器中打开文件，并点击工具提示中的**详细信息**按钮，如本列表随后的图像所示。此时将打开扩充的消息。但是，并非所有消息都具有扩充的消息。
- 使用帮助浏览器的**搜索窗格**可查找有关消息中提到的术语的文档。

下图显示了带**详细信息**按钮的工具提示。等号 (=) 下边的橙色线条指明当您将鼠标指针悬停于等号上时会显示工具提示。橙色的高亮显示方式指示可以使用自动修复。



```

40      length(unique(data(2,:)'))==1 | ...
41      length(unique(data(3,:)'))==1
42      data(3) = zeros(size(data(1)));
43      dim(nl) = 2;
44  else
45      dim(nl) = 3;
46  end
47  % Do the actual computation
48  temp = diff([data(1,:) data(2,:) data(3,:)]);
49  len(nl) = sum(sqrt(dot(temp',temp')));
50
51  end
52
53  % If some indices are not lines, fill the results with NaNs.
54  if any(notline(:))
55      warning('lengthofline:FillWithNaNs', ...
56          '\n% of non-line objects are being filled with %s.', ...

```

## 访问代码分析器消息的其他方法

使用下列任一方法均可获取代码分析器消息。每种方法都提供相同的消息，但采用的格式不同：

- 通过探查器详细信息报告来访问某文件的代码分析器报告。
- 运行 **checkcode** 函数，以便分析指定的文件并在命令行窗口中显示消息。
- 运行 **mlintrpt** 函数，以便运行 **checkcode** 并在 Web 浏览器中显示消息。
- 在编辑器中处理文件时使用代码的自动检查功能。请参阅“在编辑器中自动检查代码 - 代码分析器”（第 24-5 页）。

# MATLAB 代码兼容性报告

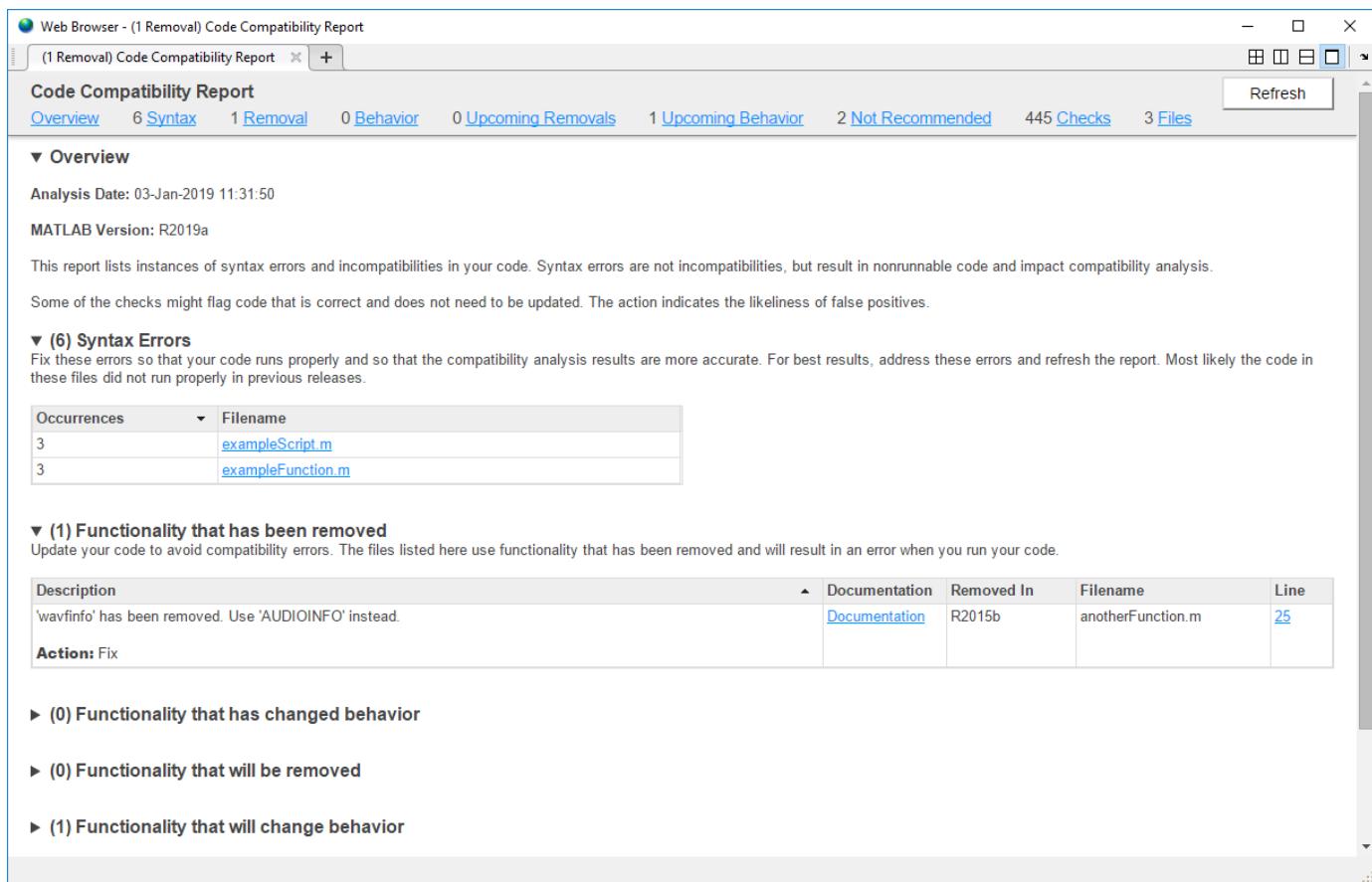
## 生成代码兼容性报告

代码兼容性报告显示您的代码中可能存在的兼容性问题。当您升级到较新版本的 MATLAB 时，此报告有助于确定现有代码中潜在的兼容性问题。例如，如果您在代码中使用了不鼓励使用的或已删除的函数或者无效的语法，报告会向您告知这些情况。升级到较新版本的 MATLAB 后，您可以使用此报告来确定现有代码中潜在的兼容性问题。

要运行代码兼容性报告，请执行下列操作：

- 1 在当前文件夹浏览器中，导航到包含您要分析的代码文件的文件夹。
- 2 在当前文件夹浏览器中，点击  或右键点击空白区域。然后选择报告 > 代码兼容性报告。

报告将显示在 MATLAB Web 浏览器中，指出潜在的兼容性问题。例如：



The screenshot shows the MATLAB Web Browser interface with a 'Code Compatibility Report' tab selected. The report summary indicates 6 Syntax errors, 1 Removal, 0 Behavior changes, 0 Upcoming Removals, 1 Upcoming Behavior, 2 Not Recommended checks, 445 Checks total, and 3 Files analyzed. The 'Overview' section shows analysis details for Jan 2019, R2019a MATLAB version, and a note about false positives. The 'Syntax Errors' section lists two files: 'exampleScript.m' and 'exampleFunction.m', each with 3 occurrences. The 'Functionality that has been removed' section shows a single entry for 'wavinfo' being removed, with an action link to 'Fix'. Below these are sections for 'Functionality that has changed behavior' (0), 'Functionality that will be removed' (0), and 'Functionality that will change behavior' (1).

- 3 更新代码以解决语法错误部分列出的每个文件中的语法错误。语法错误会导致代码不能运行。虽然代码在以前的版本中很可能运行不正常，但是语法错误会影响兼容性分析。例如，在 '}' 处出现解析错误：使用的 MATLAB 语法可能无效。。
- 4 对于报告中列出的每项功能，请认真查看问题描述和您的代码。消息包含行号，可帮助您在代码中定位问题。要在编辑器中打开文件并定位到该行，请点击行号。然后根据消息更改文件。如果您不能确定某条消息的含义或要更改代码中的哪些内容，请点击与消息对应的文档链接。

报告中列出的每项功能都会显示建议的操作。您也可以遵循以下一般性建议：

- **已经删除的功能** - 更新您的代码以避免当前版本中的兼容性错误。
- **行为已经改变的功能** - 确认行为的更改是可接受的，如果不可接受，请针对当前版本更新您的代码。
- **将要删除的功能** - 现在更新您的代码，也可在以后的版本中更新。现在更新您的代码会使将来的升级更加容易。
- **行为将要改变的功能** - 立即调查这些变化，以便将来的升级更容易。
- **不建议使用的功能** - 考虑更新您的代码。预计代码将在以后的版本中继续正常运行，但其中使用了不推荐的功能。

代码兼容性报告还包含有关对您的代码执行的检查的信息，以及进行了 MATLAB 代码兼容性分析的文件列表。

## 编程用法

当您通过当前文件夹浏览器生成代码兼容性报告时，MATLAB 会分析当前工作文件夹及其子文件夹中的代码。但是，如果以编程方式生成报告，则可以指定要分析的具体文件或从分析中排除子文件夹。要以编程方式生成报告，请使用以下方法之一。

- 要以编程方式生成在 MATLAB® Web 浏览器中打开的报告，请使用 `codeCompatibilityReport` 函数。
- 要返回包含报告信息的 `CodeCompatibilityAnalysis` 对象，请使用 `analyzeCodeCompatibility` 函数。然后，您可以使用 `codeCompatibilityReport` 函数显示所存储对象的报告。

## 另请参阅

`CodeCompatibilityAnalysis` | `analyzeCodeCompatibility` | `codeCompatibilityReport`



# 编程实用工具

---

- “确定程序依赖项” (第 25-2 页)
- “保护您的源代码” (第 25-6 页)
- “创建运行函数的超链接” (第 25-8 页)
- “创建和共享工具箱” (第 25-10 页)

## 确定程序依赖项

如果您需要知道您的程序所依赖的其他函数和脚本，请使用下面所述的一种方法。

### 简单显示程序文件依赖项

要简单显示特定函数引用的所有程序文件，请执行下列步骤：

- 1 键入 `clear functions` 以从内存中清除所有函数（请参阅下面的注释）。

**注意** `clear functions` 不清除 `mlock` 锁定的函数。如果存在锁定函数（可使用 `inmem` 检查是否存在这种函数），请使用 `munlock` 取消解除对函数的锁定，然后重复步骤 1。

- 2 执行您要检查的函数。请注意，您在此步骤中选择使用的函数参数很重要，这是因为您在调用同一函数时使用了不同的参数会得到不同的结果。
- 3 键入 `inmem` 以显示在函数运行时使用的所有程序文件。如果您还想查看使用了哪些 MEX 文件，请另外指定一个输出：

```
[mfiles, mexfiles] = inmem
```

### 详细显示程序文件依赖项

要详细显示依赖函数的信息，请使用 `matlab.codetools.requiredFilesAndProducts` 函数。除了程序文件之外，`matlab.codetools.requiredFilesAndProducts` 还显示特定函数依赖哪些 MathWorks 产品。如果有一个函数 `myFun`，它调用 Image Processing Toolbox™ 中的 `edge` 函数：

```
[fList,pList] = matlab.codetools.requiredFilesAndProducts('myFun.m');  
fList
```

```
fList =  
  
'C:\work\myFun.m'
```

唯一必需的程序文件是函数文件本身，即 `myFun`。

```
{pList.Name}'  
  
ans =  
  
'MATLAB'  
'Image Processing Toolbox'
```

文件 `myFun.m` 需要 MATLAB 和 Image Processing Toolbox。

### 文件夹中的依赖项

依存关系报告显示文件夹中的 MATLAB 代码文件之间的依存关系。使用该报告可确定：

- 同一文件夹中的哪些文件是其他文件所必需的
- 一旦将某文件删除，当前文件夹中是否有任何文件失败
- 当前文件夹中是否缺少任何调用的文件

该报告并不列出：

- **toolbox/matlab** 文件夹中的文件，这是因为每个 MATLAB 用户都有这些文件。

因此，如果您使用某个函数文件隐蔽了内置函数文件，MATLAB 会从列表中排除这两个文件。

- 从匿名函数调用的文件。
- 类文件的超类。
- 通过 eval、evalc、run、load、函数句柄和回调中调用的文件。

MATLAB 在运行时之前并不解析这些文件，因此依存关系报告无法发现它们。

- 某些方法文件。

依存关系报告查找您在 MATLAB 文件中调用的类构造函数。但是，对于该报告而言，您对生成对象执行的任何方法都是未知的。这些方法既可作为单独的方法文件，也可作为属于某个方法文件的超类的文件存在于 classdef 文件中。

要提供有意义的结果，依存关系报告要求满足以下条件：

- 您运行报告时的搜索路径与您运行文件夹中的文件时相同。（即当前文件夹位于搜索路径的顶端。）
- 您运行报告的文件夹中的文件不会改变搜索路径，否则会控制报告。
- 文件夹中的文件并不加载变量，否则会产生致使不同程序元素同名的名称冲突。

---

**注意** 请勿使用依存关系报告来确定其他人运行特定文件需要哪些 MATLAB 代码文件。请改用 **matlab.codetools.requiredFilesAndProducts** 函数。

---

## 创建依存关系报告

- 1 使用当前文件夹窗格导航到准备生成依存关系报告的文件所在的文件夹。

---

**注意** 您不能在路径为 UNC（通用命名约定）路径时运行报告；即路径以 \\ 开头。应使用您系统上的实际硬盘驱动器，或映射的网络驱动器。

---

- 2 在当前文件夹窗格上，点击 ，然后选择报告 > 依存关系报告。

依存关系报告随即在 MATLAB Web 浏览器中打开。

- 3 如果需要，请选择该报告中的一个或多个选项，如下所示：

- 要查看该文件夹（父级）中的每个文件调用的所有 MATLAB 代码文件（子级）的列表，请选择 **显示子函数**。

该报告指示每个子函数所在的位置，例如在指定工具箱中。如果该报告指定子函数的位置是未知的，则可能是因为：

- 子函数不在搜索路径上。
- 子函数不在当前文件夹中。
- 该文件已移动或删除。
- 要列出调用每个 MATLAB 代码文件的文件，请选择 **显示父函数**。

该报告将父（调用）函数限制为当前文件夹中的函数。

- 要在报告中包括局部函数，请选择 **显示子函数**。该报告在主函数后直接列出局部函数并以灰色高亮显示这些函数。

#### 4 点击对当前文件夹运行报告。

##### 阅读和使用依存关系报告

下图显示了依存关系报告。它指明 `chirpy.m` 调用 Signal Processing Toolbox™ 中的两个文件以及 Image Processing Toolbox 中的一个文件。它还显示 `go.m` 调用 `mobius.m`，后者位于当前文件夹中。

Dependency Report	
The Dependency Report shows dependencies among MATLAB files in a folder ( <a href="#">Learn More</a> ).	
<input type="button" value="Rerun This Report"/>	<input type="button" value="Run Report on Current Folder"/>
<input checked="" type="checkbox"/> Show child functions	<input type="checkbox"/> Show parent functions (current folder only)
<input type="checkbox"/> Show subfunctions	
Built-in functions and files in toolbox/matlab are not shown	
Report for Folder l:\my_MATLAB_files	
Children (called functions)	
<u>chirpy</u>	toolbox : \images\images\erode.m toolbox : \shared\siglib\chirp.m toolbox : \signal\signal\specgram.m
<u>collatz</u>	
<u>collatzall</u>	subfunction : <u>collatzplot new</u>
<u>collatzplot</u>	current dir : <u>collatz</u>
<u>collatzplot new</u>	current dir : <u>collatz</u>
<u>go</u>	current dir : <u>mobius</u>
<u>mobius</u>	

依存关系报告包括以下内容：

- MATLAB 文件列表

该文件夹中已运行依存关系报告的文件列表。点击此列中的链接可在编辑器中打开相应的文件。

- 子级

MATLAB 文件调用的一个或多个函数。

点击此列中的链接可打开列在同一行中的 MATLAB 文件，并转到对被调用函数的第一个引用。例如，假设您的依存关系报告如上一张图所示。点击 `\images\images\erode.m` 便会打开 `chirpy.m` 并将光标置于引用 `erode` 的第一行。换而言之，它不会打开 `erode.m`。

- 多个类方法

因为该报告是一种静态分析，所以它无法确定运行时数据类型，因而无法确定文件所需的特定类方法。如果有多个类方法与引用的方法相匹配，则依存关系报告会在文件名旁边插入一个问号链接。下图中显示了该问号。

<a href="#">rational differential</a>	toolbox : \rf\rf\s2sdd.m toolbox : \rf\rf\@rfckt\smith.m toolbox : \rf\rf\s2tf.m toolbox : \rf\rf\rationalfit.m toolbox : ? Multiple class methods match <a href="#">freqresp.m</a> toolbox : \rf\rf\@rfmodel\timeresp.m toolbox : \rf\rf\@rfckt\analyze.m toolbox : \signal\signal\fftfilt.m
---------------------------------------	--

点击问号链接可列出 MATLAB 可能使用的、具有指定名称的类方法。MATLAB 列出搜索路径上与指定的方法文件（本例中为 `freqresp.m`）匹配的几乎所有方法文件。如果列表包含您不熟悉的类方法和 MATLAB 内置函数，也不必担心。

您无需确定 MATLAB 将使用哪个文件。MATLAB 根据程序在运行时调用的对象确定要使用的方法。

## 保护您的源代码

虽然 MATLAB 源代码 (.m) 本身是可执行的，但 MATLAB 源文件的内容很容易被访问，会揭示设计和实现的详细信息。如果您不想以这种格式分发您的专有应用程序代码，可以改用以下选项之一：

- 以 P 代码的形式部署（第 25-6 页） - 将您的某些或全部源代码文件转换为一种名为 P 代码文件（文件扩展名为 .p）的掩盖内容的格式，并以此格式来分发您的应用程序代码。当 MATLAB 将某文件转换为 P 代码文件时，该文件将进行模糊处理而非加密。虽然 .p 文件中的内容难以理解，但不应将其视为安全的。建议不要使用 P 代码文件以便保护您的知识产权。  
MATLAB 不支持将实时脚本或实时函数转换为 P 代码文件。
- 编译为二进制格式（第 25-7 页） - 使用 MATLAB Compiler 编译您的源代码文件以生成独立应用程序。将后者分发给应用程序的最终用户。

### 使用 P 代码构建掩盖内容的格式

P 代码文件的行为方式与生成该文件的 MATLAB 源文件相同。P 代码文件的运行速度也与源文件相同。P 代码文件有意进行了模糊处理。它们未被加密。虽然 .p 文件中的内容难以理解，但不应将其视为安全的。建议不要使用 P 代码文件以便保护您的知识产权。

**注意** 因为 P 代码文件的用户不能查看 MATLAB 代码，所以考虑提供诊断以使用户能够在出现错误时继续操作。

---

#### 构建 P 代码文件

要生成 P 代码文件，请在 MATLAB 命令行窗口中输入以下命令：

```
PCODE file1 file2, ...
```

该命令生成文件 file1.p、file2.p 等。要将当前文件夹中的所有 .m 源文件转换为 P 代码文件，请使用以下命令：

```
PCODE *.m
```

有关用于生成 P 代码文件的所有语法的说明，请参阅 **PCODE** 函数参考页。

#### 调用 P 代码文件

调用生成的 P 代码文件的方式与调用派生该文件的 MATLAB .m 源文件的方式相同。例如，要调用 myfun.p 文件，请键入

```
[out, out2, ...] = myfun(in1, in2, ...);
```

要调用 myscript.p 脚本，请键入

```
myscript;
```

当您调用 P 代码文件时，MATLAB 以高于 .m 源文件的优先级来执行该代码文件。即使您在生成 P 代码文件后的某时刻碰巧更改了源代码也是如此。请记住在分发您的代码之前删除 .m 源文件。

#### 对较高版本的 MATLAB 运行较早的 P 代码文件

P 代码文件的设计旨在使得所创建的文件不受发行版本以及应用版本的影响（向后和向前兼容性）。MATLAB 的新增功能以及弃用的功能可能会造成问题，但如果使用原始的 MATLAB 输入文件，也会存

在同样的问题。要修复 P 代码文件中的此类错误，请修复相应的 MATLAB 输入文件并创建一个新的 P 代码文件。

使用 MATLAB 7.4 版本及更早版本构建的 P 代码文件与使用更高版本的 MATLAB 构建的 P 代码文件具有不同格式。这些早期的 P 代码文件不能在 MATLAB 8.6 (R2015b) 或更高版本中运行。请使用更新版本的 MATLAB 重新构建使用 MATLAB 7.4 或更早版本构建的任何 P 代码文件，然后根据需要重新进行分发。

## 构建独立的可执行文件

保护您的源代码的另一种方法是将其置于独立的可执行文件中，并将该可执行文件与所需的任何其他文件一起分发给外部客户。您必须安装 MATLAB Compiler 及受支持的 C 或 C++ 编译器，才能准备要部署的文件。但最终用户无需安装 MATLAB。

要针对您的 MATLAB 应用程序构建独立的应用程序，请按照 MATLAB 程序文件的惯常过程开发并调试您的应用程序。然后，按照“Create Standalone Application from MATLAB”(MATLAB Compiler)中的说明生成一个或多个可执行文件。

## 创建运行函数的超链接

使用特殊关键字 **matlab:** 可以在其他函数中嵌入命令。最常见的情形是，包含特殊关键字的函数会显示超链接，这样当您点击超链接文本时便会执行相应的命令。支持 **matlab:** 语法的函数包括 **disp**、**error**、**fprintf**、**help** 和 **warning**。

使用 **matlab:** 语法可在命令行窗口中创建一个超链接，以此运行一个或多个函数。例如，您可以使用 **disp** 将单词 Hypotenuse 显示为一个可执行的超链接，如下所示：

```
disp('<a href="matlab:a=3; b=4;c=hypot(a,b)">Hypotenuse</a>')
```

点击该超链接可执行 **matlab:** 后的三个命令，这将生成

```
c =  
5
```

执行该链接将在基础工作区中创建或重新定义变量 **a**、**b** 和 **c**。

**disp** 的参数是一个 **<a href>** HTML 超链接。将从 '**<a href=** 到 **</a>**' 的完整超链接文本包括在一行中，即：不要在新行上继续长文本。开头的 **<** 之后及结尾的 **>** 之前不允许使用任何空格。**a** 和 **href** 之间需要使用一个空格。

您不能直接执行 **matlab:** 语法。即，如果您键入

```
matlab:a=3; b=4;c=hypot(a,b)
```

则会收到错误，这是因为 MATLAB 将非法上下文中的冒号解释为数组运算符：

```
??? matlab:a=3; b=4;c=hypot(a,b)  
|  
Error: The expression to the left of the equals sign  
is not a valid target for an assignment.
```

您无需使用 **matlab:** 即可显示指向 Web 的活动超链接。例如，如果您希望链接到外部网页，可以使用 **disp**，如下所示：

```
disp('<a href="http://en.wikipedia.org/wiki/Hypotenuse">Hypotenuse</a>')
```

命令行窗口中的结果与上一示例看似相同，但却打开 en.wikipedia.org 上的一个页面：

**Hypotenuse**

使用 **matlab:**，您可以：

- “运行单个函数” (第 25-8 页)
- “运行多个函数” (第 25-9 页)
- “提供命令选项” (第 25-9 页)
- “包括特殊字符” (第 25-9 页)

## 运行单个函数

使用 **matlab:** 可在您点击命令行窗口中的超链接时运行指定的语句。例如，运行此命令：

```
disp('<a href="matlab:magic(4)">Generate magic square</a>')
```

它在命令行窗口中显示以下链接：

[Generate magic square](#)

当您点击该链接时， MATLAB 运行 `magic(4)`。

## 运行多个函数

您可以使用单个链接运行多个函数。例如，运行此命令：

```
disp('<a href="matlab: x=0:1:8;y=sin(x);plot(x,y)">Plot x,y</a>')
```

它在命令行窗口中显示以下链接：

[Plot x,y](#)

当您点击该链接时， MATLAB 运行此代码：

```
x = 0:1:8;
y = sin(x);
plot(x,y)
```

在基础工作区中重新定义 `x`：

```
x = -2*pi:pi/16:2*pi;
```

再次点击超链接 `Plot x,y`， 它会将 `x` 的当前值更改回 `0:1:8`。`matlab:` 在您点击 `Plot x,y` 时运行的代码会在基础工作区中定义 `x`。

## 提供命令选项

在文件中使用多个 `matlab:` 语句提供选项，例如

```
disp('<a href = "matlab:state = 0">Disable feature</a>')
disp('<a href = "matlab:state = 1">Enable feature</a>')
```

命令行窗口显示跟随的链接。根据您点击的链接， MATLAB 将 `state` 设置为 0 或 1。

[Disable feature](#)  
[Enable feature](#)

## 包括特殊字符

MATLAB 可正确解释包括特殊字符（例如大于号 (`>`)）在内的大多数文本。例如，以下语句包括大于号 (`>`)。

```
disp('<a href="matlab:str = "Value > 0"">Positive</a>')
```

并生成以下超链接。

[Positive](#)

某些符号的解释可能不正确，因而您可能需要使用该符号的 ASCII 值。例如，运行前面的语句的替代方法是使用 ASCII 62 而不是大于号。

```
disp('<a href="matlab:str=[ "Value " char(62) " 0"]">Positive</a>')
```

## 创建和共享工具箱

### 本节内容

“创建工具箱” (第 25-10 页)

“共享工具箱” (第 25-14 页)

您可以对 MATLAB 文件进行打包，以创建可与其他人共享的工具箱。这些文件可以包含 MATLAB 代码、数据、App、示例和文档。当您创建工具箱时，MATLAB 会生成一个安装文件 (.mltbx)，您或其他人可利用该文件来安装工具箱。

## 创建工具箱

如要创建工具箱安装文件：

- 1 在主页选项卡的环境部分中，从附加功能菜单中选择  工具箱打包。
- 2 在“打包为工具箱”对话框中，点击  按钮并选择您的工具箱文件夹。比较好的做法是从您工具箱文件夹的上一级文件夹创建工具箱文件包。.mltbx 工具箱文件包含了工具箱文件和文件夹路径设置的有关信息。默认情况下，您创建工具箱时所包含的位于您的路径上的文件夹和文件，将在最终用户安装工具箱后出现在他们的路径上。
- 3 在对话框中，添加有关您的工具箱的以下信息。

工具箱信息字段	说明
工具箱名称	输入工具箱名称（如有必要）。默认情况下，工具箱名称为工具箱文件夹的名称。工具箱名称即 .mltbx 的文件名。
版本	按照 Major.Minor.Bug.Build 的格式输入工具箱版本号。Bug 和 Build 为可选项。
作者姓名、电子邮件和公司	输入工具箱作者的联系信息。要保存联系信息，请点击设置为默认联系人。
工具箱图像	要选择表示您的工具箱的图像，请点击选择工具箱图像。
摘要和说明	输入工具箱的摘要和说明。比较好的做法是编写简洁的摘要文本，并在说明文本中添加详细信息。

- 4 要确保 MATLAB 检测到期望的组件，请审核工具箱内容。选择工具箱文件夹后，将会显示“打包为工具箱”对话框的以下各部分。

“打包为工具箱”对话框中的部分	说明
<b>工具箱文件和文件夹</b>	<p>您的工具箱中所包含文件夹和文件的列表。仅列出位于工具箱文件夹顶层的文件和文件夹。您无法在“工具箱打包”对话框中浏览所有文件夹。</p> <p>默认情况下，如果您的工具箱在同一个文件夹中包含同名的 P 代码文件和 MATLAB 代码文件 (.m)，则 MATLAB 会从工具箱中排除 .m 文件。要同时包含 .p 和 .m 文件，请清除<b>排除具有匹配的 P 文件的 MATLAB 脚本或函数文件</b>选项。</p> <p>要从工具箱中排除其他文件或文件夹，请点击<b>排除文件和文件夹</b>，然后在显示的文本文件中对其进行注册。比较好的做法是排除与您的工具箱相关的任何源代码管理文件。</p>
<b>要求</b>	<p>附加功能 - 工具箱所需的附加功能列表。安装工具箱时，系统会下载并安装选定的附加功能。MATLAB 用它认为工具箱需要的附加功能自动填充此列表，默认将选择所有附加功能。您可以选择忽略不想与工具箱一起安装的附加功能。</p> <p>如果 MATLAB 在列表中找不到附加功能的安装信息，您必须输入下载 URL。下载 URL 是 MATLAB 可以下载和安装附加功能的位置。安装工具箱时，MATLAB 使用指定的 URL 安装附加功能。</p>
<b>安装操作</b>	<p>MATLAB 路径 - 在用户安装工具箱时添加到其 MATLAB 路径中的文件夹列表。默认情况下，该列表包括您在创建工具箱时路径上的所有工具箱文件夹。您可以不将文件夹添加到用户的路径，只需在列表中清除它们即可。要管理安装工具箱时的路径，请点击<b>管理当前 MATLAB 路径</b>。要将列表重置为默认列表，请点击<b>重置为当前的 MATLAB 路径</b>。</p> <p>Java 类路径 - 在用户安装工具箱时添加到其 Java 类路径中的 Java 文件列表。安装工具箱时，JAR 文件会在 MATLAB 会话期间添加到动态路径中。当工具箱用户重新启动 MATLAB 时，JAR 文件将添加到静态路径中。</p>

“打包为工具箱”对话框中的部分	说明
	<p>安装其他软件 - 在用户安装工具箱时安装到用户系统上的其他软件的 ZIP 文件列表。</p> <p>指定以下字段：</p> <ul style="list-style-type: none"> <li>• <b>显示名称</b> - 在用户安装工具箱时向他们显示的名称。</li> <li>• <b>许可证 URL</b> - 在用户安装工具箱时向他们显示的其他软件许可协议的 URL。在安装过程中，系统会提示用户阅读并接受许可协议。您必须为许可协议指定有效的 URL。</li> <li>• <b>下载 URL</b> - 包含其他软件的 ZIP 文件的 URL。要为不同的平台指定不同的下载 URL，请从下载 URL 左侧的下拉菜单中选择平台名称。然后，点击<b>添加平台</b>，为其他平台添加下载 URL。</li> </ul> <p>当用户安装工具箱时，MATLAB 会将所有其他软件安装在 <code>addons\Toolboxes\AdditionalSoftware</code> 文件夹中，其中 <code>addons</code> 是附加功能的默认安装文件夹。有关附加功能默认安装文件夹位置的详细信息，请参阅“<a href="#">获取和管理附加功能</a>”。</p> <p>如果您的工具箱包含引用指定附加软件的安装文件夹的代码，需确保这些引用可移植到其他计算机。请将引用替换为对生成的函数 <code>toolboxname\getInstallationLocation.mlx</code> 的调用，其中 <code>toolboxname</code> 是工具箱的名称。例如，如果要创建名为 <code>mytoolbox</code> 的工具箱，并且要引用名为 <code>mysoftware</code> 的附加软件的安装位置，请将以下代码</p> <pre>mysoftwarelocation = 'C:\InstalledSoftware\mysoftware\'</pre> <p>替换为以下代码：</p> <pre>mysoftwarelocation = mytoolbox.getInstallationLocation('mysoftware')</pre> <p>要在打包工具箱之前在计算机上启用工具箱测试，请在<a href="#">安装其他软件</a>部分点击底部的 <code>toolboxname\getInstallationLocation.mlx</code> 链接，然后输入每个附加软件在您计算机上的安装位置。</p>
<b>工具箱可移植性</b>	当用户安装工具箱时，MATLAB 使用 <b>工具箱可移植性</b> 部分中的信息。如果由于用户使用的平台或 MATLAB 版本不受支持而导致兼容性检查失败，MATLAB 会显示警告。但是，用户仍可以安装工具箱。
	平台兼容性 - 支持工具箱的平台的列表。考虑您的工具箱是否具有针对特定平台的第三方软件或硬件要求。MATLAB Online 不能与硬件交互，包括用于图像采集和仪器控制的设备。
	版本兼容性 - 支持工具箱的 MATLAB 版本的列表。
	产品 - 您的工具箱所需的 MathWorks 产品的列表。手动创建此列表。

“打包为工具箱”对话框中的部分	说明
示例、App 和文档	<p>示例 - 已发布的与工具箱关联的 MATLAB 示例。要包含 .m 和 .mlx 文件作为示例，请点击<b>添加示例</b>按钮，选择您的代码文件，然后点击<b>发布 HTML</b>。MATLAB 将代码发布为 HTML 并将输出文件放入 html 文件夹中。</p>
	<p>您也可以在 MATLAB 中手动将代码文件发布为 HTML，然后将代码文件和 HTML 文件包含在工具箱文件夹中。</p>
	<ul style="list-style-type: none"> <li>• 对于实时脚本 (.mlx) 示例，请将其导出为 HTML。在<b>实时编辑器</b>选项卡上，选择<b>保存 &gt; 导出为 HTML</b>，然后将其保存在名为 html 的文件夹中。</li> <li>• 对于脚本 (.m) 示例，请使用 <b>publish</b> 函数将其发布为 HTML。当您发布示例时请勿指定输出文件夹。为了“打包为工具箱”工具能识别示例，输出文件夹必须是默认文件夹 (html)。</li> </ul>
	<p>如要为您的示例创建不同的类别，请将示例放置在工具箱文件夹内不同的各个子文件夹中。当您将工具箱文件夹添加到“打包为工具箱”对话框中时，MATLAB 会创建一个 <b>demos.xml</b> 文件来描述您的示例，并用示例子文件夹的名称作为示例类别名称。您也可以创建您自己的 <b>demos.xml</b> 文件。<b>demos.xml</b> 文件允许接收者通过位于帮助浏览器主页面底部的<b>补充软件</b>链接来访问您的示例。有关详细信息，请参阅“显示自定义示例”（第 31-26 页）。</p>
	<p>App - 已发布的与工具箱关联的 MATLAB 可安装 App。“打包为工具箱”工具可识别 App (.mlapp 文件) 和 App 安装程序文件 (.mlappinstall 文件)，并将它们包含在您的工具箱中。</p> <ul style="list-style-type: none"> <li>• 要指定哪些 App (.mlapp 文件) 将随工具箱安装并在用户的 MATLAB App 库中注册，请选择这些 App。</li> <li>• 工具箱文件夹中的所有 .mlappinstall 文件都将安装并在用户的 MATLAB App 库中注册。</li> </ul>
	<p>入门指南 - 工具箱快速入门指南。为了“打包为工具箱”工具能识别入门指南，请将该指南保存为实时脚本并命名为 <b>GettingStarted.mlx</b>，置于工具箱文件夹的 <b>doc</b> 子文件夹中。</p> <p>您也可以通过“打包为工具箱”对话框来生成并编辑 <b>GettingStarted.mlx</b>。</p> <p>工具箱的用户可以在附加功能管理器中通过工具箱的“选项”菜单来查看入门指南。有关详细信息，请参阅“获取和管理附加功能”。</p>
	<p>帮助浏览器集成 - 与工具箱关联的自定义文档。为了“打包为工具箱”工具能识别自定义文档，请包含一个 <b>info.xml</b> 文件来识别您的文档文件。如果您在打包工具箱之前使用 <b>builddocsearchdb</b> 函数编译文档数据库，则可以将生成的 <b>helpsearch</b> 子文件夹纳入工具箱。<b>info.xml</b> 文件和 <b>helpsearch</b> 文件夹允许接收者通过位于帮助浏览器主页面底部的<b>补充软件</b>链接来访问您的文档。有关详细信息，请参阅“显示自定义文档”（第 31-20 页）。</p> <p>您也可以通过“打包为工具箱”对话框生成 <b>info.xml</b> 和 <b>helptoc.xml</b> 模板文件。要通过帮助浏览器访问您的文档，请完成文档模板并将 <b>info.xml</b> 包含在 MATLAB 路径中。</p>

## 5 将您的工具箱打包。

- 要保存您的工具箱，请点击位于“打包为工具箱”对话框顶部的**打包**。对您的工具箱进行打包会在您的当前 MATLAB 文件夹中生成一个 **.mltbx** 文件。
- 要保存您的工具箱并将其在 MATLAB Central File Exchange 中共享，请从“打包为工具箱”对话框顶部的**打包**菜单中选择**打包和共享**。此选项会在您的当前 MATLAB 文件夹中生成 **.mltbx** 文件，并打开一个网页以将工具箱提交给 File Exchange。MATLAB 会将工具箱的相关信息填充到 File Exchange 提交表单中。检查并提交该表单以在 File Exchange 中共享您的工具箱。

当您创建工具箱时，MATLAB 会生成一个包含工具箱有关信息的 **.prj** 文件，并频繁保存该文件。比较好的做法是保存此关联的 **.prj** 文件，以便您能在未来迅速创建工具箱的修订版本。

## 共享工具箱

如要与其他人共享您的工具箱，则可以为他们提供 **.mltbx** 文件。您在打包工具箱时添加的所有文件都包含在 **.mltbx** 文件中。当最终用户安装您的工具箱时，他们无需担心 MATLAB 路径或其他安装详细信息。**.mltbx** 文件会为最终用户管理这些详细信息。

如需了解安装、卸载和查看工具箱信息的相关信息，请参阅“[获取和管理附加功能](#)”。

您可以与其他人共享您的工具箱，方法是将 **.mltbx** 文件添加到电子邮件消息的附件，或使用您通常用来共享文件的任何其他方法，例如上传到 MATLAB Central File Exchange。如果您将工具箱上传到 File Exchange，则您的用户可以从 MATLAB 中下载该工具箱。有关详细信息，请参阅“[获取和管理附加功能](#)”。

您也可以在打包工具箱时将其上传到 File Exchange。从“打包为工具箱”对话框顶部的**打包**菜单中选择**打包和共享**。

**注意** 虽然 **.mltbx** 文件可以包含您指定的任何文件，但 MATLAB Central File Exchange 对提交的文件设置了其他限制。如果您的工具箱包含了以下任意内容，则无法提交到 File Exchange：

- MEX 文件。
- 其他二进制可执行文件，例如 DLL 或 ActiveX® 控件。（数据和图像文件通常可以接受。）

---

## 另请参阅

[matlab.addons.toolbox.installToolbox](#) | [matlab.addons.toolbox.installedToolboxes](#) |  
[matlab.addons.toolbox.packageToolbox](#) | [matlab.addons.toolbox.toolboxVersion](#) |  
[matlab.addons.toolbox.uninstallToolbox](#) | [publish](#)

## 相关示例

- “[获取和管理附加功能](#)”
- “[显示自定义示例](#)”（第 31-26 页）
- “[从 MATLAB 工具条打包 App](#)”
- “[显示自定义文档](#)”（第 31-20 页）

# 函数参数验证

---

- “函数参数验证” (第 26-2 页)
- “参数验证函数” (第 26-19 页)
- “解析函数输入的方法” (第 26-21 页)
- “MATLAB 代码中的透明” (第 26-22 页)

# 函数参数验证

## 本节内容

- “参数验证简介” (第 26-2 页)
- “何时使用参数验证” (第 26-2 页)
- “arguments 代码块语法” (第 26-3 页)
- “参数验证示例” (第 26-4 页)
- “参数的种类” (第 26-6 页)
- “必需和可选位置参数” (第 26-6 页)
- “重复参数” (第 26-8 页)
- “名称-值参数” (第 26-10 页)
- “基于类属性的名称-值参数” (第 26-12 页)
- “类方法中的参数验证” (第 26-13 页)
- “参数验证的顺序” (第 26-14 页)
- “避免类和大小转换” (第 26-14 页)
- “参数验证中的 nargin” (第 26-16 页)
- “变量和函数访问的限制” (第 26-17 页)

## 参数验证简介

函数参数验证是一种对函数输入参数声明特定限制的方法。使用参数验证，您可以约束函数输入值的类、大小和其他方面，而无需在函数体中编写代码来执行这些测试。

函数参数验证是声明性的，这使得 MATLAB 桌面工具能够通过检查特定代码块来提取关于函数的信息。通过声明输入参数的要求，您可以消除繁琐的参数检查代码，并提高代码的可读性、稳健性和可维护性。

函数参数验证语法简化了定义可选参数、重复参数和名称-值参数的过程。此外，它使您能够以一致的方式定义默认值。

## 何时使用参数验证

在函数定义中，您可以选择是否使用函数参数验证。如果函数可由任何代码调用，并且在执行函数代码之前必须确定输入的有效性，则参数验证能起到很大帮助。对于那些为他人使用而设计的函数，如能对参数输入施加适当限制，并允许基于函数输入返回特定错误消息，也可使用户受益。

## 何时不需要验证

在局部和私有函数中，以及在私有或受保护方法中，调用方知道输入要求，因此可以使用有效参数调用这些类型的函数。

## 何时不允许验证

在嵌套函数、抽象方法或句柄类析构函数方法中，不能使用参数验证语法。有关方法中参数验证的详细信息，请参阅“类方法中的参数验证” (第 26-13 页)。

## arguments 代码块语法

函数在以关键字 `arguments` 和 `end` 起止的代码块中定义参数验证。如果使用，`arguments` 代码块必须在函数的第一个可执行代码行之前开始。

您可以在一个函数中使用多个 `arguments` 代码块，但是，所有这些代码块必须出现在不属于 `arguments` 代码块的代码之前。

下面代码中突出显示的部分展示了参数验证的语法。

```
function myFunction(inputArg)
    arguments
        inputArg (dim1,dim2,...) ClassName {fcn1,fcn2,...} = defaultValue
    end
    % Function code
end
```

函数参数声明可以包括以下任何一种限制：

- 大小 - 每个维度的长度，以圆括号括起
- 类 - 单个 MATLAB 类的名称
- 函数 - 以逗号分隔的验证函数列表，以花括号括起

您可以在该参数的函数验证声明中为输入参数定义默认值。确保默认值满足对该参数声明的限制。

### 大小

验证大小是输入参数的维数，用非负整数或冒号 (`:`) 指定。冒号表示该维度中允许任何长度。您不能对维度使用表达式。在函数调用中分配给参数的值必须与指定的大小兼容，否则 MATLAB 会引发错误。

MATLAB 索引赋值规则适用于大小设定。例如， $1 \times 1$  值与指定为  $(5,3)$  的大小兼容，因为 MATLAB 应用标量扩展。此外，MATLAB 行-列转换也适用，因此指定为  $(1,:)$  的大小可以接受  $1 \times n$  和  $n \times 1$  的大小。

下面给出了一些示例：

- $(1,1)$  必须精确为  $1 \times 1$
- 对于  $(3,:)$ ，第一个维度必须为 3，第二个维度可以是任何值
- 如果不指定大小，则允许任何大小，除非受到验证函数的限制。

### 类

验证类是单个类的名称。赋给函数输入的值必须属于指定的类或可转换为指定的类。允许使用任何 MATLAB 类或 MATLAB 支持的任何外部定义的类，但不包括 Java 类、COM 类和在 MATLAB 软件版本 7.6 之前定义的 MATLAB 类（其类定义不使用 `classdef` 关键字）。

下面给出了一些示例：

- **char** - 输入必须属于 **char** 类，或是 MATLAB 可以转换为 **char** 的值，例如 **string**。
- 任何精度的 **double** 数值
- **cell**, 元胞数组
- 用户定义的类（如枚举类）可以将输入限制为更具体的值，并使您能够控制支持何种转换。
- 如果不指定类，则允许任何类，除非受到验证函数的限制。

## 验证函数

验证函数是一个 MATLAB 函数，如果参数值不满足某些要求，该函数会引发错误。验证函数不返回值，因此它与类验证和大小验证不同，它不会导致所验证的参数值发生任何更改。

在验证过程中，MATLAB 将参数值传递给为该参数列出的每个验证函数。MATLAB 从左到右调用每个函数，并在遇到第一个错误时发出警告。

传递给验证函数的值是在设定类和大小时进行任意转换之后的结果。

有关预定义的验证函数的表，请参阅“参数验证函数”（第 26-19 页）。

## 默认值

参数默认值可以是满足大小、类和验证函数要求的任何常量或表达式。在参数声明中指定默认值会使参数变为可选。当参数不包含在函数调用中时，MATLAB 将使用默认值。默认值表达式在每次使用默认值时进行计算。

**注意** 由于 MATLAB 仅在不带参数值调用函数时才验证默认值，因此无效的默认值仅在不带相应参数值调用函数时才会导致错误。

可选参数必须位于函数签名中和 **arguments** 代码块中的必需参数后。有关可选参数的详细信息，请参阅“必需和可选位置参数”（第 26-6 页）。

## 验证顺序

参数在 **arguments** 代码块中自上而下进行验证。MATLAB 按照特定顺序验证参数声明的每个部分。首先验证类，然后验证大小。类和大小验证的结果传递给验证函数。每个步骤均为可选的，具体取决于类、大小和验证函数是否在参数声明中。

有关详细信息，请参阅“参数验证的顺序”（第 26-14 页）。

## 转换为声明的类和大小

由于标准 MATLAB 转换规则，类验证和大小验证都可以更改输入参数的值。因此，函数体中经过验证的值可能不同于调用函数时传递的值。转换规则派生自 MATLAB 对索引赋值应用的规则，形式如下：

**A(indices) = value**

MATLAB 可以确定左侧值对类和大小有要求，并且在某些情况下，可以将右侧值转换为所需的类和大小。

要了解相关信息，请参阅“避免类和大小转换”（第 26-14 页）。

## 参数验证示例

此 **arguments** 代码块指定三个输入的大小和类。

```

function out = myFunction(A, B, C)
    arguments
        A (1,1) string
        B (1,:) double
        C (2,2) cell
    end

    % Function code
    ...
end

```

在此函数中，变量有以下声明：

- A 是字符串标量。
- B 是 1×任意长度的双精度值向量。
- C 是 2×2 元胞数组。

### 值转换

以下函数说明如何转换输入以匹配在 **arguments** 代码块中指定的类。SpeedEnum 类是为定义第三个输入参数允许的值而创建的枚举类。

```

function forwardSpeed(a,b,c)
    arguments
        a double
        b char
        c SpeedEnum
    end

    % Function code
    disp(class(a))
    disp(class(b))
    disp(class(c))
end

```

以下代码定义该枚举类。

```

classdef SpeedEnum < int32
    enumeration
        Full (100)
        Half (50)
        Stop (0)
    end
end

```

该函数调用使用的输入值可由 MATLAB 转换为声明的类型。函数中实际的参数类型则显示为输出。

```

forwardSpeed(int8(4),"A string",'full')

double
char
SpeedEnum

```

### 使用验证函数设置具体限制

验证函数可以用更具体的方式限制输入参数。您可以将预定义的验证函数用于许多常见类型的验证，并且可以定义自己的验证函数来满足特定要求。

例如，此函数使用 `mustBeNumeric`、`mustBeReal`、`mustBeMember` 及局部函数 `mustBeEqualSize` 指定以下验证。

- 输入 `x` 必须为任意长度的实数数值行向量。
- 输入 `v` 必须为与 `x` 大小相同的实数数值行向量。
- 输入 `method` 必须为字符串向量且是三个允许的选项之一。由于 `method` 指定默认值，此参数是可选的。

```
function myInterp(x,v,method)
    arguments
        x (1,:) {mustBeNumeric,mustBeReal}
        v (1,:) {mustBeNumeric,mustBeReal,mustBeEqualSize(v,x)}
        method (1,:) char {mustBeMember(method,['linear','cubic','spline'])} = 'linear'
    end
    % Function code
    ...
end

% Custom validation function
function mustBeEqualSize(a,b)
    % Test if a and b have equal size
    if ~isequal(size(a),size(b))
        error('Size of first input must equal size of second input')
    end
end
```

避免在自定义验证函数中使用函数参数验证。如需定义验证函数的详细信息，或要查看预定义验证函数列表，请参阅“参数验证函数”（第 26-19 页）。

## 参数的种类

函数参数验证可以声明四种参数。函数可以定义四种参数中的任何一种，但参数必须按照以下顺序定义：

- 1 必需位置参数
- 2 可选位置参数
- 3 重复位置参数
- 4 可选名称-值参数

## 必需和可选位置参数

位置参数必须以特定顺序传递给函数。参数列表中传递的值的位置必须对应于参数在 `arguments` 代码块中声明的顺序。`arguments` 代码块中的所有参数名称必须唯一。

调用函数时，`arguments` 代码块中的位置参数为必需，除非此参数定义了默认值。在参数声明中指定默认值会使位置参数成为可选参数，因为 MATLAB 可以在函数调用中没有传递值时使用默认值。

默认值可以是常量，也可以是结果满足参数声明的表达式。表达式可以引用在 `arguments` 代码块中在该表达式之前声明的参数，但无法引用在它后面声明的参数。

MATLAB 仅在函数调用不包含某参数时才计算其默认值表达式。

所有可选参数都必须位于 `arguments` 代码块中的所有必需参数后。例如，在此参数代码块中，`maxlen` 和 `minlen` 具有默认值，因此是可选的。

```
function myFunction(x,y,maxval,minval)
    arguments
        x (1,:) double
        y (1,:) double
```

```

maxval (1,1) double = max(max(x),max(y))
minval (1,1) double = min(min(x),min(y))
end

% Function code
...
end

```

您可以带以下参数调用此函数：

```

myFunction(x,y,maxval,minval)
myFunction(x,y,maxval)
myFunction(x,y)

```

如果必须在函数调用中填充某可选位置参数的位置以标识其后的参数，则该可选位置参数变为必需参数。在上述实例中，这相当于：如果要指定 **minlen** 的值，必须指定  **maxlen** 的值。

### 被忽略位置参数

MATLAB 允许您通过传递波浪号字符 (~) 代替参数来忽略输入参数。您可以通过在参数代码块中添加波浪号字符 (~) 来定义一个忽略未使用位置参数的函数，添加字符的位置对应于该参数在函数签名中的位置。为函数签名中的每个被忽略参数添加一个波浪号字符 (~)。

被忽略参数不能有默认值，也不能指定类、大小或验证函数。

波浪号字符 (~) 被视为可选输入参数，除非它后跟必需位置参数。例如，在此函数中，波浪号字符 (~) 代表一个可选参数。

```

function c = f(~)
  arguments
    ~
  end

% Function code
end

```

您可以不带参数调用此函数。

```
c = f
```

也可以带一个输入参数。

```
c = f(2)
```

在以下函数中，波浪号字符 (~) 代表一个必需参数。

```

function c = f(~,x)
  arguments
    ~
    x
  end

% Function code
...
end

```

对此函数的调用必须包含两个参数。

```
c = f(2,3)
```

有关调用具有被忽略输入的函数的详细信息，请参阅“忽略函数输入”（第 21-10 页）。

## 重复参数

重复参数是可以重复指定为输入参数的位置参数。在包含 **Repeating** 属性的 **arguments** 代码块中声明重复参数。

```
arguments (Repeating)
    arg1
    arg2
    ...
end
```

函数只能有一个 **Repeating arguments** 代码块，其中可以包含一个或多个重复参数。

如果函数定义了 **Repeating arguments** 代码块，则调用该函数时，此代码块中的参数可以一次也不出现，也可以全体出现一次或多次。如果对函数的调用包含重复参数，则每次重复都必须包含此代码块中的所有参数。

对于重复参数，您不能通过指定默认值使之成为可选。但是，您可以在不带重复参数调用函数。

重复参数在函数中的声明顺序必须在位置参数之后、名称-值参数之前。您不能在 **Repeating** 代码中指定名称-值参数。有关名称-值参数的信息，请参阅“名称-值参数”（第 26-10 页）。

在函数中，每个重复参数构成一个元胞数组，其中包含的元素数与函数调用中传递的重复次数相同。验证应用于该元胞数组的每个元素。如果调用函数时该参数的出现次数为零，则元胞数组的大小为  $1 \times 0$ 。也就是说，它为空。

例如，以下函数声明一个重复参数代码块，其中包含三个参数 **x**、**y** 和 **options**。

```
function [xCell,yCell,optionCell] = fRepeat(x,y,options)
    arguments (Repeating)
        x double
        y double
        option {mustBeMember(option,[ "linear", "cubic"])}
    end

    % Function code
    % Return cell arrays
    xCell = x;
    yCell = y;
    optionCell = option;
end
```

调用此函数时，您可以不带输入，也可以将三个输入重复多次。MATLAB 为每个参数创建一个元胞数组，其中包含为该参数传递的所有值。以下对 **fRepeat** 的调用将这三个重复参数全体传递了两次。

```
[xCell,yCell,optionCell] = fRepeat(1,2,"linear",3,4,"cubic")
```

```
xCell =
```

```
1×2 cell array
```

```
{[1]} {[3]}
```

```
yCell =
```

```

1×2 cell array
{[2]} {[4]}

optionCell =
1×2 cell array
{["linear"]} {"cubic"}

```

以下函数声明 **Repeating arguments** 代码块，从而允许重复输入参数 **x** 和 **y**。在函数体中，指定为重复参数的值包含在元胞数组 **x** 和 **y** 中。此示例交错 **x** 和 **y** 中的值，以匹配 **plot** 函数的必需输入：**plot(x1,y1,...)**。

```

function myPlotRepeating(x,y)
    arguments(Repeating)
        x (1,:) double
        y (1,:) double
    end

    % Function code
    % Interleave x and y
    z = reshape([x;y],1,[]);

    % Call plot function
    if ~isempty(z)
        plot(z{:});
    end
end

```

使用重复的输入参数对来调用此函数。

```

x1 = 1:10;
y1 = sin(x1);
x2 = 0:5;
y2 = sin(x2);
myPlotRepeating(x1,y1,x2,y2)

```

### 避免对重复参数使用 varargin

不建议将 **varargin** 与使用参数验证的函数结合使用。如果您使用 **varargin** 支持原有代码，它必须为 **Repeating arguments** 代码块中的唯一参数。

如果 **varargin** 在重复参数代码块中具有大小或类方面的限制，则这些限制将应用于 **varargin** 中的所有值。

例如，此函数定义两个必需位置参数，并将 **varargin** 定义为重复参数。

```

function f(a, b, varargin)
    arguments
        a uint32
        b uint32
    end
    arguments (Repeating)
        varargin
    end

```

```
% Function code
...
end
```

## 名称-值参数

名称-值参数将名称与传递给函数的值相关联。名称-值参数：

- 可以按任何顺序传递给函数
- 始终为可选
- 必须在所有位置参数和重复参数之后声明
- 不能出现在使用 **Repeating** 属性的 **arguments** 代码块中
- 必须使用唯一名称，即便使用多个名称-值结构体时也是如此
- 不能与位置参数同名

使用圆点表示法在 **arguments** 代码块中声明名称-值参数以定义结构体的字段。例如，名为 **NameValueArgs** 的结构体定义两个名称-值参数 **Name1** 和 **Name2**。您可以使用任何有效的 MATLAB 标识符作为结构体名称。

```
arguments
    NameValueArgs.Name1
    NameValueArgs.Name2
end
```

结构体名称必须出现在函数签名中。

```
function myFunction(NameValueArgs)
```

使用名称-值结构体中的字段名称调用函数，这些名称作为字符串或字符向量传递。

```
myFunction('Name1',value1,'Name2',value2)
```

函数签名中使用的结构体名称是函数工作区中包含传递给该函数的名称和值的结构体的名称。

```
function result = myFunction(NameValueArgs)
arguments
    NameValueArgs.Name1
    NameValueArgs.Name2
end

% Function code
result = NameValueArgs.Name1 * NameValueArgs.Name2;
end
```

```
r = myFunction('Name1',3,'Name2',7)
```

```
r =
```

```
21
```

## 名称-值参数的默认值

您可以为每个名称指定一个默认值。如果没有指定默认值，并且调用函数时不带该名称-值参数，则该字段不存在于名称-值结构体中。如果没有向函数传递名称-值参数，MATLAB 仍会创建结构体，但它没有字段。

要确定在函数调用中传递了哪些名称-值参数，请使用 `isfield` 函数。

例如，以下函数定义两个必需位置参数（`width` 和 `height`）和两个名称-值参数（`LineStyle` 和 `LineWidth`）。在此示例中，`options` 结构体有两个字段（`LineStyle` 和 `LineWidth`），其中包含默认值或在调用函数时指定为名称-值参数的值。

```
function myRectangle(width,height,options)
    arguments
        width double
        height double
        options.LineStyle (1,1) string = "-"
        options.LineWidth (1,1) {mustBeNumeric} = 1
    end

    % Function code
    ...
end
```

以下均为调用此函数的有效方法。

```
myRectangle(4,5)
myRectangle(4,5,"LineStyle","-", "LineWidth",2)
myRectangle(4,5,"LineWidth",2,"LineStyle","-")
myRectangle(4,5,"LineStyle",-)
myRectangle(4,5,"LineWidth",2)
```

#### 同时使用重复参数和名称-值参数

如果函数定义了重复参数，则您必须在重复参数代码块之后另加一个 `arguments` 代码块来声明名称-值参数。例如，此函数接受两个重复参数 `x` 和 `y`。在指定 `x` 和 `y` 的所有重复项后，您可以指定一个名称-值对组，它将值 `lin` 或 `log` 赋给 `PlotType` 名称。

要确定函数调用是否包含 `PlotType` 参数，请使用 `isfield` 函数检查 `scale` 结构体中是否存在 `PlotType` 字段。

```
function linLog(x,y,scale)
    arguments(Repeating)
        x (1,:) double
        y (1,:) double
    end
    arguments
        scale.PlotType (1,1) string
    end
    z = reshape([x;y],1,[]);
    if isfield(scale,"PlotType")
        if scale.PlotType == "lin"
            plot(z{:})
        elseif scale.PlotType == "log"
            loglog(z{:})
        end
    end
end
```

以带或不带名称-值参数的方式调用此函数。

```
myLinLog(1:5,1:5)
myLinLog(1:5,1:5,1:10,1:100:1000)
myLinLog(1:5,1:5,1:10,1:100:1000,"PlotType","log")
```

## 多个名称-值结构体

函数参数代码块可以包含多个名称-值结构体。但是，字段名称在所有结构体中必须唯一。此函数有两个名称-值结构体：`lineOptions` 和 `fillOptions`。这些结构体不能有相同的字段名称。

`myRectangle` 函数中的参数声明如下。

- `width` 和 `height` 是双精度类型的必需位置参数。
- `lineOptions.LineStyle` 是标量字符串，默认值为 `"-"`。
- `lineOptions.LineWidth` 是标量数值，默认值为 1。
- `fillOptions.Color` 是字符串。
- `fillOptions.Pattern` 对其值没有任何限制。

```
function myRectangle(width,height,lineOptions,fillOptions)
    arguments
        width double
        height double
        lineOptions.LineStyle (1,1) string = "-"
        lineOptions.LineWidth (1,1) {mustBeNumeric} = 1
        fillOptions.Color string
        fillOptions.Pattern
    end

    % Function Code
    ...
end
```

## 基于类属性的名称-值参数

MATLAB 提供了一个方便的函数语法，允许您使用类的公共属性作为名称-值参数的名称。要将类定义的所有可设置属性（即具有公共 `SetAccess` 的所有属性）指定为名称-值参数，请在 `arguments` 代码块中使用以下语法。

`structName.?ClassName`

一个函数只能使用一次 "`structName.? ClassName`" 语法。因此，即便使用不同的类和结构体名称，一个函数也只能定义一个从类中获取字段名称的名称-值结构体。

如果类通过属性验证来限制可以赋给属性的值，则函数会将验证应用于单独的名称-值参数。有关属性验证的信息，请参阅“验证属性值”。

例如，此函数有两个必需参数 `x` 和 `y`，并且接受 `matlab.graphics.chart.primitive.Bar` 类的任何公共属性的名称和值。

```
function myBar(x,y,propArgs)
    arguments
        x (:,:) double
        y (:,:) double
        propArgs.?matlab.graphics.chart.primitive.Bar
    end
    propertyCell = namedargs2cell(propArgs);
    bar(x,y,propertyCell{:})
end
```

使用必需输入以及任何可设置属性的名称-值对组来调用此函数。

```
x = [1,2,3;4,5,6];
y = x.^2;
```

```
myBar(x,y)
myBar(x,y,'FaceColor','magenta','BarLayout','grouped')
```

### 覆盖特定属性

您可以通过在参数代码块中用特定名称-值参数重新定义属性名称来覆盖类属性验证。

```
structName.?ClassName
structName.PropertyName (dim1,dim2,...) ClassName {fcn1,fcn2,...}
```

特定名称-值参数验证将覆盖类为单独指定的属性名称定义的验证。

例如，以下函数将名称-值参数定义为 `matlab.graphics.chart.primitive.Bar` 类的属性。此外，它覆盖属性名称 `FaceColor` 以仅允许这些特定值：'red' 或 'blue'。

`matlab.graphics.chart.primitive.Bar` 类的 `FaceColor` 的默认值不是允许值 ('red' 或 'blue')。因此，覆盖声明必须赋予一个默认值，该默认值满足 `mustBeMember` 验证函数施加的限制。也就是说，默认值必须为 'red' 或 'blue'。

此函数使用 `namedargs2cell` 函数将名称-值结构体转换为名称与值相互交错的元胞数组。

```
function myBar(x,y,propArgs)
    arguments
        x (:,:) double
        y (:,:) double
    propArgs.?matlab.graphics.chart.primitive.Bar
    propArgs.FaceColor {mustBeMember(propArgs.FaceColor,['red','blue'])} = "blue"
end
propertyCell = namedargs2cell(propArgs);
bar(x,y,propertyCell{:})
end
```

使用两个必需参数 `x` 和 `y` 调用函数。您还可以视需要传递 `bar` 函数支持的任何名称-值对组以及 `FaceColor` 的值，该值可以是 `red` 或 `blue`。`FaceColor` 不允许使用其他值。

```
x = [1,2,3;4,5,6];
y = x.^2;
myBar(x,y)
myBar(x,y,'FaceColor','red','BarLayout','grouped')
```

## 类方法中的参数验证

在公共方法中，对方法输入参数进行验证很有用，因为对方法的调用可能不是源于类代码。您可以在具体类方法中使用函数参数验证。但是，抽象方法无法定义 `arguments` 代码块。有关类方法的信息，请参阅“[方法](#)”。

如果 `classdef` 文件包含在单独文件中定义的方法的方法原型，请在定义该方法的单独文件中包含参数块。

子类方法不继承函数参数验证。覆盖超类方法的子类方法应在子类方法中使用相同的函数参数验证，这也是一种较为理想的做法。

句柄析构函数方法无法使用参数验证。在包含 `arguments` 代码块的句柄子类中，名为 `delete` 的方法不会被视为析构函数（在销毁对象时，MATLAB 不调用它）。有关析构函数方法的详细信息，请参阅“[句柄析构函数](#)”。

## 参数验证的顺序

在调用函数时，MATLAB 按照输入参数在 `arguments` 代码块中的声明顺序从上往下验证输入参数。上一个参数完全验证后，才会继续验证下一个参数。这就确保引用较早声明的参数时，使用的是经过验证的值。在第一次验证失败时，函数会引发错误。

经过验证的值可能不同于调用函数时作为输入传递的原始值。例如，此函数将输入声明为类 `uint32` 值。第三个输入声明指定了一个默认值，该值等于前两个输入的乘积。

```
function c = f(a, b, c)
    arguments
        a uint32
        b uint32
        c uint32 = a .* b
    end

    % Function code
    ...
end
```

如果使用其他数值类（例如 `double`）的输入来调用函数，则输入会被转换为 `uint32`。

```
c = f(1.8, 1.5)
```

由于在函数调用中没有指定可选输入参数 `c`，因此在将 `a` 和 `b` 转换为 `uint32` 值后，MATLAB 会计算默认值并将其赋给 `c`。在本例中，两个输入的转换结果均为值 2。因此，`a` 和 `b` 的乘积是 4。

```
c =
uint32
4
```

如果您为第三个输入指定值，则该函数会为 `c` 赋值，并且不会计算默认值表达式。

```
c = f(1.8, 1.5, 25)
```

```
c =
uint32
25
```

## 避免类和大小转换

在验证过程中，MATLAB 在调用任何验证函数之前，会先应用类验证，然后应用大小验证。如果默认值用于函数调用中省略的可选输入，则在应用验证过程之前，会先将该值赋给参数。

当传递给函数的参数值与验证所要求的类和大小不匹配时，MATLAB 会尽可能将该值转换为声明的类和大小。然而，有些情况下转换并不是理想的行为。例如，`char` 或 `string` 会被转换为双精度，标量扩展会为了满足大小限制而更改大小，列向量会被转换为行向量。

要从输入参数验证中去除 MATLAB 执行的标准转换，请使用验证函数，而不是类和大小限制。对验证函数的调用不会返回值，也无法更改输入参数的值。

例如，此函数将第一个输入限制为双精度类的任意大小的二维数组。第二个输入必须为任何类的  $5 \times 3$  数组。

```
function f(a, b)
    arguments
        a (:,:) double
        b (5,3)
    end

    % Function code
    ...
end
```

由于存在标准 MATLAB 类型转换和标量扩展，您可以使用以下输入调用此函数，而不会收到验证错误。默认情况下，MATLAB 将字符串向量的元素转换为其等效数值，并应用标量扩展以基于标量值 144 创建一个  $5 \times 3$  数组。

```
f('character vector',144)
```

使用专用验证函数可以提供更具体的输入参数验证。例如，此函数定义两个专用验证函数，用于代替第一个和第二个参数的类和大小设定。这些局部函数帮助您避免输入值转换。

```
function fCustomValidators(a, b)
    arguments
        a {mustBeA(a,'double'), mustBeDims(a,2)}
        b {mustBeSize(b,[5,3])}
    end

    % Function code
    ...
end

% Custom validator functions
function mustBeA(input,className)
    % Test for specific class
    if ~isa(input,className)
        error('Input must be of class double.')
    end
end

function mustBeSize(input,sizeDims)
    % Test for specific size
    if ~isequal(size(input),sizeDims)
        error(['Input must be of size ',num2str(sizeDims)])
    end
end

function mustBeDims(input,numDims)
    % Test for number of dimensions
    if ~isequal(length(size(input)),numDims)
        error(['Input must have ',num2str(numDims),' dimensions.'])
    end
end
```

`mustBeSize` 和 `mustBeDims` 验证函数对输入参数执行严格的声明。

```
fCustomValidators('character vector',144)
```

```
Invalid input argument at position 1. Input must be of class double
```

在此调用中，第一个输入的维数是错误的，验证函数返回自定义的错误消息。

```
fCustomValidators(ones(2,2,4),144)
```

```
Invalid input argument at position 1. Input must have 2 dimensions
```

**mustBeSize** 验证函数将第二个输入限制为特定维度，如错误消息所示。

```
fCustomValidators(ones(2,2),144)
```

```
Invalid input argument at position 2. Input must be of size [5 3]
```

## 参数验证中的 nargin

**nargin** 函数针对当前正在执行的函数，返回函数调用中给定函数输入参数的数目。使用函数参数验证时，函数内 **nargin** 返回的值是调用函数时提供的位置参数的数量。此值不包括未包含在函数调用中的可选参数。此外，**nargin** 不对任何名称-值参数进行计数。

重复参数是位置参数，因此调用时传递给函数的重复参数的个数将计入 **nargin** 返回的值。

使用 **nargin** 确定在调用时是否将可选位置参数传递给函数。例如，此函数声明三个位置参数和一个名称-值参数。函数通过以下方式确定在调用时传递哪些参数。

- **nargin** 通过 **switch** 代码块确定是否将可选位置参数 **c** 传递给函数。
- **isfield** 确定是否将 **Format** 的名称-值参数传递给函数。

```
function result = fNargin(a, b, c, namedargs)
    arguments
        a (1,1) double
        b (1,1) double
        c (1,1) double = 1
        namedargs.Format (1,:) char
    end

    % Function code
    switch nargin
        case 2
            result = a + b;
        case 3
            result = a^c + b^c;
    end
    if isfield(namedargs,'Format')
        format(namedargs.Format);
    end
end
```

在此函数调用中：

```
result = fNargin(3,4)
```

```
result =
```

7

函数中 **nargin** 返回的值是 2。

在此函数调用中：

```
result = fNargin(3,4,7.62)
```

```
result =
```

```
4.3021e+04
```

函数中 `nargin` 返回的值是 3。

在此函数调用中：

```
result = fNargin(3,4,7.62,'Format','bank')
```

```
result =
```

```
43020.56
```

函数中 `nargin` 返回的值是 3。

## 变量和函数访问的限制

参数代码块存在于函数的工作区中。如果函数使用 `import` 命令将包、类或函数添加到该函数的作用域内，则 `arguments` 代码块中会应用同样的作用域。

对验证函数和默认值表达式可见的变量只有已声明的输入变量。在以下函数中，`c` 的默认值派生自 `a` 和 `b`。

```
function c = f(a,b,c)
    arguments
        a uint32
        b uint32
        c uint32 = a * b
    end

    % Function code
    ...
end
```

但是，您无法引用尚未在 `arguments` 代码块中声明的输入变量。例如，在上述函数中对参数 `a` 使用以下声明是无效的，因为 `b` 和 `c` 尚未声明。

```
arguments
    a uint32 = b * c
    b uint32
    c uint32
end
```

参数验证表达式只能引用此前声明过（因此经过验证）的参数。

名称-值参数的验证函数和默认值无法访问其他名称-值参数。

### `arguments` 代码块中函数的限制

对此前所声明参数的任何引用必须在验证函数和默认值的文本中可见。为了确保代码透明，请不要使用与函数工作区交互的函数。尤其不要在 `arguments` 代码块中使用嵌套函数或下表中列出的任何函数。

<code>assignin</code>	<code>builtin</code>	<code>clear</code>
-----------------------	----------------------	--------------------

<b>dbstack</b>	<b>eval</b>	<b>evalc</b>
<b>evalin</b>	<b>exist</b>	<b>feval</b>
<b>input</b>	<b>inputname</b>	<b>load</b>
<b>nargin</b>	<b>narginchk</b>	<b>nargoutchk</b>
<b>save</b>	<b>whos</b>	<b>who</b>

这些限制仅适用于 **arguments** 代码块，对函数体中的变量或函数则不适用。

## 另请参阅

[arguments](#) | [namedargs2cell](#)

## 相关示例

- “输入和输出参数”
- “参数验证函数”（第 26-19 页）
- “验证属性值”

## 参数验证函数

MATLAB 定义了一些用于参数验证的函数。这些函数支持验证的常用模式，并提供描述性错误消息。下表列出 MATLAB 验证函数及其含义，以及它们使用的 MATLAB 函数。

名称	含义	对输入调用的函数
<code>mustBePositive(value)</code>	<code>value &gt; 0</code>	<code>gt, isreal, isnumeric, islogical</code>
<code>mustBeNonpositive(value)</code>	<code>value &lt;= 0</code>	<code>ge, isreal, isnumeric, islogical</code>
<code>mustBeFinite(value)</code>	<code>value</code> 中不含 <code>Nan</code> 和 <code>Inf</code> 元素。	<code>isfinite</code>
<code>mustBeNonNan(value)</code>	<code>value</code> 中不含 <code>Nan</code> 元素。	<code>isnan</code>
<code>mustBeNonnegative(value)</code>	<code>value &gt;= 0</code>	<code>ge, isreal, isnumeric, islogical</code>
<code>mustBeNegative(value)</code>	<code>value &lt; 0</code>	<code>lt, isreal, isnumeric, islogical</code>
<code>mustBeNonzero(value)</code>	<code>value ~= 0</code>	<code>eq, isnumeric, islogical</code>
<code>mustBeGreaterThan(value,c)</code>	<code>value &gt; c</code>	<code>gt, isscalar, isreal, isnumeric, islogical</code>
<code>mustBeLessThan(value,c)</code>	<code>value &lt; c</code>	<code>lt, isreal, isnumeric, islogical</code>
<code>mustBeGreaterThanOrEqual(value,c)</code>	<code>value &gt;= c</code>	<code>ge, isreal, isnumeric, islogical</code>
<code>mustBeLessThanOrEqual(value,c)</code>	<code>value &lt;= c</code>	<code>le, isreal, isnumeric, islogical</code>
<code>mustBeNonempty(value)</code>	<code>value</code> 不为空。	<code>isempty</code>
<code>mustBeNonsparse(value)</code>	<code>value</code> 中不含稀疏元素。	<code>issparse</code>
<code>mustBeNumeric(value)</code>	<code>value</code> 包含数值。	<code>isnumeric</code>
<code>mustBeNumericOrLogical(value)</code>	<code>value</code> 包含数值或逻辑值。	<code>isnumeric, islogical</code>
<code>mustBeReal(value)</code>	<code>value</code> 没有虚部。	<code>isreal</code>

名称	含义	对输入调用的函数
<code>mustBeInteger(value)</code>	<code>value == floor(value)</code>	<code>isreal, isnan,</code> <code>floor, isnumeric,</code> <code>islogical</code>
<code>mustBeMember(value,S)</code>	<code>value</code> 的所有成员都可在 <code>S</code> 中找到。	<code>ismember</code>

## 定义验证函数

验证函数是为验证输入值而设计的普通 MATLAB 函数。用于验证的函数必须执行以下操作：

- 接受可能的值作为输入参数。
- 不返回值
- 如果验证失败，则引发错误

当使用 MATLAB 验证函数无法提供所需的特定验证时，您可以创建自己的验证函数，这非常有用。您可以将其作为局部函数包含在函数文件中，或将其放在 MATLAB 路径上。为了避免错误消息混杂，请不要在用户定义的验证函数中使用函数参数验证。

例如，下面的 `ImgData` 函数使用局部函数定义了一个验证函数，该验证函数将 `Data` 输入限制为特定范围内的数值。`mustBeNumeric` 验证函数是一个现有函数。

```
function ImgData(Data)
    % Function code
    imagesc(Data)
end

% Custom validation function
function mustBeInRange(arg,b)
    if any(arg(:) < b(1)) || any(arg(:) > b(2))
        error(['Value assigned to Data is not in range ',...
            num2str(b(1)), '...', num2str(b(2))])
    end
end
```

`Data` 输入参数可以是任意大小的数值数组，其中的值处于 0–255 的范围内。如果值不满足这些要求，MATLAB 会引发错误。

```
ImgData(randi(300,10,10))
```

```
Error using ImgData
Invalid input argument at position 1. Value assigned to Data is not in range 0..255
```

## 另请参阅

### 详细信息

- “函数参数验证”（第 26-2 页）

# 解析函数输入的方法

MATLAB 采用动态类型，这意味着变量没有声明的类型，可以保存不同类型的值。然而，值总是属于某一特定类型，并且程序总能查询变量当前值的类和大小。

函数输入参数是函数工作区中的变量，其值来自调用代码或命令行用户。当函数被他人广泛使用时，检查输入值是否与函数中代码期望的值相匹配非常重要。

借助参数检查，函数可在输入值不符合预期、函数无法按预期执行时提供更多有用信息。MATLAB 提供了几种解决方法来简化检查和处理函数输入的过程。

## 函数参数验证

MATLAB 中的许多函数对输入参数使用以下模式之一：

- 一个或多个必需输入参数
- 一个或多个必需输入参数，后跟一个或多个可选输入参数
- 以上述任一模式为基础，后跟若干名称-值对组

实现这些常见模式的有效方式是使用函数 `arguments` 代码块声明参数，如“函数参数验证”（第 26-2 页）中所述。此语法是从 R2019b 版本起新推出的，在以前的版本中无效。

函数参数验证是一种对函数输入参数声明特定限制的方法。您可以借助此语法直接约束函数输入值的类、大小和其他方面，而无需在函数体中编写代码来执行相应测试。

## `validateattributes`

您可以使用 `validateattributes` 函数验证函数的输入是否符合一系列要求。以形参指定参数要求，为每个输入参数调用 `validateattributes`。

## `inputParser`

对于复杂的函数签名，可使用 `inputParser` 对象以编程方式来表达对输入参数的要求。输入解析器解析并验证一组输入。

## 另请参阅

`arguments` | `inputParser` | `validateattributes`

## 详细信息

- “函数参数验证”（第 26-2 页）
- “通过 `validateattributes` 检查函数输入”（第 21-11 页）
- “解析函数输入”（第 21-13 页）

## MATLAB 代码中的透明

如果 MATLAB 可以在忽略注释、字符向量和字符串文字的情况下，通过扫描代码来识别每个变量访问，则代码的变量访问是透明的。变量访问包括读取、添加、删除或修改工作区变量。

在以下编码环境中，MATLAB 需要透明的变量访问。

- 函数参数验证代码块。有关详细信息，请参阅“变量和函数访问的限制”（第 26-17 页）
- **parfor** 循环或 **spmd** 代码块的主体。有关详细信息，请参阅“Ensure Transparency in parfor-Loops or spmd Statements”（Parallel Computing Toolbox）。

在这些上下文中，不透明的变量访问会导致运行时错误。

### 编写透明代码

透明代码显式引用变量名称。例如，在以下代码中，MATLAB 可以将 **X** 和 **ii** 识别为变量。

```
X = zeros(1,10);
for ii = 1:10
    X(ii) = randi(9,1);
end
```

但是，在以下对 **eval** 函数的调用中，MATLAB 无法识别传递给 **eval** 的语句中的变量，因为输入是字符串。

```
X = zeros(1,10);
for ii = 1:10
    eval('X(ii) = randi(9,1);')
end
```

在执行此代码之前，MATLAB 使用一个参数调用 **eval** 函数，该参数是字符串 '**X(ii) = randi(9,1);**'。

为了保持透明，代码必须显式引用变量名称，以便 MATLAB 通过检查来识别变量。将 **eval** 函数与字符串 '**X(ii) = randi(9,1);**' 结合使用意味着 MATLAB 必须先执行代码，才能识别 **X** 和 **ii**。

以下列出了一部分不能采用透明变量访问的函数和编码。

- **eval**、**evalc**、**evalin** 或 **assignin**。
- 脚本
- 动态访问工作区变量的 MEX 函数，例如 **mexGetVariable**。
- 内省函数，如 **who** 和 **whos**。
- **save** 和 **load** 命令，除非显式对 **load** 的结果赋值。
- 任何动态名称引用，例如使用 **clear** 命令。

使用命令形式将变量传递给函数是不透明的，因为这等效于将参数作为字符串传递。例如，对 **clear** 函数的以下调用均为不透明的。

```
clear X
clear('X')
```

如果代码创建工作区变量，但 MATLAB 只有在执行代码后才能识别这些新变量，则该代码的变量访问不透明。

例如，以下语句是不透明的，因为 MATLAB 无法确定从 MAT 文件中加载了哪些变量。

```
load foo.mat
```

然而，将加载的变量显式赋给名称的代码是透明的，因为 MATLAB 可以识别左侧的名称引用了一个工作区变量。例如，以下语句将变量 X 从 MAT 文件加载到工作区中名为 X 的变量中。

```
X = load('foo.mat','X');
```

对变量的访问在工作区中必须是透明的。例如，代码无法使用 `evalin` 或 `assignin` 函数从要求透明代码的工作区出发，在另一个工作区中创建变量。

## 另请参阅

### 详细信息

- “[函数参数验证](#)” (第 26-2 页)
- “[变量](#)”



# 软件开发



# 错误的处理方式

---

- “MATLAB 应用程序中的异常处理” (第 27-2 页)
- “捕获有关异常的信息” (第 27-4 页)
- “引发异常” (第 27-11 页)
- “对异常作出响应” (第 27-13 页)
- “在函数结束后清理” (第 27-16 页)
- “引发警告和错误” (第 27-20 页)
- “隐蔽警告” (第 27-23 页)
- “恢复警告” (第 27-25 页)
- “更改警告的显示方式” (第 27-27 页)
- “使用 try/catch 处理错误” (第 27-28 页)

# MATLAB 应用程序中的异常处理

## 本节内容

- “概述” (第 27-2 页)
- “在命令行处遇到异常” (第 27-2 页)
- “在您的程序代码中遇到异常” (第 27-3 页)
- “产生新的异常” (第 27-3 页)

## 概述

无论您如何仔细设计和测试您编写的程序，当在不同条件下执行这些程序时，它们并不能始终如期地顺利运行。最好在程序中包含错误检查机制以确保程序在所有条件下都能可靠地运行。

在 MATLAB 软件中，您可以决定您的程序如何针对不同类型的错误作出响应。您可能需要提示用户输入更多的内容，显示扩充的错误或警告信息，或者使用默认值重复某项计算。MATLAB 中的错误处理功能有助于您的程序检查特定错误条件，以及根据具体情况执行相应代码。

当 MATLAB 在运行的命令或程序中检测到严重缺陷时，它会收集有关在出现错误时所发生情况的信息，显示消息以帮助用户了解出现的故障，并终止所运行的命令或程序。这称为引发异常。当您在 MATLAB 命令提示符下输入命令或执行您的程序代码时，可能会遇到异常。

## 在命令行处遇到异常

如果您在 MATLAB 提示符下操作遇到异常，可以按如下所述使用几个选项来处理该异常。

### 根据错误消息确定故障

评估 MATLAB 显示的错误消息。多数错误消息都至少尝试解释程序故障的直接原因。通常有足够的信息来确定问题的根源以及您需要采取何种操作来修复相应的问题。

### 查看出错的代码

如果出现错误的函数是以 MATLAB 程序文件的形式实现的，则错误消息应包括一个类似如下的行：

```
surf
```

```
Error using surf (line 50)
Not enough input arguments.
```

该文本包括引发错误的函数的名称（本例中为 `surf`），并显示该函数的程序文件中出错的行号。点击行号；MATLAB 会打开该文件并将光标置于该文件中出现错误的位置。您可以通过检查这一行及其前面的代码来确定错误原因。

### 在调试器中逐行执行代码

您可以使用 MATLAB 调试器逐行执行出错的代码。点击带下划线的错误文本，即可在 MATLAB 编辑器中打开相应的文件并转到错误所在位置或附近。接下来，点击该行开头处的连字符以便在该位置设置断点。当您重新运行您的程序时，MATLAB 会在断点处暂停执行并以便允许逐行执行程序代码。命令 `dbstop on error` 也有助于查找出错位置。

有关详细信息，请参阅“调试 MATLAB 程序”（第 22-2 页）部分。

## 在您的程序代码中遇到异常

当您在程序文件中编写自己的程序时，您可以捕获异常并尝试处理或解决异常，而不是让您的程序终止。当您捕获异常时，您会中断正常终止过程并进入应对故障情形的代码节。该代码节称为 catch 块。

您可能需要在 catch 块中执行的一些操作有：

- 检查已捕获的、有关错误的信息。
- 收集更多信息以报告给用户。
- 尝试以其他某种方式完成手头任务。
- 清除错误的任何不需要的意外结果。

当您到达 catch 块的末尾时，您可以继续执行程序（如果可以）或终止程序。

“捕获有关异常的信息”（第 27-4 页）部分介绍了如何获取有关错误缘由的信息，而且“对异常作出响应”（第 27-13 页）部分还提供了一些有关如何对错误作出响应的建议。

## 产生新的异常

当您的程序代码检测到某种条件可能致使程序失败或者产生无法接受的结果时，会引发异常。在此过程中

- 将保存有关出现的问题以及在发生错误时正执行的代码的信息。
- 将收集有关错误的任何其他相关信息。
- 将指示 MATLAB 引发异常。

“捕获有关异常的信息”（第 27-4 页）部分介绍了如何使用 **MException** 对象捕获有关错误的信息，而且“引发异常”（第 27-11 页）部分还说明如何启动异常进程。

## 捕获有关异常的信息

### 本节内容

- “概述” (第 27-4 页)
- “MException 类” (第 27-4 页)
- “MException 类的属性” (第 27-5 页)
- “MException 类的方法” (第 27-10 页)

### 概述

当 MATLAB 引发异常时，它会将原因信息捕获到一个称为 **MException** 对象的数据结构体中。该对象是 MATLAB **MException** 类的实例。您可以访问 **MException** 对象，方法是通过 **catch** 命令在您的程序中止前捕获异常并访问针对此特定错误而构造的对象。当引发异常以便响应您自己的代码中的错误时，您将必须创建一个新的 **MException** 对象并将有关错误的信息存储在该对象中。

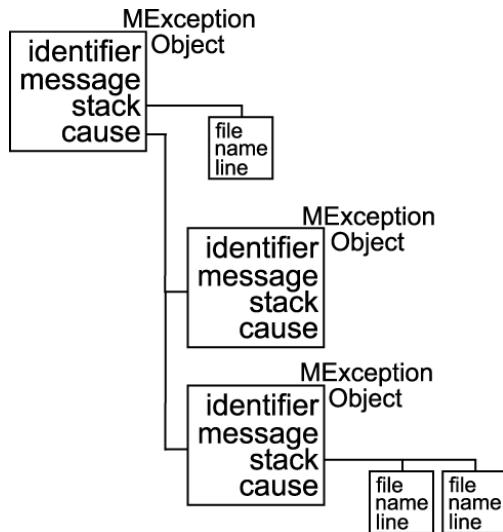
本部分介绍了 **MException** 类以及基于该类构造的对象。

后面有关 “对异常作出响应” (第 27-13 页) 和 “引发异常” (第 27-11 页) 的部分介绍了有关如何使用该类的信息。

### MException 类

下面显示的图窗展示了 **MException** 类的对象的一种可能配置。该对象有五个属性：**identifier**、**message**、**stack**、**cause** 和 **Correction**。其中每个属性都是以表示 **MException** 对象的结构体的一个字段的形式来实现的。**stack** 字段是一个由其他结构体组成的  $N \times 1$  数组，其中每个结构体标识一个函数，以及调用堆栈中的行编号。**cause** 字段是一个由 **MException** 对象组成的  $M \times 1$  数组，每个对象表示一个与当前对象相关的异常。

有关这些属性的完整描述，请参阅 “**MException** 类的属性” (第 27-5 页)。



## 对象构造函数

检测错误并引发异常的任何代码还必须构造一个 **MException** 对象，在该对象中记录并传输有关错误的信息。**MException** 构造函数的语法为

```
ME = MException(identifier, message)
```

其中 **identifier** 是括入单引号中的 MATLAB 消息标识符（第 27-6 页）：

```
component:mnemonic
```

**message** 是一个文本（也括在单引号中），用于描述错误。输出 **ME** 是生成的 **MException** 对象。

如果您要响应异常而不是引发异常，则无需构造 **MException** 对象。该对象已由最初检测到错误的代码构造并填充。

## MException 类的属性

**MException** 类有几个属性。其中每个属性都是以表示 **MException** 对象的结构体的一个字段的形式来实现的。下面各部分中介绍了其中每个属性，“对异常作出响应”（第 27-13 页）和“引发异常”（第 27-11 页）的相应部分中引用了这些属性。所有属性都是只读的，其值不能更改。

**MException** 属性为：

- **identifier**（第 27-6 页）
- **message**（第 27-7 页）
- **stack**（第 27-8 页）
- **cause**（第 27-9 页）
- **Correction**（第 27-10 页）

如果您调用不带任何输入的 **surf** 函数，则 MATLAB 会引发异常。如果您捕获异常，则可以看到 **MException** 对象结构体的属性。（本示例以非常规方式使用 **try/catch**。请参阅“**try/catch** 语句”（第 27-13 页）中的相应部分以了解有关使用 **try/catch** 的详细信息。）

```
try
    surf
catch ME
    ME
end
```

在命令行运行以下代码，MATLAB 将返回 **MException** 对象的内容：

```
ME =
```

**MException** with properties:

```
identifier: 'MATLAB:narginchk:notEnoughInputs'
    message: 'Not enough input arguments.'
    cause: {}
    stack: [1x1 struct]
    Correction: []
```

**stack** 字段显示引发异常的文件名、函数和行号：

```
ME.stack
ans =
```

```
file: 'matlabroot\toolbox\matlab\graph3d\surf.m'
name: 'surf'
line: 54
```

在本例中，**cause** 和 **Correction** 字段为空。在后续的各部分中更加详细地介绍了每个字段。

### 消息标识符

消息标识符是一个附加到错误或警告语句的标记，它使错误或警告可由 MATLAB 唯一识别。您可以将消息标识符和错误报告结合使用以更好地确定错误来源，或与警告结合使用以控制您的程序中任何选定的一小部分警告。

消息标识符是一个只读字符向量，用于指定错误或警告的组件和助记键标签。简单标识符的格式为  
**component:mnemonic**

用冒号分隔标识符的两部分：**component** 和 **mnemonic**。如果标识符使用多个 **component**，则需要额外的冒号来分隔它们。消息标识符必须始终至少包含一个冒号。

下面是消息标识符的一些示例

```
MATLAB:rmpath:DirNotFound
MATLAB:odearguments:InconsistentDataType
Simulink:actionNotTaken
TechCorp:OpenFile:notFoundInPath
```

**component** 和 **mnemonic** 字段必须遵从以下语法规则：

- 标识符中的任何位置都不允许有空白（空格字符或制表符）。
- 第一个字符必须为字母，大、小写均可。
- 其余字符可以是字母数字或下划线。

**component** 或 **mnemonic** 没有长度限制。标识符也可以是空字符串向量。

### Component 字段

**component** 字段指定一个广义类别，可以依据其生成各种错误和警告。常见组件是特定产品或工具箱名称，例如 MATLAB 或 Control，也可以是您公司的名称，例如上例中的 TechCorp。

您也可以使用此字段指定多级组件。下面的语句有一个后跟助记键标签的三级组件：

```
TechCorp:TestEquipDiv:Waveform:obsoleteSyntax
```

**Component** 字段使您能够保证每个标识符的唯一性。因此，尽管 MATLAB 的内部代码可使用诸如 MATLAB:InconsistentDataType 的特定警告标识符，但只要它前面带有唯一组件，它就不会阻止您使用同一助记键。例如，

```
warning('TechCorp:InconsistentDataType',...
'Value %s is inconsistent with existing properties.' ...
sprocketDiam)
```

### Mnemonic 字段

**mnemonic** 字段通常用作与特定消息相关的标记。例如，在报告因使用多义性语法而产生的错误时，如下所示的简单组件和助记键可能较为合适：

```
MATLAB:ambiguousSyntax
```

### MException 对象中的消息标识符

当引发异常时，在您使用以下语法构造 **MException** 对象时创建一个适当的标识符并将其保存到该对象中

```
ME = MException(identifier, text)
```

例如，

```
ME = MException('AcctError:NoClient', ...
    'Client name not recognized.');
```

```
ME.identifier
ans =
    AcctError:NoClient
```

响应异常时，您可以从 **MException** 对象提取消息标识符，如下所示。同样以 **surf** 为例，

```
try
    surf
catch ME
    id = ME.identifier
end

id =
    MATLAB:narginchk:notEnoughInputs
```

### 错误消息文本

MATLAB 中的错误消息是由程序代码发出并在 **MException** 对象中返回的只读字符串向量。此消息有助于用户确定故障原因，并进行修复（如有可能）。

当引发异常时，在您使用以下语法构造 **MException** 对象时编写一个适当的错误消息并将其保存到该对象中

```
ME = MException(identifier, text)
```

如果您的消息需要格式设置说明（如适用于 **sprintf** 函数的那些说明），则对 **MException** 构造函数使用以下语法：

```
ME = MException(identifier, formatstring, arg1, arg2, ...)
```

例如，

```
S = 'Accounts'; f1 = 'ClientName';
ME = MException('AcctError:Incomplete', ...
    'Field "%s.%s" is not defined.', S, f1);
```

```
ME.message
ans =
    Field 'Accounts.ClientName' is not defined.
```

响应异常时，您可以从 **MException** 对象提取错误消息，如下所示：

```
try
    surf
catch ME
    msg = ME.message
```

```
end

msg =
    Not enough input arguments.
```

### 调用堆栈

**MException** 对象的 **stack** 字段用于标识检测到错误的行号、函数以及文件名。如果所调用的函数出错（如下面的示例中），则 **stack** 字段不仅包含当前错误所在处的行号、函数名称和文件名，而且包含与每个调用函数对应的行号、函数名称和文件名。本例中，**stack** 是一个  $N \times 1$  数组，其中  $N$  表示调用堆栈的深度。即，在到达最顶层函数前，堆栈字段将显示出现异常的函数名称和行号、调用方的名称和行号、调用方的调用方等。

当引发异常时，MATLAB 将调用堆栈信息存储在 **stack** 字段中。您不能写入此字段；仅限只读访问。

例如，假定有三个位于两个不同文件中的函数：

```
mfileA.m
=====
.
.
.
42 function A1(x, y)
43 B1(x, y);
```

```
mfileB.m
=====
.
.
.
8 function B1(x, y)
9 B2(x, y)
.
.
.
26 function B2(x, y)
27 .
28 .
29 .
30 .
31 % Throw exception here
```

在变量 **ME** 中捕获异常，然后检查 **stack** 字段：

```
for k=1:length(ME.stack)
    ME.stack(k)
end

ans =
    file: 'C:\matlab\test\mfileB.m'
    name: 'B2'
    line: 31
ans =
    file: 'C:\matlab\test\mfileB.m'
    name: 'B1'
    line: 9
ans =
    file: 'C:\matlab\test\mfileA.m'
```

```
name: 'A1'
line: 43
```

### Cause 数组

在有些情况下，不仅要记录致使停止执行的一个命令的信息，而且要记录您的代码捕获的其他异常的信息，这一点很重要。您可以将这些其他 **MException** 对象保存在主异常的 **cause** 字段中。

**MException** 的 **cause** 字段是一个与 **MException** 对象相关的可选元胞数组。在向 **cause** 元胞数组中添加对象时，必须使用以下语法：

```
primaryException = addCause(primaryException, secondaryException)
```

此示例尝试将数组 **D** 赋给变量 **X**。如果 **D** 数组不存在，代码会尝试从 MAT 文件来加载该数组，然后重新尝试将其赋给 **X**。如果加载失败，将构造一个新的 **MException** 对象 (**ME3**) 来存储前两个错误 (**ME1** 和 **ME2**) 的原因：

```
try
    X = D(1:25)
catch ME1
    try
        filename = 'test200';
        load(filename);
        X = D(1:25)
    catch ME2
        ME3 = MException('MATLAB:LoadErr', ...
            'Unable to load from file %s', filename);
        ME3 = addCause(ME3, ME1);
        ME3 = addCause(ME3, ME2);
    end
end
```

**ME3** 的 **cause** 字段中有两个异常：

```
ME3.cause
ans =
[1x1 MException]
[1x1 MException]
```

检查 **ME3** 的 **cause** 字段以查看相关错误：

```
ME3.cause{;}
ans =

MException object with properties:

identifier: 'MATLAB:UndefinedFunction'
message: 'Undefined function or method 'D' for input
arguments of type 'double'!'
stack: [0x1 struct]
cause: {}
ans =
```

**MException** object with properties:

```
identifier: 'MATLAB:load:couldNotReadFile'
message: 'Unable to read file test204: No such file or
directory.'
```

```
stack: [0x1 struct]
cause: {}
```

### 更正

对于某些异常，您可以提供修复建议，在异常发生时显示。如果要在引发异常时提供修复建议，您可以创建一个 `matlab.lang.correction.AppendArgumentsCorrection`、`matlab.lang.correction.ConvertToFunctionNotationCorrection` 或 `matlab.lang.correction.ReplaceIdentifierCorrection` 对象并将其添加到异常的 `Correction` 字段。

向 `Correction` 字段添加修复时，必须使用以下语法：

```
primaryException = addCorrection(baseException, exceptionCorrection)
```

此示例创建需要一个输入参数的函数 `hello`。如果在没有输入的情况下调用该函数，MATLAB 会产生错误，并建议输入参数 "`world`" 作为修复。

```
function hello(audience)
if nargin < 1
    me = MException('MATLAB:notEnoughInputs', 'Not enough input arguments.');
    aac = matlab.lang.correction.AppendArgumentsCorrection("world");
    me = addCorrection(me, aac);
    throw(me)
end
fprintf("Hello, %s!\n", audience)
end
```

如果您在不带参数的情况下调用该函数，MATLAB 会建议修复。

`hello`

```
Error using hello (line 6)
Not enough input arguments.
```

```
Did you mean:
>> hello("world")
```

## MException 类的方法

有多种方法可以与 `MException` 类结合使用。这些方法的名称区分大小写。有关详细信息，请参阅 MATLAB 函数参考页。

方法名称	说明
<code>addCause</code>	将一个 <code>MException</code> 追加到另一个 <code>MException</code> 的 <code>cause</code> 字段。
<code>addCorrection</code>	为当前异常提供建议的修复。
<code>getReport</code>	根据当前异常返回一则格式化消息。
<code>MException.last</code>	返回最后未捕获的异常。这是静态方法。
<code>rethrow</code>	重新引发先前已捕获的异常。
<code>throw</code>	引发异常。
<code>throwAsCaller</code>	引发异常，但在 <code>stack</code> 字段中省略当前的堆栈帧。

## 引发异常

当您的程序检测到将阻止程序如期完成或将生成错误结果的故障时，您应该通过引发异常停止进一步执行并报告错误。采取的基本步骤是：

- 1 检测错误。这通常是使用某种类型的条件语句（如用于检查当前操作的输出的 `if` 或 `try/catch` 语句）完成的。
- 2 构造一个 `MException` 对象来表示该错误。在调用构造函数时向对象中添加一个消息标识符和一个错误消息。
- 3 如果有其他可能导致当前错误的异常，您可以将每个异常的 `MException` 对象存储在一个您打算引发的单个 `MException` 的 `cause` 字段中。要执行此操作，请使用 `addCause` 函数。
- 4 如果有针对当前错误的修复建议，您可以将其添加到要引发的 `MException` 的 `Correction` 字段。要执行此操作，请使用 `addCorrection` 函数。
- 5 使用 `throw` 或 `throwAsCaller` 函数使 MATLAB 引发异常。此时，MATLAB 将调用堆栈信息存储在 `MException` 的 `stack` 字段中，退出当前正在运行的函数，并将控制权交回给键盘或正在调用的函数中的封闭 `catch` 块。

## 关于如何引发异常的建议

以下示例展示了如何使用刚才所述的步骤引发异常。

创建一个函数 `indexIntoArray`，它使用指定的索引对指定的数组进行索引。该函数会捕获 MATLAB 引发的任何错误，并创建一个异常以提供有关错误的一般信息。当该函数捕获到错误时，它会检测该错误是否涉及输入数量或指定的索引。如果是，该函数会添加额外的异常以提供关于故障来源的更多详细信息，并尽可能提供更正建议。

```
function indexIntoArray(A, idx)

% 1) Detect the error.
try
    A(idx)
catch

    % 2) Construct an MException object to represent the error.
    msgID = 'MYFUN:BadIndex';
    msg = 'Unable to index into array.';
    baseException = MException(msgID,msg);

    % 3) Store any information contributing to the error.
    if nargin < 2
        causeException = MException('MATLAB:notEnoughInputs','Not enough input arguments.');
        baseException = addCause(baseException,causeException);
    end

    % 4) Suggest a correction, if possible.
    if(nargin > 1)
        exceptionCorrection = matlab.lang.correction.AppendArgumentsCorrection('1');
        baseException = baseException.addCorrection(exceptionCorrection);
    end

    throw(baseException);
end

try
    assert(isnumeric(idx),'MYFUN:notNumeric',...
        'Indexing array is not numeric.')
catch causeException
    baseException = addCause(baseException,causeException);
end

if any(size(idx) > size(A))
    msgID = 'MYFUN:incorrectSize';
    msg = 'Indexing array is too large.';
    causeException2 = MException(msgID,msg);
    baseException = addCause(baseException,causeException2);
end
```

```
% 5) Throw the exception to stop execution and display an error
% message.
throw(baseException)
end
end
```

如果您在未指定索引的情况下调用该函数，该函数会抛出详细的错误消息并提供更正建议：

```
A = [13 42; 7 20];
indexIntoArray(A)
```

```
Error using indexIntoArray (line 21)
Unable to index into array.
```

Caused by:

Not enough input arguments.

Did you mean:

```
>> indexIntoArray(A, 1)
```

如果使用过大的非数值索引数组调用该函数，该函数会引发详细的错误。

```
A = [13 42; 7 20];
idx = ['a' 'b' 'c'];
indexIntoArray(A, idx)
```

```
Error using indexIntoArray (line 41)
Unable to index into array.
```

Caused by:

Error using indexIntoArray (line 25)

Indexing array is not numeric.

Indexing array is too large.

# 对异常作出响应

## 本节内容

- “概述”（第 27-13 页）
- “try/catch 语句”（第 27-13 页）
- “有关如何处理异常的建议”（第 27-14 页）

## 概述

默认情况下，当引发异常时，MATLAB 软件会终止当前正在运行的程序。但如果您捕获到程序中的异常，则可以捕获有关出现的问题的信息，并以适合特定条件的方式处理该情况。这需要使用 **try/catch** 语句。

## try/catch 语句

当您的代码中包含会生成不需要的结果的语句时，请将这些语句放入 **try/catch** 块中，以捕获任何错误并对这些错误进行适当处理。

**try/catch** 语句类似于下面的伪代码。它包括两部分：

- **try** 块，其中包含 **try** 和 **catch** 语句之间的所有行。
- **catch** 块，其中包含 **catch** 和 **end** 语句之间的所有代码行。

```

try
    Perform one ...
    or more operations
A catch ME
    Examine error info in exception object ME
    Attempt to figure out what went wrong
    Either attempt to recover, or clean up and abort
end

```

### B Program continues

程序将执行 **try** 块中的语句。如果程序遇到错误，则会跳过 **try** 块中其余的任何语句并跳转到 **catch** 块的开头（此处显示为点 A）。如果 **try** 块中的所有操作均成功，则执行过程会完全跳过 **catch** 块并转至 **end** 语句后面的第一行（点 B）。

建议在不同的行上指定 **try**、**catch** 和 **end** 命令以及 **try** 和 **catch** 块的代码。如果您将这其中的任何部分合并在同一行上，请用逗号加以分隔：

```

try, surf, catch ME, ME.stack, end
ans =
  file: 'matlabroot\toolbox\matlab\graph3d\surf.m'
  name: 'surf'
  line: 54

```

**注意** 您不能在 **try** 或 **catch** 块中定义嵌套函数。

## Try 块

执行时，您的代码进入 **try** 块并执行每个语句，就好像它是正规程序的一部分一样。如果未遇到任何错误，MATLAB 会完全跳过 **catch** 块并在 **end** 语句后继续执行。如果其中任何 **try** 语句失败，MATLAB 会立即退出 **try** 块，且不执行该块中的任何其余语句，并进入 **catch** 块。

## Catch 块

**catch** 命令标记 **catch** 块的开头，并提供对包含导致异常的信息的数据结构体的访问。这在前面的伪代码中显示为变量 **ME**。此数据结构体是 MATLAB **MException** 类的对象。当出现异常时，MATLAB 构造此类的一个实例并在处理该错误的 **catch** 语句中返回它。

您不需要对 **catch** 语句指定任何参数。如果您不需要 **MException** 对象提供的任何信息或方法，仅需指定 **catch** 关键字。

**MException** 对象是由失败的程序中的内部代码构造的。该对象的属性中包含有关错误的信息，可用于确定发生了什么以及如何继续操作。**MException** 对象还提供多种方法的访问权限，从而使您能够对异常作出响应。请参阅“**MException** 类”（第 27-4 页）中的相应部分以了解有关 **MException** 类的更多信息。

进入 **catch** 块之后，MATLAB 按顺序执行这些语句。这些语句可尝试

- 解决错误。
- 捕获有关错误的更多信息。
- 打开 **MException** 对象中找到的信息并作出适当响应。
- 清理受失败代码影响的环境。

**catch** 块通常以 **rethrow** 命令结尾。**rethrow** 使 MATLAB 退出当前函数，并使调用堆栈信息保持为首次引发异常时的状态。如果此函数处于最高级别（即它不由另一个函数调用），则程序终止。如果失败的函数由另一个函数调用，则程序返回到该函数。程序执行过程会继续返回到较高级别的函数，除非其中任何调用是在较高级别的 **try** 块中进行的，在这种情况下程序会执行各自的 **catch** 块。

“捕获有关异常的信息”（第 27-4 页）部分中提供了有关 **MException** 类的更多信息。

## 有关如何处理异常的建议

以下是读取图像文件内容的示例。它包括详细的错误处理，并演示一些针对错误可以采取的建议措施。

图像读取函数以几种方式引发和捕获错误。

- 第一个 **if** 语句检查函数调用时是否带输入参数。如果没有指定输入参数，程序会引发错误，并建议提供输入参数来更正错误。
- **try** 块尝试打开和读取文件。如果打开或读取操作失败，程序会捕获生成的异常并将 **MException** 对象保存在变量 **ME1** 中。
- **catch** 块用于检查是否找不到指定的文件。如果是，则程序会提供以下可能性：通过使用修改后的扩展名重试操作，以使用常见的文件扩展名变化形式（例如 **jpeg** 而不是 **jpg**）。这是通过嵌套在原始 **try/catch** 中的 **try/catch** 语句完成的。

```
function d_in = read_image(filename)

% Check the number of input arguments.
if nargin < 1
    me = MException('MATLAB:notEnoughInputs','Not enough input arguments.');
    aac = matlab.lang.correction.AppendArgumentsCorrection("image.png");
    me = me.addCorrection(aac);
end
```

```

throw(me);
end

% Attempt to read file and catch an exception if it arises.
try
    fid = fopen(filename,'r');
    d_in = fread(fid);
catch ME1
    % Get last segment of the error message identifier.
    idSegLast = regexp(ME1.identifier, '(?<=:)\\w+$', 'match');

    % Did the read fail because the file could not be found?
    if strcmp(idSegLast,'InvalidFid') && ...
        ~exist(filename,'file')

        % Yes. Try modifying the filename extension.
        switch ext
        case '.jpg' % Change jpg to jpeg
            filename = strrep(filename,'jpg','jpeg');
        case '.jpeg' % Change jpeg to jpg
            filename = strrep(filename,'jpeg','jpg');
        case '.tif' % Change tif to tiff
            filename = strrep(filename,'.tif','.tiff');
        case '.tiff' % Change tiff to tif
            filename = strrep(filename,'.tiff','.tif');
        otherwise
            fprintf('File %s not found\\n',filename);
            rethrow(ME1);
        end
    end
end

```

该示例展示了一些可用来对异常作出响应的操作。

- 将 **MException** 对象的 **identifier** 字段与可能的错误原因进行对比。在本例中，函数检查标识符是否以 'InvalidFid' 结尾，以此结尾则表示找不到文件。
- 使用嵌套的 **try/catch** 语句，用改进的输入重试该操作。在本例中，函数使用文件扩展名的已知变体重新尝试打开和读取操作。
- 显示适当的消息。
- 将第一个 **MException** 对象添加到第二个对象的 **cause** 字段中。
- 向 **MException** 对象添加建议的更正。
- 重新引发异常。这会停止程序的执行并显示错误消息。

此外，建议清理掉任何不需要的错误结果。例如，关闭在错误发生后保持打开的图窗。

## 在函数结束后清理

### 本节内容

- “概述”（第 27-16 页）
- “退出时清理程序的示例”（第 27-17 页）
- “检索有关清理例程的信息”（第 27-18 页）
- “使用 onCleanup 与 try/catch”（第 27-19 页）
- “脚本中的 onCleanup”（第 27-19 页）

### 概述

好的编程做法是确保使您的程序环境处于干净的状态，这样不会干扰任何其他程序代码。例如，您可能需要

- 将为了导入或导出而打开的任何文件关闭。
- 恢复 MATLAB 路径。
- 锁定或取消锁定内存以防止或允许擦除 MATLAB 函数或 MEX 文件。
- 将您的工作文件夹重新设置为默认状态（如果已做了更改）。
- 确保全局变量和永久变量处于正确的状态。

为此，MATLAB 提供了 **onCleanup** 函数。在任何程序中使用此函数时，均可针对该程序建立一个清理例程。当此函数终止时，无论是正常终止、出现错误还是键入了 **Ctrl+C**，MATLAB 都会自动执行清理例程。

以下语句可针对当前运行的程序建立一个清理例程 **cleanupFun**：

```
cleanupObj = onCleanup(@cleanupFun);
```

当程序退出时，MATLAB 查找 **onCleanup** 类的任何实例并执行关联的函数句柄。生成并激活函数清理的过程涉及以下步骤：

- 1 针对正在开发的程序编写一个或多个清理例程。目前假设它只采用一个此类例程。
- 2 针对该清理例程创建一个函数句柄。
- 3 在某些时候（通常是程序代码的早期），插入对 **oncleanup** 函数的调用，并传递函数句柄。
- 4 当程序运行时，对 **onCleanup** 的调用会构造一个清理对象，其中包含步骤 1 中创建的清理例程的句柄。
- 5 当程序结束时，MATLAB 隐式清除属于局部变量的所有对象。这会对您程序中的每个局部对象（包括步骤 4 中构造的清理对象）调用析构函数方法。
- 6 该对象的析构函数方法调用此例程（如果存在）。这会执行恢复您的编程环境所需的任务。

您可以针对程序文件声明任意个清理例程。每次调用 **onCleanup** 都会为返回的每个清理对象建立一个单独的清理例程。

如果出于某种原因，**onCleanup** 返回的对象的保留期超过您程序的生命周期，则当您的函数终止时，不会运行与该对象关联的清理例程。清理例程将在毁坏该对象（例如通过清除对象变量）时运行。

您的清理例程绝不应依赖于在该例程范围外定义的变量。例如，这里左侧显示的嵌套函数在执行时不会发生任何错误，而右侧非常相似的函数在执行时会失败，并出现错误 **Undefined function or variable 'k'**。出现此错误是因为清理例程依赖的变量 **k** 是在嵌套的清理例程范围之外定义的：

```

function testCleanup          function testCleanup
k = 3;                      k = 3;
myFun                         obj = onCleanup(@myFun);
    function myFun            function myFun
        fprintf('k is %d\n', k)   fprintf('k is %d\n', k)
    end                      end
end                           end

```

## 退出时清理程序的示例

### 示例 1 - 退出时关闭打开的文件

当函数 `openFileSafely` 终止时，MATLAB 将关闭带有标识符 `fid` 的文件：

```

function openFileSafely(fileName)
fid = fopen(fileName, 'r');
c = onCleanup(@()fclose(fid));

s = fread(fid);
.
.
.

end

```

### 示例 2 - 保留所选文件夹

本示例保留当前文件夹，而不管 `functionThatMayError` 是否返回错误：

```

function changeFolderSafely(fileName)
currentFolder = pwd;
c = onCleanup(@()cd(currentFolder));

functionThatMayError;
end % c executes cd(currentFolder) here.

```

### 示例 3 - 关闭图窗并恢复 MATLAB 路径

本示例扩展 MATLAB 路径以包括 `toolbox\images` 文件夹中的文件，然后显示其中一个文件夹中的图窗。图窗显示后，清理例程 `restore_env` 会关闭该图窗并将路径恢复到初始状态。

```

function showImageOutsidePath(imageFile)
fig1 = figure;
imgpath = genpath([matlabroot '\toolbox\images']);

% Define the cleanup routine.
cleanupObj = onCleanup(@()restore_env(fig1, imgpath));

% Modify the path to gain access to the image file,
% and display the image.
addpath(imgpath);
rgb = imread(imageFile);
fprintf('\n Opening the figure %s\n', imageFile);
image(rgb);
pause(2);

% This is the cleanup routine.
function restore_env(fighandle, newpath)

```

```
disp ' Closing the figure'
close(fighandle);
pause(2)

disp ' Restoring the path'
rmpath(newpath);
end
end
```

运行如下所示的函数。您可以通过比较运行函数前后路径的长度，来验证路径是否已恢复：

```
origLen = length(path);

showImageOutsidePath('greens.jpg')
Opening the figure greens.jpg
Closing the figure
Restoring the path

currLen = length(path);
currLen == origLen
ans =
1
```

## 检索有关清理例程的信息

在上面所示的示例 3 中，清理例程及调用它所需的数据包含在匿名函数的句柄中：

```
@()restore_env(fig1, imgpath)
```

该句柄的详细信息随后包含在 `onCleanup` 函数返回的对象中：

```
cleanupObj = onCleanup(@()restore_env(fig1, imgpath));
```

您可以使用清理对象的 `task` 属性访问这些详细信息，如下所示。（通过将以下代码正好添加到提示 “% This is the cleanup routine.” 的注释行之前来修改 `showImageOutsidePath` 函数）

```
disp ' Displaying information from the function handle:'
task = cleanupObj.task;
fun = functions(task)
wsp = fun.workspace{2,1}
fprintf('\n');
pause(2);
```

运行修改后的函数以查看 `functions` 命令的输出及其中一个 `workspace` 元胞的内容：

```
showImageOutsidePath('greens.jpg')
```

```
Opening the figure greens.jpg
Displaying information from the function handle:
fun =
  function: '@()restore_env(fig1,imgpath)'
    type: 'anonymous'
    file: 'c:\work\g6.m'
  workspace: {2x1 cell}
wsp =
  imageFile: 'greens.jpg'
  fig1: 1
  imgpath: [1x3957 char]
```

```
cleanupObj: [1x1 onCleanup]
  rgb: [300x500x3 uint8]
  task: @(restore_env(fig1,imgpath))
```

Closing the figure  
Restoring the path

## 使用 onCleanup 与 try/catch

在函数意外终止时运行清理例程的另一种方法是使用 **try**, **catch** 语句。但使用此方法时有局限。如果用户通过键入 **Ctrl+C** 结束程序, MATLAB 会立即退出 **try** 块, 并且不会执行清理例程。清理例程也不会在您正常退出函数时运行。

下面的程序会在出错时进行清理, 但对于 **Ctrl+C** 并不作出响应:

```
function cleanupByCatch
try
  pause(10);
catch
  disp(' Collecting information about the error')
  disp(' Executing cleanup tasks')
end
```

与 **try/catch** 语句不同, **onCleanup** 函数不仅会对正常退出程序以及可能引发的任何错误作出响应, 还会对 **Ctrl+C** 做出响应。下一个示例将 **try/catch** 替换为 **onCleanup**:

```
function cleanupByFunc
obj = onCleanup(@()...
  disp(' Executing cleanup tasks'));
pause(10);
```

## 脚本中的 onCleanup

**onCleanup** 在脚本中的工作方式与在函数中不同。在函数中, 清理对象存储在函数工作区中。当函数退出时, 会清空此工作区, 因而执行关联的清理例程。在脚本中, 清理对象存储在基础工作区中 (即用于在命令提示符下完成的交互式工作的工作区)。因为现有脚本对基础工作区没有任何影响, 所以不会清除清理对象并且不会执行与该对象关联的例程。要在脚本中使用这种类型的清理机制, 您必须在第一个脚本终止时通过命令行或另一个脚本以显式方式清除该对象。

## 引发警告和错误

### 本节内容

- “引发警告” (第 27-20 页)
- “引发错误” (第 27-20 页)
- “向您的警告和错误中添加运行时参数” (第 27-21 页)
- “向警告和错误中添加标识符” (第 27-21 页)

## 引发警告

您可以引发警告以便给在运行程序时检测到的意外条件添加标记。`warning` 函数将一则警告消息输出到命令行。警告与错误的区别主要表现在两个方面：

- 警告不会暂停程序执行。
- 您可以隐蔽任何无用的 MATLAB 警告。

在您的代码中使用 `warning` 函数，即可在执行时生成警告消息。将消息指定为 `warning` 函数的输入参数：

```
warning('Input must be text')
```

例如，您可以在代码中插入警告来验证软件版本：

```
function warningExample1
if ~strncmp(version, '7', 1)
    warning('You are using a version other than v7')
end
```

## 引发错误

您可以引发错误以标记程序中的严重问题。使用 `error` 函数将错误消息输出到命令行。在显示消息后，MATLAB 停止执行当前的程序。

例如，假定您构造一个函数，以返回  $n$  个元素中  $k$  个元素的组合数。如果  $k > n$ ，此类函数没有意义；您不能从 4 个元素中选择 8 个元素。您必须将此事实结合到该函数中，以便让使用 `combinations` 的任何人都知道此问题：

```
function com = combinations(n,k)
if k > n
    error('Cannot calculate with given values')
end
com = factorial(n)/(factorial(k)*factorial(n-k));
end
```

如果 `combinations` 函数收到无效输入，MATLAB 在引发错误消息后立即停止执行：

```
combinations(4,8)
```

```
Error using combinations (line 3)
Cannot calculate with given values
```

## 向您的警告和错误中添加运行时参数

要使您的警告或错误消息更具体，请在执行时插入消息的各个组成部分。`warning` 函数使用的转换字符与 `sprintf` 函数所用的相同。转换字符充当子字符串或值的占位符，在执行代码之前是未知的。

例如，此警告使用 `%s` 和 `%d` 来标记在何处插入变量 `arrayname` 和 `arraydims` 的值：

```
warning('Array %s has %d dimensions.',arrayname,arraydims)
```

如果您在 `arrayname = 'A'` 且 `arraydims = 3` 时执行此命令，MATLAB 会做出如下响应：

```
Warning: Array A has 3 dimensions.
```

向您的警告和错误中添加运行时参数可阐明程序中的问题。以“引发错误”（第 27-20 页）中的函数 `combinations` 为例。您可以使用运行时参数引发一则包含更多信息的错误：

```
function com = combinations(n,k)
    if k > n
        error('Cannot choose %i from %i elements',k,n)
    end
    com = factorial(n)/(factorial(k)*factorial(n-k));
end
```

如果此函数收到无效参数，MATLAB 会引发错误消息并停止程序：

```
combinations(6,9)
```

```
Error using combinations (line 3)
Cannot choose 9 from 6 elements
```

## 向警告和错误中添加标识符

消息标识符提供一种唯一引用警告或错误的方法。

使用标识符启用或禁用警告。在 `warning` 函数中使用标识文本参数以将一个唯一标记绑定到消息：

```
warning(identifier_text,message_text)
```

例如，您可以在上一个 MATLAB 警告中添加一个有关所运行的软件版本的标识符标记：

```
minver = '7';
if ~strcmp(version,minver,1)
    warning('MYTEST:VERCHK','Running a version other than v%s',minver)
end
```

通过在错误消息中添加标识符，可以进行逆向测试。但是，在错误中添加和找回更多信息往往需要使用 `MException` 对象。

## 另请参阅

`MException` | `lastwarn` | `warndlg` | `warning`

## 相关示例

- “隐蔽警告”（第 27-23 页）
- “恢复警告”（第 27-25 页）

- “捕获有关异常的信息”（第 27-4 页）
- “MATLAB 应用程序中的异常处理”（第 27-2 页）

### 详细信息

- “消息标识符”（第 27-6 页）

## 隐蔽警告

您的程序发出的警告并非总是对执行产生不利影响。为避免混淆，您可以在执行期间将警告消息的状态从 'on' 更改为 'off' 来隐蔽这些消息。

要隐蔽特定的警告消息，必须首先查找警告标识符。每条警告消息都有一个唯一标识符。要查找与 MATLAB 警告关联的标识符，请重现该警告。例如，以下代码会重现 MATLAB 在尝试删除不存在的文件夹时引发的警告：

```
rmpath('folderthatisnotonpath')
```

```
Warning: "folderthatisnotonpath" not found in path.
```

---

**注意** 如果此语句不生成警告消息，请使用以下代码暂时显示所有警告，然后恢复初始的警告状态：

```
w = warning('on','all');
rmpath('folderthatisnotonpath')
warning(w)
```

---

要获取有关最近引发的警告的信息，请使用 `warning` 或 `lastwarn` 函数。此代码使用 `query` 状态返回一个数据结构体，其中包含消息标识符和最近引发的警告当前所处的状态：

```
w = warning('query','last')
```

```
w =
```

```
identifier: 'MATLAB:rmpath:DirNotFound'
state: 'on'
```

您可以将标识符字段保存在变量 `id` 中：

```
id = w.identifier;
```

---

**注意** `warning('query','last')` 返回最近显示的警告。MATLAB 仅显示具有 `state: 'on'` 和警告标识符的警告消息。

使用 `lastwarn` 函数，您可以检索最近的警告消息，而不管其显示状态如何：

```
lastwarn
```

```
ans =
```

```
"folderthatisnotonpath" not found in path.
```

## 开启和关闭警告

从 `query` 状态获取标识符后，使用此信息禁用或启用与该标识符关联的警告。

继续上一部分中的示例，关闭 'MATLAB:rmpath:DirNotFound' 警告，并重复操作。

```
warning('off',id)
rmpath('folderthatisnotonpath')
```

MATLAB 不显示警告。

开启警告，并尝试删除不存在的路径：

```
warning('on',id)  
rmpath('folderthatishnotinonpath')
```

Warning: "folderthatishnotinonpath" not found in path.

MATLAB 此时将引发警告。

---

**提示** 使用 `warning('off','last')` 关闭最近生成的警告。

---

### 控制所有警告

**所有**一词仅指那些在您的当前 MATLAB 会话中已引发或修改的警告。已修改的警告状态仅在整个当前会话中保留。开始新会话将会恢复默认设置。

使用标识符 '`all`' 表示包含所有警告的组。使用以下任一语法查看所有警告的状态：

```
warning('query','all')
```

```
warning
```

要启用所有警告并验证状态，请执行以下操作：

```
warning('on','all')  
warning('query','all')
```

All warnings have the state 'on'.

要禁用所有警告并验证状态，请使用此语法：

```
warning('off','all')  
warning
```

All warnings have the state 'off'.

### 另请参阅

#### 相关示例

- “恢复警告”（第 27-25 页）
- “更改警告的显示方式”（第 27-27 页）

## 恢复警告

MATLAB 允许您保存 on-off 警告状态，修改警告状态以及恢复初始的警告状态。如果您需要暂时关闭某些警告并在以后恢复初始设置，这很有用。

以下语句将所有警告的当前状态保存在一个名为 `orig_state` 的结构体数组中：

```
orig_state = warning;
```

要在修改任何警告后恢复初始状态，请使用此语法：

```
warning(orig_state);
```

您还可以用单个命令来保存当前状态并切换警告。例如，`orig_state = warning('off','all');` 语句等同于以下命令：

```
orig_state = warning;
warning('off','all')
```

## 禁用和恢复特定警告

本示例显示如何恢复特定警告的状态。

- 1 查询 `Control:parameterNotSymmetric` 警告：

```
warning('query','Control:parameterNotSymmetric')
```

The state of warning 'Control:parameterNotSymmetric' is 'on'.

- 2 关闭 `Control:parameterNotSymmetric` 警告：

```
orig_state = warning('off','Control:parameterNotSymmetric')
```

```
orig_state =
```

```
identifier: 'Control:parameterNotSymmetric'
state: 'on'
```

`orig_state` 包含 MATLAB 禁用 `Control:parameterNotSymmetric` 之前的警告状态。

- 3 查询所有警告状态：

```
warning
```

The default warning state is 'on'. Warnings not set to the default are

State Warning Identifier

```
off Control:parameterNotSymmetric
```

MATLAB 指明 `Control:parameterNotSymmetric` 为 'off'。

- 4 恢复初始状态：

```
warning(orig_state)
warning('query','Control:parameterNotSymmetric')
```

The state of warning 'Control:parameterNotSymmetric' is 'on'.

## 禁用和恢复多个警告

本示例显示如何保存和恢复多个警告状态。

- 1 禁用三个警告，并查询所有警告：

```
w(1) = warning('off','MATLAB:rmpath:DirNotFound');
w(2) = warning('off','MATLAB:singularMatrix');
w(3) = warning('off','Control:parameterNotSymmetric');
warning
```

The default warning state is 'on'. Warnings not set to the default are

State Warning Identifier

```
off Control:parameterNotSymmetric
off MATLAB:rmpath:DirNotFound
off MATLAB:singularMatrix
```

- 2 将这三个警告恢复为到各自的初始状态，并查询所有警告：

```
warning(w)
warning
```

All warnings have the state 'on'.

虽然您不必将有关之前的警告状态的信息存储在数组中，但这样做可让您通过一个命令恢复警告。

---

**注意** 当暂时禁用多个警告时，使用与 **onCleanup** 相关的方法可能更好些。

---

再者，还可以保存并恢复所有警告。

- 1 启用所有警告，并保存初始警告状态：

```
orig_state = warning('on','all');
```

- 2 将警告恢复到先前的状态：

```
warning(orig_state)
```

## 另请参阅

[onCleanup](#) | [warning](#)

## 相关示例

- “[隐蔽警告](#)”（第 27-23 页）
- “[在函数结束后清理](#)”（第 27-16 页）

## 更改警告的显示方式

您可以通过修改两种警告模式（**verbose** 和 **backtrace**）来控制警告在 MATLAB 中的显示方式。

模式	说明	默认值
<b>verbose</b>	显示有关如何隐蔽警告的消息。	<b>off (terse)</b>
<b>backtrace</b>	在生成警告后显示堆栈跟踪。	<b>on (enabled)</b>

---

注意 **verbose** 和 **backtrace** 模式存在某些局限：

- **prev\_state** 并不将有关 **backtrace** 或 **verbose** 模式的信息包含在 **prev\_state = warning('query','all')** 语句中。
  - 模式变更会影响到所有已启用的警告。
- 

## 启用 Verbose 警告

当您启用 **verbose** 警告时，MATLAB 会针对每个警告额外显示一行信息，告知您如何隐蔽该警告。

例如，您可以开启所有警告，禁用 **backtrace** 警告而启用 **verbose** 警告：

```
warning on all
warning off backtrace
warning on verbose
```

运行一条生成错误的命令会显示一则扩充的消息：

```
rmpath('folderthatishnotinpath')
```

```
Warning: "folderthatishnotinpath" not found in path.
(Type "warning off MATLAB:rmpath:DirNotFound" to suppress this warning.)
```

## 显示对特定警告的堆栈跟踪

如果警告是从淹没在多级函数调用中的代码生成的，则很难找到该警告的来源。当您启用 **backtrace** 模式时，MATLAB 显示出现警告的文件名和行号。例如，您可以启用 **backtrace** 并禁用 **verbose**：

```
warning on backtrace
warning off verbose
```

运行一条生成错误的命令会显示一个带行号的超链接：

```
Warning: "folderthatishnotinpath" not found in path.
> In rmpath at 58
```

点击该超链接可转到警告的位置。

## 使用 try/catch 处理错误

在您的程序遇到错误后，可以使用 **try/catch** 语句执行代码。**try/catch** 语句在以下情况下很有用：

- 希望以另一种方式来完成程序以避免错误
- 需要清除不必要的、意外的错误结果
- 有许多有问题的输入参数或命令

将 **try/catch** 语句安排到代码块中，类似于以下伪代码：

```
try
  try block...
catch
  catch block...
end
```

如果 **try block** 中出现错误，MATLAB 会跳过 **try** 块中其余的任何命令并执行 **catch block** 中的命令。如果 **try block** 中没有出现任何错误，MATLAB 会跳过整个 **catch block**。

例如，**try/catch** 语句可避免引发错误。以 **combinations** 函数为例，该函数返回 **n** 个元素中 **k** 个元素的组合数。

```
function com = combinations(n,k)
  com = factorial(n)/(factorial(k)*factorial(n-k));
end
```

MATLAB 会在 **k > n** 时引发错误。您不能构造一个元素数 **k** 多于拥有的元素数 **n** 的集合。使用 **try/catch** 语句，您可以避免错误并执行此函数，而不管输入顺序如何：

```
function com = robust_combine(n,k)
try
  com = factorial(n)/(factorial(k)*factorial(n-k));
catch
  com = factorial(k)/(factorial(n)*factorial(k-n));
end
end
```

**robust\_combine** 将任何顺序的整数视为有效输入：

```
C1 = robust_combine(8,4)
C2 = robust_combine(4,8)
```

C1 =

70

C2 =

70

或者，如果某变量在您的 **catch** 语句之后，则您可以捕获有关错误的更多信息：

**catch MExc**

**MExc** 是一个 **MException** 类对象，其中包含有关所引发的错误的更多信息。要详细了解如何访问 **MException** 对象中的信息，请参阅“MATLAB 应用程序中的异常处理”（第 27-2 页）。

**另请参阅**

[MException | onCleanup](#)



# 程序的安排

---

- “使用计时器安排命令的执行” (第 28-2 页)
- “计时器回调函数” (第 28-4 页)
- “处理计时器队列冲突” (第 28-7 页)

## 使用计时器安排命令的执行

### 本节内容

[“概述” \(第 28-2 页\)](#)

[“示例：显示消息” \(第 28-2 页\)](#)

### 概述

MATLAB 软件包括一个计时器对象，您可以使用该对象安排 MATLAB 命令的执行。本部分介绍如何创建计时器对象，启动计时器运行，以及指定您希望在计时器触发时执行的过程。当计时器对象指定的时间已过且计时器对象执行您指定的命令时，即触发计时器。

要使用计时器，请执行以下步骤：

**1** 创建一个计时器对象。

可使用 `timer` 函数创建计时器对象。

**2** 指定您希望在计时器触发时执行的 MATLAB 命令并控制计时器对象的其他各方面的行为。

使用计时器对象属性指定此信息。要了解计时器对象支持的所有属性，请参阅计时器和 `set`。您也可以在创建计时器对象属性（在步骤 1 中）时设置这些属性。

**3** 启动计时器对象。

创建计时器对象后，必须使用 `start` 或 `startat` 函数启动该对象。

**4** 在处理完计时器对象后将其删除。

在使用完计时器对象后，应将其从内存中删除。有关详细信息，请参阅 `delete`。

**注意** 计时器的指定执行时间和实际执行可能不同，这是因为计时器对象在 MATLAB 单线程执行环境中工作。该滞后时间的长短取决于 MATLAB 正在执行的其他过程。要强制执行事件队列中的回调函数，请在代码中包含对 `drawnow` 函数的调用。`drawnow` 函数会刷新事件队列。

### 示例：显示消息

以下示例设置一个计时器对象，该对象在 10 秒过后执行 MATLAB 命令字符串向量。该示例将创建一个计时器对象，指定两个计时器对象属性 `TimerFcn` 和 `StartDelay` 的值。`TimerFcn` 指定计时器回调函数。这是您要在计时器触发时执行的 MATLAB 命令或程序文件。在该示例中，计时器回调函数设置 MATLAB 工作区变量 `stat` 的值并执行 MATLAB `disp` 命令。`StartDelay` 属性指定计时器触发前已用的时间。

创建计时器对象后，该示例使用 `start` 函数启动计时器对象。（为了展示计时器，该示例中包括了一些其他命令，但这些命令并不是计时器操作必需的。）

```
t = timer('TimerFcn', 'stat=false; disp("Timer!")',...
    'StartDelay',10);
start(t)

stat=true;
while(stat==true)
    disp('.')
    pause(1)
end
```

当您执行此代码时，会生成以下输出：

```
.  
. .  
. .  
. .  
. .  
Timer!  
delete(t) % Always delete timer objects after using them.
```

## 另请参阅

[timer](#)

## 详细信息

- “[计时器回调函数](#)” (第 28-4 页)
- “[处理计时器队列冲突](#)” (第 28-7 页)

## 计时器回调函数

### 本节内容

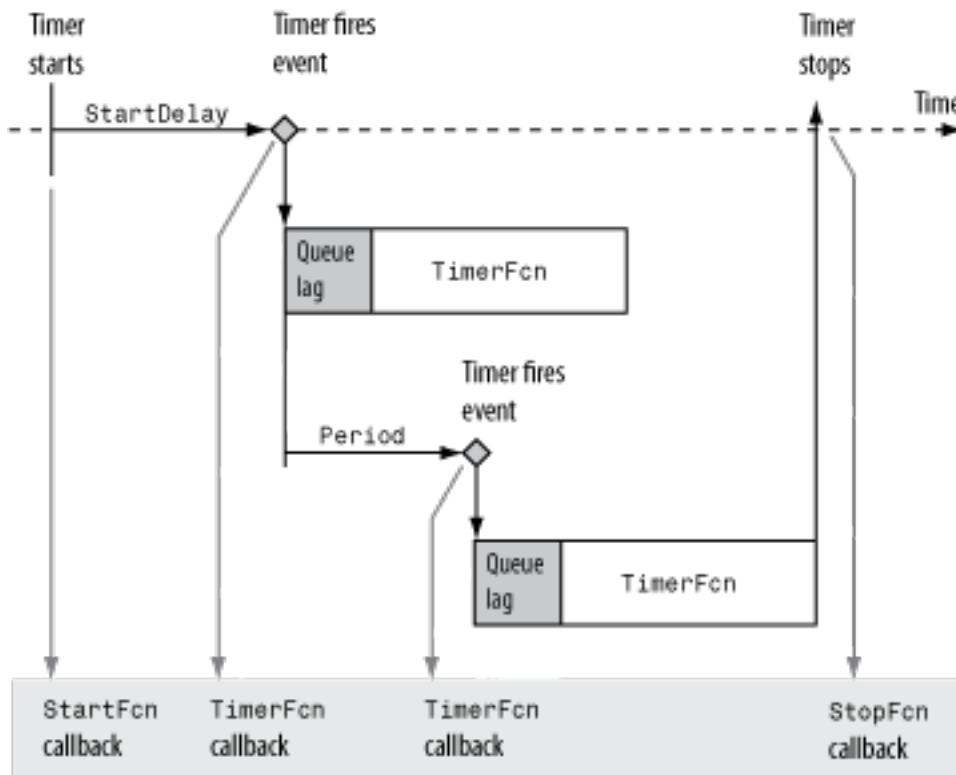
- “将命令与计时器对象事件关联”（第 28-4 页）
- “创建回调函数”（第 28-5 页）
- “指定回调函数属性的值”（第 28-5 页）

**注意** 如果回调涉及占用大量 CPU 的任务（例如更新图窗），则可能延迟执行回调函数。

### 将命令与计时器对象事件关联

计时器对象支持的属性允许您指定在计时器触发时执行的 MATLAB 命令，以及针对其他计时器对象事件（例如启动、停止或出现错误时）指定这些命令。这称为回调。要将 MATLAB 命令与计时器对象事件关联，请设置关联的计时器对象回调属性的值。

下图显示了在执行计时器对象期间何时发生事件，并提供与每个事件关联的计时器对象属性的名称。例如，要将 MATLAB 命令与启动事件关联，请为 StartFcn 回调属性赋值。错误回调随时都会发生。



计时器对象事件和相关的回调函数

## 创建回调函数

当计时器对象指定的时间段过去后，计时器对象会执行您所选择的一个或多个 MATLAB 函数。您可以将这些函数直接指定为回调属性的值。您还可以将这些命令放入函数文件中，并将该函数指定为回调属性的值。

### 直接指定回调函数

本示例创建一个在 5 秒后显示问候语的计时器对象。本示例直接指定 **TimerFcn** 回调属性的值，并将命令放入字符串中。

```
t = timer('TimerFcn',@(x,y)disp('Hello World!'),'StartDelay',5);
```

---

**注意** 当您将回调命令直接指定为回调函数属性的值时，将在 MATLAB 工作区中计算这些命令。

---

### 将命令放在调用函数中

您可以将 MATLAB 命令放在 MATLAB 程序文件中并将该文件指定为回调属性的值，而不用直接将这些命令指定为回调属性的值。

当您创建回调函数时，前两个参数必须是计时器对象句柄和事件结构体。事件结构体包含两个字段：**Type** 和 **Data**。**Type** 字段包含一个字符串，用于标识导致回调的事件类型。此字段的值可以是以下任何一项：'StartFcn'、'StopFcn'、'TimerFcn' 或 'ErrorFcn'。**Data** 字段包含事件发生的时间。

除了这两个必需的输入参数外，您的回调函数还可以接受应用程序特定的参数。要获得这些输入参数，您必须在将该函数的名称指定为回调属性的值时使用元胞数组。有关详细信息，请参阅“指定回调函数属性的值”（第 28-5 页）。

### 示例：编写回调函数

本示例实现一个简单回调函数，用以显示触发回调的事件类型及发生回调的时间。为演示如何传递应用程序特定的参数，该回调函数示例接受字符串作为额外参数并在显示输出中包含此文本。要查看本函数所使用回调属性，请参阅“指定回调函数属性的值”（第 28-5 页）。

```
function my_callback_fcn(obj, event, text_arg)

txt1 = ' event occurred at ';
txt2 = text_arg;

event_type = event.Type;
event_time = datestr(event.Data.time);

msg = [event_type txt1 event_time];
disp(msg)
disp(txt2)
```

## 指定回调函数属性的值

通过设置适当的回调属性的值可将回调函数与特定事件关联起来。可将回调函数指定为元胞数组或函数句柄。如果您的回调函数接受额外的参数，则必须使用元胞数组。

下表显示了若干回调函数示例的语法并介绍了调用方式。

回调函数语法	如何指定为对象 t 的属性值
<b>function myfile(obj, event)</b>	<b>t.StartFcn = @myfile</b>
<b>function myfile</b>	<b>t.StartFcn = @(~,~)myfile</b>
<b>function myfile(obj, event, arg1, arg2)</b>	<b>t.StartFcn = {@myfile, 5, 6}</b>

本示例演示了可以指定计时器对象回调函数属性值的多种方法，其中某些方法带参数，某些不带。要查看回调函数 **my\_callback\_fcn** 的代码，请参阅“示例：编写回调函数”（第 28-5 页）：

- 1 创建一个计时器对象。

```
t = timer('StartDelay', 4, 'Period', 4, 'TasksToExecute', 2, ...
    'ExecutionMode', 'fixedRate');
```

- 2 指定 **StartFcn** 回调的值。请注意，本示例用元胞数组指定该值，这是因为回调函数需要访问传递过来的参数：

```
t.StartFcn = {@my_callback_fcn, 'My start message'};
```

- 3 指定 **StopFcn** 回调的值。同样，用元胞数组指定该值，这是因为回调函数需要访问传递过来的参数：

```
t.StopFcn = { @my_callback_fcn, 'My stop message'};
```

- 4 指定 **TimerFcn** 回调的值。本示例用字符串来指定 MATLAB 命令：

```
t.TimerFcn = @(x,y)disp('Hello World!');
```

- 5 启动计时器对象：

```
start(t)
```

本示例的输出结果如下。

```
StartFcn event occurred at 10-Mar-2004 17:16:59
My start message
Hello World!
Hello World!
StopFcn event occurred at 10-Mar-2004 17:16:59
My stop message
```

- 6 用完计时器对象后将其删除。

```
delete(t)
```

## 另请参阅

[timer](#)

## 详细信息

- “处理计时器队列冲突”（第 28-7 页）

# 处理计时器队列冲突

在繁忙时段，在多执行情形中，计时器可能需要在先前排队的回调函数执行完毕之前，将计时器回调函数 (TimerFcn) 添加到 MATLAB 执行队列中。您可以通过将 **BusyMode** 属性设置为以下模式之一来确定计时器对象如何处理此情形：

## 本节内容

- “Drop 模式（默认值）”（第 28-7 页）
- “错误模式”（第 28-8 页）
- “Queue 模式”（第 28-9 页）

## Drop 模式（默认值）

如果将 'drop' 指定为 **BusyMode** 属性的值，计时器对象仅在执行队列为空时才将计时器回调函数添加到该队列中。如果执行队列不为空，计时器对象会不执行回调。

例如，假设创建一个时间段为 1 秒的计时器，但其回调至少需要 1.6 秒，正如此处 `mytimer.m` 所示的计时器。

```
function mytimer()
    t = timer;

    t.Period      = 1;
    t.ExecutionMode = 'fixedRate';
    t.TimerFcn    = @mytimer_cb;
    t.BusyMode    = 'drop';
    t.TasksToExecute = 5;
    t.UserData    = tic;

    start(t)
end

function mytimer_cb(h,~)
    timeStart = toc(h.UserData)
    pause(1.6);
    timeEnd = toc(h.UserData)
end
```

下表介绍了该计时器管理执行队列的方式。

所耗的大致时间 (秒)	操作
0	开始首次执行回调。
1	尝试开始第二次执行回调。第一次执行未完毕，但执行队列为空。计时器将回调添加到队列中。
1.6	第一次执行回调完毕，并开始第二次。此操作会清除执行队列。
2	尝试开始第三次执行回调。第二次执行未完毕，但该队列为空。计时器将回调添加到队列中。

所耗的大致时间 (秒)	操作
3	尝试开始第四次执行回调。第三次回调在执行队列中，因此计时器此次放弃执行函数。
3.2	第二次执行回调完毕并开始第三次，并且清除执行队列。
4	尝试开始再一次执行回调。因为队列为空，所以计时器将回调添加到队列中。这是第五次尝试，但仅运行第四个实例。
4.8	第三次执行回调完毕并开始第四个实例，并且清除队列。
5	尝试开始再一次回调。某个实例正在运行，但执行队列为空，因此计时器将其添加到队列中。这是将运行的第五个实例。
6	不执行任何操作：TasksToExecute 属性的值是 5，要运行的第五个实例在队列中。
6.4	第四次执行回调完毕并开始第五次。
8	第五次执行回调完毕。

## 错误模式

BusyMode 属性的 'error' 模式类似于 'drop' 模式：在这两种模式下，计时器仅允许执行队列中有一个回调实例。但在 'error' 模式下，当队列不为空时，计时器会调用您使用 ErrorFcn 属性指定的函数，然后停止处理。当前运行的回调函数执行完毕，但不执行队列中的回调。

例如，修改 mytimer.m (上一节中所述) 以使其包含错误处理函数并将 BusyMode 设置为 'error'。

```
function mytimer()
    t = timer;

    t.Period      = 1;
    t.ExecutionMode = 'fixedRate';
    t.TimerFcn    = @mytimer_cb;
    t.ErrorFcn    = @myerror;
    t.BusyMode    = 'error';
    t.TasksToExecute = 5;
    t.UserData    = tic;

    start(t)
end

function mytimer_cb(h,~)
    timeStart = toc(h.UserData)
    pause(1.6);
    timeEnd = toc(h.UserData)
end

function myerror(h,~)
    disp('Reached the error function')
end
```

下表介绍了该计时器管理执行队列的方式。

所耗的大致时间 (秒)	操作
0	开始首次执行回调。
1	尝试开始第二次执行回调。第一次执行未完毕，但执行队列为空。计时器将回调添加到队列中。
1.6	第一次执行回调完毕，并开始第二次。此操作会清除执行队列。
2	尝试开始第三次执行回调。第二次执行未完毕，但该队列为空。计时器将回调添加到队列中。
3	尝试开始第四次执行回调。第三次回调在执行队列中。计时器不执行第三次回调，而是调用错误处理函数。
3.2	第二次回调执行完毕并启动错误处理函数。

## Queue 模式

如果您指定 '`queue`'，则计时器对象会一直等到当前执行的回调函数执行完毕，然后才再次排队执行计时器回调函数。

在 '`queue`' 模式下，计时器对象会尝试使每次执行之间的平均时间等于 `Period` 属性中指定的时间量。如果计时器对象必须等待的时间比 `Period` 属性中指定的每次执行计时器函数回调之间的时间更长，则它会缩短该时间段以便后续执行弥补这段时间。

### 另请参阅

`timer`

### 详细信息

- “计时器回调函数”（第 28-4 页）



# 性能

---

- “测量程序性能” (第 29-2 页)
- “探查以提升性能” (第 29-4 页)
- “使用探查器可确定代码覆盖率” (第 29-9 页)
- “提升性能的方法” (第 29-11 页)
- “预分配” (第 29-13 页)
- “向量化” (第 29-15 页)

## 测量程序性能

### 本节内容

- “性能计时函数概述”（第 29-2 页）
- “计时函数”（第 29-2 页）
- “计算部分代码的时间”（第 29-2 页）
- “Cputime 函数与 tic/toc 和 timeit”（第 29-2 页）
- “有关测量性能的提示”（第 29-3 页）

### 性能计时函数概述

**timeit** 函数和秒表计时器函数 **tic** 和 **toc** 允许您计算代码运行所需的时间。使用 **timeit** 函数严格测量函数执行时间。使用 **tic** 和 **toc** 可估算运行较小部分代码而非整个函数的时间。

有关代码性能的其他详细信息（例如函数调用信息和各行代码的执行时间），请使用 MATLAB 探查器。有关详细信息，请参阅“探查以提升性能”（第 29-4 页）。

### 计时函数

要测量运行函数所需的时间，请使用 **timeit** 函数。**timeit** 函数多次调用指定的函数，并返回测量结果的中位数。它采用要测量的函数的句柄并返回典型执行时间（以秒为单位）。假设您定义了一个函数 **computeFunction**，它采用两个在工作区中定义的输入 **x** 和 **y**。您可以使用 **timeit** 计算执行函数所需的时间。

```
f = @( ) myComputeFunction(x,y); % handle to function
timeit(f)
```

### 计算部分代码的时间

要计算某部分程序需要多长时间运行或者比较各部分程序的不同实现的速度，可使用秒表计时器函数 **tic** 和 **toc**。调用 **tic** 可启动计时器，紧接着 **toc** 可读取已用时间。

```
tic
    % The program section to time.
toc
```

有时程序运行速度太快，导致 **tic** 和 **toc** 无法提供有用的数据。如果您的代码运行速度快于 1/10 秒，请考虑测量它在循环中运行的时间，然后求平均值以计算单次运行的时间。

### Cputime 函数与 tic/toc 和 timeit

建议您使用 **timeit** 或 **tic** 和 **toc** 来度量代码的性能。这些函数会返回挂钟时间。与 **tic** 和 **toc** 不同，**timeit** 函数会调用您的代码多次，因此会考虑首次成本。

**cputime** 函数会测量总 CPU 时间并跨所有线程进行汇总。此测量值不同于 **timeit** 或 **tic/toc** 返回的挂钟时间，可能会造成误解。例如：

- **pause** 函数的 CPU 时间通常很小，但挂钟时间会考虑暂停 MATLAB 执行的实际时间。因此，挂钟时间可能更长。

- 如果您的函数均匀使用四个处理核，则 CPU 时间可能约是挂钟时间的四倍。

## 有关测量性能的提示

在测量代码的性能时，请考虑以下提示：

- 计算足够大的一部分代码的时间。理想情况下，您进行计时的代码运行时间应该超过 1/10 秒。
- 将您要尝试计时的代码放在函数中，而不是在命令行或脚本内部对其计时。
- 除非您是尝试测量首次成本，否则请多次运行代码。使用 `timeit` 函数。
- 请不要在测量性能时执行 `clear all`。有关详细信息，请参阅 `clear` 函数。
- 将您的输出分配给一个变量，而不是使其保留默认值 `ans`。

## 另请参阅

`profile` | `tic` | `timeit` | `toc`

## 相关示例

- “探查以提升性能”（第 29-4 页）
- “提升性能的方法”（第 29-11 页）
- 发布在 MATLAB Central File Exchange 上的 MATLAB 性能衡量白皮书

## 探查以提升性能

### 本节内容

- “什么是探查功能？”（第 29-4 页）
- “探查过程及准则”（第 29-4 页）
- “使用探查器”（第 29-4 页）
- “探查摘要报告”（第 29-6 页）
- “探查详细信息报告”（第 29-7 页）

### 什么是探查功能？

探查是一种方法，可衡量程序在哪些位置耗用了时间。在确定哪些函数耗用大部分时间后，您可以对它们进行评估以确定可能的性能改进。另外，您还可以将代码用作调试工具来进行探查。例如，确定 MATLAB 不执行哪些代码行可帮助您开发执行该代码的测试用例。如果探查时在文件中遇到错误，您可以查看哪部分已运行，哪部分未运行，以帮助您挑出问题。

**提示** 过早优化的代码可能带来无谓的复杂性，而不会显著改善性能。第一次实施应尽可能简单。此后，如果速度成为问题，请使用探查功能来找出瓶颈所在。

您可以使用 MATLAB 探查器探查代码。探查器是一个用户界面，以 `profile` 函数返回的结果为依据。如果您要探查并行运行的代码，为获得最佳结果，请使用 Parallel Computing Toolbox™ 并行探查器。有关详细信息，请参阅 “Profiling Parallel Code” (Parallel Computing Toolbox)。

### 探查过程及准则

您可以按照以下常规过程来改善代码性能：

- 1 对您的代码运行探查器。
- 2 在探查器摘要报告中，查找使用了大量时间或调用最频繁的函数。
- 3 查看这些函数的探查详细信息报告并查找使用最多时间或调用最频繁的代码行。

考虑保留一份第一个详细信息报告的副本，以作为比较依据。在更改代码后，您可以再次运行探查器并比较报告。

- 4 确定是否可对这些代码行进行更改以改进性能。

例如，如果循环中有一个 `load` 语句，您可以将 `load` 语句移到循环外以便仅调用一次。

- 5 在您的代码中实施可能的性能改进。保存文件并运行 `clear all`。再次运行探查器并将结果与原始报告比较。

如果对完全相同的代码进行两次探查，每次获得的结果可能略有不同，这是与代码无关的固有时间波动所导致的。

- 6 要继续改进代码性能，请重复执行这些步骤。

如果代码用大部分时间来调用少数内置函数，则表示您可能已对代码进行了最大程度的优化。

### 使用探查器

要探查 MATLAB 代码文件或代码行，请执行以下操作：

- 1 使用下列方法之一打开探查器：
  - 在命令行窗口中，键入 **profile viewer**。
  - 在主页选项卡上的**代码**部分中，点击  运行并计时。
  - 在编辑器中，在**编辑器**选项卡的**运行**部分中，点击  运行并计时。如果使用此方法，探查器会自动对当前“编辑器”选项卡中的代码进行探查。如果这是您要探查的代码，请跳至步骤 4。
- 2 在**运行此代码**字段中，键入要运行的语句。

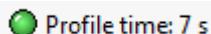
例如，您可以运行随 MATLAB 一起提供的 Lotka-Volterra 示例：

```
[t,y] = ode23('lotka',[0 2],[20;20])
```

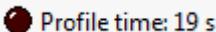
如果在当前 MATLAB 会话中，您之前对该语句进行了探查，则请从**运行此代码**列表中选择该语句。MATLAB 会自动开始探查此代码，您可以跳至步骤 4。

- 3 点击**开始探查**。

在探查器运行时，**探查时间**指示器呈绿色，所报告的秒数也会增加。**探查时间**指示器显示在探查器窗口的右上方。



当探查器运行完毕后，**探查时间**指示器将变为黑色，并显示探查器运行的时长。所探查的语句会在命令行窗口中显示为已执行的状态。



此时间不是运行语句所用的实际时间。它是从您点击**开始探查**一直到探查停止所用的时间。如果报告的时间与您预期的时间差别很大（例如，一个简单语句花费数百秒），则探查时间比所需的时间长。此时间与探查摘要报告统计信息中报告的时间不匹配，默认情况下后者基于 **performance** 时钟时间。要查看使用不同类型时钟的探查统计信息，请使用 **profile** 函数而非探查器。

- 4 探查完毕后，探查摘要报告随即显示在探查器窗口中。有关详细信息，请参阅“探查摘要报告”（第 29-6 页）。

## 在命令行窗口中探查多个语句

要探查多个语句，请执行以下操作：

- 1 在探查器中，点击**开始探查**。确保**运行此代码**字段中不显示任何代码。
- 2 在命令行窗口中，输入并运行您要探查的语句。
- 3 运行所有语句后，在探查器中点击**停止探查**，并查看探查摘要报告。

## 探查用户界面

您可以对用户界面运行探查器，例如 Signal Processing Toolbox 中包含的 Filter Design and Analysis 工具。您也可以探查您所创建的界面，例如使用 GUIDE 构建的界面。

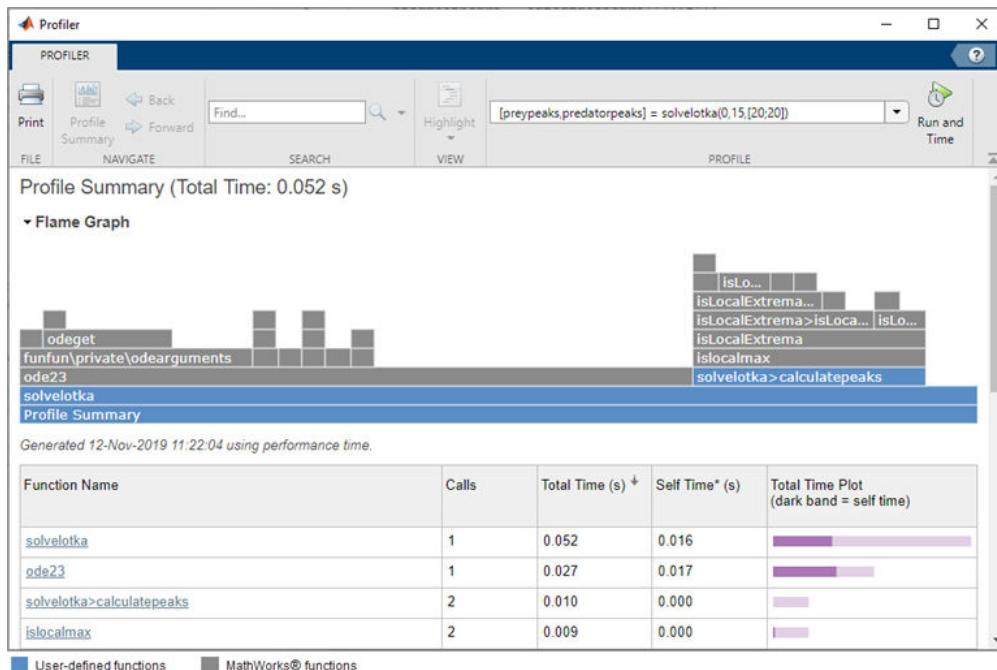
要探查用户界面，请执行以下操作：

- 1 在探查器中，点击**开始探查**。确保**运行此代码**字段中不显示任何代码。
- 2 启动用户界面。
- 3 使用该界面。完成后，在探查器中点击**停止探查**，并查看探查器摘要报告。

**注意** 要在探查中排除用户界面启动过程，请颠倒步骤 1 和 2。也就是说，在点击**开始探查**之前启动用户界面。

## 探查摘要报告

探查摘要报告显示有关函数的总体执行情况的统计信息，并提供调用的每个函数的摘要统计信息。下图显示的是 Lotka-Volterra 模型的探查摘要报告。请参阅“使用探查器”（第 29-4 页）。



探查摘要报告会显示以下信息。

列	说明
<b>函数名称</b>	探查的代码所调用的所有函数列表。起初，这些函数按处理它们时所花的时间排序。
<b>调用</b>	被探查的代码调用函数的次数。
<b>总时间</b>	函数（包括访问的所有子函数）所耗费的总时间（以秒为单位）。函数所耗用的时间包括子函数所耗费的时间。探查器本身会耗用一些时间，该时间也包括在结果中。对于运行时间无足轻重的文件来说，总时间可以是零。
<b>自用时间</b>	函数所耗费的总时间，不包括任何子函数所耗用的时间（以秒为单位）。自用时间还包括探查过程产生的一些开销。
<b>总时间图</b>	以图形方式显示自用时间与总时间的对比情况。

在摘要报告中，您可以：

- 通过点击打印按钮 打印该报告。
- 通过点击**函数名称**列中特定函数的名称，可获取有关该函数的更多详细信息。有关详细信息，请参阅“探查详细信息报告”（第 29-7 页）。
- 通过点击给定列的名称，可以按该列进行排序。要按字母顺序对这些函数进行排序，请点击**函数名称**链接。最初，结果会按**总时间**顺序显示。

## 探查详细信息报告

探查详细信息报告可显示在探查期间 MATLAB 调用的函数的探查结果。

要打开探查详细信息报告，请点击探查摘要报告中的函数名。要从探查详细信息报告返回到探查摘要报告，请点击探查窗口工具栏中的 。

探查详细信息报告标题中包含此信息。

- 探查函数的名称
- 父函数调用所探查的函数的次数
- 所探查函数花费的时间
- 在默认编辑器中打开函数的链接
- 将报告复制到单独窗口的链接。保存报告副本有助于在您更改文件后，比较函数更改的影响。

要指定探查详细信息报告包括哪些部分，请选中报告顶部的复选框，然后点击刷新按钮。使用相应复选框从这些选项中进行选择。

显示选项	详细信息
<b>显示父函数</b>	显示有关父函数的信息，且包含指向其详细信息报告的链接。要打开父函数的探查详细信息报告，请点击该函数的名称。
<b>显示正在执行的代码行</b>	列出所探查函数中耗用最长处理时间的行。
<b>显示子函数</b>	列出所探查函数调用的所有函数。要打开子函数的探查详细信息报告，请点击该子函数的名称。
<b>显示代码分析器结果</b>	显示有关所探查函数的问题以及可能的改进信息。
<b>显示文件范围</b>	显示有关 MATLAB 在探查时所执行的函数中的代码行的统计信息。
<b>显示函数列表</b>	<p>显示函数的源代码，如果它是 MATLAB 代码文件。</p> <p>对于每行代码，<b>函数列表</b>都包含以下列：</p> <ul style="list-style-type: none"> <li>• 每行代码的执行时间</li> <li>• MATLAB 执行该代码行的次数</li> <li>• 行号</li> <li>• 函数的源代码。文本颜色指示以下内容：           <ul style="list-style-type: none"> <li>• 绿色 - 注释行</li> <li>• 黑色 - 已执行的代码行</li> <li>• 灰色 - 未执行的代码行</li> </ul> </li> </ul> <p>默认情况下，探查详细信息报告高亮显示执行时间最长的代码行。高亮显示的颜色越深，执行代码行所耗用的时间越长。要更改高亮显示条件，请使用颜色高亮显示代码下拉列表。</p>

## 另请参阅

[profile](#)

### **详细信息**

- “测量程序性能” (第 29-2 页)
- “提升性能的方法” (第 29-11 页)
- “使用探查器可确定代码覆盖率” (第 29-9 页)

## 使用探查器可确定代码覆盖率

当您对某文件运行探查器时，某些代码可能不运行，例如包含 `if` 语句的块。

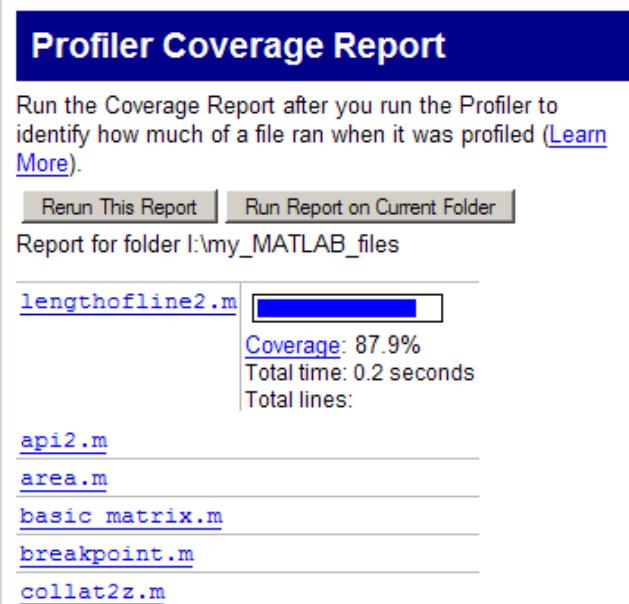
要确定探查文件时 MATLAB 执行了该文件的多大范围，请运行覆盖范围报告。

- 1 探查您的 MATLAB 代码文件。有关详细信息，请参阅“探查以提升性能”（第 29-4 页）或 `profile` 函数。
- 2 确保探查器当前未进行探查。
  - 在探查器中，如果探查器正在运行，则会显示**停止探查**按钮。如果探查器正在运行，可点击**停止探查**按钮。
  - 在命令提示符下，使用 `profile status` 检查探查器状态。如果 `ProfilerStatus` 为 'on'，则键入 `profile off` 可停止探查器。
- 3 使用当前文件夹浏览器导航到包含所探查代码文件的文件夹。

**注意** 您不能在路径为 UNC (通用命名约定) 路径时运行报告；即路径以 \\ 开头。应使用您系统上的实际硬盘驱动器，或映射的网络驱动器。

- 4 在当前文件夹浏览器上，点击 ，然后选择**报告 > 范围报告**。

将打开探查器覆盖率报告，其中提供有关所探查文件的覆盖率摘要。在下面的图像中，所探查的文件为 `lengthofline2.m`。



- 5 点击**范围**链接可查看该文件的探查详细信息报告。

**另请参阅**  
`profile`

### 详细信息

- “探查以提升性能”（第 29-4 页）

- “测量程序性能” (第 29-2 页)
- “提升性能的方法” (第 29-11 页)

# 提升性能的方法

本节内容
“环境” (第 29-11 页)
“代码结构” (第 29-11 页)
“针对性能的编程做法” (第 29-11 页)
“有关特定 MATLAB 函数的提示” (第 29-12 页)

要提升代码的性能，请考虑使用这些方法。

## 环境

请注意，共享计算资源的后台进程会降低 MATLAB 代码的性能。

## 代码结构

在组织您的代码时，请注意以下几点：

- 使用函数代替脚本。函数的速度通常更快。
- 优先使用局部函数，而不是嵌套函数。当函数不需要访问主函数中的变量时，尤其应当使用这种方法。
- 使用模块化编程。要避免产生大文件或者包含不常使用代码的文件，请将代码拆分为简单的综合函数。这种做法可降低首次运行的成本。

## 针对性能的编程做法

考虑使用以下编程做法可改进代码的性能。

- 预分配 - 您可以考虑预分配数组所需的最大空间量，而不用持续调整数组大小。有关详细信息，请参阅“[预分配](#)” (第 29-13 页)。
- 向量化 - 请考虑使用 MATLAB 矩阵和向量运算，而不是编写基于循环的代码。有关详细信息，请参阅“[向量化](#)” (第 29-15 页)。
- 将独立运算放在循环外 - 如果代码不使用每个 `for` 或 `while` 循环迭代进行不同计算，请将其移到循环外以避免冗余计算。
- 当数据类型更改时创建新变量 - 创建一个新变量，而不是将不同类型的数据分配给现有变量。更改现有变量的类或数组形状需要额外时间进行处理。
- 使用短路运算符 - 如果可能，请尽量使用短路逻辑运算符 `&&` 和 `||`。短路运算符更有效，因为仅当第一个操作数不能完全确定结果时，MATLAB 才会计算第二个操作数。有关详细信息，请参阅 [Logical Operators: Short Circuit](#)。
- 避免使用全局变量 - 尽量少使用全局变量是一种良好的编程做法，因为全局变量可能会降低 MATLAB 代码的性能。
- 避免重载内置函数 - 避免对任何标准 MATLAB 数据类重载内置函数。
- 避免使用“代码形式的数据” - 如果您有用于生成包含常量值的变量的大段代码（例如，超过 500 行），请考虑构造变量并将其保存在文件（例如 MAT 文件或 .csv 文件）中。然后，您可以加载变量而不是执行代码来生成这些变量。

## 有关特定 MATLAB 函数的提示

在编写性能关键代码时，请考虑有关特定 MATLAB 函数的以下提示。

- 请避免清除不是必须清除的代码。请不要以编程方式使用 `clear all`。有关详细信息，请参阅 `clear`。
- 请避免使用查询 MATLAB 状态的函数，例如 `inputname`、`which`、`whos`、`exist(var)` 和 `dbstack`。运行时自检会耗费大量计算资源。
- 请避免使用 `eval`、`evalc`、`evalin` 和 `feval(fname)` 等函数。尽可能使用到 `feval` 的函数句柄输入。从文本间接计算 MATLAB 表达式会耗费大量计算资源。
- 请尽可能避免以编程方式使用 `cd`、`addpath` 和 `rmpath`。在运行时更改 MATLAB 路径会导致重新编译代码。

## 另请参阅

### 详细信息

- “测量程序性能”（第 29-2 页）
- “探查以提升性能”（第 29-4 页）
- “预分配”（第 29-13 页）
- “向量化”（第 29-15 页）
- “图形性能”

## 预分配

每次经过 **for** 和 **while** 循环时，这些循环都会递增数据结构体的大小，这会对性能和内存的使用产生不利影响。反复重新调整数组大小往往需要 MATLAB 花费额外的时间来寻找更大的连续内存块，然后将数组移入这些块中。通常，您可以通过预分配数组所需的最大空间量来缩短代码的执行时间。

下面的代码显示了创建标量变量 **x**，然后在 **for** 循环中逐步增加 **x** 大小所需的时间量。

```
tic
x = 0;
for k = 2:1000000
    x(k) = x(k-1) + 5;
end
toc
```

Elapsed time is 0.301528 seconds.

如果您为 **x** 预分配一个  $1 \times 1,000,000$  的内存块并将其初始化为零，则代码的运行速度更快，这是因为无需反复为不断增长的数据结构体重新分配内存。

```
tic
x = zeros(1, 1000000);
for k = 2:1000000
    x(k) = x(k-1) + 5;
end
toc
```

Elapsed time is 0.011938 seconds.

对要初始化的数组类型使用适当的预分配函数：

- 对于数值数组，使用 **zeros**
- 对于字符数组，使用 **cell**

## 预分配非双精度类型的矩阵

当您预分配一个内存块来存储除 **double** 外的某类型的矩阵时，避免使用以下方法

```
A = int8(zeros(100));
```

该语句预分配了一个  $100 \times 100$  的 **int8** 矩阵，方法是先创建一个由 **double** 值组成的满矩阵，然后将每个元素转换为 **int8**。以 **int8** 值的形式创建数组可节省时间和内存。例如：

```
A = zeros(100, 'int8');
```

## 另请参阅

### 相关示例

- “重构和重新排列数组”
- “为元胞数组预分配内存”（第 12-15 页）
- “使用分类数组访问数据”（第 8-24 页）
- “预分配图形对象数组”

- “构造对象数组”

### 详细信息

- “提升性能的方法” (第 29-11 页)

# 向量化

## 本节内容

- “向量化的应用”（第 29-15 页）
- “数组运算”（第 29-16 页）
- “逻辑数组运算”（第 29-17 页）
- “矩阵运算”（第 29-18 页）
- “排序、设置和计数运算”（第 29-18 页）
- “向量化的常用函数”（第 29-19 页）

## 向量化的应用

MATLAB 针对涉及矩阵和向量的运算进行了优化。修正基于循环且面向标量的代码以使用 MATLAB 矩阵和向量运算的过程称为向量化。由于多种原因，代码的向量化是可取的：

- 外观：向量化的数学代码在外观上更像教科书中的数学表达式，从而使代码更便于理解。
- 不容易出错：由于不带循环，向量化的代码通常更短。代码行的减少意味着引入编程错误的机会也减少了。
- 性能：向量化代码的运行速度通常比相应的、包含循环的代码更快。

### 实现代码向量化以进行常规计算

以下代码计算 1,001 个从 0 到 10 之内的值的正弦：

```
i = 0;
for t = 0:.01:10
    i = i + 1;
    y(i) = sin(t);
end
```

这是同代码的向量化版本：

```
t = 0:.01:10;
y = sin(t);
```

第二个代码示例的执行速度通常比第一个快，是 MATLAB 的更高效的用法。通过创建包含所示代码的脚本测试您系统上的执行速度，然后使用 `tic` 和 `toc` 函数测量其执行时间。

### 针对特定任务实行代码向量化

此代码计算某向量每五个元素的累加和：

```
x = 1:10000;
ylength = (length(x) - mod(length(x),5))/5;
y(1:ylength) = 0;
for n= 5:5:length(x)
    y(n/5) = sum(x(1:n));
end
```

使用向量化，您可以编写一个更精确的 MATLAB 过程。以下代码显示了完成此项任务的一种方法：

```
x = 1:10000;
xsums = cumsum(x);
y = xsums(5:5:length(x));
```

## 数组运算

数组运算符对数据集中的所有元素执行相同的运算。这些类型的运算用于重复计算。例如，假设您通过记录各圆锥体的直径 (D) 和高度 (H) 来收集这些圆锥体的体积 (V)。如果您只收集一个圆锥体的信息，则可以只计算该圆锥体的体积：

```
V = 1/12*pi*(D.^2).*H;
```

现在，收集 10,000 个圆锥体的信息。向量 D 和 H 均包含 10,000 个元素，并且您需要计算 10,000 个体积。在绝大多数编程语言中，您需要设置类似于以下 MATLAB 代码的循环：

```
for n = 1:10000
    V(n) = 1/12*pi*(D(n).^2).*H(n));
end
```

借助 MATLAB，您可以使用与标量情形类似的语言对每个向量元素执行计算：

```
% Vectorized Calculation
V = 1/12*pi*(D.^2).*H;
```

---

**注意** 在运算符 \*、/ 和 ^ 之前放置句点 (.)，将其转换为数组运算符。

---

借助数组运算符，您还能够组合不同维度的矩阵。这样自动扩展大小为 1 的维度有助于对网格创建、矩阵和向量运算等进行向量化处理。

假设矩阵 A 代表考试分数，各种行表示不同的班级。您需要计算每个班级的平均分数与各分数的差。使用循环，该运算如下所示：

```
A = [97 89 84; 95 82 92; 64 80 99; 76 77 67; ...
88 59 74; 78 66 87; 55 93 85];
```

```
mA = mean(A);
B = zeros(size(A));
for n = 1:size(A,2)
    B(:,n) = A(:,n) - mA(n);
end
```

要执行该运算，一种更直接的方式是使用 `A - mean(A)`，后者不需要使用循环并且速度明显更快。

```
devA = A - mean(A)
```

```
devA =
```

```
18   11   0
16    4   8
-15    2  15
-3   -1  -17
 9  -19  -10
-1  -12    3
-24   15    1
```

即使  $A$  是一个  $7 \times 3$  矩阵， $\text{mean}(A)$  是一个  $1 \times 3$  向量，MATLAB 也会隐式扩展该向量，就好像其大小与矩阵相同一样，并且该运算将作为正常的按元素减法运算来执行。

根据操作数的大小要求，对于每个维度，各数组必须大小相同，或者其中一个为 1。如果满足该要求，则含有大小为 1 的数组的维度将扩展为其他数组中相应维度的大小。有关详细信息，请参阅“基本运算的兼容数组大小”（第 2-23 页）。

并且，在您处理多维数据时，隐式扩展也有助于向量化操作。假设您要计算包含两个变量  $x$  和  $y$  的函数  $F$ 。

$$F(x,y) = x \cdot \exp(-x^2 - y^2)$$

要在  $x$  和  $y$  向量中的每个点组合处计算该函数，您需要定义网格值。对于此任务，您应避免使用循环来循环访问点组合。在这种情况下，如果一个向量为列，而另一个向量为行，则当这些向量与数组运算符配合使用（例如  $x+y$  或  $x-y$ ）时，MATLAB 将会自动构造网格。在此示例中， $x$  是一个  $21 \times 1$  向量， $y$  是一个  $1 \times 16$  向量，因此该运算会通过扩展  $x$  的第二个维度和  $y$  的第一个维度来生成一个  $21 \times 16$  矩阵。

```
x = (-2:0.2:2)'; % 21-by-1
y = -1.5:0.2:1.5; % 1-by-16
F = x.*exp(-x.^2-y.^2); % 21-by-16
```

如果您要显式创建网格，可以使用 `meshgrid` 和 `ndgrid` 函数。

## 逻辑数组运算

批量处理数组的逻辑扩展是实现比较的向量化并制定决策。MATLAB 比较器接受向量输入并返回向量输出。

例如，假设在从 10,000 个圆锥体中收集数据时，您记录多个负值作为直径的值。您可以使用  $\geq$  运算符确定向量中的哪些值是有效的：

```
D = [-0.2 1.0 1.5 3.0 -1.0 4.2 3.14];
D >= 0
```

```
ans =
```

```
0 1 1 1 0 1 1
```

您可以直接利用 MATLAB 的逻辑索引功能来选择有效的圆锥体体积  $V_{\text{good}}$ ，其对应的  $D$  元素是非负的：

```
Vgood = V(D >= 0);
```

MATLAB 允许您分别使用 `all` 和 `any` 函数对整个向量的元素执行逻辑 AND 或 OR 运算。如果  $D$  的所有值都小于零，则会引发以下警告：

```
if all(D < 0)
    warning('All values of diameter are negative.')
    return
end
```

MATLAB 也可以比较两个大小兼容的向量，并允许您施加更多限制。此代码查找  $V$  为非负且  $D$  大于  $H$  的所有值：

```
V((V >= 0) & (D > H))
```

生成的向量与输入的大小相等。

为便于比较，MATLAB 包含特殊值来表示溢出、下溢和未定义的运算符，例如 **Inf** 和 **NaN**。逻辑运算符 **isinf** 和 **isnan** 有助于针对这些特殊值执行逻辑测试。例如，往往有助于从计算中排除 **NaN** 值：

```
x = [2 -1 0 3 NaN 2 NaN 11 4 Inf];
xvalid = x(~isnan(x))

xvalid =
    2   -1   0   3   2   11   4   Inf
```

---

**注意** **Inf == Inf** 返回 **true**；但 **NaN == NaN** 始终返回 **false**。

---

## 矩阵运算

在使代码向量化时，您通常需要构造一个具有特定大小或结构的矩阵。有多种方式可用来创建均匀矩阵。例如，您可能需要一个包含相等元素的  $5 \times 5$  矩阵：

```
A = ones(5,5)*10;
```

或者，您可能需要一个包含重复值的矩阵：

```
v = 1:5;
A = repmat(v,3,1)
```

```
A =
```

```
1   2   3   4   5
1   2   3   4   5
1   2   3   4   5
```

函数 **repmat** 可以灵活地根据较小的矩阵或向量来构建矩阵。**repmat** 通过重复输入矩阵来创建矩阵：

```
A = repmat(1:3,5,2)
B = repmat([1 2; 3 4],2,2)
```

```
A =
```

```
1   2   3   1   2   3
1   2   3   1   2   3
1   2   3   1   2   3
1   2   3   1   2   3
1   2   3   1   2   3
```

```
B =
```

```
1   2   1   2
3   4   3   4
1   2   1   2
3   4   3   4
```

## 排序、设置和计数运算

在许多应用程序中，对向量的某个元素执行的计算取决于同一向量中的其他元素。例如，向量 **x** 可能表示一个集。如何循环访问一个集而不使用 **for** 或 **while** 循环并不明显。如果采用向量化代码，则此过程变得更加清晰而且语法也不再过于冗长。

## 消除冗余元素

有许多不同的方法可以用来查找向量的冗余元素。一种方法涉及到函数 **diff**。在对向量元素进行排序后，当您对该向量使用 **diff** 函数时，相等的邻接元素会产生零项。因为 **diff(x)** 生成的向量的元素数比 **x** 少一个，所以您必须添加一个不等于该集中的任何其他元素的元素。NaN 始终满足此条件。最后，您可以使用逻辑索引来选择该集中的唯一元素：

```
x = [2 1 2 2 3 1 3 2 1 3];
x = sort(x);
difference = diff([x,NaN]);
y = x(difference~=0)
```

```
y =
1 2 3
```

再者，也可以使用 **unique** 函数完成同样的操作：

```
y=unique(x);
```

但是，**unique** 函数可能提供超出需要的功能并且降低代码的执行速度。如果您要衡量每个代码节的性能，请使用 **tic** 和 **toc** 函数。

## 计算向量中的元素数

您可以计算某元素在向量中的出现次数，而不只是返回 **x** 的集或子集。该向量排序后，您可以使用 **find** 函数来确定 **diff(x)** 中零值的索引并显示元素的值在何处发生了改变。**find** 函数中后续索引之间的差可指明特定元素的出现次数：

```
x = [2 1 2 2 3 1 3 2 1 3];
x = sort(x);
difference = diff([x,max(x)+1]);
count = diff(find([1,difference]));
y = x(find(difference))
```

```
count =
3 4 3
```

```
y =
1 2 3
```

**find** 函数不返回 NaN 元素的索引。您可以使用 **isnan** 和 **isinf** 函数计算 NaN 和 Inf 值的数目。

```
count_nans = sum(isnan(x(:)));
count_infs = sum(isinf(x(:));
```

## 向量化的常用函数

函数	说明
<b>all</b>	确定所有数组元素均为非零元素还是 true
<b>any</b>	确定任何数组元素是否为非零
<b>cumsum</b>	累积和

函数	说明
<b>diff</b>	差分和近似导数
<b>find</b>	查找非零元素的索引和值
<b>ind2sub</b>	线性索引的下标
<b>ipermute</b>	N 维数组的逆置换维度
<b>logical</b>	将数值转换为逻辑值
<b>meshgrid</b>	二维和三维空间中的矩形网格
<b>ndgrid</b>	N 维空间中的矩形网格
<b>permute</b>	重新排列 N 维数组的维度
<b>prod</b>	数组元素的乘积
<b>repmat</b>	重复数组副本
<b>reshape</b>	重构数组
<b>shiftdim</b>	移动维度
<b>sort</b>	对数组元素排序
<b>squeeze</b>	删除单一维度
<b>sub2ind</b>	将下标转换为线性索引
<b>sum</b>	数组元素总和

## 另请参阅

### 详细信息

- “数组索引”
- “提升性能的方法” (第 29-11 页)
- “数组与矩阵运算” (第 2-19 页)

### 外部网站

- MathWorks 新闻稿：MATLAB 中的矩阵索引

# 内存使用率

---

- “高效使用内存的策略” (第 30-2 页)
- “解决“内存不足”错误” (第 30-5 页)
- “MATLAB 如何分配内存” (第 30-9 页)
- “避免不必要的数据副本” (第 30-15 页)

## 高效使用内存的策略

本主题介绍在 MATLAB 中高效使用内存的几种方法。

### 使用适当的数据存储

MATLAB 提供了不同大小的数据类（例如 `double` 和 `uint8`），因此您无需使用大型类存储较小的数据段。例如，与使用 `double` 相比，使用 `uint8` 类存储 1,000 个无符号小整数值所用的内存少 7 KB。

#### 使用相应的数值类

您应在 MATLAB 中使用的数值类取决于您的预期操作。默认类 `double` 可提供最佳精度，但存储每个元素需要 8 字节内存。如果您计划执行复杂的数学运算（例如线性代数），则您必须使用浮点类，例如 `double` 或 `single`。`single` 类只需要 4 个字节。可使用 `single` 类执行的操作存在某些限制，但多数 MATLAB 数学运算都受支持。

如果您只需执行简单的算术运算并将原始数据表示为整数，则您可以在 MATLAB 中使用整数类。下面是数值类、内存要求（以字节为单位）及支持的运算的列表。

类（数据类型）	字节	支持的运算
<code>single</code>	4	绝大多数的数学运算
<code>double</code>	8	所有数学运算
<code>logical</code>	1	逻辑/条件运算
<code>int8, uint8</code>	1	算术和某些简单函数
<code>int16, uint16</code>	2	算术和某些简单函数
<code>int32, uint32</code>	4	算术和某些简单函数
<code>int64, int64</code>	8	算术和某些简单函数

#### 减少存储数据时的开销量

MATLAB 数组（在内部作为 `mxArrays` 实现）需要一定的空间来将有关数据的元数据信息（例如类型、维度和属性）存储在内存中。每个数组大约需要 104 字节。仅当有大量（如数百或数千）较小的 `mxArrays`（如标量）时，此开销才成问题。`whos` 命令列出了变量所用的内存，但不包括此开销。

因为简单数值数组（包括一个 `mxArray`）的开销最少，所以您应该尽可能使用它们。当数据太复杂而无法存储在简单数组（或矩阵）中时，您可以使用其他数据结构体。

元胞数组由每个元素的单独 `mxArrays` 组成。因此，包含许多小元素的元胞数组具有较大的开销。

结构体的每个字段需要类似的开销量（请参阅“数组标头”（第 30-10 页））。包含许多字段和较少内容的结构体具有较大的开销，因此应避免使用这样的结构体。由包含数值标量字段的结构体组成的大型数组所需的内存要比具有包含较大数据值数组的字段的结构体更多。

另请注意，虽然 MATLAB 将数值数组存储在连续内存中，但结构体和元胞数组则不然。

#### 将数据导入相应的 MATLAB 类

当使用 `fread` 读取二进制文件中的数据时，常见的错误是，仅指定该文件中数据的类，而不指定 MATLAB 当文件位于工作区中时使用的数据的类。因此，即使您只读取 8 位值，使用的也是默认的 `double`。例如，

```

fid = fopen('large_file_of_uint8s.bin', 'r');
a = fread(fid, 1e3, 'uint8');           % Requires 8k
whos a
  Name      Size      Bytes Class Attributes
  a    1000x1        8000  double

a = fread(fid, 1e3, 'uint8=>uint8');   % Requires 1k
whos a
  Name      Size      Bytes Class Attributes
  a    1000x1        1000  uint8

```

### 尽可能使数组稀疏

如果您的数据包含许多零，请考虑使用稀疏数组，这样仅存储非零元素。以下示例比较稀疏存储和满存储的要求：

```

A = eye(1000);      % Full matrix with ones on the diagonal
As = sparse(A);     % Sparse matrix with only nonzero elements
whos
  Name      Size      Bytes Class Attributes
  A    1000x1000    8000000  double
  As   1000x1000     24008  double  sparse

```

您可以看到，该数组只需要大约 4 KB 即可存储为稀疏数组，但要存储为满矩阵，则需要大约 8 MB。通常，对于包含 `nnz` 非零元素和 `ncol` 列的双精度型稀疏数组，所需的内存为：

- $16 * nnz + 8 * ncol + 8$  字节（在 64 位计算机上）

请注意，MATLAB 支持对稀疏数组执行大多数（但不是全部）数学运算。

## 避免临时性的数据副本

避免创建不必要的临时性数据副本，以显著减少所需的内存量。

### 避免创建临时数组

避免创建大型临时变量，并在不再需要这些临时变量时清除它们。例如，以下代码创建由零组成的、存储为临时变量 `A` 的数组，然后将 `A` 转换为单精度：

```

A = zeros(1e6,1);
As = single(A);

```

使用一个命令来执行两个操作可更高效地使用内存：

```
A = zeros(1e6,1,'single');
```

使用 `repmat` 函数、数组预分配和 `for` 循环是处理非双精度数据而不需要内存中的临时存储的其他方法。

### 使用嵌套函数减少传递的参数

处理大型数据集时，注意 MATLAB 会创建输入变量的临时副本（如果被调用函数修改其值）。这会暂时使存储数组所需的内存翻倍，从而导致 MATLAB 在没有足够内存时生成错误。

在此情形下使用较少的内存的一种方法是使用嵌套函数。嵌套函数共享所有外部函数的工作区，为嵌套函数提供对其通常范围之外的数据的访问权。在如下示例中，嵌套函数 `setrowval` 可直接访问外部函数

**myfun** 的工作区，从而无需在函数调用中传递变量副本。当 **setrowval** 修改 A 的值时，它在调用函数的工作区中修改它。无需使用额外内存为所调用函数存储一个单独数组，且无需返回 A 的修改后的值：

```
function myfun
A = magic(500);

    function setrowval(row, value)
        A(row,:) = value;
    end

    setrowval(400, 0);
    disp('The new value of A(399:401,1:10) is')
    A(399:401,1:10)
end
```

## 回收使用的内存

增加可用内存量的一种简单方法是清除您不再使用的大型数组。

### 定期将您的大型数据保存到磁盘

如果您的程序生成非常大量的数据，请考虑定期将数据写入磁盘。在保存该部分数据后，使用 **clear** 函数从内存中删除变量并继续生成数据。

### 从内存中清除不再需要的旧变量

当您重复或以交互方式处理非常大的数据集时，请首先清除旧变量以为新变量腾出空间。否则，MATLAB 需要等大小的临时存储才能覆盖此变量。例如，

```
a = rand(100e6,1)      % 800 MB array
b = rand(100e6,1)      % New 800 MB array
Error using rand
Out of memory.Type HELP MEMORY for your options.

clear a
a = rand(100e6,1)      % New 800 MB array
```

## 另请参阅

### 详细信息

- “解决“内存不足”错误”（第 30-5 页）

## 解决“内存不足”错误

本主题介绍在 MATLAB 面临内存不足问题时可以使用的几种策略。MATLAB 是在 64 位操作系统上运行的 64 位应用程序。每当它从操作系统请求的内存段大于可用内存时，它就会返回一个错误消息。

MATLAB 具有内置防护机制，可防止创建过大的数组。默认情况下，MATLAB 可使用 100% 的计算机 RAM（不包括虚拟内存）来为数组分配内存，如果数组超过该阈值，则 MATLAB 返回错误。例如，以下语句尝试创建大小不合理的数组：

```
A = rand(1e6,1e6);
```

```
Error using rand
Requested 1000000x1000000 (7450.6GB) array exceeds maximum array size
preference. Creation of arrays greater than this limit may take a long
time and cause MATLAB to become unresponsive. See array size limit or
preference panel for more information.
```

有关调整此数组大小限制的信息，请参阅“工作区和变量预设项”。如果关闭数组大小限制，则 MATLAB 返回另一种错误：

```
A = rand(1e6,1e6);
```

```
Out of memory. Type "help memory" for your options.
```

无论您由于何种原因遇到内存限制，都有几种解决方法供您选择，具体取决于您的目标。“高效使用内存的策略”（第 30-2 页）中讨论的方法可以帮助您优化可用内存，包括：

- “使用适当的数据存储”（第 30-2 页）
- “避免创建临时数组”（第 30-3 页）
- “回收使用的内存”（第 30-4 页）

如果您已在高效使用内存，但问题仍然存在，可以参考本页其余各节提供的解决办法。

## 利用 tall 数组

“使用 tall 数组处理无法放入内存的数据”旨在帮助您处理太大而无法放入内存的数据集。MATLAB 一次处理一小块数据，并在后台自动执行所有的数据分块和处理。利用 tall 数组の場合主要有两种：

- 1 如果大型数组可放入内存，但在尝试执行计算时内存不足，则您可以将该数组转换为 tall 数组：

```
B = tall(A)
```

如果大型数组可放入内存，但这些数组消耗了太多内存而无法在计算过程中容纳数据副本，则您可以使用这种方法进行处理。例如，如果您有 8GB 内存和一个 5GB 矩阵，将该矩阵转换为 tall 数组可让您在不耗尽内存的情况下对矩阵执行计算。请参阅“将可放入内存的数组转换为 tall 数组”中有关此用法的示例。

- 2 如果您有基于文件或文件夹的数据，您可以创建一个 **datastore**，然后基于该数据存储创建一个 tall 数组：

```
ds = datastore('path/to/data.csv');
tt = tall(ds);
```

这种方法可以让您充分利用 MATLAB 中 tall 数组的强大功能：数据可以有任意数量的行，且 MATLAB 不会耗尽内存。由于 **datastore** 同时支持本地和远程数据位置，因此您处理的数据不需要位于您用于分析这些数据的计算机上。有关详细信息，请参阅“处理远程数据”。

## 利用多台计算机的内存

如果您有计算机群集，您可以使用 Parallel Computing Toolbox 和 “Distributed Arrays” (Parallel Computing Toolbox) 利用群集中所有计算机的总内存来执行计算。这使您能够将整个分布式数组作为单个实体来对其执行运算。但是，工作进程仅对分配给它的部分数组执行运算，并视需要在彼此之间自动传输数据。

创建分布式数组与创建 tall 数组非常相似：

```
ds = datastore('path/to/data.csv');
dt = distributed(ds);
```

## 仅加载需要的数据

解决内存问题的另一种方法是仅将大型数据集中解决问题所需的数据导入到 MATLAB 中。从数据库等源中导入时这通常不是问题，因为您可以显式搜索匹配查询的元素。但这在加载大型简单文本或二进制文件时是个常见问题。

**datastore** 函数使您能够以增量方式处理大型数据集。此函数适用于 “使用 tall 数组处理无法放入内存的数据” 和 “Distributed Arrays” (Parallel Computing Toolbox)，但您也可以将其用于其他目的。在您需要一次将数据集的一小部分加载到内存中时，数据存储很有帮助。

要创建数据存储，您需要提供文件名，或包含一系列具有相似格式的文件的目录。例如，使用单个文件：

```
ds = datastore('path/to/file.csv')
```

或者，使用一个文件夹中的一系列文件：

```
ds = datastore('path/to/folder/')
```

您也可以使用通配符 \* 选择特定类型的所有文件，如：

```
ds = datastore('data/*.csv')
```

数据存储支持多种常见文件格式（表格数据、图像、电子表格等）。有关详细信息，请参阅 “Select Datastore for File Format or Application”。

除了数据存储，MATLAB 还提供了其他几个用于加载部分文件的函数，例如 **matfile** 可用于加载部分 MAT 文件。下表按文件类型总结了部分加载函数。

文件类型	部分加载
MAT 文件	通过对使用 <b>matfile</b> 函数创建的对象进行索引来加载部分变量。有关这种用法的示例，请参阅 MAT 文件中的大数据。
文本	使用 <b>textscan</b> 函数可通过仅读取选定的列和行来访问大型文本文件的一部分。如果您使用 <b>textscan</b> 指定行数或重复格式数字，MATLAB 会提前计算所需的确切内存量。
二进制	您可以使用低级别二进制文件 I/O 函数（例如 <b>fread</b> ）访问具有已知格式的任何文件的一部分。对于未知格式的二进制文件，请尝试通过 <b>memmapfile</b> 函数使用内存映射。

文件类型	部分加载
图像、HDF、音频和视频	许多 MATLAB 函数都支持从这些类型的文件中加载数据，使您可以选择读取部分数据。有关详细信息，请参阅“支持的导入和导出文件格式”中列出的函数参考页。

## 增加系统交换空间

可供您计算机上的应用程序使用的总内存由物理内存 (RAM) 外加磁盘上的页面文件（即或交换文件）组成。交换文件可能非常大（例如在 64 位 Windows 上为 512 TB）。操作系统将每个进程的虚拟内存分配给物理内存或交换文件，具体取决于系统和其他进程的需求。增加交换文件的大小可以增加总可用内存，但通常也会导致性能下降。

多数系统都允许您控制您的交换文件的大小。涉及的步骤取决于您的操作系统：

- Windows 系统 - 使用 Windows 控制面板可更改您系统上虚拟内存分页文件的大小。有关详细信息，请参阅 Windows 帮助。
- Linux® 系统 - 使用 **mkswap** 和 **swapon** 命令更改您的交换空间。有关详细信息，请在 Linux 提示符处键入 **man** 后跟命令名称。

没有用于直接控制 macOS 系统上的交换空间的接口。

## 设置 Linux 系统上的进程限制

进程限制是单个进程（或应用程序）可寻址的最大虚拟内存量。您只有在极特殊的情况下才需要设置此预设项，它必须足够大以容纳以下内容：

- 要处理的所有数据
- MATLAB 程序文件
- MATLAB 可执行文件自身
- 其他状态信息

64 位操作系统支持 8 TB 的进程限制。在 Linux 系统上，查找 **ulimit** 命令以查看和设置用户限制（包括虚拟内存）。

## 在 Linux 系统上禁用 Java VM

在 Linux 系统上，如果您不使用 Java JVM™ 启动 MATLAB，则可以将可用工作区内存增加大约 400 MB。要在不使用 Java JVM 的情况下启动 MATLAB，请使用命令行选项 **-nojvm**。此选项还会将最大连续内存块大小增加几乎同样的量。通过增加最大连续内存块，您可以增大矩阵大小上限。

使用 **-nojvm** 会带来某种损失，您将失去依赖 Java 软件的许多功能，包括整个开发环境。启动 MATLAB 时指定 **-nodesktop** 选项并不会节省大量内存。

## 另请参阅

[memory](#)

### 相关示例

- “高效使用内存的策略” (第 30-2 页)
- “大型文件和大数据”
- “处理远程数据”
- “Java 堆内存预设”

# MATLAB 如何分配内存

## 本节内容

- “为数组分配内存” (第 30-9 页)
- “数据结构体和内存” (第 30-11 页)

## 为数组分配内存

下面的主题提供了有关 MATLAB 软件如何在处理数组和变量时分配内存的信息。目的是帮助您在编写代码时更高效地使用内存。但在大多数时候，您不需要关注这些内部操作，因为 MATLAB 会自动为您处理数据的存储。

- “创建和修改数组” (第 30-9 页)
- “复制数组” (第 30-9 页)
- “数组标头” (第 30-10 页)
- “函数参数” (第 30-11 页)

---

**注意** 在以后的版本中，任何有关 MATLAB 软件如何在内部处理数据的信息都可能会变更。

---

### 创建和修改数组

当您将数值或字符数组分配给变量时，MATLAB 会分配一个连续的虚拟内存块并将数组数据存储在该块中。MATLAB 还将有关数组数据的信息（例如其类和维度）存储在一个名为标头的单独的小内存块中。

如果您向现有数组中添加新元素，MATLAB 会按照使内存存储保持连续的方式扩展现有数组。这通常需要查找新的大小足以容下扩展后的数组的内存块。MATLAB 随后将该数组的内容从其原始位置复制到内存中这一新块中，向该块中的数组添加新元素，并释放原始数组在内存中的位置。

如果您从现有数组中删除元素，MATLAB 会通过清除已删除的元素，然后使其在原始内存位置变得紧凑来使内存存储连续。

### 使用大数据集

如果您要使用大数据集，在增加数组大小时应小心以避免内存不足造成的错误。如果您扩展数组使其超过其原始位置的可用连续内存，MATLAB 必须创建该数组的副本并将该副本设置为新值。在此操作期间，内存中有原始数组的两个副本。这会暂时使数组所需的内存量翻倍，并增加您的程序在执行期间出现内存不足的风险。最好在开始时为可能最大的数组预分配足够的内存。请参阅“预分配” (第 29-13 页)。

### 复制数组

多个变量可能在内部指向同一数据块，从而共享该数组的值。当您将某变量复制到另一个变量（如 `B = A`）时，MATLAB 会创建数组引用而非数组本身的副本。只要您不修改数组的内容，就不需要存储其多个副本。如果您修改数组的任何元素，MATLAB 会创建该数组的副本，然后修改该副本。

此示例使用 `memory` 函数演示 MATLAB 如何处理复制数组。`memory` 仅在 Windows 系统上可用。

首先创建一个简单脚本 `memUsed.m`，以显示您的 MATLAB 进程使用了多少内存。将这两行代码放在脚本中。

```
[usr, sys] = memory;
usr.MemUsedMATLAB
```

初步了解您的 MATLAB 进程使用了多少内存：

```
format short eng;
memUsed
ans =
295.4977e+006
```

创建一个  $2000 \times 2000$  数值数组 A。这大约要使用 32MB 的内存：

```
A = magic(2000);
memUsed
ans =
327.6349e+006
```

在 B 中创建 A 的副本。由于不需要使用数组数据的两个副本，MATLAB 仅创建数组引用的一个副本。这不需要额外增加大量内存：

```
B = A;
memUsed
ans =
327.6349e+006
```

现在通过将 B 设置为原始大小的一半（即将 1000 行设置为空）对其进行修改。这需要 MATLAB 为 A 数组的至少前 1000 行创建一个副本，然后将该副本分配给 B：

```
B(1001:2000,:) = [];
format short; size(B)
ans =
1000      2000
```

再次检查使用的内存。即使 B 明显小于其原始大小，MATLAB 进程使用的内存量也增加了大约 16 MB（A 最初所需 32 MB 的 1/2），这是因为 B 不能再只作为对 A 的引用：

```
format short eng; memUsed
ans =
343.6421e+006
```

### 数组标头

当您将数组分配给变量时，MATLAB 还将有关数组的信息（例如类和维度）存储在一个称为标头的单独内存块中。对于多数数组，存储标头所需的内存可忽略不计。将大数据集存储在数量较少的大数组中比存储在数量较大的小数组中更理想。这是因为前者的配置需要的数组标头更少。

### 结构体和元胞数组

对于结构体和元胞数组，MATLAB 不仅为每个数组创建一个标头，还为结构体的每个字段及元胞数组的每个元胞创建一个标头。因此，存储结构体或元胞数组所需的内存量不仅取决于其包含的数据量，还取决于其构造方式。

例如，以一个包含字段 R、G 和 B 的标量结构体数组 S1 为例。每个大小为  $100 \times 50$  的字段都需要一个数组标头来描述总体结构，一个标头用于每个唯一字段名称，以及一个标头用于  $1 \times 1$  结构体数组的每个字段。这使整个暑假结构体总共有七个数组标头：

```
S1.R(1:100,1:50)
S1.G(1:100,1:50)
S1.B(1:100,1:50)
```

另一方面，采用一个  $100 \times 50$  结构体数组 S2，其中每个元素都有标量字段 R、G 和 B。本例中，您需要一个数组标头来描述总体结构，一个标头用于每个唯一字段名称，以及一个标头用于结构体的 5,000 个元素中的每个元素的字段，从而使整个数据结构体总共有 15,004 个数组标头：

```
S2(1:100,1:50).R
S2(1:100,1:50).G
S2(1:100,1:50).B
```

即使 S1 和 S2 包含相同数量的数据，S1 使用的内存空间也大大减少。不仅需要的内存更少，使用 S1 格式也具有速度优势。

请参阅“[数据结构体和内存](#)”（第 30-11 页）下的“元胞数组”和“结构体”。

#### **whos 函数报告的内存使用率**

**whos** 函数显示任何变量消耗的内存量。为简单起见，**whos** 仅报告用于存储实际数据的内存。例如，它不报告用于数组标头的存储。

#### **函数参数**

MATLAB 以类似方式处理传入函数调用的参数。将变量传递给函数时，您实际是传递对该变量表示的数据的引用。只要被调用的函数没有修改输入数据，调用函数中的变量和被调用函数中的变量就指向内存中的同一位置。如果被调用函数修改输入数据的值，则 MATLAB 将在内存中的新位置创建原始数组的副本，用修改后的值更新该副本，并将被调用函数中的输入变量指向此新数组。

在下面的示例中，函数 **myfun** 修改传递给其的数组值。MATLAB 在 A 指向的数组的内存中创建副本，将变量 X 设置为对此新数组的引用，然后将一行 X 设置为零。A 引用的数组保持不变：

```
A = magic(500);
myfun(A);

function myfun(X)
X(400,:) = 0;
```

如果调用函数需要其传递给 **myfun** 的数组的修改后的值，您需要以被调用函数的输出形式返回更新的数据，如此处为变量 A 显示的一样：

```
A = magic(500);
A = myfun(A);
sprintf('The new value of A is %d', A)

function Y = myfun(X)
X(400,:) = 0;
Y = X;
```

## **数据结构体和内存**

不同类型的 MATLAB 数据结构体的内存要求有所不同。通过考虑 MATLAB 存储这些结构体的方式，可以减少用于这些结构体的内存量。

#### **数值数组**

MATLAB 需要 1、2、4 或 8 个字节存储 8 位、16 位、32 位和 64 位有符号和无符号整数。对于浮点数，MATLAB 对 **single** 和 **double** 类型使用 4 或 8 个字节。要在使用数值数组时节省内存，MathWorks 建议您使用包含您的数据而不溢出的最小整数或浮点类型。有关详细信息，请参阅“[数值类型](#)”。

## 复数数组

MATLAB 将复数数据存储为单独的实部和虚部。如果您创建复数数组变量的副本，然后仅修改该数组的实部或虚部，MATLAB 会创建一个同时包含实部和虚部的数组。

## 稀疏矩阵

最好以稀疏格式存储所包含的值大多数为零的矩阵。稀疏矩阵占用的内存更少，并且可能还比满矩阵的操作速度更快。可以使用 `sparse` 函数将满矩阵转换为稀疏格式。

比较两个  $1000 \times 1000$  矩阵：`X` 是一个其元素中有  $2/3$  等于零的双精度类型的矩阵；`Y` 是 `X` 的稀疏副本。下面的示例说明稀疏矩阵需要大约一半的内存：

```
whos
  Name      Size            Bytes  Class
  X    1000x1000        8000000  double array
  Y    1000x1000        4004000  double array (sparse)
```

## 元胞数组

除了数据存储，元胞数组还需要特定量的额外内存来存储描述每个元胞的信息。这些信息记录在标头中，每个元胞数组有一个标头。您可以通过查找不含任何数据的  $1 \times 1$  元胞使用的字节数，来确定元胞数组标头所需的内存量，如下针对 32 位系统的示例所示：

```
A = {[ ];    % Empty cell array

whos A
  Name      Size            Bytes  Class  Attributes
  A    1x1              60  cell
```

本例中，MATLAB 显示 32 位系统上元胞数组中每个标头所需的字节数为 60。本部分中的所有 32 位示例使用该标头大小。对于 64 位系统，本文档中的标头大小假定为 112 个字节。可使用刚刚所示的针对 32 位的方法计算 64 位系统上的正确标头大小。

要预测整个元胞数组的大小，请将您获得的标头数值乘以该数组中的元胞总数，然后加上您打算存储在该数组中的数据所需的字节数：

`(header_size * number_of_cells) + data`

因此，一个包含 400 字节数据的  $10 \times 20$  元胞数组在 64 位系统上将需要 22,800 字节的内存：

`(112 x 200) + 400 = 22800`

---

**注意** 尽管数值数组必须存储在连续内存中，但结构体和元胞数组则不然。

---

### 示例 1 - 元胞数组的内存分配

下面的  $4 \times 1$  元胞数组记录三台笔记本电脑的品牌名称、屏幕大小、价格和促销状态：

```
Laptops = {'SuperrFast 89X', 'ReliablePlus G5', ...
'UCanA4dIt 140L6'; ...
[single(17), single(15.4), single(14.1)]; ...
[2499.99, 1199.99, 499.99]; ...
[true, true, false];}
```

在 32 位系统上，该元胞数组标头本身需要 60 个字节/元胞：

**4 cells \* 60 bytes per cell = 240 bytes for the cell array**

计算包含所有四个元胞中的数据所需的内存：

**45 characters \* 2 bytes per char = 90 bytes  
3 doubles \* 8 bytes per double = 24 bytes  
3 singles \* 4 bytes per single = 12 bytes  
3 logicals \* 1 byte per logical = 3 bytes**

**90 + 24 + 12 + 3 = 129 bytes for the data**

求二者之和，然后将您的结果与 MATLAB 返回的大小进行比较：

**240 + 129 = 369 bytes total**

```
whos Laptops
  Name      Size      Bytes Class Attributes
Laptops    4x1          369  cell
```

## 结构体

```
S.A = [];
B = whos('S');
B.bytes - 60
ans =
  64
```

按如下所示计算结构体数组所需的内存：

**32-bit systems: fields x ((60 x array elements) + 64) + data  
64-bit systems: fields x ((112 x array elements) + 64) + data**

在 64 位计算机系统上，一个包含字段 Address 和 Phone 的  $4 \times 5$  结构体 Clients 本身使用 4,608 个字节：

**2 fields x ((112 x 20) + 64) = 2 x (2240 + 64) = 4608 bytes**

要计算总和，您必须加上保存分配给每个字段的数据所需的内存。如果将一个包含 25 个字符的向量和一个包含 12 个字符的向量分别分配给  $4 \times 5$  Clients 数组的每个元素中的 Address 和 Phone，将使用 1480 个字节存储数据：

**(25+12) characters \* 2 bytes per char \* 20 elements = 1480 bytes**

求二者之和，您会发现整个结构体占用 6,088 字节的内存。

## 示例 1 - 结构体数组的内存分配

计算在 32 位系统上存储包含以下四个字段的  $6 \times 5$  结构体数组所需的内存量：

A: 5-by-8-by-6 signed 8-bit integer array  
B: 1-by-500 single array  
C: 30-by-30 unsigned 16-bit integer array  
D: 1-by-27 character array

构造数组：

```
A = int8(ones(5,8,6));
B = single(1:500);
C = uint16(magic(30));
D = 'Company Name: MathWorks';

s = struct('f1', A, 'f2', B, 'f3', C, 'f4', D);

for m=1:6
    for n=1:5
        s(m,n)=s(1,1);
    end
end
```

计算结构体本身所需的内存量，然后计算其包含的数据所需的内存量：

```
structure = fields x ((60 x array elements) + 64) =
            4 x ((60 x 30) + 64) = 7,456 bytes

data = (field1 + field2 + field3 + field4) x array elements =
            (240 + 2000 + 1800 + 54) x 30 = 122,820 bytes
```

求二者之和，然后将您的结果与 MATLAB 返回的大小进行比较：

```
Total bytes calculated for structure s: 7,456 + 122,820 = 130,276
```

```
whos s
  Name      Size      Bytes  Class     Attributes
  s         6x5      130036  struct
```

# 避免不必要的数据副本

## 本节内容

- “将值传递给函数”（第 30-15 页）
- “为什么要使用传值语义”（第 30-16 页）
- “句柄对象”（第 30-16 页）

## 将值传递给函数

当使用输入参数调用函数时，MATLAB 会将值从调用函数的工作区复制到被调用函数的参数变量中。但 MATLAB 会应用各种技术，避免在不需要时复制这些值。

MATLAB 没有像 C++ 之类语言那样提供定义值引用的方法。但 MATLAB 允许多个输出和多个输入参数，因此您知道有哪些值要输入到函数中，以及要从函数中输出哪些值。

### 写入时复制

如果函数未修改输入参数，则 MATLAB 不会复制输入变量中包含的值。

例如，假设您将一个大型数组传递给函数。

```
A = rand(1e7,1);
B = f1(A);
```

函数 `f1` 将输入数组 `X` 中的每个元素乘以 1.1，并将结果赋给变量 `Y`。

```
function Y = f1(X)
    Y = X.*1.1; % X is a shared copy of A
end
```

由于此函数并未修改输入值，因此局部变量 `X` 和调用方工作区中的变量 `A` 共享数据。执行 `f1` 后，赋给 `A` 的值不变。调用方工作区中的变量 `B` 包含按元素相乘的结果。输入通过值进行传递。但是，调用 `f1` 时没有制作副本。

函数 `f2` 确实会修改输入变量的本地副本，从而导致本地副本与输入 `A` 不共享。现在，函数中的 `X` 值在调用方工作区中是输入变量 `A` 的独立副本。当 `f2` 将结果返回到调用方的工作区时，局部变量 `X` 将被销毁。

```
A = rand(1e7,1);
B = f2(A);

function Y = f2(X)
    X = X.*1.1; % X is an independent copy of A
    Y = X;       % Y is a shared copy of X
end
```

### 以 MATLAB 表达式方式传递输入

您可以将一个函数的返回值用作另一个函数的输入参数。例如，使用 `rand` 函数直接为函数 `f2` 创建输入。

```
B = f2(rand(1e7,1));
```

保存 `rand` 返回值的唯一变量是函数 `f2` 的工作区中的临时变量 `X`。在调用方的工作区中，不存在这些值的共享副本或独立副本。直接传递函数输出可以节省在被调用函数中创建输入值副本所需的时间和内存。当输入值不会再次使用时，可以使用此方法。

## 就地赋值

当您不需要保留原始输入值时，可以将函数的输出赋给与输入相同的变量。

```
A = f2(A);
```

重新赋给相同的变量名称发生在前面介绍的写入时复制行为之后：修改输入变量值会生成这些值的临时副本。但是，MATLAB 在某些条件下可以应用内存优化。

请参考以下示例。**memoryOptimization** 函数在变量 A 中生成一个很大的随机数数组。然后它调用局部函数 **fLocal**，传递 A 作为输入，并将局部函数的输出赋给相同的变量名称。

由于对局部函数的调用 A = fLocal(A) 将输出赋给变量 A，因此 MATLAB 在执行函数的过程中不需要保留 A 的原始值。在 **fLocal** 内，输入 X 具有原来由 A 保存的值的唯一副本。

因此，对 **fLocal** 内的 X 所做的修改不会产生数据副本。赋值 X = X.\*1.1 只是原地修改 X，不会为乘法结果分配新数组。

```
function memoryOptimization
A = rand(1e7,1);
A = fLocal(A);
end
function X = fLocal(X)
X = X.*1.1;
end
```

消除局部函数中的副本可以节省内存并提高大型数组的执行速度。但是，在函数引发错误时可能会用到变量的情况下，MATLAB 不能应用内存优化。因此，这种优化不适用于脚本、命令行、对 eval 的调用以及 try/catch 模块内的代码。

此外，当执行被调用函数过程中可以直接访问原始变量时，MATLAB 也不会应用内存优化。例如，如果 **fLocal** 是嵌套函数，则 MATLAB 无法应用优化，因为它会与父函数共享变量。

最后，当指定的变量声明为全局变量或持久变量时，MATLAB 也不会应用内存优化。

## 为什么要使用传值语义

MATLAB 在向函数传递参数以及从函数返回值时使用传值语义。在某些情况下，传值会在被调用函数中生成原始值的副本。但是，传值语义也有一些好处。

当调用函数时，您知道输入变量不会在调用方工作区中被修改。因此，不需要只是为了防止这些值可能被修改而在函数内或在调用位置创建输入副本。只有赋给返回值的变量会被修改。

此外，如果通过引用传递变量的函数中发生错误，可以避免出现损坏工作区变量的可能性。

## 句柄对象

有一些特殊的对象称为句柄。保存同一个句柄的副本的所有变量都可以访问和修改同一个底层对象。在特定的情况下，即当对象表示物理对象（例如窗口、绘图、设备或人）而不是数学对象（如数字或矩阵）时，句柄对象很有用。

句柄对象从 **handle** 类派生而来，该类提供事件和侦听程序、析构函数方法以及动态属性支持等功能。

有关值和句柄的详细信息，请参阅“[句柄类和值类的比较](#)”和“[Which Kind of Class to Use](#)”。

## 另请参阅

handle

## 相关示例

- “句柄对象行为”
- “Avoid Copies of Arrays in MEX Functions”
- “高效使用内存的策略”（第 30-2 页）



# 自定义帮助和文档

---

- “为类创建帮助” (第 31-2 页)
- “检查哪些程序具有帮助” (第 31-8 页)
- “创建帮助摘要文件 - Contents.m” (第 31-10 页)
- “自定义代码建议和自动填充” (第 31-12 页)
- “显示自定义文档” (第 31-20 页)
- “显示自定义示例” (第 31-26 页)

## 为类创建帮助

### 本节内容

“`doc` 命令显示的帮助文本”（第 31-2 页）

“自定义帮助文本”（第 31-3 页）

### doc 命令显示的帮助文本

当您使用 `doc` 命令显示类的帮助时，MATLAB 会自动显示其从类定义获得的信息。

例如，创建一个名为 `someClass.m` 的具有多个属性和方法的类定义文件，如下所示。

```
classdef someClass
    % someClass Summary of this class goes here
    % Detailed explanation goes here

    properties
        One    % First public property
        Two    % Second public property
    end
    properties (Access=private)
        Three  % Do not show this property
    end

    methods
        function obj = someClass
            % Summary of constructor
        end
        function myMethod(obj)
            % Summary of myMethod
            disp(obj)
        end
    end
    methods (Static)
        function myStaticMethod
            % Summary of myStaticMethod
        end
    end
end
```

使用 `doc` 命令查看类定义中的帮助文本和详细信息。

```
doc someClass
```

MATLAB File Help: someClass    [View code for someClass](#)    [Default Topics](#)

## someClass

`someClass` Summary of this class goes here  
Detailed explanation goes here

### Class Details

Sealed	false
Construction load	false

### Constructor Summary

[someClass](#) Summary of constructor

### Property Summary

<a href="#">One</a>	First public property
<a href="#">Two</a>	Second public property

### Method Summary

<a href="#">myMethod</a>	Summary of myMethod
<a href="#">myStaticMethod</a>	Summary of myStaticMethod

## 自定义帮助文本

您可以添加有关 `doc` 命令和 `help` 命令均包括在其显示中的类的信息。`doc` 命令在生成的 HTML 页的顶部、从类定义获得的信息之上显示帮助文本。`help` 命令在命令行窗口中显示帮助文本。有关详细信息，请参阅：

- “类” (第 31-3 页)
- “方法” (第 31-4 页)
- “属性” (第 31-5 页)
- “枚举” (第 31-5 页)
- “事件” (第 31-6 页)

### 类

通过在紧随文件中 `classdef` 语句之后的行上包括注释来为类创建帮助文本。例如，创建一个名为 `myClass.m` 的文件，如下所示。

```
classdef myClass
% myClass Summary of myClass
% This is the first line of the description of myClass.
% Descriptions can include multiple lines of text.
%
% myClass Properties:
%   a - Description of a
%   b - Description of b
%
% myClass Methods:
%   doThis - Description of doThis
```

```
% doThat - Description of doThat

properties
    a
    b
end

methods
    function obj = myClass
    end
    function doThis(obj)
    end
    function doThat(obj)
    end
end

end
```

初始注释块中的属性和方法的列表和说明是可选的。如果您的注释行包含类名称且后跟 **Properties** 或 **Methods** 以及一个冒号 (:), 则 MATLAB 将创建指向这些属性或方法的帮助的超链接。

使用 **help** 命令在命令行窗口中查看类的帮助文本。

```
help myClass
```

```
myClass Summary of myClass
This is the first line of the description of myClass.
Descriptions can include multiple lines of text.

myClass Properties:
    a - Description of a
    b - Description of b

myClass Methods:
    doThis - Description of doThis
    doThat - Description of doThat
```

### 方法

通过在函数定义语句后插入注释来为方法创建帮助。例如，修改类定义文件 **myClass.m** 以包括 **doThis** 方法的帮助。

```
function doThis(obj)
% doThis Do this thing
% Here is some help text for the doThis method.
%
% See also DOTHT.
```

```
    disp(obj)
end
```

使用 **help** 命令在命令行窗口中查看方法的帮助文本。指定类名称和方法名称，用点分隔开。

```
help myClass.doThis
```

```
doThis Do this thing
Here is some help text for the doThis method.
```

```
See also doThat.
```

## 属性

可通过两种方法为属性创建帮助：

- 在属性定义上方插入注释行。将此方法用于多行帮助文本。
- 在属性定义旁边添加一行注释。

定义上方的注释比定义旁边的注释具有更高的优先级。

例如，修改类定义文件 `myClass.m` 中的属性定义。

```
properties
  a      % First property of myClass

  % b - Second property of myClass
  % The description for b has several
  % lines of text.
  b      % Other comment
end
```

使用 `help` 命令在命令行窗口中查看属性的帮助。指定类名称和属性名称，用点分隔开。

`help myClass.a`

a - First property of myClass

`help myClass.b`

b - Second property of myClass  
The description for b has several  
lines of text.

## 枚举

和属性一样，可通过两种方法为枚举创建帮助：

- 在枚举定义上方插入注释行。将此方法用于多行帮助文本。
- 在枚举定义旁边添加一行注释文本。

定义上方的注释比定义旁边的注释具有更高的优先级。

例如，在名为 `myEnumeration.m` 的文件中创建一个枚举类。

```
classdef myEnumeration
  enumeration
    uno,      % First enumeration

    % DOS - Second enumeration
    % The description for DOS has several
    % lines of text.
    dos      % A comment (not help text)
  end
end
```

使用 `help` 命令在命令行窗口中查看帮助。指定类名称和枚举成员，用点分隔开。

`help myEnumeration.uno`

```
uno - First enumeration

help myEnumeration.dos

dos - Second enumeration
The description for dos has several
lines of text.
```

### 事件

和属性和枚举一样，可通过两种方法为事件创建帮助：

- 在事件定义上方插入注释行。将此方法用于多行帮助文本。
- 在事件定义旁边添加一行注释文本。

定义上方的注释比定义旁边的注释具有更高的优先级。

例如，在名为 `hasEvents.m` 的文件中创建一个类。

```
classdef hasEvents < handle
events
    Alpha % First event

    % Beta - Second event
    % Additional text about second event.
    Beta % (not help text)
end

methods
    function fireEventAlpha(h)
        notify(h,'Alpha')
    end

    function fireEventBeta(h)
        notify(h,'Beta')
    end
end
end
```

使用 `help` 命令在命令行窗口中查看帮助。指定类名称和事件，用点分隔开。

```
help hasEvents.Alpha

Alpha - First event

help hasEvents.Beta

Beta - Second event
Additional text about second event.
```

### 另请参阅

[doc](#) | [help](#)

### 详细信息

- “类在 MATLAB 中的角色”

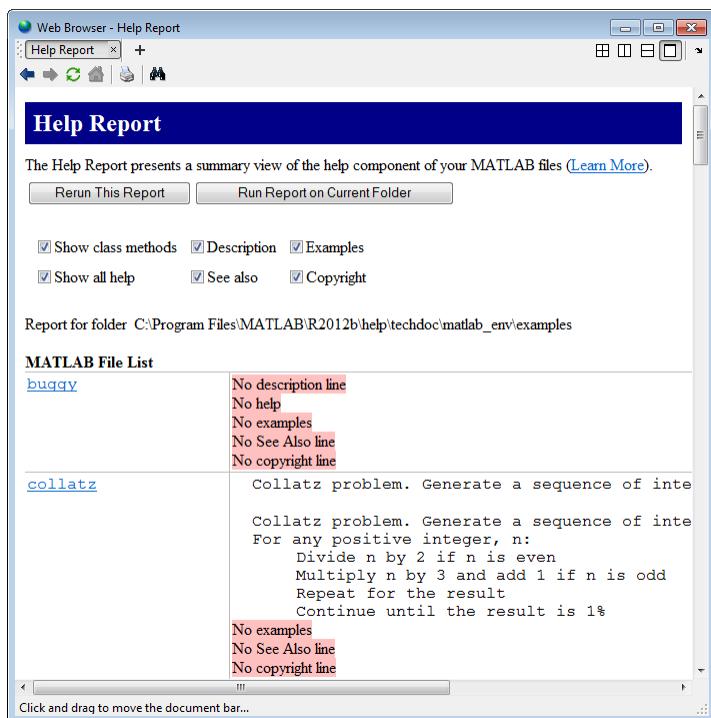
- “用户定义的类”

## 检查哪些程序具有帮助

要确定您的哪些程序文件包含帮助文本，可以使用帮助报告。

在帮助报告中，指定一组要搜索的帮助组件，例如示例或 **See Also** 行。对于搜索的每个文件，MATLAB 显示其查找的组件的帮助文本。否则，MATLAB 显示一则高亮消息以指示缺失该组件。

要生成帮助报告，请在当前文件夹浏览器中导航到要检查的文件夹，点击 ，然后选择报告 > 帮助报告。帮助报告显示在 MATLAB Web 浏览器中。



**注意** 您不能在路径为 UNC（通用命名约定）路径时运行报告；即路径以 \\ 开头。应使用您系统上的实际硬盘驱动器，或映射的网络驱动器。

下表描述了帮助报告的可用选项。

帮助报告选项	说明
<b>显示类方法</b>	在报告中包括方法。如果您不选择此选项，则报告包括类的结果，而不是类定义文件中的方法的结果。
<b>显示所有帮助</b>	显示每个文件中找到的所有帮助文本。如果您还选择各个帮助组件（例如 <b>说明</b> ），则帮助文本在每个文件的报告中显示两次：一次针对总体帮助文本，一次针对相应组件。  如果您的程序与 MATLAB 搜索路径上的其他程序同名，则 <b>help</b> 命令生成这些重载项目的列表。MATLAB 会自动添加指向这些项目的帮助的链接。
<b>说明</b>	检查文件中的初始非空注释行。该行有时称为 H1 行。

帮助报告选项	说明
<b>示例</b>	检查帮助文本中的示例。帮助报告对包含 <code>example</code> 的单个单词变体的帮助行执行不区分大小写的搜索。该报告显示该行和后续非空注释行，以及初始行号。
<b>另请参阅</b>	检查帮助中以单词 <code>See also</code> 开头的行。该报告显示文本和初始行号。 如果在 <code>See also</code> 之后列出的程序在搜索路径上，则 <code>help</code> 命令生成指向这些程序的帮助的超链接。帮助报告指示 <code>See also</code> 行中的程序何时不在该路径上。
<b>版权</b>	检查文件中以单词 <code>Copyright</code> 开头的注释行。当存在版权行时，该报告还检查终止年份是否为当前年份。日期检查需要版权行包括单个年份（例如 <code>2012</code> ）或不含空格的一系列年份（例如 <code>2001-2012</code> ）。 建议的做法是包括一系列年份，从您创建文件的年份开始一直到当前年份。

## 另请参阅

### 相关示例

- “为程序添加帮助” (第 20-5 页)
- “创建帮助摘要文件 - `Contents.m`” (第 31-10 页)

## 创建帮助摘要文件 - Contents.m

### 本节内容

- “什么是 Contents.m 文件？”（第 31-10 页）
- “创建 Contents.m 文件”（第 31-10 页）
- “检查现有的 Contents.m 文件”（第 31-11 页）

### 什么是 Contents.m 文件？

Contents.m 文件提供特定文件夹中的程序的摘要。help、doc 和 ver 函数引用 Contents.m 文件以显示有关文件夹的信息。

Contents.m 文件仅包含注释行。前两行是描述该文件夹的标题。后续行列出该文件夹中的程序文件及其说明。您也可以对文件分组并包括分类说明。例如，查看位于 codetools 文件夹中的函数：

```
help codetools
```

```
Commands for creating and debugging code
MATLAB Version 9.3 (R2017b) 24-Jul-2017

Editing and publishing
edit           - Edit or create a file
grabcode       - Copy MATLAB code from published HTML
mlint          - Check files for possible problems
publish        - Publish file containing cells to output file
snapshot       - Force snapshot of image for published document

Directory tools
mlintrpt      - Run mlint for file or folder, reporting results in browser
visdiff        - Compare two files (text, MAT, or binary) or folders

...
```

如果您不想让其他人看到您的程序文件的摘要，请在该文件夹中放置一个空 Contents.m 文件。空 Contents.m 文件会导致 help foldername 报告 No help found for foldername。如果没有 Contents.m 文件，help 和 doc 命令会显示生成的文件夹中所有程序文件列表。

### 创建 Contents.m 文件

当某文件夹中包含一组现有程序文件时，创建 Contents.m 文件的最简单方法是使用内容报告。内容报告的主要用途是检查现有的 Contents.m 文件是否是最新的。但是，它还检查 Contents.m 是否存在，并可基于该文件夹的内容生成一个新文件。执行下列步骤创建一个文件：

- 1 在当前文件夹浏览器中，导航到包含您的程序文件的文件夹。
- 2 点击 ，然后选择报告 > 内容报告。
- 3 在该报告中，如果提示生成一个 Contents.m 文件，请点击是。新文件包括该文件夹中所有程序文件的名称，并使用描述行（第一个非空注释行，如果可用）。
- 4 在编辑器中打开生成的文件，并修改该文件以使第二个注释行采用以下格式：

```
% Version xxx dd-mmm-yyyy
```

不要在日期中包括任何空格。此注释行使 ver 函数能够检测版本信息。

---

**注意** MATLAB 在创建内容报告时不包括实时脚本或函数。

## 检查现有的 Contents.m 文件

使用内容报告验证您的 **Contents.m** 文件是否反映该文件夹的当前内容，如下所示：

- 1 在当前文件夹浏览器中，导航到包含 **Contents.m** 文件的文件夹。
- 2 点击 ，然后选择**报告 > 内容报告**。

**注意** 您不能在路径为 UNC (通用命名约定) 路径时运行报告；即路径以 \\ 开头。应使用您系统上的实际硬盘驱动器，或映射的网络驱动器。

内容报告执行以下检查。

检查 Contents.m 文件是否...	详细信息
存在	如果该文件夹中没有 <b>Contents.m</b> 文件，您可以从报告中创建一个。
包括该文件夹中的所有程序	缺少的程序以灰色高亮显示。您无需添加不希望公开给最终用户的程序。
错误地列出不存在的文件	不在该文件夹中的所列程序以粉色高亮显示。
匹配程序文件说明	该报告将 <b>Contents.m</b> 中的文件说明与对应文件中的第一个非空注释行进行比较。不一致之处以粉色高亮显示。您可以更新程序文件或 <b>Contents.m</b> 文件。
在文件名和说明之间使用一致的间距	通过点击报告顶部的 <b>固定间距</b> 使对齐方式固定。

您可以通过点击**全部固定**进行所有建议的更改，或通过点击**编辑 Contents.m** 在编辑器中打开文件。

## 另请参阅

[doc](#) | [help](#) | [ver](#)

## 自定义代码建议和自动填充

要为您的函数自定义代码建议和自动填充，请向 MATLAB 提供有关您的函数签名的信息。函数签名描述函数的可接受语法和允许的数据类型。MATLAB 使用此信息来改善交互功能，如 Tab 键自动填充和函数提示。在名为 `functionSignatures.json` 的 JSON 格式文件中定义该函数信息。

要使 MATLAB 检测函数签名信息，您必须将 `functionSignatures.json` 放入包含函数代码的文件夹中。您可以在同一个文件中为多个函数定义签名。

`functionSignatures.json` 文件只包含一个 JSON 对象。JSON 使用花括号来定义对象，并以名称-值对组集合的形式引用对象。由于这些术语在函数签名的语境下使用过多，因而此处使用“属性”而非“名称”。`functionSignatures.json` 中的 JSON 对象包含架构版本（可选）和函数对象列表。每个函数对象都包含一个签名对象列表，每个签名对象包含一个参数对象数组。JSON 使用方括号来定义数组。

```
{
  "_schemaVersion": "<major#>.<minor#>.<patch#>",

  "functionName1":
  {
    "inputs":
    [
      {"name": "A", "kind": "required", "type": ["numeric"]},
      {"name": "dim", "kind": "ordered", "type": ["numeric", "integer", "scalar", ">0"]},
      {"name": "nanflag", "kind": "flag", "type": ["char"], "choices": {'includenan', 'omitnan'}}
    ]
  }

  "functionName2":
  {
    "inputs":
    [
      {"name": "A", "kind": "required", "type": ["numeric"]},
      {"name": "B", "kind": "required", "type": ["numeric", "scalar"]}
    ]
  }
  ...
}
```

要指定架构版本（可选），请使用 `_schemaVersion` 作为第一个属性，使用版本号作为其值。将版本号指定为 `major#.minor#.patch#` 格式的 JSON 字符串，其中每个数字指定为非负整数。当前架构版本是 **1.0.0**。如果文件未指定架构版本，则 MATLAB 假定版本为 **1.0.0**。

如果 `functionSignatures.json` 包含语法错误，MATLAB 在读取文件时会在命令行窗口中显示错误消息。使用 `validateFunctionSignaturesJSON` 函数根据 JSON 架构和 MATLAB 函数签名架构验证 `functionSignatures.json` 文件。

### 函数对象

要为函数定义信息，请创建一个与函数名称相同的属性。它的值是签名对象（第 31-13 页）。

```
{
  "functionName1": { signatureObj1 },
  "functionName2": { signatureObj2 }
}
```

要为类方法或包函数定义信息，请使用函数或方法的全名。示例属性为 `"MyClass.myMethod"` 或 `"myPackage.myFunction"`。通过定义具有相同属性（函数名称）的多个函数对象，可以为同一个函数定义多个函数签名。有关详细信息，请参阅“多个签名”（第 31-18 页）。

## 签名对象

签名对象定义函数的输入和输出参数以及支持的平台。每个属性的值（`platforms` 属性除外）都是一个参数对象（第 31-13 页）数组。

```
{
  "functionName1":
  {
    "inputs": [ argumentObj1, argumentObj2 ]
  }
}
```

每个签名可以包含以下属性。

属性	说明	值的 JSON 数据类型
<code>inputs</code>	函数输入参数的列表。MATLAB 将此属性用于代码建议和自动填充。	参数对象数组
<code>outputs</code>	函数输出参数的列表。MATLAB 当前不将此属性用于代码建议和自动填充。	参数对象数组
<code>platforms</code>	<p>支持该函数的平台的列表。如果平台不支持该函数，MATLAB 不会显示自定义代码建议和自动填充。</p> <p>默认是所有平台。列表的元素必须与 <code>computer</code> 函数返回的 <code>archstr</code> 匹配。该列表要么列出所有支持的平台，要么列出所有不支持的平台，但不能同时列出支持和不支持的平台。示例值为 "<code>win64,maci64</code>" 或 "<code>-win64,-maci64</code>"。</p>	逗号分隔值的字符串

## 参数对象

参数对象定义每个输入和输出参数的信息。

```
{
  "functionName1":
  {
    "inputs":
    [
      {"name":"in1", "kind":"required", "type":["numeric"]},
      {"name":"in2", "kind":"required", "type":["numeric","integer","scalar"]}
    ]
  }
}
```

各个输入在 JSON 文件中的显示顺序非常重要。例如，在对 `functionName1` 函数的调用中，`in1` 必须出现在 `in2` 之前。

每个参数对象可以包含以下属性。

### name - 参数的名称

输入或输出参数的名称，指定为 JSON 字符串。此属性和值是必需的。`name` 属性不需要与源代码中的参数名称相匹配，但最佳做法是匹配某一帮助或文档。

示例："`name":"myArgumentName"`

**kind - 参数的种类**

参数的种类，指定为具有下列值之一的 JSON 字符串。MATLAB 使用 **kind** 属性的值来确定是否以及何时在函数签名中显示参数。

值	说明
<b>required</b>	参数是必需的，其位置相对于签名对象中的其他必需参数。
<b>ordered</b>	参数是可选的，其位置相对于签名对象中的必需参数和前面的可选参数。
<b>namevalue</b>	参数是可选的名称-值对组。名称-值对组参数出现在函数签名的末尾，但这些对组可以按任意顺序指定。

**required** 和 **ordered** 种类的参数出现在函数签名的开头，后跟 **namevalue** 参数。

**required**、**ordered** 和 **namevalue** 参数是最常见的。您还可以为 **kind** 指定以下值。

- **positional** - 如果此类参数出现在参数列表末尾，则为可选参数；但如果要在其后指定一个位置参数，则它会变为必需参数。任何 **positional** 参数必须与 **required** 和 **ordered** 参数一起出现，而且位于 **namevalue** 参数之前。
- **flag** - 参数是可选的常量字符串，通常用作开关。例如，'ascend' 或 'descend'。标志出现在函数签名的末尾。所有 **flag** 参数必须出现在 **namevalue** 参数之前。
- **properties** - 参数是可选的，用于指定另一不同 MATLAB 类的公共可设置属性。使用参数对象 **type** 属性指示类。在代码建议中，这些属性显示为名称-值对组。**properties** 参数都必须是签名中的最后一个参数。

示例："kind":"required" 或 "kind":"namevalue"

**type - 参数的类和/或属性**

参数的类或属性，指定为 JSON 字符串、列表或者字典的列表。

**type** 属性可以定义参数是哪个类以及参数必须具有哪些属性。

- 要匹配一个类或属性，请使用单个 JSON 字符串。例如，如果参数必须是数值，则指定 "type":"numeric"。
- 要匹配所有类或属性，请使用 JSON 字符串列表。例如，如果某个参数必须既是数值又是正数，则指定 "type":["numeric", ">=0"]。
- 要匹配多个类或属性中的任意多个，请使用 JSON 字符串列表的列表。在内层列表，MATLAB 对各值执行逻辑 AND 运算。在外层列表，MATLAB 对各值执行逻辑 OR 运算。例如，如果参数必须要么是正数，要么是 **containers.Map** 对象，则指定 "type":[[{"type": "numeric", ">=0"}, {"type": "containers.Map"}]]。

值	参数说明
"classname"	必须是类 <b>classname</b> 的对象，其中 <b>classname</b> 是 <b>class</b> 函数返回的类的名称。例如，"double" 或 "function_handle"。
"choices=expression"	必须是指定选项之一的匹配项（不区分大小写）。 <b>expression</b> 是返回字符向量元胞、字符串数组或整数值元胞的有效 MATLAB 表达式。例如，"choices={'on','off'}" 或 "choices={8, 16, 24}"。  <b>expression</b> 可以按名称引用出现在参数列表中的其他输入参数。由于 <b>expression</b> 是在运行时计算的，因此允许的选项可以随其他输入参数的值动态变化。

值	参数说明
"file=*.ext,..."	必须是字符串或字符向量，表示具有指定扩展名的现有文件的名称。文件名相对于当前工作文件夹。例如，要允许当前文件夹中的所有 .m 和 .mlx 文件，请使用 "file=*.m,*.mlx"。要匹配当前文件夹中的所有文件，请使用 "file"。
"folder"	必须是相对于当前工作文件夹的某一现有文件夹的名称的字符串或字符向量。
"matlabpathfile=*.ext,..."	必须是指名 MATLAB 路径上某一现有文件的字符串或字符向量。该值至少需要一个文件扩展名。例如，要允许路径中的所有 .mat 文件，请使用 "matlabpathfile=*.mat"。
"size=size1,size2,...,sizeN"	必须符合大小限制。该值要求至少两个维度。每个大小维度只能是正整数（指示允许的维度大小）或冒号（表示允许任何大小）。例如，"size=2,:,2" 约束参数在第 1 个和第 3 个维度中的大小为 2。
"numel=integerValue"	必须有指定数量的元素。
"nrows=integerValue"	必须有指定数量的行。
"ncols=integerValue"	必须有指定数量的列。
"numeric"	必须是数值。数值型的值是用 <code>isa</code> 函数判断其是否属于 'numeric' 类时返回 <code>true</code> 的值。
"logical"	必须是数值或逻辑值。
"real"	必须是实数值或字符或逻辑值。
"scalar"	必须是标量。
"integer"	必须是 <code>double</code> 类型的整数。
"square"	必须是方阵。
"vector"	必须是列或行向量。
"column"	必须是列向量。
"row"	必须是行向量。
"2d"	必须是二维的。
"3d"	不得超过三个维度。
"sparse"	必须是稀疏类型。
"positive"	必须大于零。
">expression"	必须是数值且满足不等式。expression 必须返回全双精度标量。
">=expression"	
"<expression"	
"<=expression"	
@(args) expression	必须满足函数句柄。要使值满足函数句柄，句柄的计算结果必须为 <code>true</code> 。

### repeating - 多次指定参数

指示某个参数可以多次指定，指定为 JSON `true` 或 `false`（不含引号）。默认值为 `false`。如果指定为 `true`，则参数或参数集（元组）可以多次指定。必需的重复参数必须出现一次或多次，可选的重复参数可以出现零次或多次。

示例："`repeating":true`

### **purpose - 参数说明**

参数说明，指定为 JSON 字符串。使用此属性来表明参数的目的。

示例： "purpose":"Product ID"

对于更复杂的函数签名，各个参数对象均可使用以下属性。

### **platforms - 支持的平台的列表**

支持该参数的平台的列表，指定为 JSON 字符串。默认是所有平台。列表的元素必须与 **computer** 函数返回的 **archstr** 匹配。该列表要么列出所有支持的平台，要么列出所有不支持的平台，但不能同时列出支持和不支持的平台。

示例： "platforms":"win64,maci64" 或 "platforms": "-maci64"

### **tuple - 参数集的定义**

必须始终一起出现的一组参数的定义，指定为参数对象的列表。该属性仅用于定义多个重复参数的集合。对于这些函数签名，定义元组并将 **repeating** 属性设置为 **true**。

### **mutuallyExclusiveGroup - 互斥参数集的定义**

一组不能一起使用的参数集的定义，指定为参数对象的列表。该属性用于提供有关具有多个函数签名的函数的信息。但是，使用多个函数对象定义多个函数签名通常会更容易。有关详细信息，请参阅“多个签名”（第 31-18 页）。

## **创建函数签名文件**

此示例说明如何为函数创建自定义代码建议和自动填充。

创建一个函数，其签名将在后续步骤中由 JSON 文件加以描述。以下函数接受：

- 两个必需参数
- 一个可选的位置参数（通过 **varargin**）
- 两个可选的名称-值对组参数（通过 **varargin**）

此处使用 **myFunc** 来演示代码建议，其中不包含参数检查。

```
% myFunc Example function
% This function is called with any of these syntaxes:
%
% myFunc(in1, in2) accepts 2 required arguments.
% myFunc(in1, in2, in3) also accepts an optional 3rd argument.
% myFunc(___, NAME, VALUE) accepts one or more of the following name-value pair
% arguments. This syntax can be used in any of the previous syntaxes.
% * 'NAME1' with logical value
% * 'NAME2' with 'Default', 'Choice1', or 'Choice2'
function myFunc(reqA,reqB,varargin)
    % Initialize default values
    NV1 = true;
    NV2 = 'Default';
    posA = [];

    if nargin > 3
        if rem(nargin,2)
            posA = varargin{1};
            V = varargin(2:end);
        else
            V = varargin;
        end
    for n = 1:2:size(V,2)
```

```

switch V{n}
  case 'Name1'
    NV1 = V{n+1};
  case 'Name2'
    NV2 = V{n+1}
  otherwise
    error('Error.')
end
end
end
end

```

在 **myFunc** 所在的文件夹中，在名为 **functionSignatures.json** 的文件中创建以下函数签名描述。输入名称与 **myFunc** 主体中的名称不匹配，但与帮助文本一致。

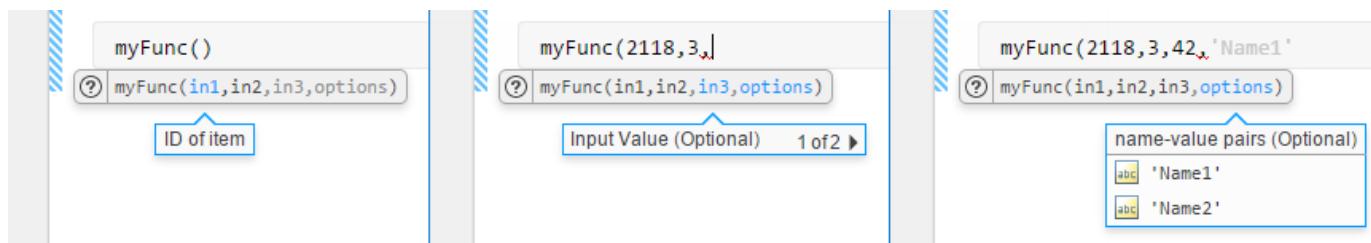
```

{
  "_schemaVersion": "1.0.0",
  "myFunc":
  {
    "inputs":
    [
      {"name":"in1", "kind":"required", "type":["numeric"], "purpose":"ID of item"},
      {"name":"in2", "kind":"required", "type":["numeric"], "purpose": "# Items"},
      {"name":"in3", "kind":"ordered", "type":["numeric"], "purpose":"Input Value"},
      {"name":"Name1", "kind":"namevalue", "type":["logical","scalar"], "purpose":"Option"},
      {"name":"Name2", "kind":"namevalue", "type":["char"], "choices={'Default','Choice1','Choice2'}"}
    ]
  }
}

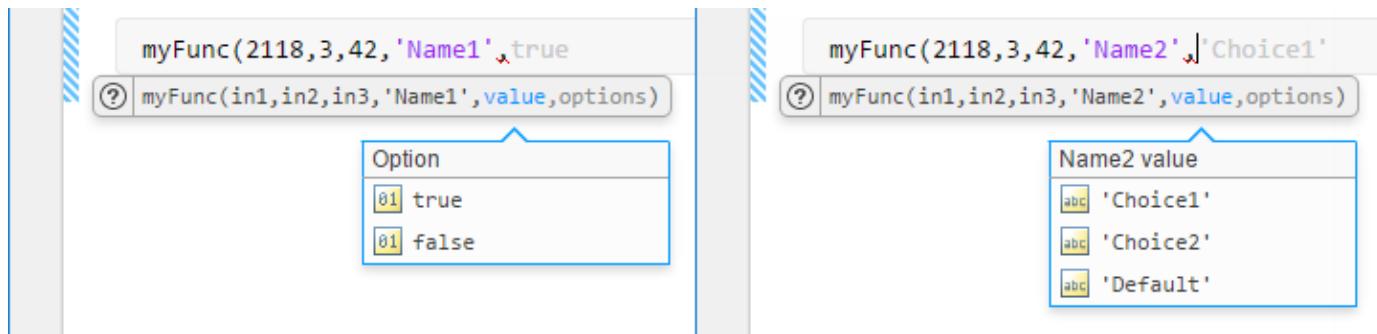
```

MATLAB 使用此函数签名描述来提供代码建议和自动填充。

要试用代码建议，请从实时脚本中调用该函数并观察建议。例如，建议会显示取自 JSON 文件的名称和用途。MATLAB 会指示参数是否为可选以及是否有多个建议（如第三个位置参数或一个名称-值对组）。列出名称-值对组选项。



向函数调用添加名称-值对组参数时，MATLAB 基于 JSON 文件提供选项。由于 'Name1' 定义为逻辑标量，因此 MATLAB 会自动填充选项 (`true` 或 `false`)。MATLAB 基于 JSON 文件显示 'Name2' 参数的三个值。



## 多个签名

如果一个函数有许多语法，则可以在代码建议中将语法分组为多个函数签名（不考虑函数实现）。要为多个签名提供代码建议和自动填充，请在 JSON 文件中创建具有相同属性的多个函数对象。

以如下函数为例，它根据第二个输入的类采用不同代码路径。此函数仅用作代码建议的示例，因此不执行任何计算或错误检查。

```
function anotherFunc(arg1,arg2,arg3)
switch class(arg2)
    case 'double'
        % Follow code path 1
    case {'char','string'}
        % Follow code path 2
    otherwise
        error('Invalid syntax.')
end
end
```

从代码建议的角度来看，将函数视为具有两个函数签名。第一个签名接受两个必需的数值。第二个签名接受一个必需的数值，然后是一个字符或字符串，最后是一个必需的数值。要定义多个函数签名，请使用相同的属性（函数名称）在 JSON 文件中定义多个函数对象。

```
{
    "_schemaVersion": "1.0.0",
    "anotherFunc":
    {
        "inputs":
        [
            {"name":"input1", "kind":"required", "type":["numeric"]},
            {"name":"input2", "kind":"required", "type":["numeric"]}
        ],
        "anotherFunc":
        {
            "inputs":
            [
                {"name":"input1", "kind":"required", "type":["numeric"]},
                {"name":"input2", "kind":"required", "type":["char"], ["string"]},
                {"name":"input3", "kind":"required", "type":["numeric"]}
            ]
        }
    }
}
```

您也可以使用参数对象的 **mutuallyExclusiveGroup** 属性定义多个函数签名。通常情况下，实现多个函数对象更容易且更易读，但使用互斥组能够重用常见参数对象，如 **input1**。

```
{
    "_schemaVersion": "1.0.0",
    "anotherFunc":
    {
        "inputs":
```

```
[  
  {"name":"input1", "kind":"required", "type":["numeric"]},  
  {"mutuallyExclusiveGroup":  
    [  
      [  
        {"name":"input2", "kind":"required", "type":["numeric"]}  
      ],  
      [  
        {"name":"input2", "kind":"required", "type":["char","string"]},  
        {"name":"input3", "kind":"required", "type":["numeric"]}  
      ]  
    ]  
  }  
}
```

## 另请参阅

[validateFunctionSignaturesJSON](#)

## 详细信息

- “在键入时检查语法”

## 外部网站

- <https://www.json.org/>

## 显示自定义文档

### 本节内容

- “概述” (第 31-20 页)
- “创建 HTML 帮助文件” (第 31-20 页)
- “创建 info.xml 文件” (第 31-21 页)
- “创建 helptoc.xml 文件” (第 31-22 页)
- “编译搜索数据库” (第 31-24 页)
- “解决 info.xml 文件的验证错误” (第 31-24 页)

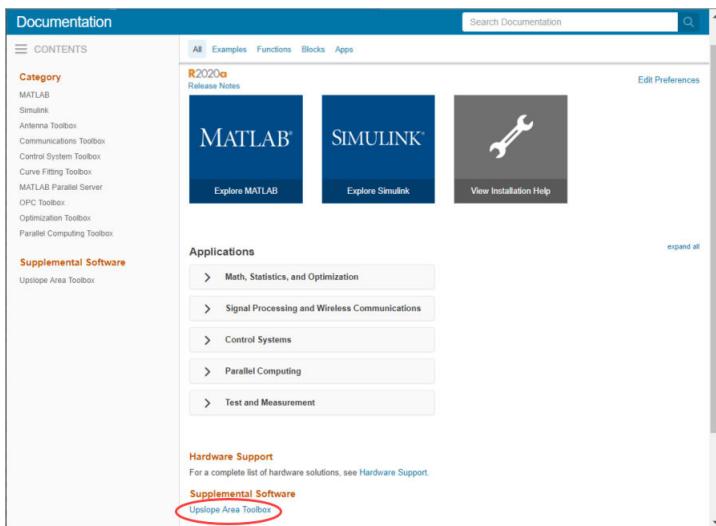
### 概述

如果您创建一个使用 MathWorks 产品的工具箱（即使它只包含一些函数），则可以包含 HTML 帮助文件格式的自定义文档。您的工具箱的自定义文档可以包括图窗、图表、屏幕捕获、方程和格式设置，以使您的工具箱帮助更实用。

要正常显示，您的自定义文档必须包含以下文件：

- **HTML 帮助文件** - 这些文件包含您的自定义文档信息。
- **info.xml 文件** - 此文件使 MATLAB 能够查找并标识您的 HTML 帮助文件。
- **helptoc.xml 文件** - 此文件包含您的文档的目录，该目录显示在帮助浏览器的**目录**窗格中。此文件必须存储在包含您的 HTML 帮助文件的文件夹中。
- **搜索数据库 (可选)** - 这些文件用于在您的 HTML 帮助文件中进行搜索。

要查看您的自定义文档，请打开帮助浏览器并导航到主页。在主页右下角的**补充软件**下面，点击您的工具箱的名称。您的帮助将在当前窗口中打开。



### 创建 HTML 帮助文件

您可以在任何文本编辑器或 Web 发布软件中创建 HTML 帮助文件。要在 MATLAB 中创建帮助文件，请使用以下两种方法之一：

- 创建实时脚本 (\*.mlx) 并将其导出为 HTML。有关详细信息, 请参阅“共享实时脚本和函数”(第 19-57 页)。
- 创建脚本 (\*.m) 并将其发布为 HTML。有关详细信息, 请参阅“发布和共享 MATLAB 代码”(第 23-2 页)。

将所有 HTML 帮助文件都存储在一个文件夹中, 例如您的工具箱文件夹中的 html 子文件夹。此文件夹必须:

- 位于 MATLAB 搜索路径中
- 位于 matlabroot 文件夹之外
- 位于任何已安装的硬件支持包帮助文件夹之外

文档集通常包含:

- 路线图页 (即文档的初始登录页)
- 说明如何使用工具箱的示例和主题
- 函数或块参考页

## 创建 info.xml 文件

info.xml 文件用于描述您的自定义文档, 包括文档的显示名称。它还标识查找您的 HTML 帮助文件和 helptoc.xml 文件的位置。为所记录的每个工具箱创建一个名为 info.xml 的文件。

要创建 info.xml 以描述您的工具箱, 您可以采用以下模板:

```
<productinfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="optional">
<?xmlstylesheet type="text/xsl" href="optional"?>

<matlabrelease>R2016b</matlabrelease>
<name>MyToolbox</name>
<type>toolbox</type>
<icon></icon>
<help_location>html</help_location>

</productinfo>
```

此外, 也可以使用 MATLAB 文档随附的模板 info\_template.xml 来创建 info.xml。要在当前文件夹中创建并编辑模板文件副本, 请在命令行窗口中运行以下代码:

```
copyfile(fullfile(matlabroot,'help','techdoc','matlab_env',...
'examples','templates','info_template.xml'),pwd)
fileattrib('info_template.xml','+w')
edit('info_template.xml')
```

下表介绍 info.xml 文件的必需元素。

XML 标记	说明	模板中的值	注释
<matlabrelease>	MATLAB 发行版	R2016b	指示何时添加了帮助文件。不显示在浏览器中。
<name>	工具箱标题	MyToolbox	您的自定义文档在浏览器目录窗格中的显示名称。
<type>	工具箱的标签	toolbox	允许的值: matlab、toolbox、simulink、blockset、links_targets、other。

XML 标记	说明	模板中的值	注释
<icon>	开始按钮的图标 (未使用)	无	不再使用，但 MATLAB 仍然需要使用<icon>元素来解析 info.xml 文件。
<help_location>	帮助文件的位置。	html	包含工具箱的 helptoc.xml 和 HTML 帮助文件的子文件夹的名称。如果该帮助位置不是 info.xml 文件位置的子文件夹，请将路径指定为 info.xml 文件对应的 help_location。如果您为多个工具箱提供 HTML 帮助文件，则每个 info.xml 文件中的 help_location 都必须为不同的文件夹。
<help_contents_icon>	显示在内容窗格中的图标	无	在 MATLAB R2015a 和更新版本中已忽略。出现在 info.xml 文件中（但非必须）时不会导致错误。

您也可以在 info.xml 文件中包括注释，例如版权和联系信息。通过将某行上的文本放在 <!-- 和 --> 之间来创建注释。

当您创建 info.xml 文件时，请确保：

- 包括所有必需元素。
- 这些条目的排列顺序与上表相同。
- XML 中的文件和文件夹名称与您的文件和文件夹的名称完全匹配并使用完全相同的大小写。
- info.xml 文件所在的文件夹位于 MATLAB 搜索路径上。

---

**注意** MATLAB 解析 info.xml 文件并在您向该路径中添加包含 info.xml 的文件夹时显示您的文档。如果您在已位于该路径上的文件夹中创建了 info.xml 文件，请从该路径中删除该文件夹。然后重新添加该文件夹，以便 MATLAB 解析该文件。确保您要添加的文件夹不是您的当前文件夹。

---

## 创建 helptoc.xml 文件

helptoc.xml 文件用于定义补充软件浏览器的**目录**窗格中显示的帮助文件层次结构。

您可以使用 MATLAB 文档随附的模板来创建 helptoc.xml 文件。要在当前文件夹中创建并编辑模板文件 helptoc\_template.xml 的副本，请在命令行窗口中运行以下代码：

```
copyfile(fullfile(matlabroot,'help','techdoc','matlab_env',...
'examples','templates','helptoc_template.xml'),pwd)
fileattrib('helptoc_template.xml','+w')
edit('helptoc_template.xml')
```

将 helptoc.xml 文件放在包含您的 HTML 文档文件的文件夹中。此文件夹在 info.xml 文件中必须作为<help\_location>进行引用。

helptoc.xml 文件中的每个 <tocitem> 条目引用您的 HTML 帮助文件中的一个。helptoc.xml 文件中的第一个 <tocitem> 条目用作您的文档的初始登录页。

在顶层 <toc> 元素中，嵌套的 <tocitem> 元素定义您的目录的结构。每个 <tocitem> 元素都有一个提供文件名的 target 属性。文件名和路径名区分大小写。

当您创建 **helptoc.xml** 文件时，请确保：

- **helptoc.xml** 文件的位置在 **info.xml** 文件中是作为 **<help\_location>** 列出的。
- 所有文件名和路径名都与文件和文件夹的名称完全匹配，包括大小写也完全匹配。
- 所有路径名都使用 URL 文件路径分隔符 (/)。Windows 样式文件路径分隔符 (\) 可导致目录显示不正确。例如，如果有一个 HTML 帮助页 **firstfx.html** 位于主文档文件夹中称为 **refpages** 的子文件夹内，则该页面的 **<tocitem> target** 属性值将为 **refpages/firstfx.html**。

### **helptoc.xml** 文件示例

假设您创建了以下 HTML 文件：

- 您的工具箱的路线图或开始页，**mytoolbox.html**。
- 列出您的函数的页，**funclist.html**。
- 三个函数参考页：**firstfx.html**、**secondfx.html** 和 **thirdfx.html**。
- 示例页，**myexample.html**。

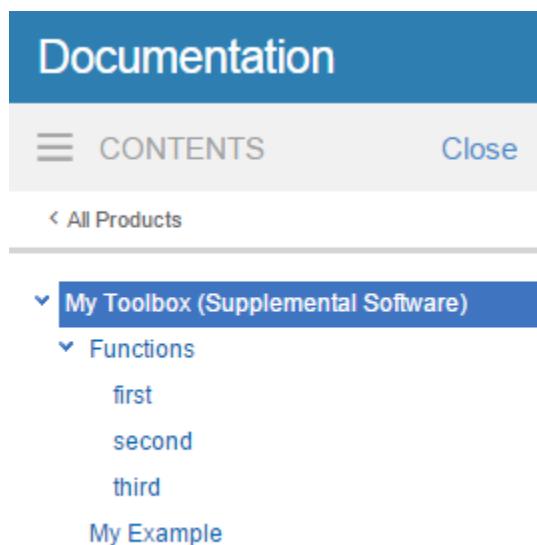
将文件名和说明包括在 **helptoc.xml** 文件中，如下所示：

```
<?xml version='1.0' encoding="utf-8"?>
<toc version="2.0">

<tocitem target="mytoolbox.html">My Toolbox
  <tocitem target="funclist.html">Functions
    <tocitem target="firstfx.html">first</tocitem>
    <tocitem target="secondfx.html">second</tocitem>
    <tocitem target="thirdfx.html">third</tocitem>
  </tocitem>
  <tocitem target="myexample.html">My Example
  </tocitem>
</tocitem>

</toc>
```

此 **helptoc.xml** 文件（配有正常构建的 **info.xml** 文件）在帮助浏览器中生成了以下显示。



## 编译搜索数据库

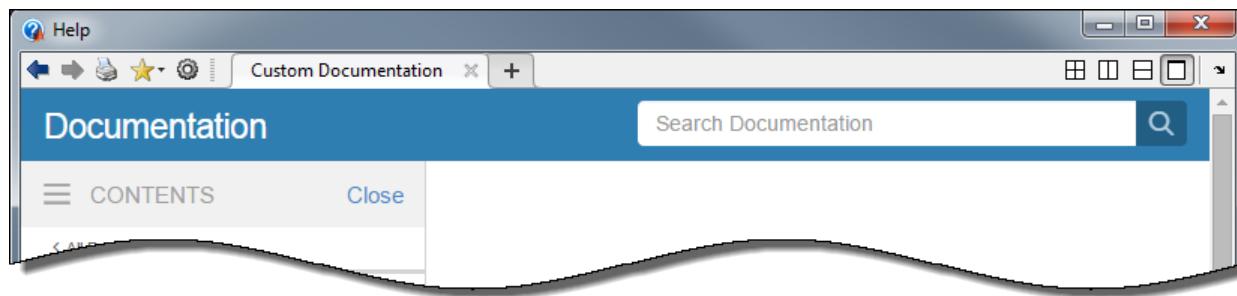
要使您的文档可供搜索，请使用 **builddocsearchdb** 命令创建一个搜索数据库（也称为搜索索引）。使用该命令时，请指定包含您的 HTML 文件的文件夹的完整路径。

例如，假设您的 HTML 文件位于 C:\MATLAB\MyToolbox\html 中。以下命令将为这些文件创建一个可搜索的数据库：

```
builddocsearchdb('C:\MATLAB\MyToolbox\html')
```

**builddocsearchdb** 会在 C:\MATLAB\MyToolbox\html 中创建一个名为 **helpsearch-v3** 的子文件夹，其中包含数据库文件。

您可以通过帮助浏览器中的**搜索文档**字段来搜索工具箱中的术语。



从 MATLAB R2014b 开始，您可以并排放置搜索索引。例如，如果您已有一个用于 MATLAB R2014a 或更早版本的搜索索引，则可以使用 MATLAB R2014b 根据帮助文件运行 **builddocsearchdb**。然后，当您运行任何 MATLAB 版本时，帮助浏览器会自动使用相应的索引搜索您的文档数据库。

## 解决 info.xml 文件的验证错误

### 什么是 XML 验证错误？

当 MATLAB 在搜索路径上或当前文件夹中找到 **info.xml** 文件时，它会自动根据支持的架构来验证该文件。如果 **info.xml** 文件中有无效的构造，MATLAB 会在命令行窗口中显示错误。该错误的形式通常为：

**Warning: File <yourxmlfile.xml> did not validate.**

...

当您启动 MATLAB 或在搜索路径中添加文件夹时，会发生 **info.xml** 验证错误。

XML 文件验证错误的主要原因如下：

- **info.xml** 文件中缺少实体或实体顺序有误。
- 存在不相关的 **info.xml** 文件。
- **info.xml** 文件中存在语法错误。
- MATLAB 尝试访问 MathWorks 产品的过时 **info.xml** 文件。

### info.xml 中缺少实体或实体顺序有误

如果您没有按规定的顺序列出必需的 XML 元素，则会遇到 XML 验证错误：

Often, errors result from incorrect ordering of XML tags. Correct the error by updating the info.xml file contents to follow the guidelines in the MATLAB help documentation.

有关您在 `info.xml` 文件中所需的元素以及元素所需的排序方式的说明，请参阅“创建 `info.xml` 文件”(第 31-21 页)。

### 不相关的 `info.xml` 文件

假设有一个名为 `info.xml` 的文件与自定义文档毫不相关。由于此 `info.xml` 文件是一个不相关的文件，因此如果它导致错误，则您完全可以将其忽略。为防止再次出现此错误消息，请对不相关的 `info.xml` 文件进行重命名。或者，确保该文件不在搜索路径上或当前文件夹中。

### `info.xml` 文件中存在语法错误。

使用错误消息隔离问题或使用任何 XML 架构验证器。有关 `info.xml` 文件的结构的详细信息，请在 `matlabroot/sys/namespace/info/v1/info.xsd` 上参考该文件的架构。

### MathWorks 产品的 `info.xml` 文件过时

如果您的 `info.xml` 文件来自不同版本的 MATLAB，该文件中包含的构造可能对您的版本无效。要识别来自另一版本的 `info.xml` 文件，请查看错误消息中报告的完整路径名。该路径通常包含版本号，例如 ...\\MATLAB\\R14\\...。在本例中，该错误实际上不会产生任何问题，因此您完全可以忽略该错误消息。为确保错误不反复出现，请删除产生问题的 `info.xml` 文件。或者，从搜索路径和当前文件夹中删除已过时的 `info.xml` 文件。

## 另请参阅

### 相关示例

- “显示自定义示例” (第 31-26 页)
- “创建和共享工具箱” (第 25-10 页)
- “为程序添加帮助” (第 20-5 页)

## 显示自定义示例

### 本节内容

- “如何显示示例”（第 31-26 页）  
“demos.xml 文件的元素”（第 31-27 页）

### 如何显示示例

要在 MATLAB 帮助浏览器中显示视频、已发布的程序脚本等示例，或者其他演示您的程序用法的文件，请执行下列步骤：

- 1 创建您的示例文件。将这些文件存储在一个位于 MATLAB 搜索路径上、但在 `matlabroot` 文件夹之外的文件夹中。

**提示** MATLAB 包括一种功能，通过该功能可将脚本或函数转换为可按示例形式显示的格式化 HTML 文件。要在 MATLAB 中创建这些 HTML 文件，请使用以下两种方法之一：

- 创建实时脚本 (\*.mlx) 并将其导出为 HTML。有关详细信息，请参阅“共享实时脚本和函数”（第 19-57 页）。
- 创建脚本 (\*.m) 并将其发布为 HTML。有关详细信息，请参阅“发布和共享 MATLAB 代码”（第 23-2 页）。

- 2 创建一个 `demos.xml` 文件来描述您的示例的名称、类型以及显示信息。

例如，假设您有一个名为 My Sample 的工具箱，其中包含一个您发布为 HTML 的名为 `my_example` 的脚本。此 `demos.xml` 文件允许您显示 `my_example`：

```
<?xml version="1.0" encoding="utf-8"?>
<demos>
  <name>My Sample</name>
  <type>toolbox</type>
  <icon>HelpIcon.DEMOS</icon>
  <description>This text appears on the main page for your examples.</description>
  <website><a href="https://www.mathworks.com">Link to your Web site</a></website>

  <demosection>
    <label>First Section</label>
    <demoitem>
      <label>My Example Title</label>
      <type>M-file</type>
      <source>my_example</source>
    </demoitem>
  </demosection>

</demos>
```

**注意** `<demosection>` 元素是可选的。

- 3 查看您的示例。
  - a 在帮助浏览器中，导航到主页。
  - b 在页面底部的**补充软件**下面，点击对应您示例的链接。

您的示例将在帮助主窗口中打开。

## demos.xml 文件的元素

- “<demos> 中的一般信息”（第 31-27 页）
- “使用 <demosection> 的类别”（第 31-27 页）
- “有关 <demoitem> 中每个示例的信息”（第 31-27 页）

### <demos> 中的一般信息

在 `demos.xml` 文件中，根标记是 `<demos>`。该标记包括可确定您的示例的主页内容的元素。

XML 标记	注释
<code>&lt;name&gt;</code>	工具箱或示例集合的名称。
<code>&lt;type&gt;</code>	可能的值包括 <code>matlab</code> 、 <code>simulink</code> 、 <code>toolbox</code> 或 <code>blockset</code> 。
<code>&lt;icon&gt;</code>	在 MATLAB R2015a 和更新版本中已忽略。 在之前的版本中，此图标是对应您示例的图标。在那些版本中，您可以使用标准图标 <code>HelpIcon.DEMOS</code> 。您也可以通过指定相对于 <code>demos.xml</code> 文件位置的路径提供自定义图标。
<code>&lt;description&gt;</code>	显示在您的示例的主页上的说明。
<code>&lt;website&gt;</code>	(可选) 链接到网站。例如，MathWorks 示例包括一个指向位于 <a href="https://www.mathworks.com">https://www.mathworks.com</a> 的产品页的链接。

### 使用 <demosection> 的类别

或者，通过包含每个类别的 `<demosection>` 来定义您的示例的类别。如果您包含任意类别，则所有示例都必须在类别中。

每个 `<demosection>` 示例都包含一个 `<label>`，后者提供类别名称以及相关联的 `<demoitem>` 元素。

### 有关 <demoitem> 中每个示例的信息

XML 标记	注释
<code>&lt;label&gt;</code>	定义要显示在浏览器中的标题。
<code>&lt;type&gt;</code>	可能的值包括 <code>M-file</code> 、 <code>model</code> 、 <code>M-GUI</code> 、 <code>video</code> 或 <code>other</code> 。 通常，如果您使用发布函数发布了您的示例，则相应的 <code>&lt;type&gt;</code> 为 <code>M-file</code> 。
<code>&lt;source&gt;</code>	如果 <code>&lt;type&gt;</code> 为 <code>M-file</code> 、 <code>model</code> 、 <code>M-GUI</code> ，则 <code>&lt;source&gt;</code> 是关联的 <code>.m</code> 文件或模型文件的名称且不带扩展名。否则，请不要包括 <code>&lt;source&gt;</code> 元素，而应包括 <code>&lt;callback&gt;</code> 元素。
<code>&lt;file&gt;</code>	当您要显示描述示例的 HTML 文件时，仅对包含除 <code>M-file</code> 之外的 <code>&lt;type&gt;</code> 值的示例使用此元素。指定相对于 <code>demos.xml</code> 位置的路径。
<code>&lt;callback&gt;</code>	仅对 <code>&lt;type&gt;</code> 值为 <code>video</code> 或 <code>other</code> 的示例使用此元素，以指定可执行文件或 MATLAB 命令执行此示例。
<code>&lt;dependency&gt;</code>	(可选) 指定运行该示例所需其他产品，例如另一个工具箱。文本必须与位于搜索路径上或当前文件夹中的 <code>info.xml</code> 文件中指定的产品名匹配。



# 工程

---

- “[创建工程](#)” (第 32-2 页)
- “[自动执行启动和关闭任务](#)” (第 32-8 页)
- “[管理工程文件](#)” (第 32-10 页)
- “[查找工程文件](#)” (第 32-12 页)
- “[创建常见任务的快捷方式](#)” (第 32-14 页)
- “[将标签添加到工程文件](#)” (第 32-16 页)
- “[创建自定义任务](#)” (第 32-18 页)
- “[大型工程组件化](#)” (第 32-20 页)
- “[共享工程](#)” (第 32-22 页)
- “[升级工程](#)” (第 32-26 页)
- “[分析工程依存关系](#)” (第 32-29 页)
- “[对工程使用源代码管理](#)” (第 32-35 页)
- “[以编程方式创建和编辑工程](#)” (第 32-44 页)
- “[了解示例工程](#)” (第 32-51 页)

## 创建工程

### 什么是工程?

工程是一种可扩展的环境，您可以在一个工程中集中管理 MATLAB 文件、数据文件、要求、报告、电子表格、测试和生成的文件。

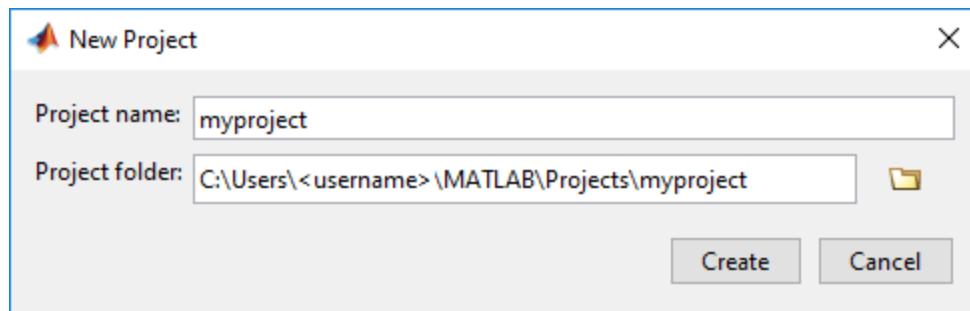
工程可以帮助您整理工作和进行协作。工程通过帮助您完成共同的任务来提高工作效率和团队合作。

- 查找属于您的工程的所有文件。
- 创建标准方法来设置和关闭整个团队的 MATLAB 环境。
- 创建、存储和轻松访问常见操作。
- 查看和标记经过修改的文件以用于同行评审工作流。
- 使用与 Git™、Subversion® (SVN) 的内置集成或使用外部源代码管理工具共享工程。

## 创建工程

要创建空白工程，请在**主页**选项卡上，点击**新建 > 工程 > 空白工程**。要从现有文件夹创建工程，请在**主页**选项卡上，点击**新建 > 工程 > 从文件夹**。

将打开“新建工程”对话框。输入工程名称，选择工程文件夹，然后点击**创建**。



## 打开工程

要打开现有工程，请在**主页**选项卡上，点击**打开**，并浏览到现有工程的 .prj 文件。或者，在当前文件夹浏览器中，双击工程的 .prj 文件。

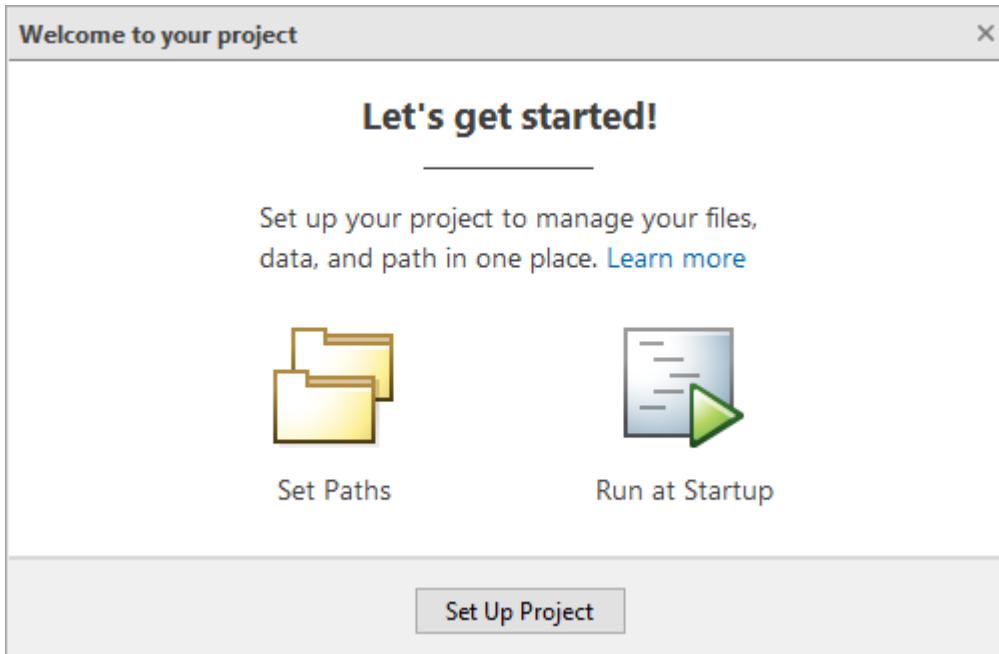
**注意** 为了避免冲突，在同一时刻只能有一个工程处于打开状态。如果打开另一个工程，则任何当前打开的工程都将关闭。

要打开最近的工程，请在**主页**选项卡上，点击**打开**箭头，然后在**最近创建的工程**列表下选择您的工程。

## 设置工程

创建工程后，将打开“欢迎使用您的工程”对话框，并提示您设置工程。

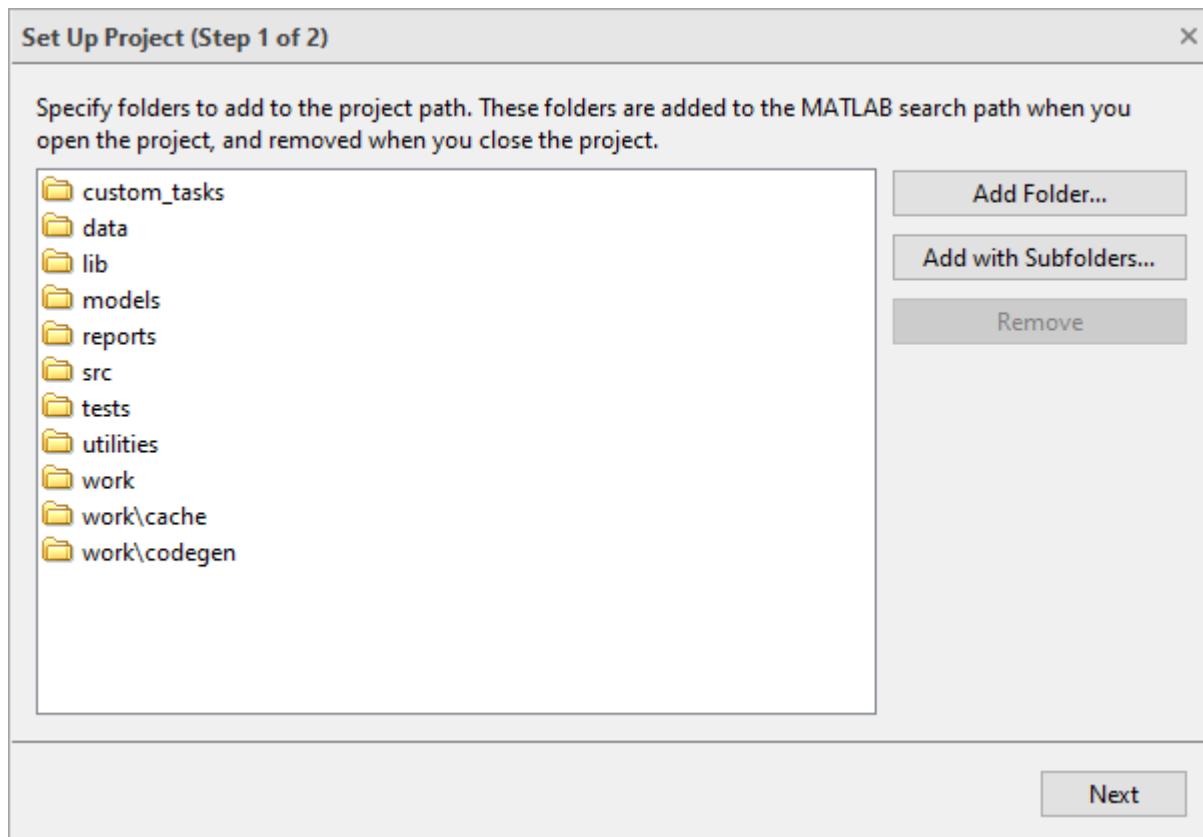
- 1 点击**设置工程**以开始设置您的工程。



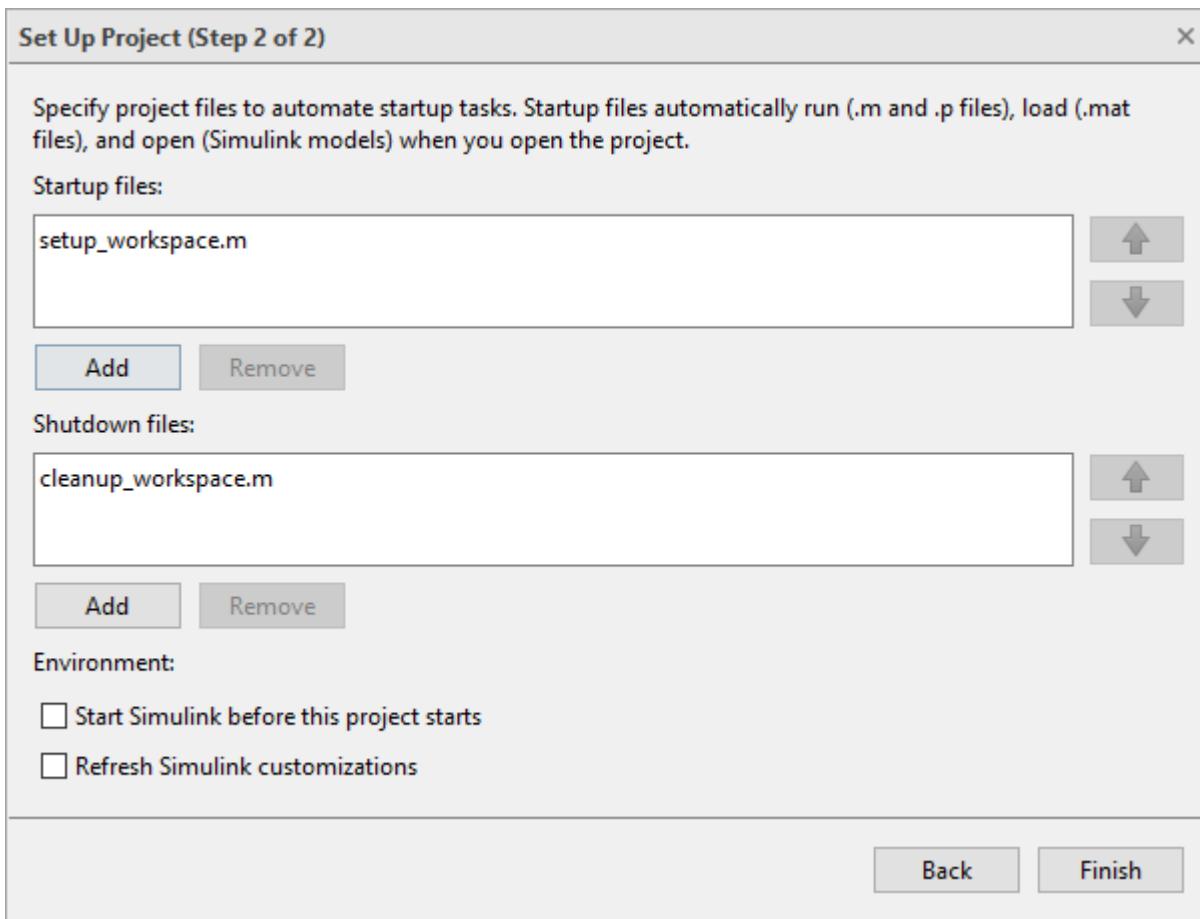
- 2 在“设置工程（第 1 步，共 2 步）”对话框中，您可以选择要添加到工程路径的文件夹。将工程文件夹添加到工程路径可确保工程的所有用户都能访问其中的文件。MATLAB 会在您打开工程时将这些文件夹添加到搜索路径中，并会在您关闭工程时将其删除。

要将工程文件夹中的所有文件夹添加到工程路径，请点击**添加并包含子文件夹**，然后选择包含所有子文件夹的工程根文件夹。

有关将文件夹添加到工程路径的详细信息，包括在完成工程设置后如何添加文件夹，请参阅“指定工程路径”（第 32-8 页）。



- 3 指定工程路径后，点击**下一步**按钮继续。
- 4 在“设置工程（第 2 步，共 2 步）”对话框中，可以指定启动和关闭文件。启动文件帮助您设置工程环境。关闭文件帮助您在完成后清理环境。使用关闭文件可撤消在启动文件中发生的设置。  
使用**添加**和**删除**按钮来管理启动和关闭文件列表。这些文件将从上到下运行。如果运行文件的顺序很重要，请使用箭头按钮在列表中上下移动文件。  
有关指定启动和关闭文件的详细信息，包括完成工程设置后如何指定它们，请参阅“指定启动和关闭文件”（第 32-8 页）。



5 点击**完成**以完成工程设置并打开您新创建的工程。

## 将文件添加到工程

**文件**视图显示工程中的文件列表。请注意，在创建新工程后，该列表为空。工程根文件夹中的文件不会包含在工程中，需要您自行添加。

要在工程中创建新文件或文件夹，请在**文件**视图中，右键点击空白区域，然后选择**新建文件夹**或**新建文件**。新文件或文件夹将被创建并添加到工程中。

要将现有文件添加到工程中，请在**工程**选项卡上的**文件**部分中，点击  **添加文件**。然后，从工程文件夹中尚未添加到工程的文件列表中进行选择。要从**文件**视图添加现有文件，请点击**全部**。然后，右键点击一个或多个文件和文件夹，并选择**添加到工程**或**将文件夹添加到工程(包括子文件)**。要从文件浏览器或当前文件夹浏览器添加现有文件，请将这些文件剪切并粘贴或拖放到工程的**文件**视图中。如果从工程根文件夹外部拖动文件，这会移动该文件并将其添加到工程中。拖动工程根文件夹中的文件以移动它们。

要以编程方式添加和删除工程文件，请使用 **addFile** 函数。

您可能不想在工程中包含所有文件。例如，您可能不希望在工程根文件夹中包含某些文件，例如 SVN 或 CVS 源代码管理文件夹。要确定哪些文件需要包含在工程中，请参阅“分析工程依存关系”（第 32-29 页）。

## 创建工程的其他方法

创建工程有几种替代方法。您可以：

- 从存档工程创建工程。
- 使用 Simulink 模板创建工程。
- 使用从现有源代码管理存储库中检索的文件创建工程。有关这种替代方法的详细信息，请参阅“对工程使用源代码管理”（第 32-35 页）。

### 从存档工程创建工程

有些工程以存档工程的方式进行共享。在与无权访问所连接的源代码管理工具的用户共享工程时，存档工程非常有用。要查看和编辑存档工程的内容，请从存档工程创建一个新工程。

要从存档工程创建新工程，请在当前文件夹浏览器中双击扩展名为 .mlproj 的存档工程文件。将打开“将工程提取到”对话框。指定新工程的位置，然后点击**选择文件夹**。例如，C:\myNewProject。

新工程将打开。当前文件夹（例如 C:\myNewProject）包含导入的工程文件夹。如果存档工程包含引用的工程，MATLAB 会将文件导入两个子文件夹 mains 和 refs。mains 工程文件夹（例如 C:\myNewProject\mains）包含工程文件夹。refs 文件夹（例如 C:\myNewProject\refs）包含引用的工程文件夹。

### 使用 Simulink 创建工程

如果您有 Simulink，您可以使用 Simulink 模板来创建和重用标准工程结构。

要从在 R2014b 或更高版本中创建的 Simulink 模板创建工程，请执行以下操作：

- 1 在**主页**或**工程**选项卡上，点击**新建 > 工程 > 从 Simulink 模板**。Simulink Start Page 将打开。
- 2 Start Page 显示 MATLAB 路径上的所有工程模板 (\*.sltx)。在列表中选择一个模板以读取模板说明。

如果您的模板没有出现，请点击**打开**找到它们。在“打开”对话框中，通过将文件类型设置为**所有 MATLAB 文件**使 \*.sltx 文件可见，并浏览到您的模板。

- 3 选择一个模板，然后点击**Create Project**。将打开“Create Project”对话框。

如果缺失必需的产品，在 R2017b 或更高版本中创建的模板会向您发出警告。点击链接打开附加功能资源管理器并安装所需的产品。

- 4 指定工程根文件夹，编辑工程名称，然后点击**创建工程**。

要使用在 R2014a 或更早版本中创建的工程模板 (.zip 文件)，请使用 **Simulink.exportToTemplate** 将它们升级到 .sltx 文件。

## 另请参阅

### 详细信息

- “**管理工程文件**”（第 32-10 页）
- “**分析工程依存关系**”（第 32-29 页）
- “**共享工程**”（第 32-22 页）
- “**对工程使用源代码管理**”（第 32-35 页）

- “以编程方式创建和编辑工程” (第 32-44 页)

## 自动执行启动和关闭任务

在您打开工程时，MATLAB 会将工程路径添加到 MATLAB 搜索路径，然后运行或加载指定的启动文件。工程路径和启动文件可帮助您为工程设置环境。同样，当您关闭工程时，MATLAB 会从 MATLAB 搜索路径中删除工程路径，并运行指定的关闭文件。关闭文件可帮助您清理工程环境。使用关闭文件可撤销在启动文件中发生的设置。

更具体地说，当您打开工程时，MATLAB 会将当前文件夹更改为工程启动文件夹，并运行（.m 和 .p 文件）或加载 (.mat 文件）任何指定的启动文件。有关配置启动文件夹的详细信息，请参阅“设置启动文件夹”（第 32-8 页）。

### 指定工程路径

您可以在工程路径中添加或删除文件夹。将工程文件夹添加到工程路径可确保工程的所有用户都能访问其中的文件。

要将文件夹添加到工程路径，请在**工程**选项卡上的**环境**部分中，点击**工程路径**。点击**添加文件夹**，然后选择要添加的文件夹。要添加一个文件夹及其所有子文件夹，请点击**添加并包含子文件夹**。

要从工程路径中删除文件夹，请从显示的列表中选择该文件夹，然后点击**删除**。

您也可以在工程的**文件**视图中添加或删除文件夹。右键点击文件夹，选择**工程路径**，然后从可用选项中进行选择。

工程路径上的文件夹在**文件**视图的**状态**列中显示有 工程路径图标。

### 设置启动文件夹

打开工程时，当前工作文件夹将更改为工程启动文件夹。默认情况下，启动文件夹设置为工程根目录。

要编辑工程启动文件夹，请在**工程**选项卡上的**环境**部分中，点击 **详细信息**。然后，在**启动**部分中，输入工程启动文件夹的路径。

### 指定启动和关闭文件

要将文件配置为在打开工程时运行，请右键点击该文件，然后选择**启动时运行**。要将文件配置为在关闭工程时运行，请右键点击该文件，然后选择**关闭时运行**。**状态**列会显示一个图标，指示文件是在启动时还是在关闭时运行。

要停止文件在启动或关闭时运行，请右键点击该文件并选择**从启动中删除**或**从关闭中删除**。

或者，转至**工程**选项卡，然后点击**启动关闭**。然后，在“管理工程的启动和关闭”对话框中，使用**添加**和**删除**按钮来管理启动和关闭文件列表。这些文件将从上到下运行。如果运行文件的顺序很重要，请使用箭头按钮在列表中上下移动文件。

**注意** 将修改后的文件提交到源代码管理时，会包括启动和关闭文件。配置启动和关闭文件后，它们对所有其他工程用户同样有效。

启动文件的名称可以是除 `startup.m` 之外的任何名称。在您启动 MATLAB 时，MATLAB 路径上名为 `startup.m` 的文件会运行。如果您用 `startup.m` 文件调用工程，将会发生错误，因为工程尚未加载。有关使用 `startup.m` 文件的详细信息，请参阅“在 MATLAB 启动文件中指定启动选项”。

要以编程方式创建新的启动和关闭文件，请参阅 `addStartupFile` 和 `addShutdownFile`。

在 Simulink 中，您可以指定其他工程启动选项。有关详细信息，请参阅“Automate Startup Tasks”(Simulink)。

## 另请参阅

`addStartupFile`

## 详细信息

- “管理工程文件” (第 32-10 页)
- “以编程方式创建和编辑工程” (第 32-44 页)
- “什么是 MATLAB 搜索路径？”

## 管理工程文件

下表显示如何添加、移动、重命名和打开工程文件及文件夹。其中某些操作可能会引发自动更新，而这些更新会影响到其他文件。所有操作都可以撤消和重做。

操作	过程
查看工程文件。	在 <b>文件</b> 视图中，点击 <b>工程</b> 以仅显示工程中包含的文件和文件夹。
查看工程文件夹中的所有文件。	要显示工程文件夹中的所有文件和文件夹，请在 <b>文件</b> 视图中，点击 <b>全部</b> 。  您可能不想在工程中包含所有文件。例如，您可能要排除 SVN 或 CVS 源代码管理文件夹。有关详细信息，请参阅“处理工程中的派生文件”（第 32-42 页）。
新建工程文件夹。	在 <b>文件</b> 视图中，右键点击空白区域，然后点击 <b>新建 &gt; 文件夹</b> 。
将文件添加到工程中。	在 <b>工程</b> 选项卡的 <b>文件</b> 部分中，点击  <b>添加文件</b> 。从工程文件夹中的非托管文件列表中进行选择。  您也可以将文件和文件夹从操作系统文件浏览器或当前文件夹浏览器中粘贴或拖动到工程的 <b>文件</b> 视图中。将文件拖到 <b>文件</b> 视图时，MATLAB 会将文件添加到工程中。  要以编程方式添加文件，请使用 <b>addFile</b> 和 <b>addFolderIncludingChildFiles</b> 函数。例如，要将名为 <b>myfile.m</b> 的文件添加到 <b>proj</b> 工程对象，请键入 <code>addFile(proj,'myfile.m');</code>
删除工程文件或文件夹。	在 <b>文件</b> 视图中，右键点击该文件，然后选择 <b>从工程中删除</b> 。  要以编程方式删除文件，请使用 <b>removeFile</b> 函数。
移动工程文件或文件夹。	剪切并粘贴或拖动工程中的文件。
重命名工程文件或文件夹。	在 <b>文件</b> 视图中，右键点击文件，然后点击 <b>重命名</b> 。
打开工程文件。	在 <b>文件</b> 视图中，右键点击文件，然后点击 <b>打开</b> 。  您也可以双击该文件。
预览工程文件内容而不打开文件。	在 <b>文件</b> 视图中，选择该文件。 <b>文件</b> 视图右下角的面板会显示文件信息和标签。要在面板最小化的情况下还原面板，请点击“还原”  按钮。
删除工程文件或文件夹。	在 <b>文件</b> 视图中，右键点击该文件，然后点击 <b>删除</b> 。
撤消或重做操作	点击工具条右上角的  或  。  如果您使用源代码管理，还可以还原到文件或工程的特定版本。有关详细信息，请参阅“还原更改”（第 32-41 页）。

## 重命名、删除或移除文件时自动更新

重命名、删除或移除工程中的文件或文件夹时，工程将运行依存关系分析，以检查对其他工程文件的影响。分析完成后，工程将显示受影响的文件。如果您尚未对工程运行依存关系分析，则运行该分析可能需要一些时间。后续分析是增量更新，因此运行速度更快。

重命名文件时，您可以选择自动更新文件引用。重命名时自动更新文件引用可防止因手动更改名称或路径以及忽略或错误键入名称而导致的错误。

例如：

- 重命名类时，您可以选择自动更新从该类继承的所有类。
- 重命名 .m 或 .mlx 文件时，您可以选择自动更新调用该文件的任何文件和回调。
- 重命名 C 文件时，工程会提示您更新使用该文件的 S-Function。

有关重命名、删除或移除 Simulink 文件（如库链接、模型引用和模型回调）时自动更新的详细信息，请参阅“Automatic Updates When Renaming, Deleting, or Removing Files”（Simulink）。

## 另请参阅

### 详细信息

- “查找工程文件”（第 32-12 页）
- “将标签添加到工程文件”（第 32-16 页）
- “创建自定义任务”（第 32-18 页）

## 查找工程文件

可通过几种方法在工程中查找文件和文件夹。您可以：

- 对工程文件进行分组和排序。
- 搜索和筛选工程文件。
- 搜索工程文件中的内容。

### 对工程文件进行分组和排序

要更改文件和文件夹在**文件**视图中的分组或排序方式，请从**布局**字段和操作菜单  的选项中进行选择。

例如，要按类型对文件进行分组，请在**布局**字段中，选择**列表**。然后，点击操作按钮 ，并选择**分组依据 > 类型**。要按大小对分组的文件进行排序，请点击操作  按钮，然后选择**排序依据 > 大小**。

### 搜索和筛选工程文件

在**文件**视图和**依存关系分析**视图中，您都可以搜索工程文件和文件夹。

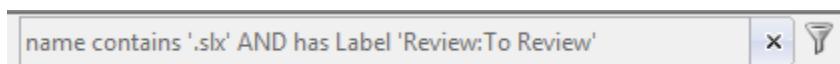
要搜索文件或文件夹，请在视图顶部的搜索字段中，开始键入文件或文件夹名称。星号字符 (\*) 是通配符。例如，如只显示以 **coll** 开头且扩展名为 .m 的文件名，请键入 **coll\*.m**。

要筛选当前视图中的文件和文件夹，请点击搜索字段旁边的“筛选器”按钮 。在“筛选器生成器”对话框中，选择要作为筛选依据的名称、文件类型、工程状态和标签。

筛选器生成器将显示生成的筛选器。例如：

```
Filter = type=='MATLAB Code files (*.m, *.mlx)' AND project status=='In project'  
AND label=='Classification:Test'
```

点击**应用**以筛选当前视图中的文件和文件夹。搜索字段将显示应用的筛选器。



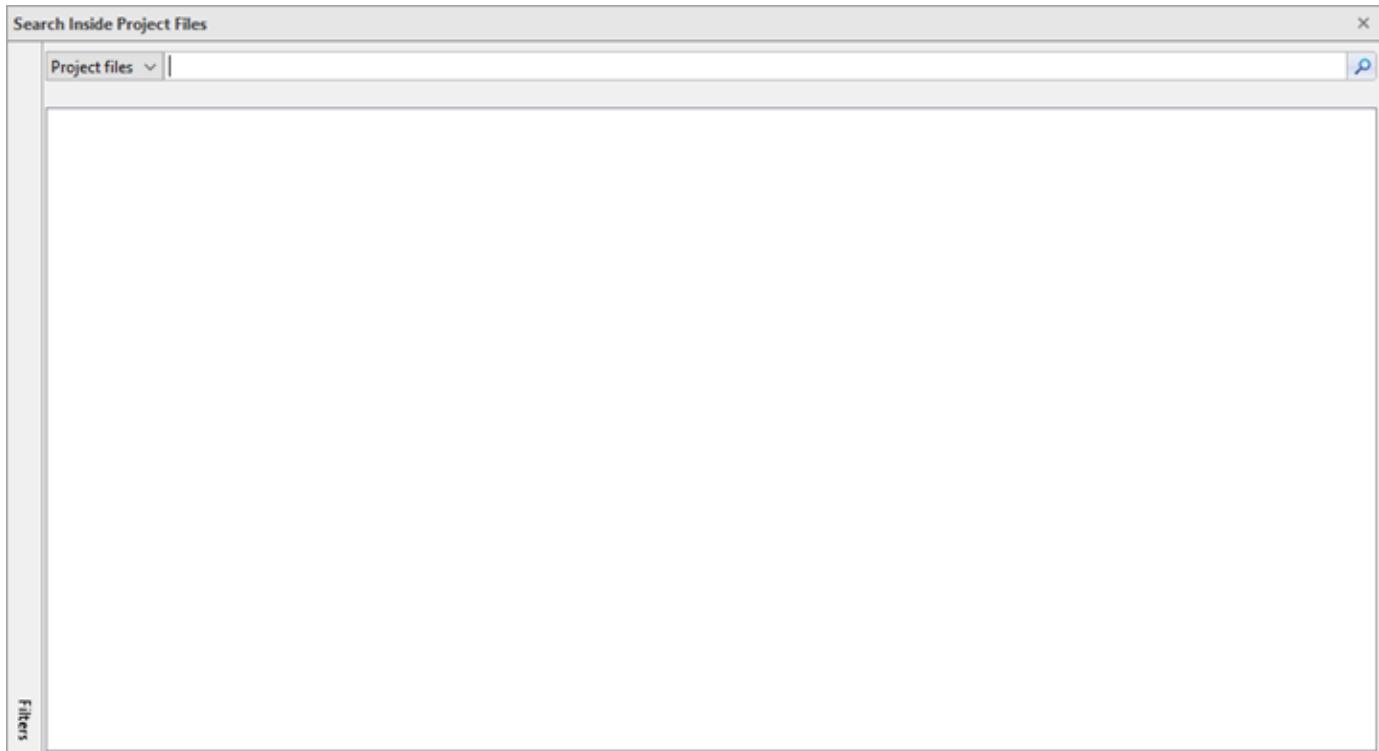
要清除搜索或筛选器，请点击搜索字段右侧的“清除”按钮 。

### 搜索工程文件中的内容

您可以在 MathWorks 产品文件（如 MATLAB 文件、Simulink 模型文件和数据字典）及一些其他类型的工程文件（如 PDF 和 Microsoft Word 文件）中搜索文本。MATLAB 只搜索工程中的文件。要搜索引用工程中的文件内容，请打开引用工程。

要搜索工程文件的内容，请执行以下操作：

- 在**工程**选项卡上，点击**搜索**。将打开“在工程文件内搜索”对话框。



- 2 输入要搜索的文本。MATLAB 会在工程文件中搜索您在搜索字段中提供的准确文本并显示结果。  
不要使用引号将短语引起来。
- 3 展开结果列表中的结果，以查看搜索窗口显示的匹配项和上下文。
- 4 双击搜索结果以打开文件并定位到匹配项。
- 5 点击结果左下角的**筛选器**，按文件类型、状态或标签优化结果。

## 另请参阅

### 详细信息

- “管理工程文件”（第 32-10 页）
- “创建常见任务的快捷方式”（第 32-14 页）
- “将标签添加到工程文件”（第 32-16 页）

## 创建常见任务的快捷方式

您可以在工程中创建快捷方式来执行常见的工程任务，例如打开重要文件和加载数据。

### 运行快捷方式

要运行快捷方式，请在**工程快捷方式**选项卡中，点击该快捷方式。点击**工程快捷方式**选项卡中的快捷方式将对该文件类型执行默认操作。例如，MATLAB 运行 .m 快捷方式文件并加载 .mat 快捷方式文件。如果快捷方式文件不在路径上，MATLAB 会将当前文件夹更改为快捷方式文件的父文件夹，运行快捷方式，然后将当前文件夹更改回原始文件夹。

或者，在**文件**视图中，您可以右键点击快捷方式文件并选择**运行**。如果脚本不在路径上，则 MATLAB 会询问您是要更改文件夹还是要将文件夹添加到路径中。

### 创建快捷方式

要从现有工程文件中创建快捷方式，请执行以下操作：

- 1 在**文件**视图中，右键点击该文件并选择**创建快捷方式**。或者，在**工程快捷方式**选项卡上，点击**新建快捷方式**并浏览以选择文件。  
将打开“创建新快捷方式”对话框。
- 2 选择一个图标并输入名称。如果使用快捷方式来定义工作流中的步骤，可以考虑在其名称前面加上数字。
- 3 要将快捷方式添加到一个现有组，请在**组**字段的下拉列表中选择一个组。有关快捷方式组的详细信息，请参阅“整理快捷方式”（第 32-14 页）。
- 4 点击**确定**。

快捷方式将以选定的名称和图标显示在**工程快捷方式**选项卡上。在**文件**视图中，状态列会显示图标 ，表示该文件是一个快捷方式。

**注意** 将修改后的文件提交到源代码管理时会包含快捷方式，因此您可以与其他工程用户共享快捷方式。

### 整理快捷方式

您可以通过将快捷方式分组来整理快捷方式。例如，您可以创建一组快捷方式用于加载数据，一组快捷方式用于打开文件，一组快捷方式用于生成代码，一组快捷方式用于运行测试，等等。

要创建快捷方式组，请执行以下操作：

- 1 在**工程快捷方式**选项卡上，点击**整理组**。
- 2 点击**创建**按钮。
- 3 输入组的名称，然后点击**确定**。

新快捷方式组将出现在**工程快捷方式**选项卡上。

要将快捷方式移至某个组中，请执行以下操作：

- 1 在**工程快捷方式**选项卡上，右键点击快捷方式并选择**编辑快捷方式**。或者，在**文件**视图中，右键点击文件并选择**编辑快捷方式**。

将打开“创建新快捷方式”对话框。

2 在**组**字段中，从下拉列表中选择一个组，然后点击**确定**。

快捷方式在**工程快捷方式**选项卡中按组来组织。

## 另请参阅

### 详细信息

- “**管理工程文件**”（第 32-10 页）
- “**查找工程文件**”（第 32-12 页）
- “**将标签添加到工程文件**”（第 32-16 页）

## 将标签添加到工程文件

您可以使用标签来整理工程文件并向工程用户传达信息。

### 添加标签

要向工程文件添加标签，请在**文件**视图中选择该文件。然后，将所需标签从工程左下角的**标签面板**拖到选定文件的“**标签编辑器**”面板中。“**标签编辑器**”面板位于**文件**视图的右下角。要在面板最小化的情况下还原面板，请点击  图标。

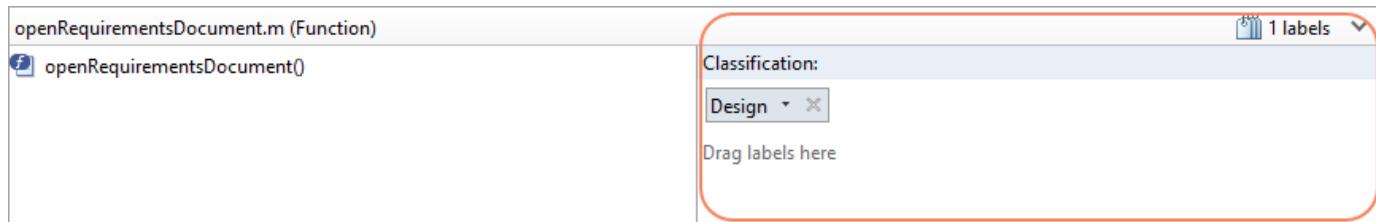
要为多个工程文件添加标签，请在**文件**或**依存关系分析**视图中选择这些文件，右键点击，然后选择**添加标签**。从列表中选择一个标签，然后点击**确定**。

**注意** 为文件添加标签后，该标签将在文件的各修订版中保持不变。

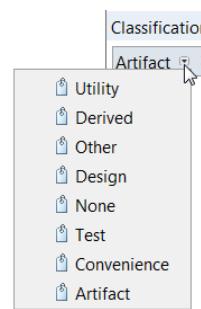
要以编程方式添加标签（例如，在自定义任务函数中），请参阅 `addLabel`。

### 查看和编辑标签数据

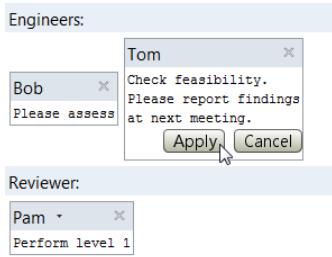
当您在**文件**视图中选择工程文件时，文件标签会出现在“**标签编辑器**”视图中。



要更改属于单值类别的标签，请从标签列表中选择新值。



您可以从创建的类别中向标签添加附加注释。在“**标签编辑器**”面板中，点击标签并插入或修改文本。然后，点击**应用**。



## 创建标签

标签有两个类别：

- **单值** - 只能将该类别中的一个标签附加到一个文件。
- **多值** - 可以将该类别中的多个标签附加到一个文件。

所有工程都包含一个名为**分类**的内置标签类别，其中包含几个内置标签。这些内置标签为只读标签。

要创建您自己的标签类别，请执行以下操作：

- 1 在工程左下角的**标签**面板中，右键点击并选择**创建新类别**。将打开“创建类别”对话框。
- 2 输入新类别的名称。
- 3 要创建单值标签类别，请选中**单值**复选框。否则，MATLAB 将创建多值标签类别。
- 4 要指定默认“字符串”数据类型以外的标签数据类型，请从**类型**列表的可用选项中进行选择。
- 5 点击**创建**。

要在标签类别中创建您自己的标签，请执行以下操作：

- 1 在工程左下角的**标签**面板中，右键点击标签类别，然后选择**创建新标签**。将打开“创建标签”对话框。
- 2 输入新标签的名称，然后点击**确定**。

要重命名或删除类别或标签，请右键点击该类别或标签，然后选择**重命名或删除**。

要以编程方式创建新标签或标签类别，请参阅 `createLabel` 或 `createCategory`。

## 另请参阅

### 详细信息

- “管理工程文件” (第 32-10 页)
- “查找工程文件” (第 32-12 页)

## 创建自定义任务

自定义任务是允许您对一个或多个文件执行一系列操作的 MATLAB 函数。您可以创建自定义任务函数，然后对工程中的一组选定文件运行自定义任务。例如，您可以创建自定义任务来检查所有代码文件中的错误或运行工程中的所有测试。

### 创建自定义任务函数

要创建自定义任务函数，请执行以下操作：

- 1 在工程选项卡中，点击**自定义任务**，然后选择**管理自定义任务**。将打开“管理自定义任务”对话框。
- 2 点击**添加**，然后选择**使用新函数添加**。如果要将现有脚本添加为自定义任务，请选择**使用现有函数添加**。
- 3 指定脚本的文件名，并将新文件保存到 MATLAB 路径上。MATLAB 编辑器将打开包含示例自定义任务函数的新文件。
- 4 编辑该函数以对每个文件执行所需的操作。使用文件顶部的说明来指导您创建具有正确函数签名的自定义任务。自定义任务必须接受文件的完整路径作为单个输入参数，并返回单个输出参数。

例如，以下自定义任务函数使用 `checkcode` 函数提取每个文件的代码分析器信息。

```
[~,~,ext] = fileparts(file);
switch ext
    case {' .m', '.mlx', '.mlapp'}
        result = checkcode(file, '-string');
    otherwise
        result = [];
end
```

- 5 保存文件。

您可以使用 MATLAB 编辑器来设置断点和调试自定义任务函数，就像对任何其他 MATLAB 函数一样。

### 运行自定义任务

要对工程中的一组选定文件运行自定义任务，请执行以下操作：

- 1 在工程选项卡上，点击**自定义任务**，然后选择**运行自定义任务**。
- 2 在表的**包括**列中，选择要对其运行自定义任务的工程文件。

要在表中一次包括或排除多个文件，请按住 **Shift** 或 **Ctrl** 键，选择文件，然后右键点击并选择**包括**或**排除**。如果自定义任务函数可以识别要对其进行操作的文件，则请包括所有文件。

- 3 在**自定义任务**字段中，从可用的自定义任务函数中进行选择。您也可以直接在字段中输入任务的名称，或者点击**浏览**。
- 4 点击**运行任务**以运行任务。“自定义任务报告”窗口将显示结果。
- 5 检查表中的**结果**列，以确保对所有文件正确运行了自定义任务。要查看文件的详细结果信息，请在表中选择该文件。“自定义任务报告”底部的结果窗格将显示详细信息。

### 保存自定义任务报告

如果您要保存自定义任务结果的记录，或者要与其他人共享结果，则保存自定义任务报告非常有用。

要保存自定义任务报告，请点击“自定义任务报告”底部的**发布报告**按钮。您可以将报告保存为 HTML 文件或 Microsoft Word 文件。如果您有 MATLAB Report Generator™，也可以将报告保存为 PDF 文件。

要查看报告文件并将其添加到工程中，请切换到**所有文件**视图。

## 另请参阅

### 详细信息

- “[创建常见任务的快捷方式](#)”（第 32-14 页）
- “[管理工程文件](#)”（第 32-10 页）

## 大型工程组件化

MATLAB 允许您从父工程引用其他工程，从而支持将大型工程组件化。将大型工程组织为若干组件有助于代码重用、模块化和基于团队的开发、单元测试以及组件的独立发布。

工程可以以分层方式引用多个其他工程。工程引用层次结构在[引用视图](#)中显示为树。

从父工程中，您可以

- 访问所有引用工程的工程路径、入口函数快捷方式和源代码管理信息。
- 查看、编辑和运行属于引用工程的文件。
- 使用检查点检测引用工程中的更改。

### 添加或删除对工程的引用

可以通过引用其他工程将新组件添加到工程中。

要添加对工程的引用，请执行以下操作：

1



在工程选项卡上的环境部分中，点击 **引用**。将打开“添加引用”对话框。

2 浏览以选择所需的工程 (.prj) 文件。

3 在**引用类型**字段中，选择**相对或绝对**。如果您的工程层次结构具有相对于工程根目录明确定义的根，请选择**相对**。例如，工程根目录可能是源代码管理下的文件夹。如果要引用的工程位于您的计算机可访问的位置，例如网络驱动器，请选择**绝对**。

4 要在添加工程时创建检查点，请选择**设置检查点以用于检测将来的更改**。有关检查点的详细信息，请参阅“[使用检查点管理引用工程中的更改](#)”（第 32-21 页）。

5 点击**添加**。

加载引用工程时，MATLAB 将引用工程路径添加到 MATLAB 搜索路径，然后运行或加载指定的启动文件。同样，当引用工程关闭时，MATLAB 会从搜索路径中删除工程路径，并运行指定的关闭文件。MATLAB 将先加载引用工程，再加载其父工程。这样父工程就可以在启动和关闭文件中访问引用工程。

要从工程层次结构中删除引用工程，请在**引用**树中右键点击引用工程，然后选择**删除引用**。

### 查看、编辑或运行引用工程文件

如果您的工程引用其他工程，则可以直接从父工程查看、修改或运行属于引用工程的文件。

要查看引用工程，请在父工程中选择**引用**视图。在**引用**树中，选择一个引用工程。

要显示引用工程文件，请在**引用**视图的右上角，点击**显示文件**。

要修改或运行文件，请右键点击该文件，然后从可用选项列表中进行选择。

### 提取文件夹以创建引用工程

您可以提取工程中的现有文件夹来创建引用工程。提取文件夹后，仍可从父工程访问引用工程中的文件和文件夹内容以及快捷方式。

要从工程中提取文件夹并将该文件夹转换为引用工程，请执行以下操作：

- 1 在**文件**视图中，右键点击文件夹，然后选择**提取到引用工程**。将打开“将文件夹提取到新工程”对话框。
- 2 指定工程名称和位置
- 3 在**引用类型**字段中，选择**相对**或**绝对**。如果通过引用当前工程根目录指定新工程位置，请选择**相对**。如果指定新位置的完整路径，例如在网络驱动器上，请选择**绝对**。
- 4 要禁用任何默认的内容迁移操作，请点击**更多选项**，并清除对应的复选框。
- 5 点击**提取**。
- 6 在两个打开的“警告”对话框中，点击**确定**。

选定的文件夹及其内容将从工程中删除。在**工程快捷方式**选项卡上，**引用工程**部分显示引用工程的新快捷方式。

## 使用检查点管理引用工程中的更改

要检测和比较引用工程中的更改，请创建检查点。然后，您可以将引用工程与检查点进行比较来检测更改。

默认情况下，当您添加对工程的引用时，MATLAB 会创建一个检查点。要创建其他检查点，请执行以下操作：

- 1 在引用工程的父工程中，选择**引用**视图。
- 2 要创建检查点，请转至**引用**选项卡，在**检查点**部分中，点击  **更新**。在**详细信息**视图中，**检查点**字段显示最新检查点的时间戳。

要检测引用工程中的更改，请转至**引用**选项卡，在**检查点**部分中，点击  **检查点报告**。“与检查点的差异”对话框显示自创建该检查点以来在磁盘上已发生更改的文件。

要删除检查点，请在**引用**选项卡的**检查点**部分中，点击  **清除**。

## 另请参阅

### 详细信息

- “**创建工程**”（第 32-2 页）
- “**共享工程**”（第 32-22 页）

## 共享工程

您可以通过打包和共享工程与他人协作。您还可以使用标签和导出配置文件来控制共享工程中包含哪些文件。（有关详细信息，请参阅“创建导出配置文件”（第 32-24 页）。）

下表说明打包工程的方式。

共享工程的方式	过程
创建一个存档以进行共享。	<p>您可以将工程转换为 .mlproj 或 ZIP 存档，并与协作者共享该存档。与无法访问所连接的源代码管理工具的人员一起工作时，共享存档非常有用。</p> <p>要存档工程，请执行以下操作：</p> <ol style="list-style-type: none"> <li>1 加载工程后，在<b>工程</b>选项卡上，点击<b>共享 &gt; 存档</b>。</li> <li>2 如果只想导出一组指定的文件，请选择<b>导出配置文件</b>。有关详细信息，请参阅“创建导出配置文件”（第 32-24 页）。</li> <li>3 如果您引用了工程并且想导出引用工程文件，请选中<b>包括引用工程</b>。</li> <li>4 点击<b>另存为</b>并指定文件路径。</li> <li>5 在<b>保存类型</b>字段中，选择存档的工程文件类型。默认情况下，MATLAB 将工程存档为 .mlproj 文件。您可以选择将工程存档为 ZIP 文件。</li> <li>6 点击<b>保存</b>以创建工程存档。</li> </ol> <p>现在，您可以像共享任何其他文件一样共享存档文件。</p>
一步通过电子邮件发送 .mlproj 存档。	<p>在 Windows 系统上，您可以一步生成 .mlproj 存档并将其附加到电子邮件中。与无法访问所连接的源代码管理工具的人员一起工作时，共享存档非常有用。</p> <p>要以 .mlproj 文件的形式通过电子邮件发送工程，请执行以下操作：</p> <ol style="list-style-type: none"> <li>1 加载工程后，在<b>工程</b>选项卡上，选择<b>共享 &gt; 电子邮件</b>。</li> <li>2 如果只想导出一组指定的文件，请选择<b>导出配置文件</b>。有关详细信息，请参阅“创建导出配置文件”（第 32-24 页）。</li> <li>3 如果您引用了工程并且想导出引用工程文件，请选中<b>包括引用工程</b>复选框。</li> <li>4 点击<b>附加到电子邮件</b>。MATLAB 会在默认电子邮件客户端中打开一封新电子邮件，该电子邮件附有一份 .mlproj 文件。</li> <li>5 编辑并发送该电子邮件。</li> </ol>

共享工程的方式	过程
创建一个工具箱以进行共享。	<p>您可以从您的工程创建一个工具箱，并与协作者共享该工具箱。</p> <p>要从工程创建工具箱，请执行以下操作：</p> <ol style="list-style-type: none"><li><b>1 加载工程后，在工程选项卡上，选中共享 &gt; 工具箱。</b></li><li><b>2 工具箱信息字段填充有工程名称、作者和描述。根据需要编辑信息。</b></li><li><b>3 要包括尚未包含在工程文件中的文件，请编辑排除的文件和文件夹。</b></li><li><b>4 点击打包。</b></li></ol> <p>现在，您可以像共享任何其他文件一样共享工具箱文件。</p>

共享工程的方式	过程
在 GitHub® 上发布。	<p>您可以通过在 GitHub 上公开发布工程来共享工程。首先，您必须有 GitHub 帐户。</p> <p>在 GitHub 上共享工程会将 Git 源代码管理添加到工程中。如果您的工程已在源代码管理之下，共享会用 Git 取代源代码管理配置，GitHub 将成为工程的远程存储库。</p> <p><b>注意</b> 如果您不想更改所打开工程中的当前源代码管理，请以其他方式共享该工程。</p> <p>要在 GitHub 上共享工程，请执行以下操作：</p> <ol style="list-style-type: none"> <li>加载工程后，在工程选项卡上，选中<b>共享 &gt; GitHub</b>。将打开“创建 GitHub 存储库”对话框。 如果<b>共享</b>菜单中未包含<b>GitHub</b>选项，请选择<b>共享 &gt; 更改共享选项</b>。然后，在“管理共享”对话框中，选择<b>GitHub</b>并点击<b>关闭</b>。</li> <li>输入您的 GitHub 用户名和密码，并编辑新存储库的名称。点击<b>创建</b>。 将出现一则警告，提示您确认是否要创建公共存储库并修改当前工程的远程存储库位置。要继续，请点击<b>是</b>。</li> <li>“创建 GitHub 存储库”对话框将显示新存储库的 URL。点击该链接可在 GitHub 网站上查看新存储库。存储库包含工程文件的初始签入。</li> <li>当前工程中的源代码管理现在引用 GitHub 上的新存储库作为远程存储库。要使用具有新存储库的工程，请在“创建 GitHub 存储库”对话框中点击<b>重新加载工程</b>。</li> </ol> <p>要查看远程存储库的 URL，请在工程选项卡的<b>源代码管理</b>部分中，点击<b>Git 详细信息</b>按钮。</p> <p>您无需进行其他安装即可使用 Git。要在 Git 中执行合并操作，需要额外的设置步骤。有关详细信息，请参阅“设置 Git 源代码管理”（第 32-23 页）。</p>
使用源代码管理进行协作。	您可以在工程中使用源代码管理进行协作。有关详细信息，请参阅“对工程使用源代码管理”（第 32-35 页）。

在与其他人共享工程之前，通过执行依存关系分析来检查工程所需的工具箱会很有用。有关详细信息，请参阅“查找必需的工具箱”（第 32-31 页）。

## 创建导出配置文件

如果要在共享工程之前从工程中排除文件，请创建导出配置文件。导出配置文件允许您排除具有特定标签的文件。有关创建标签并将其添加到工程文件的详细信息，请参阅“创建标签”（第 32-17 页）。

要创建导出配置文件，请执行以下操作：

- 1 在**工程**选项卡上，点击**共享 > 管理导出配置文件**。
- 2 点击**+**，并指定导出配置文件的名称。
- 3 在**文件**窗格中，点击**+**，为不想导出的文件选择标签，然后点击**确定**。
- 4 在**标签**窗格中，点击**+**，选择不想导出的自定义标签，然后点击**确定**。
- 5 点击**应用**，并关闭“管理导出配置文件”对话框。

**注意** 导出配置文件不会应用对引用工程的更改。当您共享工程时，MATLAB 会导出整个引用工程。

## 另请参阅

### 详细信息

- “将标签添加到工程文件”（第 32-16 页）
- “创建工程”（第 32-2 页）
- “分析工程依存关系”（第 32-29 页）

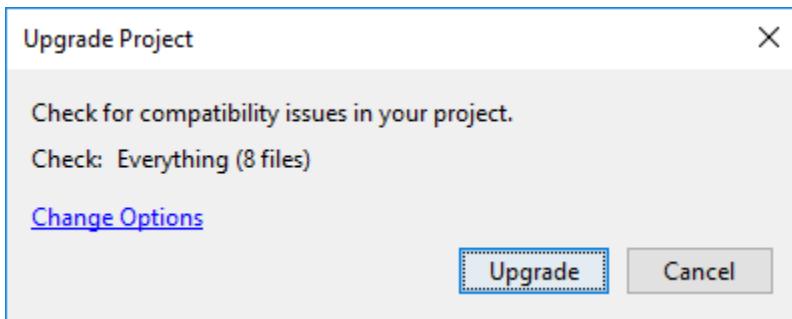
## 升级工程

升级工程工具可帮助您检查兼容性问题，或将工程升级到当前 MATLAB 版本。该工具会尽可能自动应用修复并生成报告。

**提示** 要执行以后可还原的升级，请在升级之前在工程中添加源代码管理。有关详细信息，请参阅“对工程使用源代码管理”（第 32-35 页）。

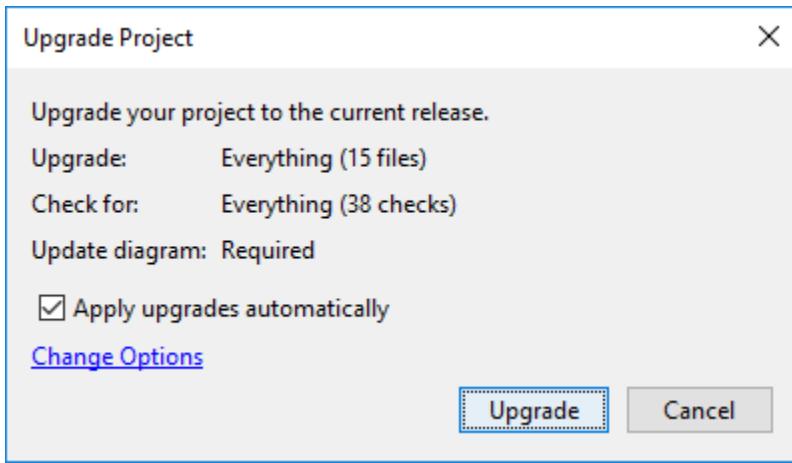
### 运行升级工程工具

- 对于仅包含 MATLAB 文件的工程，您可以检查与当前 MATLAB 版本的兼容性问题。在**工程**选项卡上，**点击运行检查 > 升级**。



点击**升级**以检查兼容性问题。

- 对于同时还包含 Simulink 模型和库的工程，您可以应用修复，并自动将工程升级到当前版本。在**工程**选项卡上，**点击运行检查 > 升级**。



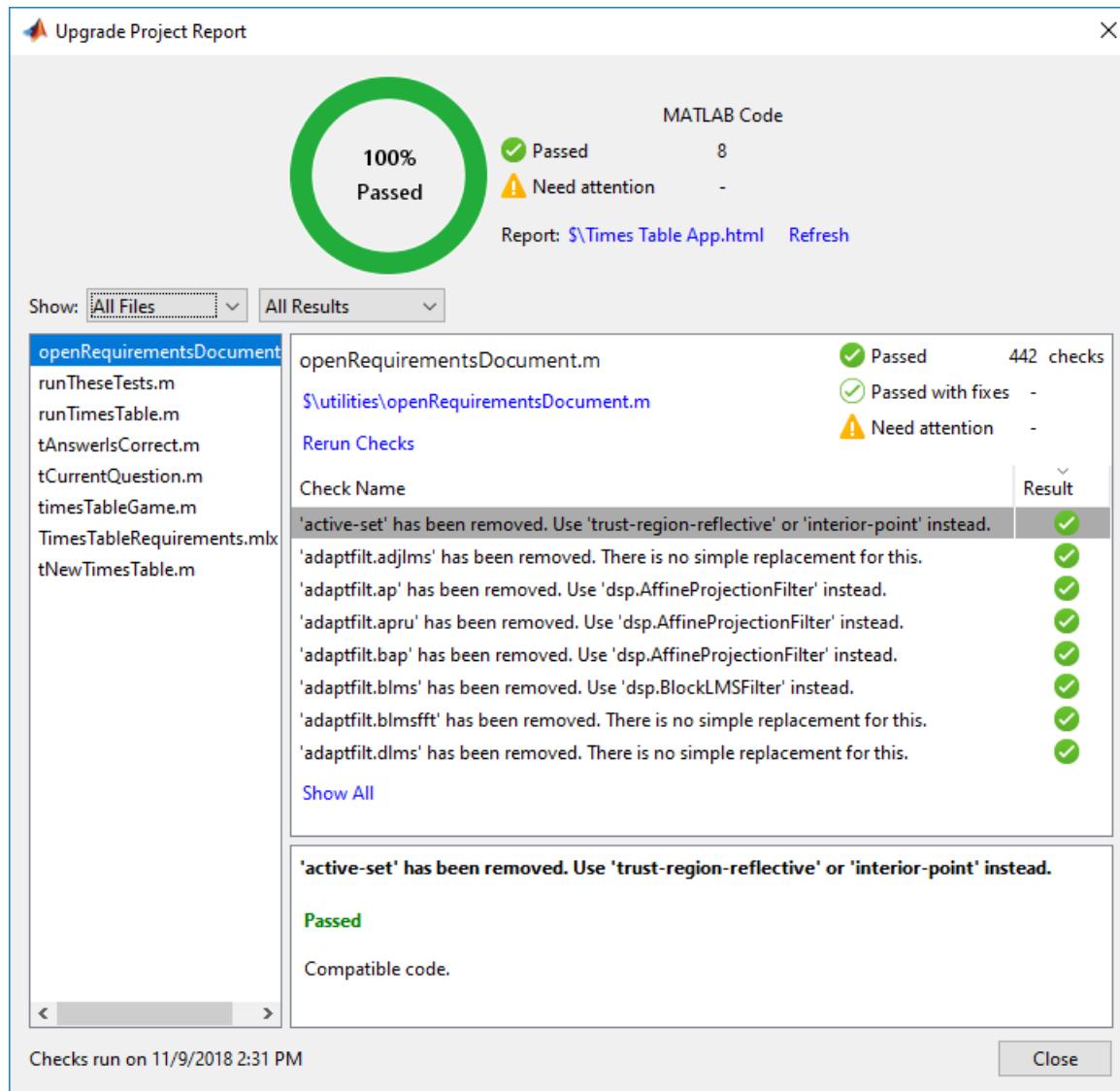
如果您要运行工程升级工具而不自动应用修复，请不要选中**自动应用升级**。

如果要指定升级哪些文件和运行哪些检查，请**点击更改选项**。在“升级选项”对话框中，清除要从升级中排除的任何文件或检查的复选框。例如，您可能要排除需要执行更新图的检查，因为这可能很耗时。

升级工程工具在升级工程报告中显示兼容性检查或升级的结果。

## 检查升级工程报告

升级工程后，请检查升级工程报告以确保升级已按计划进行。



摘要显示有多少文件通过了所有升级检查，有多少文件需要注意。要查看某个文件的升级结果，请在左侧文件列表中选中该文件。默认情况下，文件列表显示需要注意的所有文件。要更改显示哪些文件，请从**显示**下拉列表的选项中进行选择。

对于工程升级报告中的每个文件，请认真查看标记为需要注意的检查。这些文件在**结果**列中用橙色圆圈图标来标记。在**检查名称**列中选择一个检查，以在下方面板中显示检查结果和任何自动应用的修复。

要查看升级前后的差异，请在升级工程报告中，点击**查看更改**。如果您的工程在源代码管理之下，您还可以通过比较文件的前后版本来查看差异。

MATLAB 会将升级结果以 HTML 报告的形式保存在工程根文件夹中。要打开保存的报告，请点击“升级工程报告”顶部的**报告**链接。

### 另请参阅

#### 详细信息

- “对工程使用源代码管理” (第 32-35 页)
- “分析工程依存关系” (第 32-29 页)

# 分析工程依存关系

要分析工程的结构并发现工程所需的文件，请运行依存关系分析。当您要检查工程是否所有必需的文件时，可以在工作流中的任意点运行依存关系分析。例如，在设置工程时、在向源代码管理提交工程版本之前以及在共享工程之前，我们都建议您检查依存关系。

运行依存关系分析可以在您进行更改之前向您显示更改会对其他文件产生怎样的影响。例如，我们经常使用依存关系分析来检查工程结构，查找、调查和解决工程文件中的问题，或者查找工程所需的工具箱。当调查需求变化的潜在影响时，可使用依存关系分析来查找链接到需求文档的设计文件。您还可以在提交更改之前查找修改后的文件的上游和下游依存关系，从而使用依存关系来调查更改集的影响。找到这些依存关系可以帮助您识别需要修改的设计和测试文件，并帮助您找到需要运行的测试。

在进行依存关系分析后，您可以打开文件或对文件进行标记。您也可以将结果导出为工作区变量、图像或可重新加载的文件。您还可以发送这些文件以用于自定义任务处理。导出结果允许您进一步处理或存档影响分析结果。您可以将导出的文件列表添加到描述更改影响的报告或其他文档中。

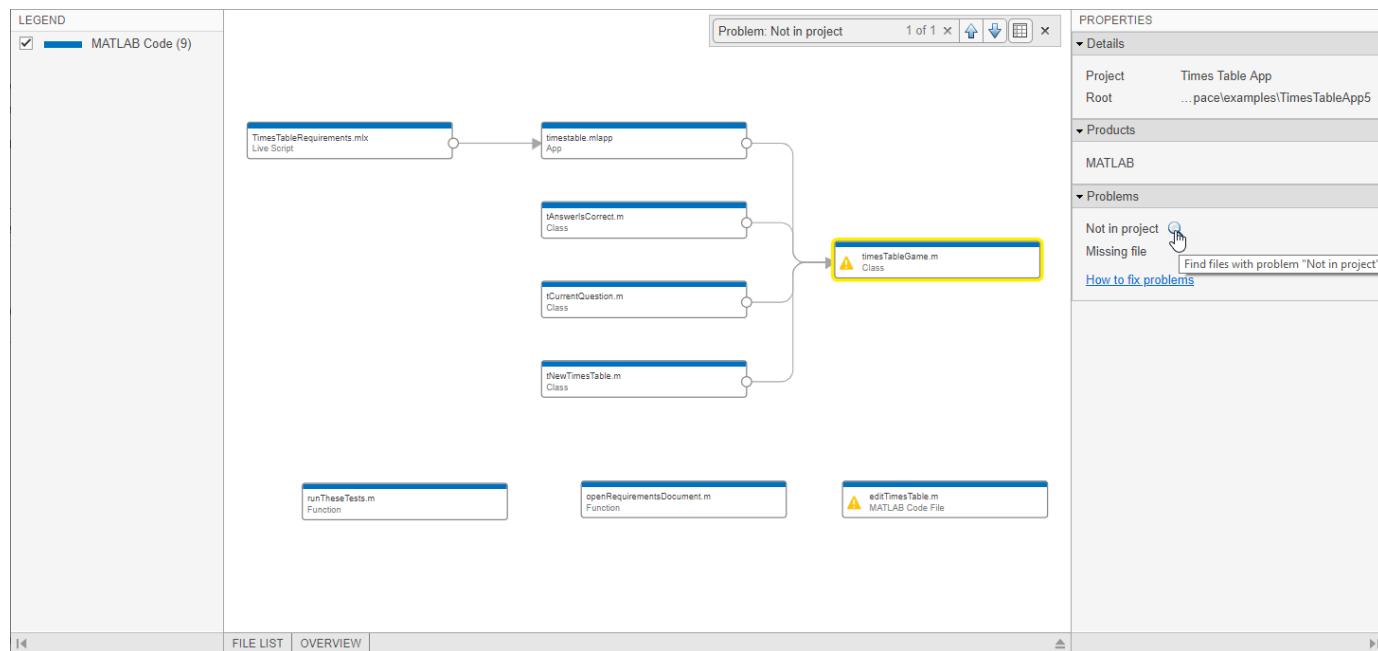
## 运行依存关系分析

在对工程运行依存关系分析之前，请确保已将所有文件添加到工程中。依存关系分析仅检查工程中的文件。有关详细信息，请参阅“将文件添加到工程”（第 32-5 页）。

要对工程中的所有文件运行依存关系分析，请选择**依存关系分析**视图。然后，在**依存关系分析**选项卡上， **分析**。

要仅分析特定文件，请在**依存关系分析**选项卡上，**点击分析 > 选择要分析的文件**。选择要分析的文件，然后**点击分析**。

要分析外部工具箱的依存关系，请在**依存关系分析**选项卡上，**点击选项 > 分析外部工具箱**。



执行依存关系分析后，**影响视图**会显示：

- 您的工程结构及其文件依存关系，显示模型、库、函数、数据文件、源文件和派生文件等文件之间的相互关系。
- 所需的产品和工具箱。
- 源文件和派生文件（如 .m 和 .p 文件、.slx 和 .slxp、.ssc 和 .sscp 或 .c 和 .mex 文件）之间的关系，以及 C/C++ 源文件和头文件之间的关系。您可以查看每个模型生成的代码，并查找在修改模型后需要重新生成哪些代码。
- 关于问题文件的警告，例如缺失文件、不在工程中的文件、未保存更改的文件以及过期的派生文件。

要检查工程依存关系和问题文件，请在**依存关系分析**视图的右上角，点击**表视图**。

对工程运行第一次依存关系分析后，后续分析为增量更新。但是，如果您更新了外部工具箱并希望发现其中的依存关系变化，则必须执行完整分析。要执行完整分析，请转至**依存关系分析**选项卡，然后点击**分析 > 全部重新分析**。

有关对 Simulink 模型和库运行依存关系分析的详细信息，请参阅“Perform an Impact Analysis”(Simulink)。

## 调查和解决问题

运行依存关系分析后，整个工程的影响图将出现在**影响视图**中。工程依存关系分析会确定一些问题，例如缺失文件、不在工程中的文件、未保存的更改以及过期的派生文件。您可以使用**影响视图**或**表视图**检查问题文件。

使用**影响视图**以图形方式调查问题文件。视图的中心显示工程的影响图。**影响视图**的右窗格显示工具箱依存关系和问题文件列表。

- 要仅显示图中的问题文件，请将鼠标悬停在**影响视图**右窗格中的**问题标题**上，然后点击**查找全部**。要返回完整的工程视图，请清除搜索框中的筛选器（例如，**Dependencies of "filename"**）。
- 要查看特定问题文件的详细信息，包括路径、类型和问题消息，请在图中选择该文件。要清除文件选择并查看工程的所有问题文件，请点击图的空白区域。
- 要查看文件的依存关系，请将鼠标悬停在问题文件上，然后点击**查找全部**。图会更新以仅显示问题文件及其依存关系。要返回完整的工程视图，请清除搜索框中的筛选器（例如，**problem=="\*"**）。

使用**表视图**以表的形式调查工程依存关系和问题文件。要使用**表视图**，请在**依存关系分析**视图的右上角，选择**表视图**。

- 要在列表顶部查看有问题的文件，请按**问题说明**列对表进行排序。
- 要查看文件的问题，请查看**问题说明**列中针对该文件的消息。
- 要查看文件的依存关系，请在表中选择该文件。下窗格显示依赖于所选文件的文件。您可以查看**直接受影响的文件**，或者切换到**直接需要的文件**。

## 解决问题

对于每个问题文件，请采取相应措施以解决问题。下表列出了常见问题，并介绍了如何修复这些问题。

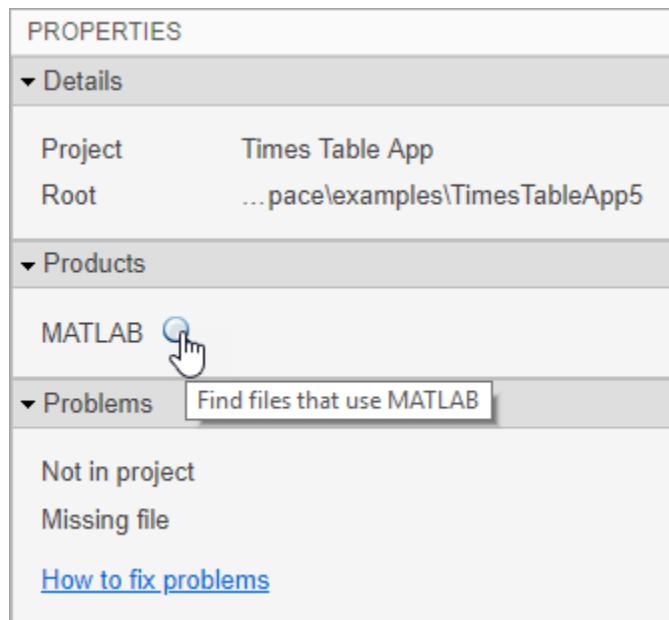
问题消息	说明	修复
不在工程中	文件不在工程中。	<p>将文件添加到工程中。</p> <p>您不需要将所有必需的文件都添加到工程中。例如，您可以排除工程中源代码生成的派生 S-Function 二进制文件。有关详细信息，请参阅“将文件添加到工程”(第 32-5 页)。</p> <p>要从问题列表中删除文件而不将其添加到工程中，请右键点击该文件并选择<b>添加外部文件</b>。</p>
缺失工程文件	文件在工程中，但不在磁盘上。	创建该文件，或使用源代码管理恢复该文件。
在工程根目录外部	文件在工程根文件夹之外。	<p>如果可以，请将该文件设置为外部文件。否则，请将其移到工程根目录下。</p> <p>如果您需要将工程根目录之外的文件包括在工程中，请将该文件复制或移动到工程根目录，并将其添加到工程和路径。从路径中删除原始文件位置。</p> <p>如果所需文件在工程根目录之外，则无法将这些文件添加到工程中。如果文件位于您的路径上，并且是不属于工程的实用工具或资源，则此依存关系可能不会指示问题。使用依存关系分析确保您了解设计依存关系。</p>
在未引用的工程中	文件位于当前工程未引用的工程中。	将包含该文件的工程添加为工程引用。
缺失文件	找不到文件或变量。	如果此状态可接受，请右键点击该文件，然后选择 <b>添加外部文件</b> 。
未保存的更改	文件在 MATLAB 编辑器中有未保存的更改。	保存文件。
派生文件已过期	派生的文件比其源文件更早。	<p>重新生成派生文件。如果它是 .p 文件，您可以通过运行工程检查自动重新生成它。在<b>工程</b>选项卡上，点击<b>检查工程</b>，并按照提示重新编译文件。</p> <p>如果重命名源文件，工程将检测对派生文件的影响，并提示您更新它。</p>
缺失产品	工程所依赖的产品缺失。	安装缺失产品。

## 查找必需的工具箱

对工程运行依存关系分析后，**影响视图**将显示整个工程或选定文件所需的工具箱。您可以看到需要哪些产品才能使用该工程，或查明哪个文件正在引入产品依存关系。

在**依存关系分析**视图的右窗格中，**产品**部分显示整个工程所需的产品。要查看选定文件所需的产品，请点击图或图例来选择一些文件。

要查找哪个文件正在引入产品依存关系，请将鼠标悬停在产品名称上，然后点击**查找使用情况**。



图会更新以仅显示使用选定产品的文件。要返回完整的工程视图，请清除搜索框中的筛选器（例如，**Usages of "productname"**）。

如果所需的产品缺失，产品列表会将其标记为**缺失**。要解决产品缺失问题，请安装该产品并重新运行依存关系分析。

## 查找文件依存关系

要在运行依存关系分析后调查某文件的依存关系，请在**影响视图**中，右键点击图中的该文件，然后选择**查找所有依存关系**、**查找受影响的文件**或**查找需要的文件**。该图显示选定的文件和文件依存关系。

或者，选择该文件，转至**依存关系分析**选项卡，在**影响分析**部分中，点击**查找**。从可用选项中进行选择。

要选择文件组，请在**影响视图**左上角的**文件类型**图例中，从可用选项中进行选择。例如，要选择所有函数文件，请点击**文件类型**，然后选择**函数**。要更改视图中文件的分组方式，请转至**依存关系分析**选项卡，在**视图**部分中，点击**分组依据**。从可用选项中进行选择。

要选择修改后的文件、问题文件或外部文件，请转至**依存关系分析**选项卡，在**影响分析**部分中，点击**选择**。从可用选项中进行选择。

要检查或编辑文件依存关系，请点击图中文件名称旁边的箭头来展开文件。然后，双击引用组件（如引用问题文件的 MATLAB 代码行）将其打开。要展开图中的所有文件，请在**视图**部分中，点击**全部展开**。

要重置图以显示工程中所有分析的依存关系，请在**依存关系分析**选项卡上的**影响分析**部分中，选择**查找 > 所有文件**。

**提示** 对于大型工程，在表中查看结果可以使导航更轻松。要使用**表视图**，请在**依存关系分析**视图的右上角，选择**表视图**。

## 查找需求文档

在工程中，依存关系分析会查找使用需求管理接口链接的需求文档。

- 您可以查看和浏览链接的需求文档。
- 仅当您有 Simulink Requirements™ 时，才能创建或编辑需求管理链接。

要在图中突出显示需求文档，请在**影响视图**的右上角，点击**依存关系类型**图例，然后选择**需求链接**。箭头通过需求链接将需求文档连接到文件。

要查找特定文件的需求链接，请点击图中文件名旁边的箭头以展开该文件。要展开图中的所有文件，请在**视图部分**中，点击**全部展开**。箭头将包含需求链接的组件连接到需求文档。

要打开需求文档，请双击图中的文档。

## 保存、打开和比较依存关系分析结果

在工程中运行依存关系分析后，可以保存分析结果。然后，您可以打开并查看结果，而不必重复分析。

要将结果保存为 **.graphml** 文件，请转至**依存关系分析**选项卡，在**文件部分**中，点击**另存为**。选择文件名和位置。

要打开保存的依存关系分析结果，请在**依存关系分析**选项卡上的**文件部分**中，点击**打开**。

要将当前依存关系分析的结果与以前保存的结果进行比较，请转至**依存关系分析**选项卡，在**文件部分**中，点击**与保存版比较**。选择一个 **.graphml** 文件并检查比较报告中的差异。

### 在影响视图中导出文件

要导出依存关系分析的**影响视图**中显示的所有文件，首先点击图的背景以清除对所有文件的选择。然后，转至**依存关系分析**选项卡，在**影响分析部分**中，点击**导出**。从可用选项中选择：

- **保存到工作区** - 将选定的文件路径保存到变量。
- **发送到自定义任务** - 打开“自定义任务”对话框且这些文件处于选定状态。
- **在文件视图中显示** - 切换到**文件**视图且这些文件处于选定状态。

要导出图中的部分文件，请选择所需的文件，然后点击**导出**。菜单会显示选择了多少个文件：**所选文件数：文件数**。您也可以右键点击选定的文件，并从**导出**菜单的选项中进行选择。

### 将图导出到图像文件

要将**影响视图**图导出为图像文件以进行共享或存档，请转至**依存关系分析**选项卡，在**文件部分**中，选择**另存为 > 另存为图像**。使用“保存”对话框指定名称、文件类型和位置。默认文件类型为 SVG，它支持图像缩放。

您也可以使用键盘将图像复制到剪贴板。然后，您可以将剪贴板内容粘贴到其他文档中。

## 另请参阅

### 详细信息

- “共享工程”（第 32-22 页）

- “对工程使用源代码管理” (第 32-35 页)

## 对工程使用源代码管理

您可以使用工程处理源代码管理下的文件。源代码管理将文件存储在存储库中，允许您签出存储库中的文件以进行处理，并返回存储库中以保留更改，还允许您查看已签入的不同文件版本的历史记录。有关在 MathWorks 产品中使用源代码管理的详细信息，请参阅“关于 MathWorks 源代码管理集成”（第 33-2 页）。

工程集成了两个源代码管理系统，即 Git 和 Subversion (SVN)。

要使用源代码管理设置工程，请使用以下任一工作流：

- 从现有存储库中创建新工程。
- 将现有工程添加到源代码管理中。
- 在已在源代码管理下的文件夹中创建新工程。
- 为新工程或现有工程创建一个新 GitHub 存储库。

然后，当您的工程在源代码管理下时，您可以从 MATLAB 内执行操作，例如签入和签出文件、运行检查以及提交和还原更改。

## 设置源代码管理

可以通过四种方法来使用源代码管理设置工程。

### 从现有存储库中创建新工程

通过从源代码管理检索文件来从现有存储库创建工程的本地新副本。您可以克隆一个 Git 存储库，或者从 SVN 存储库中签出文件，或者使用其他源代码管理集成。

要从现有存储库中创建新工程，请执行以下操作：

- 1 在**主页**选项卡上，点击**新建 > 工程 > 从 Git 或 新建 > 工程 > 从 SVN**。将打开“从源代码管理新建工程”对话框。
- 2 如果您知道存储库位置，请将其粘贴到**存储库路径**字段中。

否则，要浏览并验证从中检索文件的存储库路径，请点击**更改**。

- a 在对话框中，通过在字段中输入或粘贴 URL、从最近使用的存储库列表中选择或点击  按钮，指定存储库 URL。
- b 点击**验证**以检查存储库路径。如果路径无效，请对照您的源代码管理存储库浏览器检查 URL。
- c 如果您看到存储库的身份验证对话框，请输入登录信息以继续。
- d 如有必要，请选择存储库树中较深的文件夹。在 SVN 中，您可能希望从 `trunk` 或 `tags` 下的分支文件夹中签出。
- e 当您完成指定希望检索的 URL 路径后，请点击**确定**。对话框关闭，您会返回到“从源代码管理新建工程”对话框。
- 3 在**沙盒**字段中，选择工作文件夹，以将检索到的用于新工程的文件放入该文件夹中。在 SVN 中，使用本地文件夹可以获得最佳效果，因为使用网络文件夹速度较慢。
- 4 点击**检索**。

如果存储库已包含工程，则当工具完成将文件检索到所选沙盒文件夹时，该工程即准备就绪。

如果沙盒中尚未包含工程，则会出现对话框询问您是否在文件夹中创建工程。要创建工程，请指定工程名称，然后点击**确定**。将出现欢迎屏幕，帮助您设置新工程。有关设置工程的详细信息，请参阅“[设置工程](#)”（第 32-2 页）。

如果您在克隆大型 Git 存储库时遇到 **OutOfMemoryError: Java heap space** 之类的错误，请编辑您的 MATLAB 预设项以增加堆大小。

- 1 在[主页](#)选项卡的**环境**部分，点击**预设**。
- 2 选择 **MATLAB > 常规 > Java 堆内存**。
- 3 移动滑块以增加堆大小，然后点击**确定**。
- 4 重新启动 MATLAB。

### 将现有工程添加到源代码管理

如果您有现有工程，可以将其添加到 Git 或 SVN 源代码管理中。

要将工程添加到源代码管理，请执行以下操作：

- 1 在[工程](#)选项卡上的**源代码管理**部分中，点击**使用源代码管理**。将打开“[源代码管理信息](#)”对话框。
- 2 点击**将工程添加到源代码管理**按钮。将打开“[添加到源代码管理](#)”对话框。
- 3 在**源代码管理工具**列表中，为您的存储库选择合适的工具。如果您选择 Git，则跳过步骤 4，直接进入步骤 5。
- 4 如果您正在使用 SVN 远程存储库，请点击**更改**按钮以选择现有存储库或创建新存储库。
  - 要指定现有存储库，请点击  按钮以浏览到您的存储库，将 URL 粘贴到字段中，或者使用列表选择最近的存储库。
  - 要创建新存储库，请点击  按钮。有关创建新存储库的详细信息，请参阅“[创建新的存储库](#)”（第 33-5 页）。

点击**验证**以检查所选存储库的路径，然后点击**确定**。

- 5 点击**转换**以完成将工程添加到源代码管理的操作。

工程将运行完整性检查。

- 6 运行完整性检查后，点击**打开工程**以返回工程。

工程将显示当前源代码管理工具和存储库位置的详细信息。

- 7 如果创建了新存储库，请选择**文件 > 已修改**视图，然后点击**提交**以将文件的第一个版本提交到新存储库。请根据需要在对话框中输入注释，然后点击**提交**。

如果要将分支与 Git 合并，您需要执行其他设置步骤。有关详细信息，请参阅“[设置 Git 源代码管理](#)”（第 33-23 页）。

如果您要使用 SVN 的某个非内置版本，请参阅“[设置 SVN 源代码管理](#)”（第 33-14 页）。

### 在已在源代码管理下的文件夹中创建新工程

如果您从已在源代码管理下的文件夹创建新工程，MATLAB 可以自动将新工程添加到源代码管理中。创建工程后，点击**检测**按钮。有关从文件夹创建工程的详细信息，请参阅“[创建工程](#)”（第 32-2 页）。

## 创建新的 GitHub 存储库

创建 GitHub 存储库会将 Git 源代码管理添加到您的新工程或现有工程中。您创建的 GitHub 存储库将成为工程的远程存储库。要创建 GitHub 存储库，您必须有 GitHub 帐户。

要创建空白工程和 GitHub 远程存储库，请执行以下操作：

- 1 在**主页**选项卡上，点击**新建 > 工程 > 从 Git**。
- 2 选择**新建 > GitHub 存储库**。在 GitHub 对话框中，输入您的**用户名和密码**。填写**存储库名称**和**说明**字段，然后点击**创建**。

MATLAB 创建一个新的公共 GitHub 存储库，并用 <https://github.com/myusername/mynewrepository> 格式的信息填充**存储库路径**字段。

- 3 在**沙盒**字段中，指定沙盒的位置。所选文件夹必须为空。点击**检索**以创建沙盒。

要确认工程名称并创建工程，请点击**确定**。

在创建 GitHub 存储库和沙盒后，将文件添加到沙盒中。请参阅“标记文件以添加到源代码管理”（第 33-8 页）。将文件的第一个版本提交到本地存储库中，然后将所有修改推送到远程 GitHub 存储库中。

---

**提示** 如果您要为现有工程创建远程 GitHub 存储库，请将您的工程共享到 GitHub。

加载工程后，在**工程**选项卡上，选择**共享 > GitHub**。有关详细说明，请参阅“共享工程”（第 32-22 页）中的“在 GitHub 上发布”。

---

## 执行源代码管理操作

### 检索版本和签出工程文件

下表显示如何检查修改的工程文件、更新修订版、获取和管理文件锁以及为工程文件加标签。

操作	过程
刷新工程文件的状态。	要检查本地修改的文件，请在 <b>工程</b> 选项卡上的 <b>源代码管理</b> 部分中，点击 <b>刷新</b> 。刷新会查询本地沙盒状态，并检查使用 MATLAB 之外的其他工具所做的更改。 有关详细信息，请参阅“更新 SVN 文件状态和修订版本”（第 33-21 页）或“更新 Git 文件状态和修订版本”（第 33-28 页）。
检查工程文件中的修改。	要了解存储库中是否有工程的新版本，请在 <b>文件</b> 视图中，右键点击该文件并选择 <b>源代码管理 &gt; 检查修改</b> 。 在 SVN 中，此选项会联系存储库以检查外部修改。工程会比较本地文件和存储库版本的修订版编号。如果存储库中的修订版编号大于本地沙盒文件夹中的修订版编号，则工程会在本地文件的修订版编号旁边显示 <b>(非最新)</b> 。

操作	过程
更新所有工程文件。	<p>使用 SVN 时，要获取所有工程文件的最新更改，请转至<b>工程</b>选项卡，在<b>源代码管理</b>部分中，点击<b>更新</b>。工程会显示一个对话框，其中列出在磁盘上发生了更改的所有文件。您可以使用工程预设项<b>显示关于源代码管理更新的更改</b>来控制此行为。有关详细信息，请参阅“更新 SVN 文件状态和修订版本”（第 33-21 页）。</p> <p>使用 Git 时，要从源代码管理存储库中获取所有工程文件的最新更改并将其合并到当前分支中，请转至<b>工程</b>选项卡，在<b>源代码管理</b>部分中，点击<b>取回</b>。要手动获取更改并进行合并，请在<b>工程</b>选项卡上的<b>源代码管理</b>部分中，点击<b>提取</b>。此操作将更新本地存储库中的所有原分支。当您点击<b>提取</b>时，沙盒文件不会更改。要查看来自其他人的更改，请将原始更改合并到本地分支。有关详细信息，请参阅“使用 Git 取回、推送和提取文件”（第 33-33 页）。</p>
更新选定工程文件的修订版。	<p>要更新选定的一组文件，请在<b>文件</b>视图中，右键点击这些文件，然后为您使用的源代码管理系统选择<b>源代码管理 &gt; 更新</b>命令。例如，如果您使用 SVN，请选择<b>源代码管理 &gt; 从 SVN 更新</b>，以从存储库中获取所选文件的最新本地副本。</p>
获取 SVN 文件锁。	<p>要获取 SVN 文件锁，请在<b>文件</b>视图中，选择要签出的文件。右键点击选定的文件，然后选择<b>源代码管理 &gt; 获取文件锁</b>。SVN 源代码管理列中将出现一个锁符号。其他用户在其沙盒中看不到锁符号，但当您拥有锁时，他们无法获取文件锁，也无法签入更改内容。要查看或打开锁，请在<b>工程</b>选项卡上，点击<b>锁</b>。</p> <p><b>获取文件锁</b>仅适用于 SVN。Git 没有锁。</p>
管理 SVN 存储库锁。	<p>要管理存储库的全局 SVN 锁，请在<b>工程</b>选项卡上的<b>源代码管理</b>部分中，点击<b>锁</b>。有关详细信息，请参阅“获取 SVN 文件锁”（第 33-22 页）。</p>
对工程文件的版本加标签。	<p>要标识所有工程文件的特定版本，请在<b>工程</b>选项卡上的<b>源代码管理</b>部分中，点击<b>添加标签</b>。指定标签文本并点击<b>确定</b>。标签会被添加到每个工程文件。如果您的存储库中没有 tags 文件夹，则会显示错误。有关详细信息，请参阅“设置 SVN 源代码管理”（第 33-14 页）。</p>

### 查看工程文件中的更改

您可以使用**文件 > 已修改**视图查看工程文件中的更改。下表显示如何查看修改后的工程文件列表、查看文件的历史记录以及比较两个文件的修订版。

操作	过程
查看修改后的工程文件。	<p>在<b>文件</b>视图中，选择<b>已修改(文件数)</b>选项卡。仅当在工程中使用源代码管理时，<b>文件</b> &gt; <b>已修改</b>视图才可见。</p> <p><b>提示</b> 使用<b>列表</b>布局查看文件，而无需展开文件夹。</p>
	<p>您可以使用源代码管理摘要状态来标识已修改或发生冲突的文件夹内容。在<b>文件</b>视图中，文件夹显示汇总的源代码管理状态。这更便于定位文件中的更改，尤其是冲突文件。您可以将鼠标悬停在文件夹的源代码管理状态（例如，<b>Git</b> 或 <b>SVN</b> 列）上，以查看工具提示，其中显示有多少文件被修改、添加、删除和发生冲突。</p>
更新修改的文件列表。	<p>要更新修改的文件列表，请在<b>工程</b>选项卡上的<b>源代码管理</b>部分中，点击<b>刷新</b>。</p>
查看修订历史记录。	<p>在<b>文件</b>视图中，右键点击文件，然后选择<b>源代码管理</b> &gt; <b>显示修订版</b>。</p> <p>要浏览和比较提交的 SVN 更改集中的文件，请在<b>工程</b>选项卡上的<b>源代码管理</b>部分中，选择<b>显示日志</b>。在“文件修订版”对话框中，选择一个修订版以查看修改文件的列表。右键点击下方列表中的文件以查看更改或保存修订版。</p>
对修订版进行比较。	<p>在<b>文件</b>视图中，右键点击一个文件，然后选择<b>比较</b> &gt; <b>与父级比较</b>，以与本地存储库 (Git) 或沙盒中的上一个签出版本 (SVN) 进行比较。“比较工具”会显示一份报告。</p> <p>要比较文件的其他修订版，请选择<b>比较</b> &gt; <b>与修订版比较</b>。</p> <p>要查看比较报告，请选择要比较的修订版，然后点击<b>与选定项比较</b>。或者，选择一个修订版并点击<b>与本地文件比较</b>。有关详细信息，请参阅“比较文件和文件夹以及合并文件”。</p>

## 工程定义文件

**resources/project** 文件夹中的文件是首次创建或更改工程时生成的工程定义文件。工程定义文件允许您在不签出文件的情况下向文件添加元数据，例如，通过创建快捷方式、添加标签和添加工程说明。工程定义文件还指定添加到工程中的文件。这些文件不是工程的一部分。

您对工程（例如，对工程中的快捷方式、标签、类别或文件）所做的任何更改都会在 **resources/project** 文件夹中产生更改。这些 XML 文件存储工程定义，格式可能发生更改。

您不需要直接查看工程定义文件，除非源代码管理工具需要合并。显示这些文件是为了让您了解要提交到源代码管理系统的所有文件。

如果要更改工程定义文件的类型，请执行以下操作：

- 1 在**主页**选项卡的**环境**部分，点击**预设**。选择**MATLAB** > **工程**，在**新建工程**部分中，选择一个**工程定义文件**：选项。
- 2 为工程创建一个 **.mlproj** 存档。有关详细信息，请参阅“共享工程”（第 32-22 页）或 **export**。
- 3 从存档工程创建一个新工程。有关详细信息，请参阅“创建工程”（第 32-2 页）。

要停止使用工程管理文件夹并删除 **resources/project** 文件夹，请参阅 **matlab.project.deleteProject**。

## 运行工程检查

要运行工程检查，请转至**工程**选项卡，然后点击**运行检查 > 检查工程**。工程会检查工程完整性方面的问题，例如缺失文件、未保存的文件或未进行源代码管理的文件。将出现一个报告结果的对话框。您可以点击以查看详细信息，并按照提示修复问题。

如果要检查所需的文件，请点击**依存关系分析**，以分析修改的文件的依存关系。使用依存关系工具来分析工程的结构。

有关检查可以修复的问题的详细信息，请参阅“处理工程中的派生文件”（第 32-42 页）和“分析工程依存关系”（第 32-29 页）。

## 提交修改的工程文件

查看更改并运行工程检查后，您就可以将修改后的工程文件提交到源代码管理了。下表显示如何提交修改的工程文件。

操作	过程
将所有修改后的文件提交到源代码管理。	在 <b>文件</b> 视图中，选择 <b>已修改(文件数)</b> 选项卡。在 <b>工程</b> 选项卡上的 <b>源代码管理</b> 部分中，点击 <b>提交</b> 。在对话框中输入注释，然后点击 <b>提交</b> 。如果您使用的是 Git 源代码管理，文件将提交到您的本地存储库。如果您使用的是 SVN 源代码管理，则会将更改提交到存储库。  如果由于存储库之前移动了而导致您无法提交，会显示一条消息。您必须将文件的修订版更新到最新的 HEAD 修订版，才能提交文件。如果您使用的是 Git 源代码管理，请点击 <b>取回</b> 。如果您使用的是 SVN 源代码管理，请点击 <b>更新</b> 。请在提交前解决所有冲突。
将选定的文件提交到源代码管理。	在 <b>文件</b> 视图中，选择文件，右键点击，然后选择 <b>源代码管理 &gt; 提交</b> 。如果提交单个文件，则存在未提交用于跟踪文件的相关工程定义文件的风险。为了避免这种情况，请提交所有修改过的文件。
使用 Git 推送工程文件。	要将本地提交发送到远程存储库，请在 <b>工程</b> 选项卡上的 <b>源代码管理</b> 部分中，点击 <b>推送</b> 。如果由于存储库已发生变化导致您无法直接推送更改内容，则会显示消息。点击 <b>提取</b> 以从远程存储库中提取所有更改。合并更改内容并解决冲突，然后您就能推送您的更改内容了。有关详细信息，请参阅“使用 Git 取回、推送和提取文件”（第 33-33 页）。
使用 Git 推送空工程文件夹。	您不能向 Git 源代码管理添加空文件夹，因此您无法点击 <b>推送</b> 然后克隆空文件夹。您可以在工程中创建一个空文件夹，但如果推送更改然后同步新沙盒，则空文件夹不会出现在新沙盒中。在这种情况下，您可以运行 <b>检查工程</b> 来为您创建空文件夹。  要将空文件夹推送到存储库以供其他用户同步，也可以在文件夹中创建一个 <code>gitignore</code> 文件，再推送您的更改内容。
创建 Git 暂存文件。	Git 暂存文件用于存储未提交的更改，以供日后使用。要创建暂存文件，请在 <b>工程</b> 选项卡上的 <b>源代码管理</b> 部分中，点击 <b>暂存文件</b> 。将打开“暂存文件”对话框。点击 <b>新建暂存文件</b> 以创建包含当前已修改文件的暂存文件。有关详细信息，请参阅“使用 Git 暂存文件”（第 33-34 页）。

操作	过程
使用 Git 创建分支。	<p>要创建分支，请在<b>工程</b>选项卡上的<b>源代码管理</b>部分中，点击<b>分支</b>。将出现“分支”对话框，您可以在其中查看、切换、创建和合并分支。</p> <p>为新分支选择源。在分支浏览器图中点击一个节点，或在源文本框中输入一个唯一标识符。您可以输入标签、分支名称或 SHA1 哈希码的唯一前缀（例如 73c637）以标识一次特定提交。保留默认值可从当前文件分支的头部创建分支。在<b>分支名称</b>文本框中输入一个名称并点击<b>创建</b>。</p>
使用 Git 切换、比较、保存和合并分支。	<p>要切换、比较、保存和合并分支，请在<b>工程</b>选项卡上的<b>源代码管理</b>部分中，点击<b>分支</b>。将出现“分支”对话框，您可以在其中查看、切换、创建和合并分支。有关详细信息，请参阅“Git 的分支和合并”（第 33-29 页）。</p>
解决冲突。	<p>如果您和其他用户在不同的沙盒中或在不同的分支上更改了相同的文件，则当您试图提交已修改文件时会显示冲突消息。如有必要，请提取冲突标记，比较导致冲突的差异并解决冲突。</p> <p>在<b>文件 &gt; 已修改</b>视图中查找冲突的文件。使用源代码管理摘要状态标识冲突的文件夹内容。文件夹显示汇总的源代码管理状态。这更便于定位文件中的更改，尤其是冲突文件。您可以将鼠标悬停在文件夹的源代码管理状态上，以查看工具提示，其中显示有多少文件被修改、添加、删除和发生冲突。</p> <p><b>提示</b> 使用<b>列表</b>布局查看文件，而无需展开文件夹。</p> <p>检查源代码管理状态栏（<b>Git</b> 或 <b>SVN</b>）中是否有带红色警告符号的文件，该符号指示存在冲突。右键点击存在冲突的文件，然后选择<b>查看冲突</b>以比较各个版本。将打开比较报告，显示冲突文件之间的差异。</p> <p>当您已解决冲突并要提交您沙盒中的版本时，请右键点击文件并选择<b>源代码管理 &gt; 标记冲突已解决</b>。</p> <p>对于 Git，<b>Git</b> 窗格中的<b>分支</b>状态会从 <b>MERGING</b> 更改为 <b>SAFE</b>。</p> <p>选择<b>文件 &gt; 已修改</b>视图以检查更改。</p>

## 还原更改

下表显示如何还原工程文件中的更改。有关还原更改的详细信息，请参阅“在源代码管理中还原更改内容”（第 33-13 页）。

操作	过程
还原本地更改。	<p>要解除锁定并还原到上次沙盒更新中的版本（即您从存储库同步或检索的上一个版本），请在<b>文件</b>视图中，右键点击要还原的文件并选择<b>源代码管理 &gt; 放弃本地更改并解锁</b>。</p> <p>要在使用 Git 时放弃本地更改，请右键点击文件，然后选择<b>源代码管理 &gt; 还原本地更改</b>。要删除所有本地更改，请点击<b>Git</b> 窗格中的<b>分支</b>，然后点击<b>还原到 HEAD</b>。</p>

操作	过程
将文件还原到指定修订版本	<p>要将文件还原到指定的修订版，请右键点击文件并选择<b>源代码管理 &gt; 使用 SVN 还原或源代码管理 &gt; 使用 Git 还原</b>。</p> <p>在“还原文件”对话框中，选择要还原到的修订版本。选择一个修订版以查看更改内容的有关信息，例如作者、日期和日志消息。点击<b>还原</b>。</p> <p>如果您将文件还原到早期的修订版本后又进行了更改，则在解决与存储库历史记录的冲突之前，将无法提交该文件。</p>
将工程还原为指定的修订版。	<p>要将工程还原为指定的修订版，请在<b>工程</b>选项卡上的<b>源代码管理</b>部分中，点击<b>还原工程</b>。在“还原文件”对话框中，选择要还原到的修订版本。</p> <p>列表中的每个修订版均为修改后的文件的更改集。选择一个修订版以查看更改内容的有关信息，例如作者、日期和日志消息。</p> <p>选择一个修订版并且确定信息正确后，请点击<b>还原</b>。</p>

## 处理工程中的派生文件

一般情况下，最好从工程中省略派生文件和临时文件，或者从源代码管理中排除它们。使用**提交之前的操作**窗格或**工程**选项卡中的**检查工程**检查派生文件或临时文件。如果您将 **resources** 文件夹添加到工程中，工程检查会建议您从工程中删除该文件夹，并提供修复选项。

另一个好做法是将派生文件排除在源代码管理之外，因为它们可能会导致问题。这包括 **.mex** 文件、**resources** 文件夹的内容或其他代码生成文件夹。例如：

- 如果源代码管理系统可以执行文件锁定，您可能会遇到冲突。如果 **resources** 在源代码管理之下，并且您生成了代码，则 **resources** 下的大多数文件都会更改并变为锁定状态。由于文件权限错误，其他用户无法生成代码。锁定二进制文件（如 **.mex** 文件）可能会对团队产生影响。
- 经常需要删除 **resources**。但是，如果文件夹在某些源代码管理工具（例如 SVN）下，删除 **resources** 可能会导致问题，例如发生 **not a working copy** 等错误。
- 如果您要将生成的代码作为过程产生的项目签入，通常可将一些文件从 **resources** 缓存文件夹复制到工程中的一个单独位置。这样，您可以在需要时删除临时缓存文件夹。请使用 **packNGo** 函数列出生成的代码文件，并使用工程 API 将它们与适当的元数据一起添加到工程中。
- resources** 文件夹可能包含许多小文件。当检查所有这些文件是否为最新时，这可能会影响某些源代码管理工具的性能。

## 查找具有未保存更改的工程文件

您可以检查工程中是否有未保存更改的文件。在**工程**选项卡上的**文件**部分中，点击**未保存的更改**。

在“未保存的更改”对话框中，您可以看到所有包含未保存更改的工程文件。如果有引用的工程，文件将按工程分组。您可以保存或放弃所有更改。

## 关闭工程时管理打开的文件

关闭工程时，如果有未保存更改的文件，将显示一条消息提示您保存或放弃更改。您可以看到所有包含未保存更改的文件，如果有引用的工程，这些包含未保存更改的文件将按工程分组。为了避免丢失工作，您可以按文件或按工程保存或放弃更改，也可以全部保存或放弃更改。

要控制此行为，请在**主页**选项卡的**环境**部分中，点击  **预设**。转至 **MATLAB > 工程**，在**工程关闭**部分中，选中或清除标签为**检查打开的工程模型，只要不是脏模型就关闭它们**的复选框。

## 另请参阅

### 详细信息

- “关于 MathWorks 源代码管理集成”（第 33-2 页）
- “设置 SVN 源代码管理”（第 33-14 页）
- “设置 Git 源代码管理”（第 33-23 页）

## 以编程方式创建和编辑工程

以下示例说明如何使用 Project API 创建新工程，并自动执行工程任务来操作文件。它讲述如何从命令行创建工程，添加文件和文件夹，设置工程路径，定义工程快捷方式，以及在另一个工程中创建对新工程的引用。它还展示如何以编程方式处理修改的文件、依存关系、快捷方式和标签。

### 设置示例文件

创建 Times Table App 示例工程文件的工作副本并打开该工程。MATLAB® 将这些文件复制到示例文件夹中以便您编辑它们。该工程将文件置于 Git™ 源代码管理下。使用 `currentProject` 从当前加载的工程创建一个工程对象。

```
matlab.project.example.timesTable
mainProject = currentProject;
```

### 检查工程文件

检查工程中的文件。

```
files = mainProject.Files

files =
1x14 ProjectFile array with properties:

    Path
    Labels
    Revision
    SourceControlStatus
```

使用索引访问此列表中的文件。例如，获取编号为 10 的文件。每个文件都有说明其路径和附加的标签的属性。

```
mainProject.Files(10)

ans =
ProjectFile with properties:

    Path: "C:\workSpace\examples\TimesTableApp\tests\tNewTimesTable.m"
    Labels: [1x1 matlab.project.Label]
    Revision: ""
    SourceControlStatus: Unmodified
```

检查第十个文件的标签。

```
mainProject.Files(10).Labels

ans =
Label with properties:

    File: "C:\workSpace\examples\TimesTableApp\tests\tNewTimesTable.m"
    DataType: 'none'
    Data: []
    Name: "Test"
    CategoryName: "Classification"
```

按名称获取特定文件。

```
myfile = findFile(mainProject,"source/timestable.mlapp")
myfile =
  ProjectFile with properties:

    Path: "C:\workSpace\examples\TimesTableApp\source\timestable.mlapp"
    Labels: [1×1 matlab.project.Label]
    Revision: ""
  SourceControlStatus: Unmodified
```

## 创建新工程

创建 Times Table Game 工程。此工程用于存储 Times Table App 背后的游戏逻辑。Times Table Game 工程将通过工程引用由 Times Table App 工程使用。

创建工作并设置工程名称。

```
timesTableGameFolder = fullfile(mainProject.RootFolder,"refs","TimesTableGame");
timesTableGame = matlab.project.createProject(timesTableGameFolder);
timesTableGame.Name = "Times Table Game";
```

将 Times Table App 游戏逻辑从主工程文件夹移至新工程文件夹，并将其添加到 Times Table Game 工程中。然后，从 Times Table App 工程中删除该文件。

```
movefile("../\source\timesTableGame.m");
addFile(timesTableGame,"timesTableGame.m");

reload(mainProject);
removeFile(mainProject,"source\timesTableGame.m");
```

将 Times Table Game 工程根文件夹添加到 Times Table Game 工程路径。这样，在加载 Times Table App 工程或任何引用 Times Table App 工程的工程时，timesTableGame.m 文件处于可用状态。

```
reload(timesTableGame);
addPath(timesTableGame,timesTableGame.RootFolder);
```

## 添加工程引用

将新 Times Table Game 工程作为工程引用添加到 Times Table App 工程中。这允许 Time Table App 工程查看、编辑和运行 Times Table Game 工程中的文件。

```
reload(mainProject);
addReference(mainProject,timesTableGame);
```

## 获取修改的文件

获取 Times Table App 工程中的所有修改的文件。将此列表与工程中的文件 > 已修改视图进行比较。您可以看到新 Times Table Game 工程的文件，以及 Times Table App 工程中已删除和修改的文件。

```
modifiedfiles = listModifiedFiles(mainProject)
modifiedfiles =
  1×19 ProjectFile array with properties:
```

```
Path
Labels
```

```
Revision
SourceControlStatus
```

获取列表中的第二个修改的文件。注意 `SourceControlStatus` 属性为 `Added`。`listModifiedFiles` 函数返回所添加、修改、删除以及发生冲突的任何文件。

#### `modifiedfiles(2)`

```
ans =
ProjectFile with properties:
```

```
Path: "C:\workSpace\examples\TimesTableApp\refs\TimesTableGame\resources\project\Project.x
Revision: ""
SourceControlStatus: Added
```

在查询单个文件之前刷新源代码管理状态。在调用 `listModifiedFiles` 之前，您不需要这样做。

#### `refreshSourceControl(mainProject)`

获取状态为 `Unmodified` 的所有工程文件。使用 `ismember` 函数获得一个逻辑值数组，其中列出 Times Table App 工程中未修改的文件。使用该数组获取未修改文件的列表。

```
unmodifiedStatus = ismember([mainProject.Files.SourceControlStatus], matlab.sourceforge.Status.Unmodified)
mainProject.Files(unmodifiedStatus)
```

```
ans =
1×9 ProjectFile array with properties:
```

```
Path
Labels
Revision
SourceControlStatus
```

## 获取文件依存关系

运行依存关系分析以更新工程文件之间的已知依存关系。

#### `updateDependencies(mainProject)`

获取 Times Table App 工程中的依存关系列表。`Dependencies` 属性包含工程文件之间的依存关系图，存储为 MATLAB `digraph` 对象。

```
g = mainProject.Dependencies
```

```
g =
digraph with properties:
```

```
Edges: [5×1 table]
Nodes: [9×1 table]
```

获取 `timestable.mlapp` 文件所需的文件。

```
requiredFiles = bfsearch(g, which('source/timestable.mlapp'))
```

```
requiredFiles = 2×1 cell array
{'C:\workSpace\examples\TimesTableApp\source\timestable.mlapp' }
```

```
{'C:\workSpace\examples\TimesTableApp\refs\TimesTableGame\timesTableGame.m'}
```

获取图中所有类型的顶层文件。`indegree` 函数查找不被任何其他文件依赖的所有文件。

```
top = g.Nodes.Name(indegree(g)==0)

top = 7×1 cell array
{'C:\workSpace\examples\TimesTableApp\requirements\TimesTableRequirements mlx'}
{'C:\workSpace\examples\TimesTableApp\tests\tAnswerIsCorrect.m'      }
{'C:\workSpace\examples\TimesTableApp\tests\tCurrentQuestion.m'      }
{'C:\workSpace\examples\TimesTableApp\tests\tNewTimesTable.m'        }
{'C:\workSpace\examples\TimesTableApp\utilities\editTimesTable.m'     }
{'C:\workSpace\examples\TimesTableApp\utilities\openRequirementsDocument.m'  }
{'C:\workSpace\examples\TimesTableApp\utilities\runTheseTests.m'       }
```

获取具有依存关系的顶层文件。`indegree` 函数查找不被任何其他文件依赖的所有文件，`outdegree` 函数查找具有依存关系的所有文件。

```
top = g.Nodes.Name(indegree(g)==0 & outdegree(g)>0)

top = 4×1 cell array
{'C:\workSpace\examples\TimesTableApp\requirements\TimesTableRequirements mlx'}
{'C:\workSpace\examples\TimesTableApp\tests\tAnswerIsCorrect.m'      }
{'C:\workSpace\examples\TimesTableApp\tests\tCurrentQuestion.m'      }
{'C:\workSpace\examples\TimesTableApp\tests\tNewTimesTable.m'        }
```

通过创建转置图查找受影响的（即“上游”）文件。使用 `flipedge` 函数反转图中边的方向。

```
transposed = flipedge(g)

transposed =
digraph with properties:

Edges: [5×1 table]
Nodes: [9×1 table]

impacted = bfsearch(transposed,which('source/timestable.mlapp'))
```

```
impacted = 2×1 cell array
{'C:\workSpace\examples\TimesTableApp\source\timestable.mlapp'      }
{'C:\workSpace\examples\TimesTableApp\requirements\TimesTableRequirements mlx'}
```

获取工程文件的信息，例如依存关系和孤立项的数量。

```
averageNumDependencies = mean(outdegree(g));
numberOfOrphans = sum(indegree(g)+outdegree(g)==0);
```

更改依存关系图的排序顺序，以自下而上显示工程更改。

```
ordered = g.Nodes.Name(flip(toposort(g)));
```

您可以在工程的**影响视图**中查看相同的依存关系分析图。要绘制图，请将影响视图保存为图像文件。

## 查询快捷方式

您可以使用快捷方式保存常见任务和经常访问的文件，或者自动执行启动和关闭任务。

获取 Times Table App 工程快捷方式。

```
shortcuts = mainProject.Shortcuts
```

```
shortcuts =
1×4 Shortcut array with properties:
```

```
Name
Group
File
```

检查列表中的快捷方式。

```
shortcuts(2)
```

```
ans =
Shortcut with properties:
```

```
Name: "Edit Times Table App"
Group: "Launch Points"
File: "C:\workSpace\examples\TimesTableApp\utilities\editTimesTable.m"
```

获取快捷方式的文件路径。

```
shortcuts(3).File
```

```
ans =
"C:\workSpace\examples\TimesTableApp\utilities\openRequirementsDocument.m"
```

检查快捷方式列表中的所有文件。

```
{shortcuts.File}'
```

```
ans = 4×1 cell array
{["C:\workSpace\examples\TimesTableApp\source\timestable.mlapp"]}
{["C:\workSpace\examples\TimesTableApp\utilities\editTimesTable.m"]}
{["C:\workSpace\examples\TimesTableApp\utilities\openRequirementsDocument.m"]}
{["C:\workSpace\examples\TimesTableApp\utilities\runTheseTests.m"]}
```

## 为文件加标签

创建 **char** 类型的新标签类别。在 Times Table App 工程中，新的 **Engineers** 类别将出现在**标签**窗格中。

```
createCategory(mainProject,'Engineers','char')
```

```
ans =
Category with properties:
```

```
Name: "Engineers"
SingleValued: 0
DataType: "char"
```

```
LabelDefinitions: [1×0 matlab.project.LabelDefinition]
```

在新类别中定义一个新标签。

```
category = findCategory(mainProject,'Engineers');
createLabel(category,'Bob');
```

获取新标签的标签定义对象。

```
ld = findLabel(category,'Bob')
```

```
ld =
LabelDefinition with properties:
```

```
    Name: "Bob"
    CategoryName: "Engineers"
```

将标签附加到工程文件。如果您在 Times Table App 工程中选中该文件，您可以在**标签编辑器**窗格中看到该标签。

```
myfile = findFile(mainProject,"source/timetable.mlapp");
addLabel(myfile,'Engineers','Bob');
```

获取某特定标签，并在该标签上附加文本数据。

```
label = findLabel(myfile,'Engineers','Bob');
label.Data = 'Email: Bob.Smith@company.com'
```

```
label =
Label with properties:
```

```
    File: "C:\workSpace\examples\TimesTableApp\source\timetable.mlapp"
    DataType: 'char'
    Data: 'Email: Bob.Smith@company.com'
    Name: "Bob"
    CategoryName: "Engineers"
```

检索标签数据并将其存储在变量中。

```
mydata = label.Data
```

```
mydata =
'Email: Bob.Smith@company.com'
```

创建数据类型为 **double** 的新标签类别，MATLAB 通常将该数据类型用于数值数据。

```
createCategory(mainProject,'Assessors','double');
category = findCategory(mainProject,'Assessors');
createLabel(category,'Sam');
```

将新标签附加到指定的文件，并为该标签分配数据值 2。

```
myfile = mainProject.Files(10);
addLabel(myfile, 'Assessors', 'Sam', 2)
```

```
ans =
Label with properties:
```

```
File: "C:\workSpace\examples\TimesTableApp\utilities"
DataType: 'double'
Data: 2
Name: "Sam"
CategoryName: "Assessors"
```

### 关闭工程

关闭工程以运行关闭脚本并检查未保存的文件。

```
close(mainProject)
```

### 另请参阅

```
currentProject
```

### 详细信息

- “创建工程” (第 32-2 页)
- “管理工程文件” (第 32-10 页)
- “分析工程依存关系” (第 32-29 页)

# 了解示例工程

此示例通过 Times Table App 示例工程来展示工程工具如何帮助您整理您的工程。

使用 Times Table App 示例，我们将了解如何：

- 1 设置和浏览源代码管理下的一些示例工程文件。
- 2 检查工程快捷方式以访问常用的文件和任务。
- 3 分析工程中的依存关系，并找到工程中尚不存在的必需文件。
- 4 修改一些工程文件，找到和查看修改后的文件，将其与早期版本进行比较，并将修改过的文件提交到源代码管理。
- 5 查看仅包含工程文件的视图、包含修改过的文件的视图，以及包含工程根文件夹下所有文件的视图。

## 设置示例文件

创建 Times Table App 示例工程文件的工作副本并打开该工程。MATLAB® 将这些文件复制到示例文件夹中以便您编辑它们。该工程将文件置于 Git™ 源代码管理下。

`matlab.project.example.timesTable`

## 查看、搜索和排序工程文件

您可以使用**文件**视图查看、搜索工程文件和对其进行排序。

要查看工程中的文件，请在**文件**视图中，点击**工程 (number of files)**。如果选择此视图，将仅显示您的工程中的文件。

要查看工程文件夹中的所有文件，请点击**全部**。此视图将显示工程根文件夹下的所有文件，而不仅是工程中的文件。因此，此视图对于向工程添加文件非常有用。

要以列表而不是树的形式查看文件，请在**文件**视图右上角的**布局**字段中，选择**列表**。

有几种方法可以在工程中查找文件和文件夹：

- 要按名称搜索特定文件或文件类型，请在任何文件视图的搜索框中键入内容或点击**筛选器**按钮。例如，在搜索字段中，输入文本 `timestable`。工程会返回包含单词 `timestable` 的所有文件和文件夹。点击  以清除搜索。
- 要搜索文件内容，请转至**工程**选项卡，然后点击**搜索**按钮。在搜索字段中输入值，然后按 **Enter** 键。例如，输入单词 `tests`。工程将显示包含单词 `tests` 的所有文件和文件夹。点击  以清除搜索。
- 要更改文件的分组或排序方式以及自定义列，请点击操作  按钮并从可用选项中进行选择。

## 打开并运行常用文件

您可以使用快捷方式在大型工程中更轻松地查找文件。在**工程快捷方式**选项卡上查看并运行快捷方式。您可以将快捷方式分组。

Times Table App 工程包含几个快捷方式，包括打开工程需求的快捷方式和运行工程中所有测试的快捷方式。快捷方式使工程用户能够更轻松地完成这些任务。

要执行某项操作，请在**工程快捷方式**选项卡上，点击关联的快捷方式。例如，要打开工程需求，请点击**文档 > 需求**。要运行测试，请点击**测试 > 运行所有测试**。

要创建新快捷方式，请选择**文件**视图，右键点击文件，然后选择**创建快捷方式**。

### 将文件夹添加到工程

创建一个新文件夹并将其添加到工程路径中。将工程文件夹添加到工程路径可确保工程的所有用户都能访问其中的文件。

- 1 选择**文件**视图。
- 2 右键点击空白区域，然后选择**新建 > 文件夹**。输入文件夹的名称。该文件夹会自动添加到工程中。
- 3 右键点击新文件夹，然后选择**工程路径 > 添加到工程路径(包括子文件夹)**。

### 查看已修改文件中的更改

打开文件、进行更改并查看更改。

- 1 选择**文件**视图。使用树布局查看文件夹，然后展开 utilities 文件夹。
- 2 右键点击 `source/timesTableGame.m`，然后选择**打开**。
- 3 在编辑器中进行更改（例如添加注释），然后保存文件。
- 4 在**文件**视图中，选择**已修改 (number of files)** 选项卡。编辑文件后，您会看到**已修改 (2)**。您更改的文件将显示在列表中。
- 5 要查看更改，请在**已修改**文件视图中右键点击 `source/timesTableGame.m`，然后选择**比较 > 与父级比较**。MATLAB 比较工具将打开一个报告，将您的沙盒中的文件修改版与存储在版本控制中的原版进行比较。比较报告类型可能因您选择的具体文件而异。如果选择 Simulink® 模型来进行比较，此命令将运行 Simulink 模型比较。

\* 注意 - 当您打开 Times Table App 示例工程时，该工程会在 `resources` 文件夹中显示修改后的文件。这是打开示例工程的副作用。在您自己的工程中编辑文件时，只有影响文件元数据的更改（例如向文件添加标签）才会在 `resources` 文件夹中创建修改后的文件。

### 分析依存关系

要检查所有必需的文件是否都在工程中，请对修改过的文件运行文件依存关系分析。

- 1 在**视图**窗格中，选择**依存关系分析**。
- 2 点击**分析**。影响图显示工程中所有分析的依存关系的结构。右窗格列出所需的工具箱和任何问题文件。请注意，没有列出任何问题文件。

现在，删除一个必需的文件。选择**文件**视图，右键点击 `source/timesTableGame.m` 文件，然后选择**从工程中删除**。点击“从工程中删除”对话框中的**删除**。

再次检查是否有问题。

- 1 再次选择**依存关系分析**视图。
- 2 在**依存关系分析**选项卡中，点击**分析 > 全部重新分析**。
- 3 将鼠标悬停在**问题**下的消息上，然后点击**查找全部**以查看使用问题文件的文件。图会更新以仅显示问题文件 `timesTableGame.m`，并在右窗格中显示问题消息 `Not in project`。
- 4 要查看问题文件的依存关系，请在**依存关系分析**选项卡上的**影响分析**部分中，选择**查找 > 所选项的所有依存关系**。

现在您已看到问题，请通过将缺失文件放回到工程中来修复问题。右键点击该文件，然后选择**添加到工程**。下次运行依存关系分析时，该文件不会显示为问题文件。

运行依存关系分析后，要调查修改后的文件的依存关系，请执行影响分析。

- 1 在**依存关系分析**选项卡的**影响分析**部分中，选择**查找 > 所有文件**。
- 2 在**影响分析**部分中，选择**选择 > 修改的文件**。
- 3 要查看修改的文件的依存关系，请在**影响分析**部分中，选择**查找 > 所选项的所有依存关系**。

## 运行工程完整性检查

为确保您的更改已准备好提交，请检查您的工程。在**工程**选项卡上，点击**运行检查**以运行工程完整性检查。这些检查会查找缺失的文件、要添加到源代码管理或从源代码管理检索的文件及其他问题。“**检查**”对话框提供自动修复发现的问题的功能（如果可能的话）。当您点击“**检查**”对话框中的**详细信息**按钮时，可以查看推荐的操作并决定是否进行更改。

## 提交修改的文件

修改文件并对检查结果满意后，可以将更改提交到源代码管理存储库。

- 1 在文件视图中，选择**已修改 (number of files)** 选项卡。您更改的文件会显示在列表中。
- 2 要将更改提交到源代码管理，请在**工程**选项卡上的**源代码管理**部分中，点击**提交**。
- 3 为您的提交输入注释，然后点击**提交**。在源代码管理提交您的更改时查看状态栏中的消息。在 Git 中，您可以同时拥有本地和远程存储库。这些说明会提交到本地存储库。要提交到远程存储库，请在**源代码管理**部分中，点击**取回**。

## 查看工程和源代码管理信息

要查看和编辑工程详细信息，请在**工程**选项卡上的**环境**部分中，点击**详细信息**。查看和编辑工程详细信息，如名称、说明、工程根目录、启动文件夹以及包含生成文件的文件夹的位置。

要查看有关源代码管理集成和存储库位置的详细信息，请在**工程**选项卡上的**源代码管理**部分中，点击**Git 详细信息**。Times Table App 示例工程使用 Git 源代码管理。

## 关闭工程

点击  (位于工程窗口的右上角) 以关闭工程。

```
proj = currentProject;  
close(proj);
```

## 另请参阅

### 详细信息

- “**创建工程**”（第 32-2 页）
- “**管理工程文件**”（第 32-10 页）
- “**分析工程依存关系**”（第 32-29 页）



# 源代码管理接口

---

通过源代码管理接口可以从 MATLAB 桌面访问您的源代码管理系统。

- “关于 MathWorks 源代码管理集成” (第 33-2 页)
- “选择或禁用源代码管理系统” (第 33-4 页)
- “创建新的存储库” (第 33-5 页)
- “在源代码管理中审核更改” (第 33-7 页)
- “标记文件以添加到源代码管理” (第 33-8 页)
- “解决源代码管理冲突” (第 33-9 页)
- “将已修改文件提交到源代码管理” (第 33-12 页)
- “在源代码管理中还原更改内容” (第 33-13 页)
- “设置 SVN 源代码管理” (第 33-14 页)
- “从 SVN 存储库签出” (第 33-19 页)
- “更新 SVN 文件状态和修订版本” (第 33-21 页)
- “获取 SVN 文件锁” (第 33-22 页)
- “设置 Git 源代码管理” (第 33-23 页)
- “从 Git 存储库克隆” (第 33-27 页)
- “更新 Git 文件状态和修订版本” (第 33-28 页)
- “Git 的分支和合并” (第 33-29 页)
- “使用 Git 取回、推送和提取文件” (第 33-33 页)
- “移动、重命名或删除源代码管理下的文件” (第 33-36 页)
- “自定义外部源代码管理以使用 MATLAB 执行差异分析和合并” (第 33-37 页)
- “MSSCCI 源代码管理接口” (第 33-39 页)
- “设置 MSSCCI 源代码管理” (第 33-40 页)
- “从 MSSCCI 源代码管理中签入和签出文件” (第 33-45 页)
- “其他 MSSCCI 源代码管理操作” (第 33-47 页)
- “从编辑器访问 MSSCCI 源代码管理” (第 33-52 页)
- “MSSCCI 源代码管理问题故障排除” (第 33-53 页)

## 关于 MathWorks 源代码管理集成

您可以使用 MATLAB 处理源代码管理下的文件。您可以直接从当前文件夹浏览器执行更新、提交、合并更改和查看修订历史记录等操作。

MATLAB 集成了：

- Subversion (SVN)
- Git

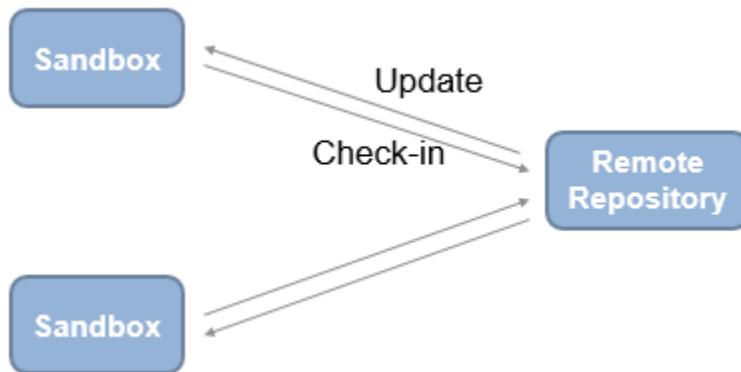
如要在您的工程中使用源代码管理，请使用以下任意工作流：

- 从现有存储库中检索文件。请参阅“从 SVN 存储库签出”（第 33-19 页）或“从 Git 存储库克隆”（第 33-27 页）。
- 为文件夹添加源代码管理。请参阅“创建新的存储库”（第 33-5 页）。
- 在已经处于源代码管理下的文件夹中添加新文件。请参阅“标记文件以添加到源代码管理”（第 33-8 页）。

其他源代码管理集成（例如 Microsoft 源代码管理接口 (MSSCCI)）可从附加功能资源管理器下载。有关详细信息，请参阅“获取和管理附加功能”。

### 典型和分布式源代码管理

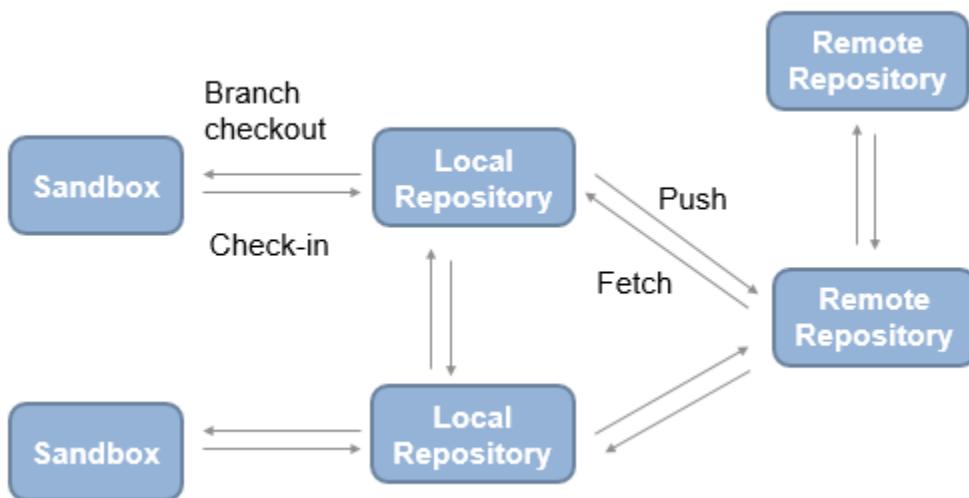
下图展示了典型源代码管理的工作流（例如，使用 SVN）。



典型源代码管理的优点：

- 基于每个文件的锁定和用户权限（例如，您可以对模型文件强制执行锁定）
- 中央服务器，减少本地存储需求
- 简单易学

下图展示了分布式源代码管理的工作流（例如，使用 Git）。



分布式源代码管理的优点：

- 离线工作
- 提供完整历史记录的本地存储库
- 分支
- 多个远程存储库，实现大规模分层访问控制

在选择典型或分布式源代码管理时，请考虑以下提示。

在以下情况下，典型源代码管理可能会比较有帮助：

- 需要文件锁定。
- 初次使用源代码管理。

在以下情况下，分布式源代码管理可能会比较有帮助：

- 需要脱机工作，定期提交，并需要访问整个存储库历史记录。
- 需要本地分支。

## 选择或禁用源代码管理系统

### 选择源代码管理系统

如果您刚开始在 MATLAB 中使用源代码管理，请选择 MathWorks 当前文件夹浏览器的源代码管理集成中所包含的源代码管理系统，例如 Subversion 或 Git。这样您就能发挥集成所具有的内置特性优势。默认情况下 MathWorks 源代码管理集成是开启的。

- 1 在**主页**选项卡上，点击**环境**部分中的**预设**。
- 2 在预设对话框中，导航到**MATLAB > 常规 > 源代码管理**窗格。
- 3 要使用 MathWorks 源代码管理集成（可通过当前文件夹浏览器访问），请选择**启用 MathWorks 源代码管理集成**。启用此选项可使用 Subversion 和 Git 等源代码管理系统。这是默认选项，除非您以前设置过 MATLAB 的源代码管理。

### 禁用源代码管理

当您禁用源代码管理时，MATLAB 不会破坏存储库信息。例如，不会删除 .svn 文件夹。您可以通过重新启用源代码管理集成，再次对该文件夹进行源代码管理。

- 1 在**主页**选项卡上，点击**环境**部分中的**预设**。
- 2 在“**预设项**”对话框的**MATLAB > 常规 > 源代码管理**窗格中，选择**无**。

# 创建新的存储库

## 在您的本地系统上创建 Git 存储库

如果您希望为您的文件添加版本控制但又不希望与其他用户共享，最快捷的方法在本地系统上创建 Git 存储库和沙盒。要克隆现有远程 Git 存储库，请参阅“从 Git 存储库克隆”（第 33-27 页）。

---

**注意** 在使用源代码管理之前，您必须在您的源代码管理工具中注册二进制文件以避免损坏。有关详细信息，请参阅“在 Git 中注册二进制文件”（第 33-24 页）。

---

要在本地系统上创建 Git 存储库和沙盒，请执行以下操作：

- 1 右键点击 MATLAB 当前文件夹浏览器的空白处（任意空白区域），然后选择**源代码管理 > 管理文件**。MATLAB 打开“使用源代码管理系统管理文件”对话框。
- 2 在**源代码管理集成**列表中，选择“Git”。
- 3 点击**更改**按钮。MATLAB 打开“选择存储库”对话框。
- 4 点击在磁盘上创建一个 Git 存储库 (+) 按钮。
- 5 选择您希望在其中创建存储库的空文件夹或创建一个新文件夹，然后点击**选择文件夹**。MATLAB 会创建存储库，关闭对话框并返回“选择存储库”对话框。
- 6 点击**验证**以检查新存储库的路径，然后点击**确定**。MATLAB 会关闭对话框，并返回“使用源代码管理系统管理文件”对话框。
- 7 在**沙盒**字段中，指定沙盒的位置。所选文件夹必须为空。
- 8 点击**检索**以创建沙盒。

在创建 Git 存储库和沙盒后，将文件添加到沙盒中。然后，将文件的第一个版本提交到新存储库中。有关详细信息，请参阅“标记文件以添加到源代码管理”（第 33-8 页）。

您也可以在创建存储库后更改存储库 URL。在当前文件夹浏览器的源代码管理下的文件夹中，右键点击并选择**源代码管理 > 远程**，然后指定一个新 URL。

要合并 Git 中的分支，您需要完成一些其他设置步骤。有关详细信息，请参阅“安装命令行 Git 客户端”（第 33-24 页）。

要将 Git 服务器用于本地系统上的存储库，您可以设置自己的 Apache™ Git 服务器或使用 Git 服务器托管解决方案。如果您无法设置服务器且必须通过使用 file:/// 协议的文件系统来使用远程存储库，则请确保该存储库是不含任何签出工作副本的裸存储库。

## 创建 SVN 存储库

您还可以通过创建 SVN 存储库来对文件添加版本控制。要签出现有的 SVN 存储库，请参阅“从 SVN 存储库签出”（第 33-19 页）。

---

### 注意

- 在使用源代码管理之前，您必须在您的源代码管理工具中注册二进制文件以避免损坏。请参阅“使用 SVN 注册二进制文件”（第 33-14 页）。

- 检查您要用于 SVN 沙盒的文件夹是否在本地硬盘上。配合 SVN 使用网络文件夹既慢又不可靠。
- 

要创建 SVN 存储库，请执行以下操作：

- 1 右键点击 MATLAB 当前文件夹浏览器的空白处（任意空白区域），然后选择**源代码管理 > 管理文件**。MATLAB 打开“使用源代码管理系统管理文件”对话框。
- 2 在**源代码管理集成**列表中，选择“SVN”。
- 3 点击**更改**按钮。MATLAB 将打开“指定 SVN 存储库 URL”对话框。
- 4 点击在文件夹中创建一个 SVN 存储库 (+) 按钮。
- 5 选择您希望在其中创建存储库的空文件夹或创建一个新文件夹，然后点击**选择文件夹**。

MATLAB 在所选文件夹中创建存储库，关闭对话框并返回到“指定 SVN 存储库 URL”对话框。新存储库的 URL 在**存储库**框中，且工程会自动选择 trunk 文件夹。请仅为单个用户指定 file:// URL 和创建新存储库。对于多个用户，请参阅“共享 Subversion 存储库”（第 33-17 页）。

- 6 点击**验证**以检查新存储库的路径，然后点击**确定**。MATLAB 会关闭对话框，并返回“使用源代码管理系统管理文件”对话框。

如果您的 SVN 存储库中有文件 URL，则会显示警告消息，指明文件 URL 适用于单个用户。点击**确定**继续。

- 7 在**沙盒**字段中，指定沙盒的位置。例如，选择要添加到源代码管理的文件所在的文件夹。
- 8 点击**检索**以创建沙盒。

在创建 SVN 存储库和沙盒后，您可以将文件的第一个版本提交到新存储库中。有关详细信息，请参阅“标记文件以添加到源代码管理”（第 33-8 页）。

## 另请参阅

### 相关示例

- “设置 Git 源代码管理”（第 33-23 页）
- “设置 SVN 源代码管理”（第 33-14 页）
- “从 Git 存储库克隆”（第 33-27 页）
- “从 SVN 存储库签出”（第 33-19 页）
- “将已修改文件提交到源代码管理”（第 33-12 页）

## 在源代码管理中审核更改

进行源代码管理的文件在更改后，会在当前文件夹浏览器中显示“修改的文件”符号 。在当前文件夹浏览器中右键点击文件，选择**源代码管理**，并根据需要进行相应操作：

- 选择**显示修订**可打开“文件修订”对话框，并浏览文件的历史记录。您可以查看以前提交文件的人员、提交时间、日志消息以及每个更改集中的文件列表等有关信息。您可以选择多个文件并查看每个文件的修订历史记录。

使用 SVN，您可以选择一个修订版并浏览下方更改集中的文件列表。右键点击文件以查看更改或保存修订版。

- 选择**与修订版比较**可打开一个对话框，在此可以选择您要进行比较的修订版本并查看比较报告。您可以：
  - 选择一个修订版本并点击**与局部项比较**。
  - 选择两个修订版本并点击**与选定项比较**。

使用 SVN，您可以选择一个修订版并浏览下方更改集中的文件列表。右键点击文件以查看更改或保存修订版。

- 选择**与父级比较**可以运行与沙盒 (SVN) 中最近签出版本的比较，或者运行与本地存储库 (Git) 的比较。“比较工具”会显示一份报告。

如果您需要更新已修改文件的状态，请参阅“更新 SVN 文件状态和修订版本”（第 33-21 页）或“更新 Git 文件状态和修订版本”（第 33-28 页）。

## 另请参阅

### 相关示例

- “解决源代码管理冲突”（第 33-9 页）
- “将已修改文件提交到源代码管理”（第 33-12 页）
- “在源代码管理中还原更改内容”（第 33-13 页）

## 标记文件以添加到源代码管理

当您在源代码管理的文件夹中创建一个新文件时，在当前文件夹浏览器的状态栏中会显示“未进行源代码管理”符号 。要将文件添加到源代码管理，请在当前文件夹浏览器中右键点击文件，并选择源代码管理，然后选择您的源代码管理系统适用的“添加”选项。例如，选择添加到 Git 或添加到 SVN。

当文件被标记为添加到源代码管理时，符号会更改为“已添加” 。

# 解决源代码管理冲突

## 检查和解决冲突

如果您和其他用户在不同的沙盒中或在不同的分支上更改了相同的文件，则当您试图提交已修改文件时会显示冲突消息。如有必要，请按照程序“解决冲突”（第 33-9 页）提取冲突标记，比较导致冲突的差异并解决冲突。

要解决冲突，您可以：

- 使用“比较工具”合并修订版本之间的更改。
- 决定用一套更改来覆盖另一套更改。
- 通过编辑文件来进行手动更改。

有关使用“比较工具”来合并更改的详细信息，请参阅“合并文本文件”（第 33-9 页）。

当您对标记为冲突的文件满意后，就能将冲突标记为已解决并提交文件。

## 解决冲突

- 1 在当前文件夹浏览器中查找存在冲突的文件。
- 2 检查源代码管理状态栏（SVN 或 Git）中是否有带红色警告符号  的文件，该符号指示存在冲突。
- 3 右键点击存在冲突的文件，然后选择源代码管理 > 查看冲突以比较各个版本。
- 4 检查冲突。会打开一份比较报告，显示冲突文件之间的差异。

在 SVN 中，比较报告会显示该文件与冲突文件版本之间的差异。

在 Git 中，比较报告会显示您的分支与您要合并到的分支上文件之间的差异。

- 5 使用“比较工具”报告来决定如何解决冲突。

您可以使用“比较工具”来合并各个修订版本之间的更改，该方法在“合并文本文件”（第 33-9 页）中有说明。

- 6 当您已解决更改并要提交您沙盒中的版本时，在当前文件夹浏览器中，右键点击文件并选择源代码管理 > 标记为冲突已解决。

在 Git 中，“源代码管理详细信息”对话框中的分支状态将从 MERGING 更改为 SAFE。

- 7 提交已修改文件。

## 合并文本文件

比较文本文件时，您可以将来自一个文件的更改内容合并到另一个文件。在解决不同版本文件之间的冲突时，合并更改内容很有用处。

如果您在文本比较报告中看到类似如下的冲突标记：

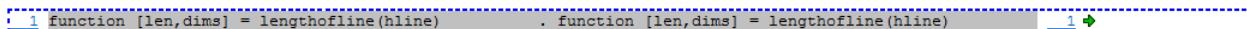
```
<<<<< .mine
```

则可按照“提取冲突标记”（第 33-10 页）中介绍的方法在合并前提取冲突标记。

**提示** 在源代码管理中将一个文件与另一个版本作比较时，默认情况下右边的文件是您沙盒中的版本，左边的文件是上一个版本的临时副本或导致冲突的其他版本（例如：filename\_theirs）。您可以交换文件的位置，因此请务必观察位于比较报告顶部的左边和右边文件的文件路径。将临时副本中的差异合并到您沙盒中的版本来解决冲突。

- 1 在“比较工具”报告中，选择报告中的差异并点击**合并**。选中的差异从左边的文件复制到右边的文件中。

已合并差异的行将以灰色高亮显示，并显示绿色合并箭头。



如果报告顶部的合并文件名称显示有星号 (filename.m\*)，则表示该文件包含有尚未保存的更改内容。

- 2 点击**保存合并的文件**以保存您沙盒中的文件。要解决冲突，请保存合并文件以覆盖冲突文件。
- 3 如要在编辑器中检查文件，请点击报告中的行号链接。

**注意** 如果您在编辑器中又进行了任何更改，则比较报告不会更新来反映更改内容，报告链接可能出现错误。

- 4 当您处理了更改内容后，请将其标记为冲突已解决。在当前文件夹浏览器中右键点击文件，然后选择**源代码管理 > 标记为冲突已解决**。

## 提取冲突标记

- “什么是冲突标记？”（第 33-10 页）
- “提取冲突标记”（第 33-11 页）

### 什么是冲突标记？

源代码管理工具可以在您未注册为二进制文件（例如文本文件）的文件中插入冲突标记。您可以使用 MATLAB 来提取冲突标记并对导致冲突的文件进行比较。以下流程将帮助您决定如何解决冲突。

**小心** 在源代码管理工具上注册文件，以防文件被插入冲突标记或被损毁。请参阅“使用 SVN 注册二进制文件”（第 33-14 页）或“在 Git 中注册二进制文件”（第 33-24 页）。如果您的文件已包含冲突标记，使用 MATLAB 工具可以帮助您解决冲突。

冲突标记具有以下形式：

```
<<<<<["mine" file descriptor]
["mine" file content]
=====
["theirs" file content]
<<<<<["theirs" file descriptor]
```

如果您试图打开含有冲突标记的文件，会弹出“发现冲突标记”对话框。请按照提示通过提取冲突标记来修复文件。提取冲突标记后，请按照“检查和解决冲突”（第 33-9 页）中介绍的方法解决冲突。

要查看冲突标记，在“发现冲突标记”对话框中点击**加载文件**。请勿尝试加载文件，因为 MATLAB 不能识别冲突标记。而是要点击**修复文件**来提取冲突标记。

MATLAB 只检查冲突文件中的冲突标记。

### 提取冲突标记

当您打开冲突文件或选择查看冲突时，MATLAB 会检查文件中的冲突标记并提供提取冲突标记的功能。MATLAB 只检查冲突文件中的冲突标记。

但是，未标记为冲突的部分文件中仍可能含有冲突标记。如果您或其他用户未删除冲突标记就标记为冲突已解决然后将文件提交，则可能发生此种情况。如果您在未标记为有冲突的文件中看到了冲突标记，则您可以提取该冲突标记。

- 1 在当前文件夹浏览器中，右键点击该文件，并选择源代码管理 > **将冲突标记提取到文件**。
- 2 在“将冲突标记提取到文件”对话框中，保持默认选项不变以复制“我的”文件版本覆盖冲突文件。选中**比较所提取的文件**复选框。点击**提取**。
- 3 照常使用“比较工具”报告来继续解决冲突。

## 将已修改文件提交到源代码管理

在提交已修改文件之前，请审核更改内容并对要添加到源代码管理的任何新文件进行标记。处于源代理管理下且可提交到存储库的文件会在当前文件夹浏览器中显示“已添加到源代码管理”符号 或“修改的文件”符号 。

- 1 在当前文件夹浏览器中点击右键，然后选择**源代码管理 > 查看并提交更改**。在“查看并提交更改”对话框中，选择要提交到存储库的文件。
  - 2 在对话框中输入注释，然后点击**提交**。
  - 3 如果由于存储库之前移动了而导致您无法提交，会显示一条消息。您必须将修订版本更新到最新的 HEAD 修订版本后才能提交文件。
    - 如果您使用的是 SVN 源代码管理，请在当前文件夹浏览器中点击右键。选择**源代码管理 > 全部从 SVN 更新**。
    - 如果您使用的是 Git 源代码管理，请在当前文件夹浏览器中点击右键。选择**源代码管理 > 取回**。
- 请在提交前解决所有冲突。

### 另请参阅

#### 相关示例

- “标记文件以添加到源代码管理” (第 33-8 页)
- “在源代码管理中审核更改” (第 33-7 页)
- “解决源代码管理冲突” (第 33-9 页)
- “更新 SVN 文件状态和修订版本” (第 33-21 页)
- “更新 Git 文件状态和修订版本” (第 33-28 页)
- “使用 Git 取回、推送和提取文件” (第 33-33 页)

# 在源代码管理中还原更改内容

## 还原本地更改内容

在 SVN 中，如果您要还原文件中的本地更改内容，请右键点击文件并选择**源代码管理 > 还原本地更改并解锁**。此命令会解除锁定并还原到最近沙盒更新中的版本（也即：您从存储库同步或检索的最近版本）。如果您的文件未锁定，则菜单选项为**源代码管理 > 还原本地更改**。要放弃所有本地更改内容，请在当前文件夹浏览器中选择所有文件，然后选择此命令。

在 Git 中，右键点击文件并选择**源代码管理 > 还原本地更改**。Git 没有锁定。要删除所有本地更改内容，请在当前文件夹浏览器中右键点击空白处，并选择**源代码管理 > 分支**。在“分支”对话框中，点击**还原到 HEAD**。

## 将文件还原到指定修订版本

- 1 在当前文件夹浏览器中右键点击文件，然后选择**源代码管理 > 使用 SVN 还原或使用 Git 还原**。
- 2 在“还原文件”对话框中，选择要还原到的修订版本。选择一个修订版本查看更改内容的有关信息，例如作者、日期和日志消息。
- 3 点击**还原**。

如果您将文件还原到早期的修订版本后又进行了更改，则在解决与存储库历史记录的冲突之前，将无法提交该文件。

## 另请参阅

### 相关示例

- “解决源代码管理冲突”（第 33-9 页）

## 设置 SVN 源代码管理

MATLAB 提供了配合 Subversion (SVN) 沙盒和存储库使用的内置 SVN 集成。由于该实现已内置到 MATLAB，因此无需安装 SVN。内置的 SVN 集成支持安全登录。该集成忽略任何现有的 SVN 安装。

### SVN 源代码管理选项

如果要使用随 MATLAB 提供的 SVN 版本，请在从源代码管理中检索文件时，选择**源代码管理集成**列表中的“SVN”。有关说明，请参阅“从 SVN 存储库签出”（第 33-19 页）。当您使用 MATLAB 内置的 SVN 集成创建新的沙盒时，新沙盒会使用由 MATLAB 提供的 SVN 最新版本。

**小心** 在使用源代码管理之前，您必须在您的源代码管理工具中注册二进制文件以避免损坏。请参阅“使用 SVN 注册二进制文件”（第 33-14 页）。

如果您需要使用内置版本以外的其他 SVN 版本，您可以使用“命令行 SVN 集成（兼容模式）”**源代码管理集成**选项来创建一个存储库，但是您还必须安装一个命令行 SVN 客户端。

命令行 SVN 集成可与支持命令行界面的任何 Subversion (SVN) 客户端通信。在“命令行 SVN 集成（兼容模式）”中，如果您尝试将文件或文件夹重命名为包含 @ 字符的名称，会发生错误，这是因为命令行 SVN 将 @ 符号后的所有字符视为 peg 修订版本值。

### 使用 SVN 注册二进制文件

如果您使用第三方源代码管理工具，则必须

将 .mlx、.mat、.fig、.mlapp、.mdl、.slx、.mdlp、.slxp、.sldd 和 .p 等 MATLAB 和 Simulink 文件扩展名注册为二进制格式。如果不注册这些扩展名，则当您通过更改行尾字符、扩展标记、替换关键字或尝试自动合并来提交文件时，这些工具可能会损坏您的文件。不论您是在 MATLAB 外部使用该源代码管理工具，还是在未先注册文件格式的前提下尝试从 MATLAB 提交文件，都可能发生损坏。

还要检查其他文件扩展名是否已注册为二进制文件，以避免在签入时损坏。检查并注册 MEX 文件、.xlsx、.jpg、.pdf、.docx 等文件。

如果您使用任何 SVN 版本，包括由 MATLAB 提供的内置 SVN 集成，则必须注册二进制文件。如果未将扩展名注册为二进制，则 SVN 可能向存在冲突的 MATLAB 文件中添加注释并尝试自动合并。为了避免在使用 SVN 时出现此问题，请注册文件扩展名。

1 定位您的 SVN config 文件。在以下这些位置中查找文件：

- Windows 上的 C:\Users\myusername\AppData\Roaming\Subversion\config 或 C:\Documents and Settings\myusername\Application Data\Subversion\config
- Linux 或 macOS 上的 ~/.subversion

2 如果您未找到 config 文件，请创建一个新文件。请参阅“创建 SVN 配置文件”（第 33-14 页）。

3 如果找到现有的 config 文件，则您以前已经安装了 SVN。编辑 config 文件。请参阅“更新现有的 SVN 配置文件”（第 33-15 页）。

#### 创建 SVN 配置文件

1 如果未找到 SVN config 文件，则创建一个包含以下各行的文本文件：

```
[miscellany]
enable-auto-props = yes
```

```
[auto-props]
*.mlx = svn:mime-type=application/octet-stream
*.mat = svn:mime-type=application/octet-stream
*.fig = svn:mime-type=application/octet-stream
*.mdl = svn:mime-type=application/octet-stream
*.slx = svn:mime-type= application/octet-stream
*.mlapp = svn:mime-type= application/octet-stream
*.p = svn:mime-type=application/octet-stream
*.mdlp = svn:mime-type=application/octet-stream
*.slxp = svn:mime-type=application/octet-stream
*.sldd = svn:mime-type=application/octet-stream
*.slxc = svn:mime-type=application/octet-stream
*.mlproj = svn:mime-type=application/octet-stream
*.mldatx = svn:mime-type=application/octet-stream
*.slreqx = svn:mime-type=application/octet-stream
*.sfx = svn:mime-type=application/octet-stream
*.slt = svn:mime-type=application/octet-stream
```

- 2 检查配置文件中是否有您使用的其他文件类型需要注册为二进制，以免签入时遭到损坏。检查 MEX 文件 (.mexa64、.mexmaci64、.mexw64)、.xlsx、.jpg、.pdf、.docx 等文件。在 config 文件中为您需要的每个文件类型添加一行。示例：

```
*.mexa64 = svn:mime-type=application/octet-stream
*.mexw64 = svn:mime-type=application/octet-stream
*.mexmaci64 = svn:mime-type=application/octet-stream
*.xlsx = svn:mime-type=application/octet-stream
*.docx = svn:mime-type=application/octet-stream
*.pdf = svn:mime-type=application/octet-stream
*.jpg = svn:mime-type=application/octet-stream
*.png = svn:mime-type=application/octet-stream
```

- 3 为文件 config 命名并将其保存到恰当的位置：

- Windows 上的 C:\Users\myusername\AppData\Roaming\Subversion\config 或 C:\Documents and Settings\myusername\Application Data\Subversion\config
- Linux 或 macOS 上的 ~/.subversion 。

当您创建 SVN config 文件后，SVN 将带有这些扩展名的新文件视为二进制文件。如果存储库中已经有二进制文件，请参阅“[注册存储库中已经拥有的文件](#)”（第 33-16 页）。

## 更新现有的 SVN 配置文件

如果找到现有的 config 文件，则您以前已经安装了 SVN。编辑 config 文件以便将文件注册为二进制。

- 1 在文本编辑器中编辑 config 文件。
- 2 找到 [miscellany] 部分，并确认在下行中通过 yes 启用了 auto-props：

```
enable-auto-props = yes
```

确保此行未被注释掉（也即：此行的开头没有 # 符号）。配置文件中可能含有被注释掉的示例行。如果在此行的开头有 # 字符，请将其删除。

- 3 找到 [auto-props] 部分。确保 [auto-props] 未被注释掉。如果在此行的开头有 # 字符，请将其删除。
- 4 在 [auto-props] 部分的末尾添加以下各行：

```
*.mlx = svn:mime-type=application/octet-stream
*.mat = svn:mime-type=application/octet-stream
```

```
*.fig = svn:mime-type=application/octet-stream  
*.mdl = svn:mime-type=application/octet-stream  
*.slx = svn:mime-type= application/octet-stream  
*.mlapp = svn:mime-type= application/octet-stream  
*.p = svn:mime-type=application/octet-stream  
*.mdlp = svn:mime-type=application/octet-stream  
*.slxp = svn:mime-type=application/octet-stream  
*.sldd = svn:mime-type=application/octet-stream  
*.slxc = svn:mime-type=application/octet-stream  
*.mlproj = svn:mime-type=application/octet-stream  
*.mldatx = svn:mime-type=application/octet-stream  
*.sreqx = svn:mime-type=application/octet-stream  
*.sfx = svn:mime-type=application/octet-stream  
*.slt = svn:mime-type=application/octet-stream
```

上述各行将阻止 SVN 向存在冲突的 MATLAB 和 Simulink 文件中添加注释，并阻止其自动合并。

- 5 检查配置文件中是否有您使用的其他文件类型需要注册为二进制，以免签入时遭到损坏。检查 MEX 文件 (.mexa64、.mexmaci64、.mexw64)、.xlsx、.jpg、.pdf、.docx 等文件。在 config 文件中为您使用的每个文件类型添加一行。示例：

```
*.mexa64 = svn:mime-type=application/octet-stream  
*.mexw64 = svn:mime-type=application/octet-stream  
*.mexmaci64 = svn:mime-type=application/octet-stream  
*.xlsx = svn:mime-type=application/octet-stream  
*.docx = svn:mime-type=application/octet-stream  
*.pdf = svn:mime-type=application/octet-stream  
*.jpg = svn:mime-type=application/octet-stream  
*.png = svn:mime-type=application/octet-stream
```

- 6 保存 config 文件。

当您创建或更新了 SVN config 文件后，SVN 会将新文件视为二进制文件。如果存储库中已经拥有文件，请按照“[注册存储库中已经拥有的文件](#)”（第 33-16 页）中介绍的方法为其注册。

### 注册存储库中已经拥有的文件

**小心** 更改您的 SVN config 文件不会影响已经提交到 SVN 存储库的文件。如果文件未注册为二进制，请使用 `svn propset` 将这些文件手动注册为二进制文件。

要将存储库中的文件手动注册为二进制，请通过命令行 SVN 使用以下命令：

```
svn propset svn:mime-type application/octet-stream binaryfilename
```

### 标准存储库结构

使用标准的 `tags`、`trunk` 和 `branches` 文件来创建您的存储库，并从 `trunk` 签出文件。Subversion 工程推荐使用此结构。请参阅 <https://svn.apache.org/repos/asf/subversion/trunk/doc/user/svn-best-practices.html>。

如果使用 MATLAB 创建 SVN 存储库，它创建的是标准的存储库结构。要启用标记，存储库必须包含标准文件夹 `trunk/` 和 `tags/`。使用此结构创建存储库后，您可以通过点击[源代码管理](#)上下文菜单中的[添加标签](#)，向您的所有文件中添加标签。有关详细信息，请参阅“[文件的标签版本](#)”（第 33-17 页）。

## 文件的标签版本

在 SVN 中，您可以使用标签来标识所有文件的特定修订版本。要在 SVN 中使用标签，您需要在存储库中使用标准的文件夹结构，并需要从 **trunk** 签出您的文件。请参阅“标准存储库结构”（第 33-16 页）。

- 1 在当前文件夹浏览器中点击右键，然后选择**源代码管理 > 标记**。
- 2 指定标签文本并点击**提交**。标签被添加到文件夹中的每个文件。如果您的存储库中没有 **tags** 文件夹，则会显示错误。

---

**注意** 您可以从源代码管理中检索文件的标记版本，但无法用新的标签来再次对其进行标记。您必须从 **trunk** 签出以创建新的标签。

---

## 强制实施编辑前锁定文件

如果要求用户在编辑文件前记得获取文件锁，请配置 SVN 将带有指定扩展名的文件设置为只读。当您的文件为只读时，您需要在当前文件夹浏览器中点击右键，并选择**源代码管理 > 获取文件锁**，然后才能对文件进行编辑。此设置将阻止用户在未获取文件锁的情况下编辑文件。当文件有锁时，其他用户就知道该文件正在被编辑，并且您可以避免合并问题。

要强制锁定文件，请修改 SVN **config** 文件中的条目。要定位您的 SVN **config** 文件，请参阅“使用 SVN 注册二进制文件”（第 33-14 页）。

- 1 要将带有 **.m** 扩展名的文件设置为只读，请在 SVN **config** 文件的 **[auto-props]** 部分中添加一个属性。如果带有 **.m** 扩展名的文件不存在条目，请使用 **needs-lock** 属性进行添加。

```
*.m = svn:needs-lock=yes
```

如果存在条目，则可以按任意顺序组合属性，但多个条目必须位于一行上并用分号分隔。

- 2 要将带有 **.mlx** 扩展名的文件设置为只读，请在 SVN **config** 文件的 **[auto-props]** 部分中添加一个属性。由于您必须将带有 **.mlx** 扩展名的文件注册为二进制文件，因此存于此文件类型的条目。按任意顺序将 **needs-lock** 属性添加到该条目，但必须位于同一行上并用分号分隔。

```
*.mlx = svn:mime-type=application/octet-stream;svn:needs-lock=yes
```

- 3 重新创建沙盒使配置生效。

在此设置下，您需要选择**获取文件锁**后才能编辑带有 **.m** 扩展名的文件。请参阅“获取 SVN 文件锁”（第 33-22 页）。

## 共享 Subversion 存储库

您可以使用 **file://** 协议指定存储库位置。但是，Subversion 文档强烈建议仅单一用户通过 **file://** URL 直接访问存储库。请参阅网页：

<http://svnbook.red-bean.com/en/1.7/svn-book.html#svn.serverconfig.choosing.recommendations>

---

**小心** 请勿允许多个用户通过 **file://** URL 直接访问存储库，否则您会面临存储库被损坏的风险。请只对单一用户的存储库使用 **file://** URL。

---

如果您使用 MATLAB 创建存储库，请注意此警告。MATLAB 采用的是 **file://** 协议。创建新存储库的功能仅为本地、单一用户访问而提供，用于测试和调试目的。通过 **file://** URL 访问存储库比使用服务器要慢。

当您要共享存储库时，需要设置一台服务器。您可以使用 `svnserve` 或 Apache SVN 模块。请参阅网页参考内容：

<http://svnbook.red-bean.com/en/1.7/svn-book.html#svn.serverconfig.svnserve>  
<http://svnbook.red-bean.com/en/1.7/svn-book.html#svn.serverconfig.httpd>

### 另请参阅

#### 相关示例

- “从 SVN 存储库签出”（第 33-19 页）

## 从 SVN 存储库签出

通过从源代码管理检索文件来创建存储库的本地新副本。

- 1 右键点击当前文件夹浏览器的空白处（任意空白区域），然后选择**源代码管理 > 管理文件**。
- 2 在“使用源代码管理系统管理文件”对话框中，于**源代码管理集成**列表中选择源代码管理接口。要使用 SVN，请保留默认的“SVN”。
- 3 如果您知道存储库位置，请将其粘贴到**存储库路径**字段中。

否则，要浏览并验证从中检索文件的存储库路径，请点击**更改**。

- a 在“指定 SVN 存储库 URL”对话框中指定存储库 URL，您可以输入 URL、从最近使用的存储库列表选择，或使用**存储库**按钮 。

---

**小心** 请只对单一用户的存储库使用 `file://` URL。有关详细信息，请参阅“共享 Subversion 存储库”（第 33-17 页）。

---

- b 点击**验证**以检查存储库路径。

如果您看到存储库的身份验证对话框，请输入登录信息以继续。

- c 如果路径无效，请对照您的源代码管理存储库浏览器检查 URL。

如有必要，请选择存储库树中较深的文件夹。如果您的存储库包含文件的标记版本，则您可能希望从 `trunk` 或 `tags` 下的分支文件夹中签出。您可以从分支签出，但内置 SVN 集成不支持分支合并。使用 TortoiseSVN 等外部工具执行分支合并。

- d 当您完成指定希望检索的 URL 路径后，请点击**确定**。对话框关闭，您返回到“使用源代码管理系统管理文件”对话框。

- 4 在“使用源代码管理系统管理文件”对话框中，选择用于存储检索到的文件的沙盒文件夹，并点击**检索**。

如果您看到存储库的身份验证对话框，请输入登录信息以继续。

---

**小心** 使用本地沙盒文件夹。在 SVN 中使用网络文件夹会拖慢源代码管理操作的速度。

---

从源代码管理中检索文件时，“使用源代码管理系统管理文件”对话框会显示消息。

---

**注意** 要从源代码管理更新现有的沙盒，请参阅“更新 SVN 文件状态和修订版本”（第 33-21 页）。

---

## 检索存储库的标记版本

要在 SVN 中使用标签，您需要在存储库中使用标准的文件夹结构。有关详细信息，请参阅“标准存储库结构”（第 33-16 页）。

- 1 右键点击当前文件夹浏览器的空白处，然后选择**源代码管理 > 管理文件**。
- 2 在“使用源代码管理系统管理文件”对话框中，于**源代码管理集成**列表中选择源代码管理接口。要使用 SVN，请保留默认的“SVN”。
- 3 点击**更改**以选择您要从其中检索文件的“存储库路径”。

- 4 在“指定 SVN 存储库 URL”对话框中，执行以下操作：
  - a 从**存储库**列表中选择一个最近使用的存储库，或点击**存储库**按钮  以浏览存储库位置。
  - b 点击**验证**以显示存储库浏览器。
  - c 在存储库树中展开**tags**文件夹，并选择您需要的标签版本。如果 URL 包含**trunk**，则在存储库中向上导航一级目录。
  - d 点击**确定**继续，返回“使用源代码管理系统管理文件”对话框。
- 5 选择用于接收标记文件的沙盒文件夹。您必须使用一个空的沙盒文件夹或指定一个新文件夹。
- 6 点击**检索**。

### 另请参阅

#### 相关示例

- “设置 SVN 源代码管理”（第 33-14 页）
- “更新 SVN 文件状态和修订版本”（第 33-21 页）

# 更新 SVN 文件状态和修订版本

## 本节内容

- “刷新文件状态” (第 33-21 页)
- “更新文件的修订版本” (第 33-21 页)

## 刷新文件状态

要刷新文件的源代码管理状态, 请在当前文件夹浏览器中选择一个或多个文件, 右键点击并选择**源代码管理 > 刷新 SVN 状态**。

要刷新文件夹中所有文件的状态, 请右键点击当前文件夹浏览器的空白处, 然后选择**源代码管理 > 刷新 SVN 状态**。

**注意** 对于 SVN, 刷新源代码管理状态的操作并不会联系存储库。要获取最新的修订版本, 请参阅 “**更新文件的修订版本**” (第 33-21 页)。

## 更新文件的修订版本

要更新选定文件的本地副本, 请在当前文件夹浏览器中选择一个或多个文件, 右键点击并选择**源代码管理 > 从 SVN 更新所选内容**。

要更新文件夹中的所有文件, 请右键点击当前文件夹浏览器, 然后选择**源代码管理 > 全部从 SVN 更新**。

## 另请参阅

### 相关示例

- “从 SVN 存储库签出” (第 33-19 页)
- “在源代码管理中审核更改” (第 33-7 页)

## 获取 SVN 文件锁

要编辑文件，比较好的做法是先获取该文件的文件锁。文件锁会告诉其他用户该文件正在被编辑，从而可以避免合并问题。当您设置源代码管理时，可以通过配置 SVN 将带有某些扩展名的文件设置为只读。用户必须在这些只读文件上获取文件锁后才能进行编辑。

在当前文件夹浏览器中，选择您要签出的文件。右键点击选定的文件，然后选择**源代码管理 > 获取文件锁**。源代码管理状态栏中会出现一个锁符号 。其他用户无法在其沙盒中看到锁符号，但当您拥有锁时，他们无法获取文件锁，也无法签入更改内容。要查看或打开锁，请右键点击当前文件夹浏览器并选择**源代码管理 > 锁**。

如果您看到一条 SVN 消息，报告工作副本被锁定错误，请将旧锁删除。在当前文件夹浏览器中，右键点击并选择**源代码管理 > SVN 清理**。SVN 在内部使用工作副本锁，这些锁不是您使用**源代码管理 > 获取文件锁**控制的文件锁。

## 管理 SVN 存储库锁

要管理存储库的全局 SVN 锁，请从顶层存储库文件夹中右键点击当前文件夹浏览器中的空白区域（任意空白区域），然后选择**源代码管理 > 锁**。

在**SVN 存储库锁**对话框中，您可以：

- 查看哪些用户对文件加了锁。
- 右键点击以打开锁。
- 按用户或文件对锁进行分组。

## 另请参阅

### 相关示例

- “强制实施编辑前锁定文件”（第 33-17 页）

# 设置 Git 源代码管理

## 本节内容

- “关于 Git 源代码管理”（第 33-23 页）
- “安装命令行 Git 客户端”（第 33-24 页）
- “在 Git 中注册二进制文件”（第 33-24 页）
- “添加 Git 子模块”（第 33-26 页）

## 关于 Git 源代码管理

Git 与 MATLAB 的集成提供了分布式源代码管理，支持创建和合并分支。Git 是一个分布式源代码管理工具，因此您能将更改内容提交到本地存储库，并稍后与其他远程存储库同步。

Git 支持分布式开发，因为每个沙盒均包含了完整的存储库。在本地保存每个文件的完整的修订历史记录。如此就能支持离线工作，因为您无需再为每一处本地编辑而联系远程存储库和提交文件，仅在推送批量更改内容时需要这样做。此外，您还可以创建您自己的分支和提交本地编辑。这种方法更快，而您也无需在每次提交时合并其他更改内容。

Git 源代码管理的功能：

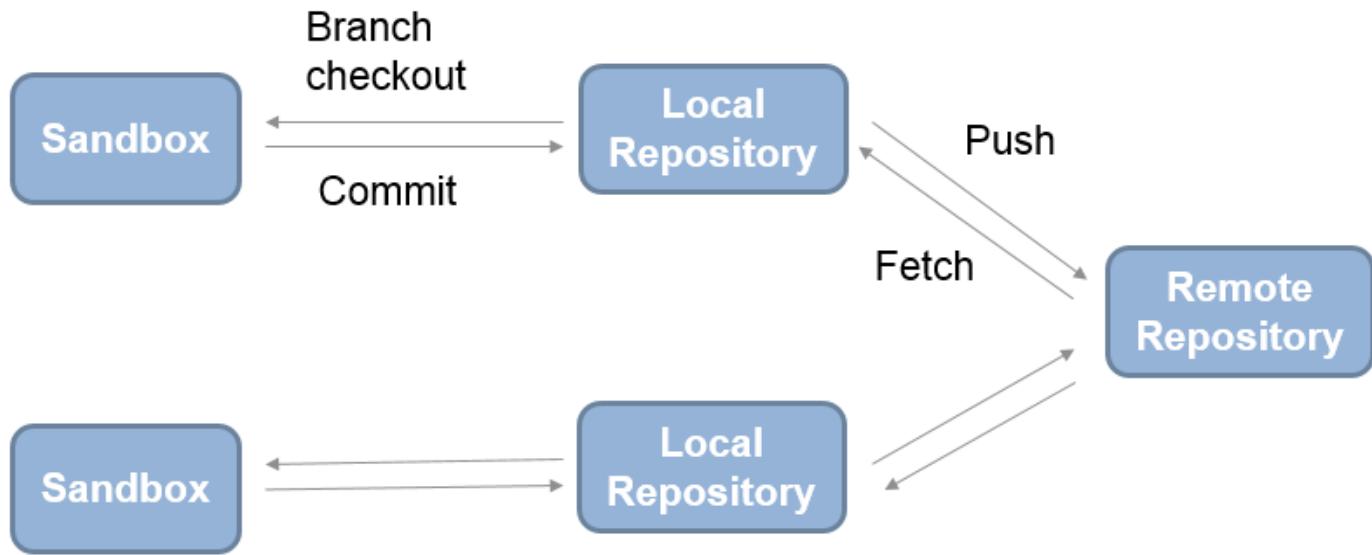
- 分支管理
- 本地完整的修订历史记录
- 本地访问比远程访问更快
- 离线工作
- 分别跟踪文件名称和内容
- 强制实施更改日志以用于责任追踪
- 整合批量更改功能

上述功能并不适用所有情形。例如，如果您的工程不适合离线工作，或者保存完整的本地修订历史记录会导致您的存储库太大，则 Git 不是理想的源代码管理工具。此外，如果您需要在编辑前强制锁定文件，Git 不具备此功能。在此类情形中，SVN 是更佳选择。

当您在 MATLAB 中使用 Git 时，您可以：

- 创建本地 Git 存储库。
- 从远程 Git 存储库取回文件。
- 创建和切换分支。
- 本地合并分支。
- 本地提交。
- 向远程 Git 存储库推送文件。

下图展示了分布式 Git 的工作流。



## 安装命令行 Git 客户端

如果您希望在 MATLAB 中使用 Git 合并分支，则还必须安装一个系统范围内可用的命令行 Git 客户端。无需进行其他安装，您就可以使用其他 Git 功能。

某些客户端不可在系统范围内使用，包括 GitHub（开始菜单上的 **Git Shell**）提供的 mingw32 环境。安装命令行 Git 会使其在系统范围内可用，然后 MATLAB 可以找到标准 ssh 键。

通过在 MATLAB 中使用 `!git` 命令来检查 Git 是否可用。如果 Git 不可用，请安装它。安装命令行 Git 客户端并将文件注册为二进制文件后，您就能在 MATLAB 中使用 Git 的合并功能。

在 Windows 上：

1 下载 Git 安装程序并运行。您可以在以下网址找到命令行 Git：

<https://msysgit.github.io/>

2 在调整 PATH 的部分中，选择从 Windows 命令提示符使用 Git 的安装选项。此选项将 Git 添加到您的 PATH 变量，从而使 MATLAB 能够与 Git 通信。

3 在配置行结束符转换的部分中，选择选项原样签出，原样提交，以避免转换文件中的任何行结束符。

4 为了避免损坏二进制文件，在使用 Git 合并分支前，请先注册二进制文件。

在大多数的 Linux 发行版中都可以使用 Git。安装适用于您的发行版的 Git。例如，在 Debian® 上，通过输入以下命令即可安装 Git：

```
sudo apt-get install git
```

在 Mac 上的 Mavericks (10.9) 或更高版本中，尝试从终端运行 `git`。如果您尚未安装 Git，系统将提示您安装 Xcode 命令行工具。有关更多选项，请参阅 <https://git-scm.com/doc>。

## 在 Git 中注册二进制文件

如果您使用第三方源代码管理工具，则必须将 `.mlx`、`.mat`、`.fig`、`.mlapp`、`.mdl`、`.slx`、`.mdlp`、`.slxp`、`.sldd` 和 `.p` 等 MATLAB 和 Simulink 文

件扩展名注册为二进制格式。如果不注册这些扩展名，则当您通过更改行尾字符、扩展标记、替换关键字或尝试自动合并来提交文件时，这些工具可能会损坏您的文件。不论您是在 MATLAB 外部使用该源代码管理工具，还是在未先注册文件格式的前提下尝试从 MATLAB 提交文件，都可能发生损坏。

还要检查其他文件扩展名是否已注册为二进制文件，以避免在签入时损坏。检查并注册 MEX 文件、.xlsx、.jpg、.pdf、.docx 等文件。

安装了命令行 Git 客户端后，您可以阻止 Git 插入冲突标记，以免该工具损坏您的文件。为此，请编辑您的 .gitattributes 文件来注册二进制文件。有关详细信息，请参阅：

<https://git-scm.com/docs/gitattributes>

- 如果您的沙盒文件夹中尚无 .gitattributes 文件，请通过 MATLAB 命令提示符创建一个：

edit .gitattributes

- 将以下各行添加到 .gitattributes 文件：

```
*.mlx -crlf -diff -merge
*.mat -crlf -diff -merge
*.fig -crlf -diff -merge
*.mdl -crlf -diff -merge
*.slx -crlf -diff -merge
*.mlapp -crlf -diff -merge
*.p -crlf -diff -merge
*.mdlp -crlf -diff -merge
*.slxp -crlf -diff -merge
*.sldd -crlf -diff -merge
*.slxc -crlf -diff -merge
*.mlproj -crlf -diff -merge
*.mldatx -crlf -diff -merge
*.sreqx -crlf -diff -merge
*.sfx -crlf -diff -merge
*.slt -crlf -diff -merge
```

上述各行指定不要对这些文件类型尝试自动换行、比较差异和尝试合并内容。

- 检查配置文件中是否有您使用的其他文件类型需要注册为二进制，以免签入时遭到损坏。检查 MEX 文件（.mexa64、.mexmaci64、.mexw64）、.xlsx、.jpg、.pdf、.docx 等文件。在属性文件中为您需要的每个文件类型添加一行。

示例：

```
*.mexa64 -crlf -diff -merge
*.mexw64 -crlf -diff -merge
*.mexmaci64 -crlf -diff -merge
*.xlsx -crlf -diff -merge
*.docx -crlf -diff -merge
*.pdf -crlf -diff -merge
*.jpg -crlf -diff -merge
*.png -crlf -diff -merge
```

- 重新启动 MATLAB 后您就能开始使用 Git 客户端。

---

**提示** 您可以无压缩保存 Simulink 模型以减小 Git 存储库的大小。关闭压缩会导致磁盘上的 SLX 文件变大，但可以减小存储库的大小。

要对新 SLX 文件使用此设置，请在“SLX 压缩”设置为无的状态下使用模型模板创建模型。对于现有 SLX 文件，请设置压缩，然后保存模型。有关详细信息，请参阅“设置 SLX 压缩级别”(Simulink)。

---

## 添加 Git 子模块

要重用其他存储库中的代码，您可以指定 Git 子模块。

要将外部 Git 存储库克隆为子模块，请执行以下操作：

- 1 在 MATLAB 当前文件夹浏览器中点击右键，然后选择**源代码管理 > 子模块**。
- 2 在“子模块”对话框中点击**+**按钮。
- 3 在“添加子模块”对话框的**远程**框中，指定一个存储库位置。您也可以点击**验证**。
- 4 在**路径**框中，为子模块指定位置并点击**确定**。“子模块”对话框将会显示子模块的状态和详细信息。
- 5 检查状态消息并点击**关闭**。

## 更新子模块

在使用**取回**从远程存储库中获取最新更改后，点击“子模块”检查子模块是否为最新，然后点击**更新**。如果子模块定义有任何更改，则更新将确保子模块文件夹包含正确的文件。更新应用于子模块层次结构中的所有子级子模块。

## 对子模块使用提取和合并

当您要管理所添加的子模块时，请打开“子模块”对话框。

- 1 要获取子模块的最新版本，请在“子模块”对话框中点击**提取**。
- 2 提取之后，您必须进行合并。检查“子模块”对话框中的**状态**消息，了解有关您的当前分支相对于存储库中远程跟踪分支的信息。当您看到在后面消息时，您需要将更改从存储库合并到您的本地分支。
- 3 点击**分支**，然后使用“分支”对话框将原分支中的更改合并到您的本地分支。请参阅“提取和合并”（第 33-34 页）。

## 使用推送将更改发送到子模块存储库

如果您在子模块中进行了更改，并且要将更改发送回存储库，请执行以下操作：

- 1 在父文件夹中执行本地提交。
- 2 打开“子模块”对话框并点击**推送**。

如果您希望其他用户在克隆父文件夹时获取子模块中的更改，请确保索引和标头匹配。

- 1 在“子模块”对话框中，检查索引和标头值。索引指向在您首次克隆子模块时的标头提交，或者指向在您最后提交父文件夹时的标头提交。如果索引和标头不匹配，您必须更新索引。
- 2 要更新索引，请提交父文件夹中的更改，然后点击“子模块”对话框中的**推送**。此操作会将索引和标头设置为相同。

## 另请参阅

### 相关示例

- “从 Git 存储库克隆”（第 33-27 页）

## 从 Git 存储库克隆

克隆远程 Git 存储库以检索存储库文件。

- 1 右键点击当前文件夹浏览器的空白处（任意空白区域），然后选择**源代码管理 > 管理文件**。
- 2 在“使用源代码管理系统管理文件”对话框中，从**源代码管理集成**列表中选择“Git”。
- 3 如果您知道存储库位置，请将其粘贴到**存储库路径**字段中。  
否则，要浏览并验证从中检索文件的存储库路径，请点击**更改**。
  - a 在“选择存储库”对话框中，使用**远程**按钮  浏览到您的存储库。
  - b 点击**验证**以检查存储库路径。  
如果您看到存储库的身份验证对话框，请输入登录信息以继续。
  - c 如果路径有效，点击**确定**。对话框关闭，您返回到“使用源代码管理系统管理文件”对话框。
- 4 在“使用源代码管理系统管理文件”对话框中，选择要存储检索到的文件的沙盒文件夹，并点击**检索**。  
如果您看到存储库的身份验证对话框，请输入登录信息以继续。

## 故障排除

如果您遇到 **OutOfMemoryError: Java heap space** 之类的错误（例如在克隆大型 Git 存储库时），请编辑您的 MATLAB 预设项以增加内存大小。

- 1 在**主页**选项卡的**环境**部分，点击**预设**。
- 2 选择 **MATLAB > 常规 > Java 堆内存**。
- 3 移动滑块以增加堆大小，然后点击**确定**。
- 4 重新启动 MATLAB。

## 另请参阅

### 相关示例

- “设置 Git 源代码管理”（第 33-23 页）
- “更新 Git 文件状态和修订版本”（第 33-28 页）
- “Git 的分支和合并”（第 33-29 页）

# 更新 Git 文件状态和修订版本

## 本节内容

- “刷新文件状态”（第 33-28 页）
- “更新文件的修订版本”（第 33-28 页）

## 刷新文件状态

要刷新文件的源代码管理状态，请在当前文件夹浏览器中选择一个或多个文件，右键点击并选择**源代码管理 > 刷新 Git 状态**。

要刷新存储库中所有文件的状态，请右键点击当前文件夹浏览器的空白处，然后选择**源代码管理 > 刷新 Git 状态**。

## 更新文件的修订版本

要更新存储库中的所有文件，请在当前文件夹浏览器中点击右键，然后选择**源代码管理 > 取回**。

**小心** 在使用**取回**前，请确保您已在 Git 中注册了二进制文件。如未注册，冲突标记可能损坏您的文件。有关详细信息，请参阅“[在 Git 中注册二进制文件](#)”（第 33-24 页）。

取回最新更改，然后将它们合并到当前分支中。如果您不确定将出现在存储库中的内容，请先使用提取功能检查更改，然后手动合并更改。有关详细信息，请参阅“[使用 Git 取回、推送和提取文件](#)”（第 33-33 页）。

如果出现冲突，取回可能会失败。对于复杂更改，您可能要从源创建一个分支，进行一些兼容性更改，然后将该分支合并到主跟踪分支中。

## 另请参阅

### 相关示例

- “[从 Git 存储库克隆](#)”（第 33-27 页）
- “[在源代码管理中审核更改](#)”（第 33-7 页）

# Git 的分支和合并

## 本节内容

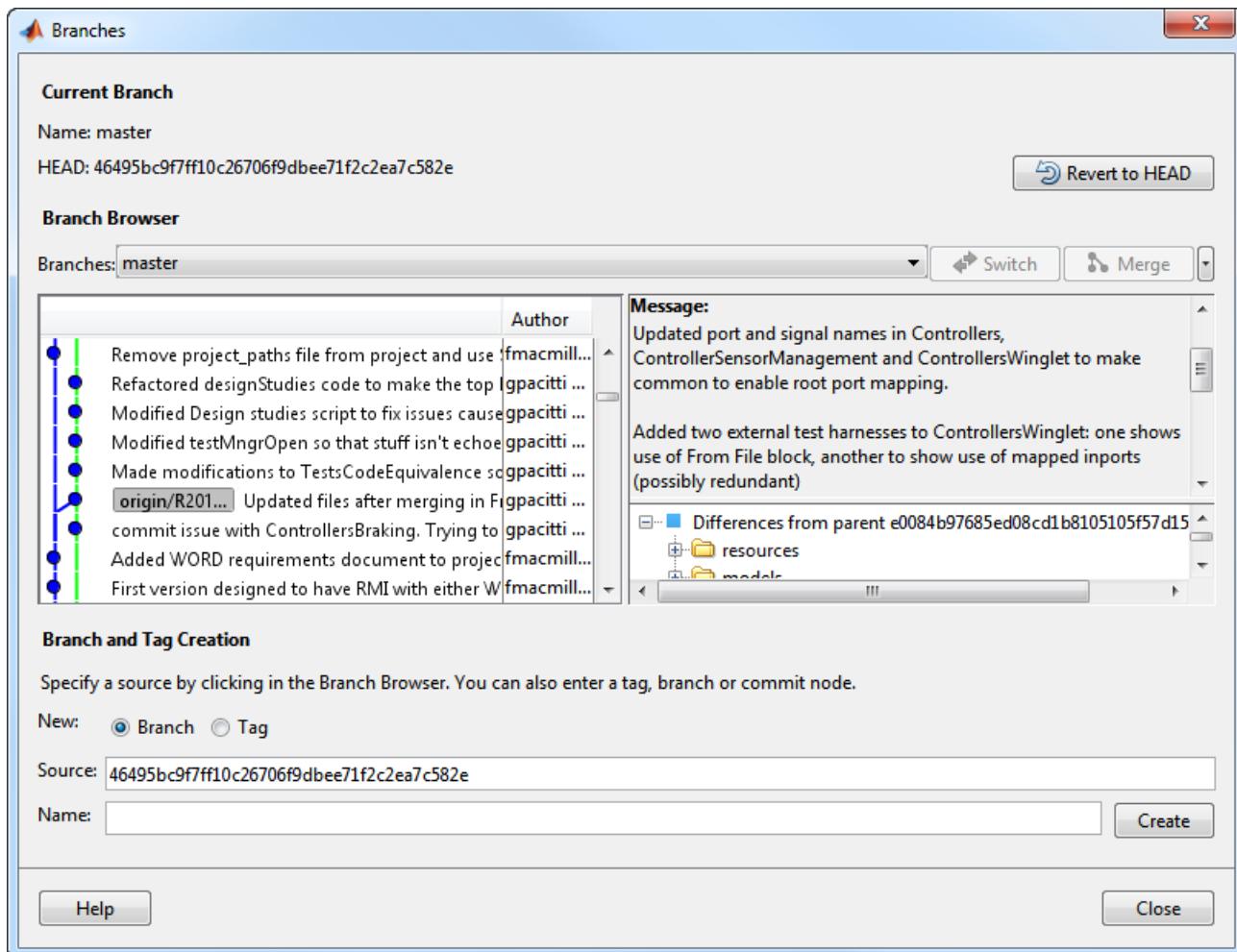
- “创建分支” (第 33-29 页)
- “切换分支” (第 33-30 页)
- “比较分支” (第 33-31 页)
- “合并分支” (第 33-31 页)
- “还原到 HEAD” (第 33-31 页)
- “删除分支” (第 33-32 页)

## 创建分支

- 1 在 Git 存储库文件夹中，右键点击当前文件夹浏览器的空白处，并选择源代码管理 > 分支。在 “分支” 对话框中，您可以查看、切换、创建和合并分支。

**提示** 您可以检查关于每个提交节点的信息。选择**分支浏览器**图中的节点，以查看作者、日期、提交消息和更改的文件。

下图中的**分支浏览器**演示了一个示例分支历史记录。



- 2 为新分支选择源。在**分支浏览器**图中点击一个节点，或在**源**文本框中输入一个唯一标识符。您可以输入标签、分支名称或 SHA1 哈希码的唯一前缀（例如：73c637 可标识一次特定提交）。保留默认值可从当前文件分支的头部创建分支。
- 3 在**分支名称**文本框中输入一个名称并点击**创建**。
- 4 要在新分支的文件上工作，请将您的工程切换至该分支。
- 在**分支**下拉列表中，选择您要切换到的分支并点击**切换**。
- 5 关闭“分支”对话框，然后处理您的分支中的文件。

有关后续步骤，请参阅“使用 Git 取回、推送和提取文件”（第 33-33 页）。

## 切换分支

- 1 在 Git 存储库文件夹中，右键点击当前文件夹浏览器的空白处，并选择**源代码管理 > 分支**。
- 2 在“分支”对话框中，从**分支**下拉列表中选择您要切换到的分支并点击**切换**。
- 3 关闭“分支”对话框，然后处理您的分支中的文件。

## 比较分支

在 Git 存储库文件夹中，右键点击当前文件夹浏览器的空白处，并选择**源代码管理 > 分支**。

- 要检查一个文件的当前修订版本与其父级之间的差异，请右键点击树中与父级的差异下的文件，然后选择**显示差异**。
- 要检查一个文件的任何两个修订版本之间的差异，包括两个不同开发分支上的修订版本，请按住 **Ctrl** 键并选择两个不同修订版本。右键点击树中与所选的差异下的文件，然后选择**显示差异**。

MATLAB 会打开一份比较报告。您可以保存所选文件的任一修订版本的副本。右键点击文件并选择**另存为**，保存文件的所选修订版本的副本。选择**将父级另存为**，保存文件的前一修订版本的副本。此功能可帮助您测试代码在以前的修订版本或其他分支上的运行情况。

## 合并分支

在合并分支前，您必须在您的系统路径上安装命令行 Git 并注册二进制文件以防止 Git 插入冲突标记。请参阅“[安装命令行 Git 客户端](#)”（第 33-24 页）。

---

**提示** 提取更改后，您必须合并。有关详细信息，请参阅“[提取和合并](#)”（第 33-34 页）。

---

要合并任何分支，请执行以下操作：

- 1 在 Git 存储库文件夹中，右键点击当前文件夹浏览器的空白处，并选择**源代码管理和分支**。
- 2 在“分支”对话框中，从**分支**下拉列表中选择您要合并到当前分支的分支，并点击**合并**。
- 3 关闭“分支”对话框，然后处理您的分支中的文件。

如果分支合并引发了 Git 无法自动解决的冲突，会弹出错误对话框报告自动合并失败。请解决冲突后继续操作。

---

**小心** 请勿移动或删除 MATLAB 外部的文件，因为这会导致出现合并错误。

---

### 保留您的版本

- 1 要保留您的文件版本，右键点击文件并选择**标记为冲突已解决**。
- 2 点击**提交已修改文件**，提交标记为冲突已解决的更改内容。

### 查看分支版本中的冲突

如果您合并了分支且文件中存在冲突，则 Git 会将文件标记为冲突且不会修改文件内容。右键点击文件，然后选择**源代码管理 > 查看冲突**。此操作会打开比较报告，显示您的分支与您要合并到的分支上文件之间的差异。确定如何解决冲突。请参阅“[解决源代码管理冲突](#)”（第 33-9 页）。

### 还原到 HEAD

- 1 在 Git 存储库文件夹中，右键点击当前文件夹浏览器的空白处，并选择**源代码管理 > 分支**。
- 2 在“分支”对话框中，点击**还原到 HEAD** 以删除所有本地更改。

## 删除分支

- 1 在“分支”对话框的**分支浏览器**下面，展开**分支**下拉列表，然后选择要删除的分支。
- 2 在最右侧，点击向下箭头并选择**删除分支**。

---

**小心** 分支删除操作无法撤消。

---

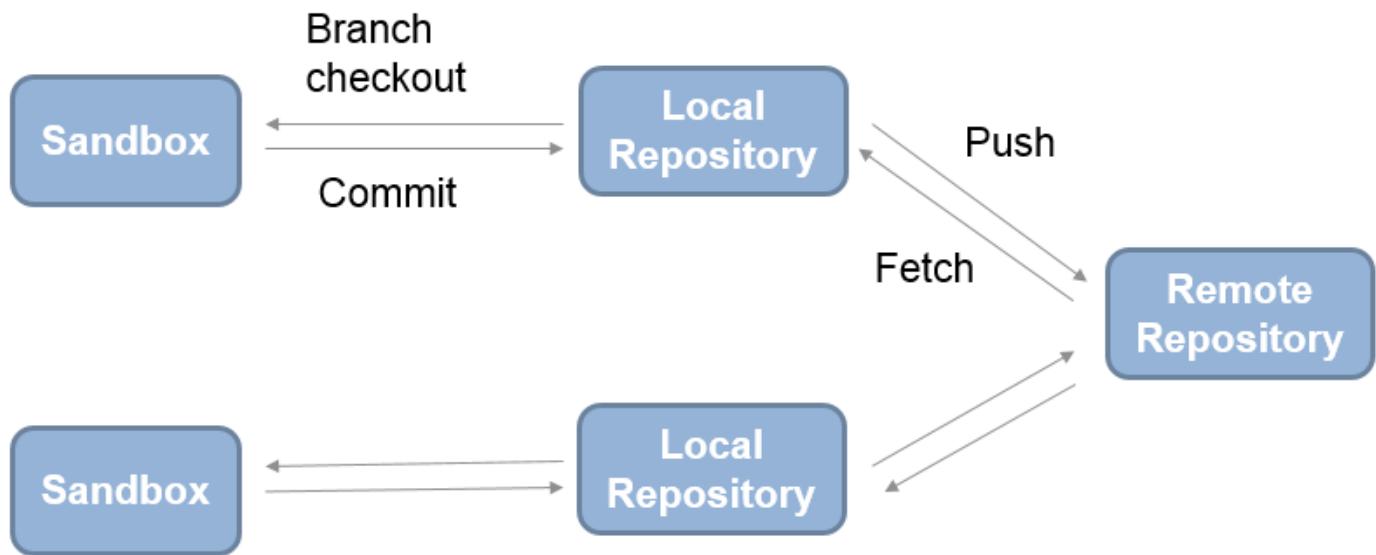
## 另请参阅

### 相关示例

- “设置 Git 源代码管理”（第 33-23 页）
- “使用 Git 取回、推送和提取文件”（第 33-33 页）
- “解决源代码管理冲突”（第 33-9 页）

## 使用 Git 取回、推送和提取文件

按照以下工作流使用远程存储库开展工作。使用 Git 时的工作流有两步：提交本地更改，然后推送到远程存储库。在 MATLAB 中，访问远程存储库的唯一途径是通过**取回**、**推送**和**提取**菜单选项。**与父级比较**和**提交**等所有其他动作都使用本地存储库。下图显示了 Git 的工作流。



### 取回和推送

要获取最新更改，请右键点击当前文件夹浏览器，然后选择**源代码管理 > 取回**。取回最新更改，然后将它们合并到当前分支中。如果您不确定将出现在存储库中的内容，请先使用提取功能检查更改，然后手动合并更改。

---

**注意** 在合并前，您必须先安装命令行 Git 并注册二进制文件，以防止 Git 插入冲突标记。请参阅“[安装命令行 Git 客户端](#)”（第 33-24 页）。

---

如果出现冲突，取回可能会失败。对于复杂更改，您可能要从源创建一个分支，进行一些兼容性更改，然后将该分支合并到主跟踪分支中。

要将更改提交到本地存储库，请右键点击当前文件夹浏览器，然后选择**源代码管理 > 查看并提交更改**。

要查看您的本地更改是否已移到远程跟踪分支之前，请右键点击该文件或当前文件夹浏览器的空白区域，然后选择**源代码管理 > 查看详细信息**。Git 信息字段指示您提交的本地更改是位于远程跟踪分支之前、之后还是与之重合。

要将本地提交发送到远程存储库，请在当前文件夹浏览器中点击右键，并选择**源代码管理 > 推送**。如果由于存储库已发生变化导致您无法直接推送更改内容，则会显示消息。在当前文件夹浏览器中点击右键，然后选择**源代码管理 > 提取**，以从远程存储库提取所有更改内容。合并更改内容并解决冲突，然后您就能推送您的更改内容了。

使用 Git，您无法向源代码管理添加空文件夹，因此您无法选择**推送**然后克隆空文件夹。您可以在 MATLAB 中创建一个空文件夹，但如果要在推送更改后同步新沙盒，则空文件夹不会在新沙盒中出现。要将空文件夹推送到存储库以供其他用户同步，请在文件夹中创建一个 `gitignore` 文件，再推送您的更改内容。

## 提取和合并

使用**提取**可获取更改并手动合并。使用**取回**则可提取最新更改并将它们合并到当前分支中。

**注意** 提取之后，您必须进行合并。在合并分支前，您必须先安装命令行 Git 并注册二进制文件，以防止 Git 插入冲突标记。请参阅“[安装命令行 Git 客户端](#)”（第 33-24 页）。

要从远程存储库提取更改内容，请在当前文件夹浏览器中点击右键，并选择**源代码管理 > 提取**。获取操作将更新本地存储库中的所有原分支。您的沙盒文件不会发生更改。要查看其他更改，您需要将原分支中的更改合并到您的本地分支。

有关您的当前分支相对于存储库中远程跟踪分支的信息，请右键点击该文件或当前文件夹浏览器的空白区域，然后选择**源代码管理 > 查看详细信息**。**Git 信息**字段指示您提交的本地更改是位于远程跟踪分支之前、之后还是与之重合。当您看到**Behind**消息时，您需要将更改从存储库合并到您的本地分支。

例如，如果您位于主分支上，则从远程存储库中的主分支中获取所有更改内容。

- 1 在当前文件夹浏览器中点击右键，然后选择**源代码管理 > 提取**
- 2 在当前文件夹浏览器中点击右键，并选择**源代码管理 > 分支**。
- 3 在“分支”对话框中，选择**分支**列表中的**origin/master**。
- 4 点击**合并**。原分支的更改内容即合并到您沙盒中的主分支。

如果您右键点击当前文件夹浏览器并选择**源代码管理 > 查看详细信息**，则**Git 信息**字段会指示**Coincident with /origin/master**。现在，您可以在本地沙盒中查看从远程存储库提取和合并的更改。

## 使用 Git 暂存文件

通过创建 Git 暂存文件来存储未提交的更改，以供以后使用。暂存文件可用于：

- 存储修改的文件而不提交。
- 将更改轻松移动到新的分支。
- 在暂存文件内浏览和检查更改。

要创建和管理暂存文件，请在当前文件夹浏览器中，右键点击由 Git 管理的文件夹中的空白区域，然后选择**源代码管理 > 暂存文件**。

在“暂存文件”对话框中：

- 要创建包含当前已修改文件的暂存文件，请点击**新建暂存文件**。
- 要查看暂存文件中的已修改文件，请从**可用暂存文件**下选择暂存文件。右键点击修改的文件，以查看更改或保存副本。
- 要将暂存文件应用于当前分支，然后删除暂存文件，请点击**恢复**。
- 要应用并保留暂存文件，请点击**应用**。
- 要删除暂存文件，请点击**丢弃**。

## 另请参阅

### 相关示例

- “Git 的分支和合并”（第 33-29 页）
- “解决源代码管理冲突”（第 33-9 页）

## 移动、重命名或删除源代码管理下的文件

请使用 MATLAB 源代码管理上下文菜单选项或其他源代码管理客户端应用程序执行移动、重命名或删除操作。

要移动处于源代码管理下的某个文件，请在当前文件夹浏览器中右键点击该文件，选择**源代码管理 > 移动**，然后输入新的文件位置。

要重命名处于源代码管理下的某个文件，请在当前文件夹浏览器中右键点击该文件，选择**源代码管理 > 重命名**，然后输入一个新文件名。

要从存储库中删除某个文件，请将该文件标记为删除。

- 要将某文件标记为从存储库中删除并保留本地副本，请在当前文件夹浏览器中右键点击该文件。选择**源代码管理**，然后选择**从 SVN 删除或从 Git 删除**。当文件被标记为从源代码管理中删除时，符号会更改  
为“已删除”。此文件会在下一提交时从存储库中删除。
- 要将某文件标记为从存储库和磁盘中删除，请在当前文件夹浏览器中右键点击该文件。选择**源代码管理**，然后选择**从 SVN 和磁盘删除或从 Git 和磁盘删除**。此文件会从当前文件夹浏览器中消失，并立即  
从磁盘中删除。此文件会在下一提交时从存储库中删除。

## 另请参阅

### 相关示例

- “标记文件以添加到源代码管理”（第 33-8 页）
- “将已修改文件提交到源代码管理”（第 33-12 页）

## 自定义外部源代码管理以使用 MATLAB 执行差异分析和合并

您可以自定义外部源代码管理工具，以使用 MATLAB 比较工具执行差异分析和合并。如果您要通过源代码管理工具来比较 MATLAB 文件（例如实时脚本、MAT、SLX 或 MDL 文件），则可以对源代码管理工具进行配置以打开 MATLAB 比较工具。

MATLAB 比较工具为 MathWorks 文件提供了有用的合并工具，并且与所有常用的软件配置管理和版本控制系统兼容。

通过执行以下步骤，将您的源代码管理工具设置为使用 MATLAB 作为所需文件扩展名（例如 .mlx、.mat、.slx 或 .mdl）的差异分析和合并应用程序。

- 1 要获取所需文件路径并设置预设以重用打开的 MATLAB 会话，请在 MATLAB 中运行以下命令：

```
comparisons.ExternalSCMLink.setup()
```

该命令会在**比较**下设置称为**允许外部源代码管理工具使用打开的 MATLAB 会话执行差异分析和合并**的 MATLAB 预设。

该命令还会显示您将复制并粘贴到源代码管理工具设置中的文件路径：

- 在 Windows 上：

```
matlabroot\bin\win64\mlDiff.exe
matlabroot\bin\win64\mlMerge.exe
```

- 在 Linux 上：

```
matlabroot/bin/glnxa64/mlDiff
matlabroot/bin/glnxa64/mlMerge
```

在 Mac 上：

```
matlabroot/bin/mac64/mlDiff
matlabroot/bin/mac64/mlMerge
```

其中，matlabroot 将替换为您的安装的完整路径，例如 C:\Program Files\MATLAB\R2016b。

- 2 设置差异分析。

- a 在 MATLAB 命令行窗口中，复制 mlDiff 的文件路径，例如 C:\Program Files\MATLAB\R2016b\bin\win64\mlDiff.exe。
- b 在源代码管理工具中，找到差异分析设置，然后添加一个条目以指定要对特定文件扩展名（例如 .slx）执行的操作。粘贴您在 MATLAB 命令行窗口中复制的 mlDiff 的文件路径。
- c 在脚本路径之后或在参数框中，添加参数以指定输入文件。查找特定于源代码管理工具的参数名称。按照以下顺序指定用于差异分析的输入：leftFile、rightFile。

例如，对于 SourceTree Git：

```
"C:\Program Files\MATLAB\R2016b\bin\win64\mlDiff.exe" $LOCAL $PWD/$REMOTE
```

对于 Tortoise SVN：

```
"C:\Program Files\MATLAB\R2016b\bin\win64\mlDiff.exe" %base %mine
```

对于 Perforce® P4V：

```
"C:\Program Files\MATLAB\R2016b\bin\win64\mlDiff.exe" %1 %2
```

**3 设置合并。**

- a 在 MATLAB 命令行窗口中，复制 mlMerge 的文件路径。**
- b 在源代码管理工具中，找到合并设置，然后添加一个条目以指定要对特定文件扩展名（例如 .slx）执行的操作。粘贴您在 MATLAB 命令行窗口中复制的 mlMerge 的文件路径。**
- c 在脚本路径之后或在参数框中，添加参数以指定输入文件。查找特定于源代码管理工具的参数名称。按照以下顺序指定用于合并的输入：base、mine、theirs 和 merged 目标文件。**

例如，对于 SourceTree Git：

```
"C:\Program Files\MATLAB\R2016b\bin\win64\mlMerge.exe" $PWD/$BASE $PWD/$LOCAL $PWD/$REMOTE $MERGED
```

对于 Tortoise SVN：

```
"C:\Program Files\MATLAB\R2016b\bin\win64\mlMerge.exe" %base %mine %theirs %merged
```

对于 Perforce P4V：

```
"C:\Program Files\MATLAB\R2016b\bin\win64\mlMerge.exe" %b %2 %1 %r
```

**4 进行该设置后，当您使用差异分析或合并时，您的外部源代码管理工具将会在 MATLAB 比较工具中打开一个报告。使用该报告查看更改并解决合并问题。**

您的差异分析和合并操作将使用打开的 MATLAB 会话（如果有），并且仅在必要时打开 MATLAB。比较仅使用指定的 MATLAB 安装。

## 另请参阅

### 相关示例

- “[比较和合并文本](#)”
- “[比较和合并 MAT 文件](#)”
- “[比较变量](#)”
- “[从比较报告合并 Simulink 模型](#)” (Simulink)

## MSSCCI 源代码管理接口

**注意** MSSCCI 支持已删除。将此功能替换为以下选项之一。

- 将作为 MathWorks “源代码管理集成”一部分的源代码管理系统与当前文件夹浏览器配合使用。
- 使用源代码管理软件开发包为源代码管理创建插件。
- 使用 MATLAB `system` 函数访问您的源代码管理工具的命令行 API。此选项不会提供与 MATLAB 当前文件夹浏览器菜单或源代码管理状态栏的集成。

---

如果您使用源代码管理系统管理您的文件，则可以与这些系统对接，以便从 MATLAB、Simulink 和 Stateflow 产品中执行源代码管理操作。使用 MATLAB、Simulink 或 Stateflow 产品中的菜单项，或在 MATLAB 命令行窗口中运行函数，便可与源代码管理系统对接。

Windows 上的源代码管理接口使用符合 Microsoft 常用源代码管理标准 1.1 版的任何源代码管理系统。如果您的源代码管理系统不符合该标准，请使用适合您的源代码管理系统的 Microsoft 源代码管理 API 封装程序产品，以便从 MATLAB、Simulink 和 Stateflow 产品中与其对接。

本文档使用 Microsoft Visual SourceSafe® 软件作为示例。您的源代码管理系统可能使用不同的术语并且不支持相同的选项，或者可能以不同方式使用它们。无论如何，您应该可以基于本文档使用您的源代码管理系统执行类似操作。

从当前文件夹浏览器中执行大多数源代码管理接口操作。您也可以从 MATLAB 编辑器、Simulink 模型窗口或 Stateflow 图窗窗口中对单个文件执行下述众多操作 - 有关详细信息，请参阅“从编辑器访问 MSSCCI 源代码管理”（第 33-52 页）。

## 设置 MSSCCI 源代码管理

**注意** MSSCCI 支持已删除。将此功能替换为以下选项之一。

- 将作为 MathWorks “源代码管理集成”一部分的源代码管理系统与当前文件夹浏览器配合使用。
- 使用源代码管理软件开发包为源代码管理创建插件。
- 使用 MATLAB `system` 函数访问您的源代码管理工具的命令行 API。此选项不会提供与 MATLAB 当前文件夹浏览器菜单或源代码管理状态栏的集成。

### 本节内容

- “在源代码管理系统中创建工程”（第 33-40 页）
- “通过 MATLAB 软件指定源代码管理系统”（第 33-41 页）
- “向 MATLAB 软件中注册源代码管理工程”（第 33-42 页）
- “向源代码管理中添加文件”（第 33-44 页）

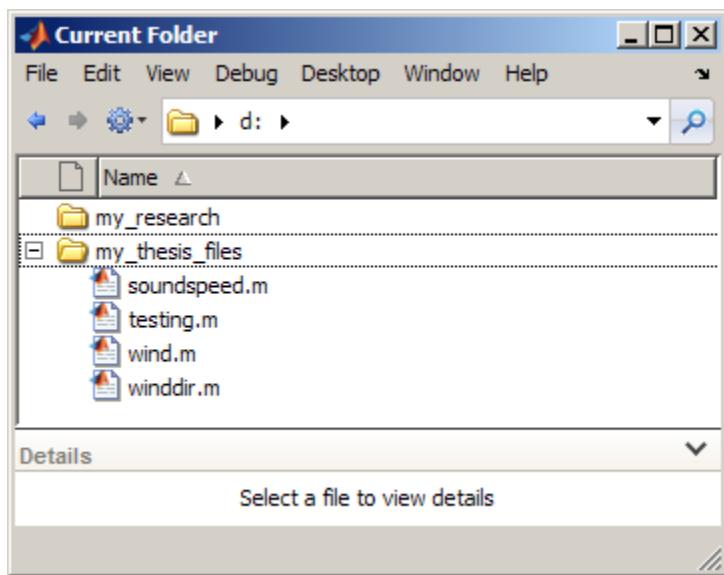
## 在源代码管理系统中创建工程

在您的源代码管理系统中，创建要与文件夹和文件相关联的工程。

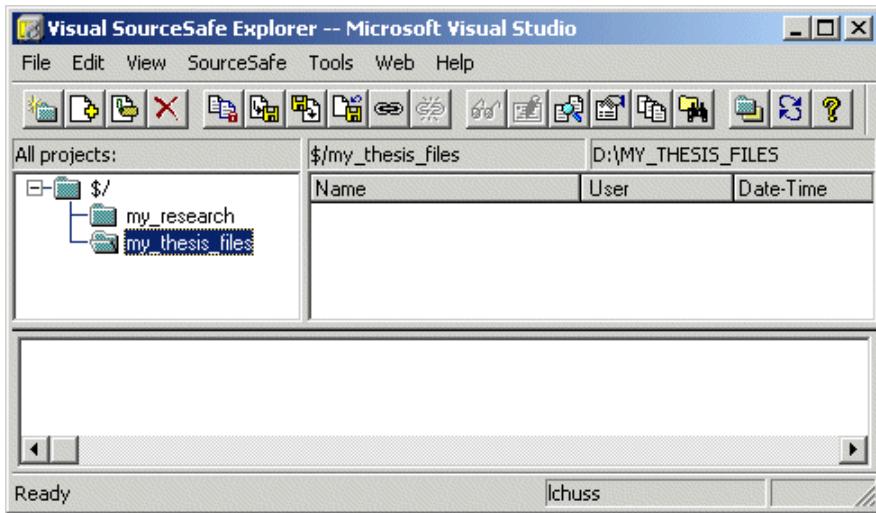
文件夹中的所有文件必须属于同一源代码管理工程。确保源代码管理系统中的工程的工作文件夹指定了该文件夹在磁盘上的正确路径。

### 创建源代码管理工程的示例

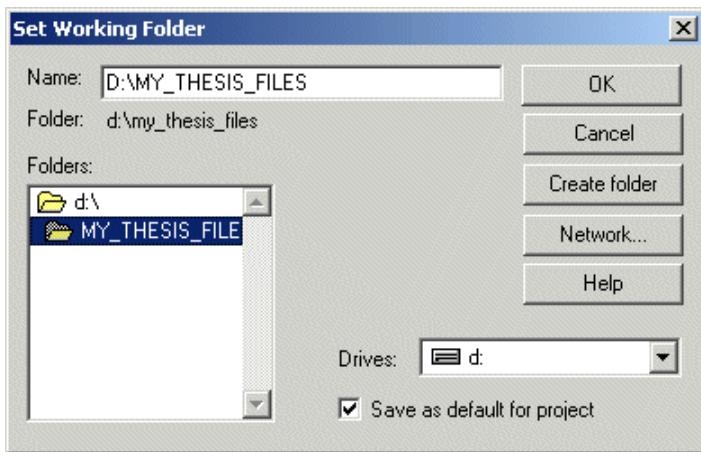
此示例使用 Microsoft Visual SourceSafe 中的工程 `my_thesis_files`。其中展示的当前文件夹浏览器显示了该文件夹在磁盘上的路径 `D:\my_thesis_files`。



下面的插图显示了源代码管理系统中的工程示例。



要在 Microsoft Visual SourceSafe 中为此示例设置工作文件夹, 请选择 my\_thesis\_files 并点击右键, 从上下文菜单中选择**设置工作文件夹**, 然后在出现的对话框中指定 D:\my\_thesis\_files。

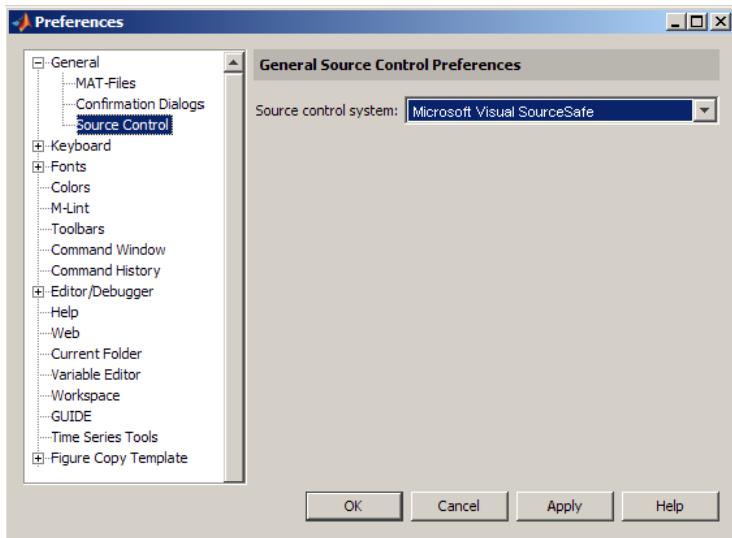


## 通过 MATLAB 软件指定源代码管理系统

在 MATLAB 中, 指定您要访问的源代码管理系统。在**主页**选项卡上的**环境**部分中, 点击**预设 > MATLAB > 常规 > 源代码管理**。

当前选择的系统显示在“**预设项**”对话框中。该列表包括支持 Microsoft 常用源代码管理标准的所有已安装的源代码管理系统。

选择您要与其对接的源代码管理系统, 并点击**确定**。



MATLAB 会记住不同会话之间的预设，因此您只需在要访问不同源代码管理系统时再次执行此操作。

### 用于 64 位版本的 MATLAB 的源代码管理

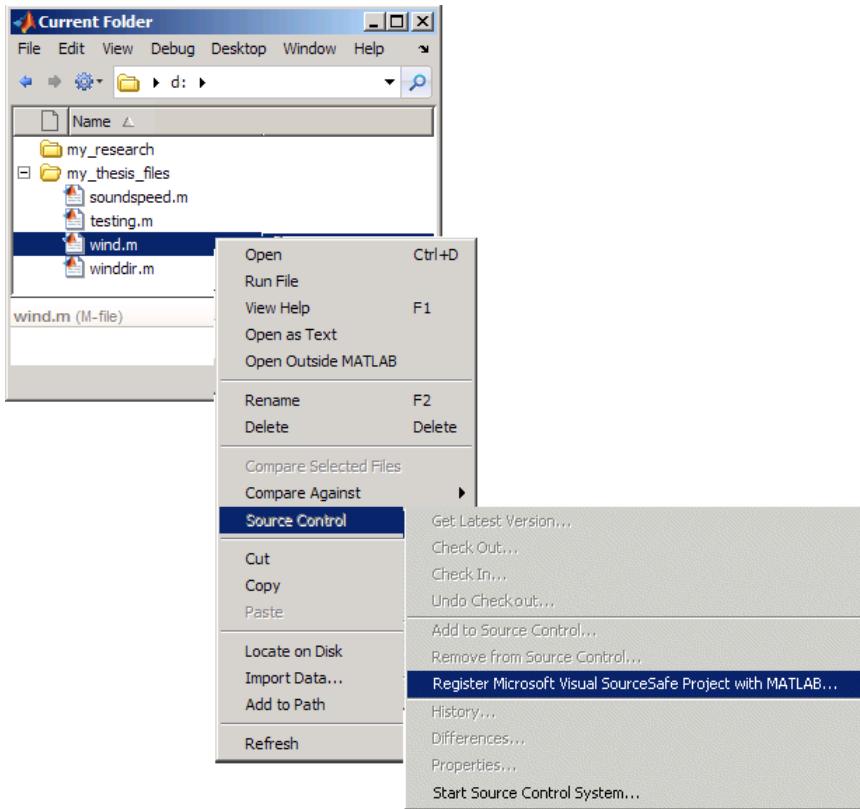
如果您运行 64 位版本的 MATLAB 并希望 MATLAB 与您的源代码管理系统对接，您的源代码管理系统必须与 64 位软件兼容。如果安装的是 32 位源代码管理，或者有以 32 位兼容模式运行的 64 位源代码管理系统，则 MATLAB 不能使用它。在此情况下，MATLAB 会在源代码管理预设窗格中显示有关该问题的警告。

### 向 MATLAB 软件中注册源代码管理工程

在 MATLAB 内的一个文件夹中注册源代码管理系统工程，即，将源代码管理系统工程与该文件夹及其中的所有文件发生关联。对该文件夹中的任何文件仅执行此操作一次，这会注册该文件夹中的所有文件：

- 1 在 MATLAB 当前文件夹浏览器中，选择一个位于您要与源代码管理系统中的某工程相关联的文件夹中的文件。例如，选择 D:\my\_thesis\_files\wind.m。这将与 my\_thesis\_files 文件夹中的所有文件相关联。
- 2 右键点击，然后从上下文菜单中选择**源代码管理 > 在 MATLAB 中注册 Name of Source Control System 工程**。**Name of Source Control System** 是使用“通过 MATLAB 软件指定源代码管理系统”（第 33-41 页）中所述的预设项选择的源代码管理系统。

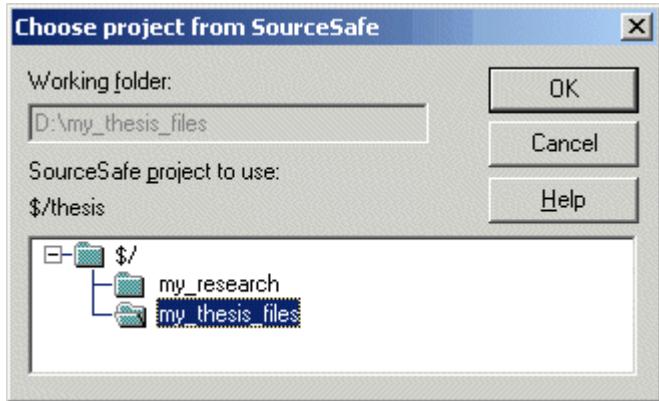
下面的示例显示了 Microsoft Visual SourceSafe。



- 3 在生成的 **Name\_of\_Source\_Control\_System Login** 对话框中，提供用于访问您的源代码管理系统的用户名和密码，并点击确定。



- 4 在生成的从 **Name\_of\_Source\_Control\_System** 中选择工程对话框中，选择要与该文件夹相关联的源代码管理系统工程，并点击确定。此示例显示了 my\_thesis\_files。

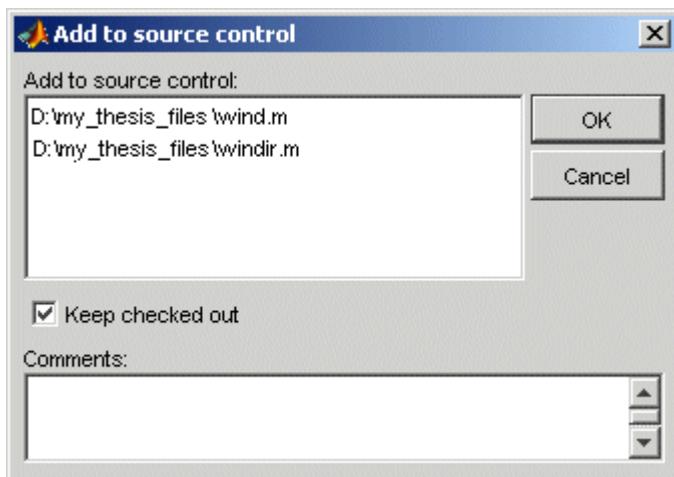


选择的文件、该文件所在的文件夹以及该文件夹中的所有文件都与您选择的源代码管理系统工程相关联。例如，MATLAB 将 D:\my\_thesis\_files 中的所有文件与源代码管理工程 my\_thesis\_files 相关联。

## 向源代码管理中添加文件

向源代码管理系统中添加文件。对每个文件仅执行此操作一次：

- 1 在当前文件夹浏览器中，选择要添加到源代码管理系统中的文件。
- 2 右键点击，然后从上下文菜单中选择**源代码管理 > 添加到源代码管理**。
- 3 生成的**添加到源代码管理**对话框中列出了您选择要添加的文件。您可以在**注释**字段中添加文本。如果您预计很快将使用这些文件，请选中**保持签出**复选框（默认处于选中状态）。点击**确定**。



如果您尝试添加未保存的文件，该文件将在添加时自动保存。

# 从 MSSCCI 源代码管理中签入和签出文件

**注意** MSSCCI 支持已删除。将此功能替换为以下选项之一。

- 将作为 MathWorks “源代码管理集成”一部分的源代码管理系统与当前文件夹浏览器配合使用。
- 使用源代码管理软件开发包为源代码管理创建插件。
- 使用 MATLAB `system` 函数访问您的源代码管理工具的命令行 API。此选项不会提供与 MATLAB 当前文件夹浏览器菜单或源代码管理状态栏的集成。

## 本节内容

“将文件签入到源代码管理系统” (第 33-45 页)

“签出源代码管理系统中的文件” (第 33-45 页)

“撤消签出” (第 33-46 页)

在从 MATLAB 桌面将文件签入或迁出源代码管理之前，务必按“设置 MSSCCI 源代码管理” (第 33-40 页) 中所述设置您的系统以与 MATLAB 配合使用。

## 将文件签入到源代码管理系统

使用 MATLAB 软件或相关产品创建或修改文件后，通过执行以下步骤将这些文件签入到源代码管理系统：

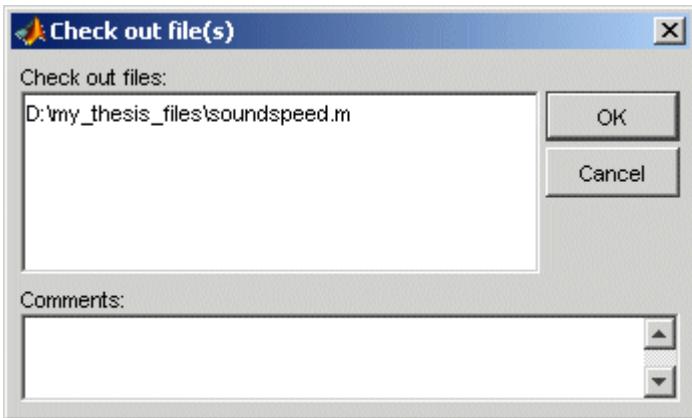
- 1 在当前文件夹浏览器中，选择要签入的文件。在签入文件时该文件可能处于打开或关闭状态，但它必须已保存，即，它不能包含未保存的更改。
- 2 右键点击，然后从上下文菜单中选择**源代码管理 > 签入**。
- 3 在生成的**签入文件**对话框中，您可以在**注释**字段中添加文本。如果您要继续使用这些文件，请选中复选框**保持签出**。点击**确定**。

如果在您尝试签入某文件时该文件包含未保存的更改，则系统会提示您保存这些更改以完成签入。如果您未使该文件保持签出状态并且将该文件保持打开状态，此时应注意它是只读版本。

## 签出源代码管理系统中的文件

从 MATLAB 中，要签出您要修改的文件，请执行以下步骤：

- 1 在当前文件夹浏览器中，选择要签出的文件。
- 2 右键点击，然后从上下文菜单中选择**源代码管理 > 签出**。
- 3 生成的**签出文件**对话框列出了您选择要签出的文件。在**注释**字段中输入注释文本，如果您的源代码管理系统支持对签出添加注释，则会显示该文本。点击**确定**。



签出文件后，在 MATLAB 中或另一产品中对其进行更改，并保存文件。例如，在编辑器中编辑文件。

如果您尝试在不事先签出文件的情况下更改该文件，则该文件是只读的（如标题栏中所示），并且您将无法保存任何更改。这样可防止您意外覆盖该文件的源代码管理版本。

如果您结束 MATLAB 会话，则文件将保持签出状态。您可以在以后的会话期间从 MATLAB 中签入该文件，或从源代码管理系统中签入文件夹。

## 撤消签出

您可以撤消签出文件。这些文件仍保持签入状态，并且不包含自您上次将其签出后所做的任何更改。要保存自签出某特定文件以来所做的任何更改，请在撤消签出之前点击**编辑器**或**实时编辑器**选项卡上的**保存**，选择**另存为**，使用不同的文件名保存文件。

要撤消签出，请执行下列步骤：

- 1 在 MATLAB 当前文件夹浏览器中，选择您要撤消签出的文件。
- 2 右键点击，然后从上下文菜单中选择**源代码管理** > **撤消签出**。

MATLAB **撤消签出**对话框随即打开，其中列出了您选择的文件。



- 3 点击确定。

## 其他 MSSCCI 源代码管理操作

**注意** MSSCCI 支持已删除。将此功能替换为以下选项之一。

- 将作为 MathWorks “源代码管理集成”一部分的源代码管理系统与当前文件夹浏览器配合使用。
- 使用源代码管理软件开发包为源代码管理创建插件。
- 使用 MATLAB `system` 函数访问您的源代码管理工具的命令行 API。此选项不会提供与 MATLAB 当前文件夹浏览器菜单或源代码管理状态栏的集成。

### 本节内容

- “获取最新版本的文件以便查看或编译”（第 33-47 页）
- “删除源代码管理系统中的文件”（第 33-47 页）
- “显示文件历史记录”（第 33-48 页）
- “将工作副本与源代码管理中的最新版本进行比较”（第 33-49 页）
- “查看文件的源代码管理属性”（第 33-50 页）
- “启动源代码管理系统”（第 33-51 页）

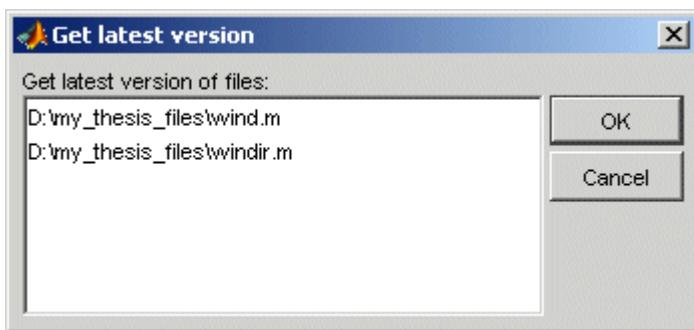
## 获取最新版本的文件以便查看或编译

您可以从源代码管理系统中获取最新版本的文件以便查看或运行。获取文件不同于签出文件。当您获取某文件时，它受写保护，因此您无法编辑它，但当您签出文件时，您可以编辑它。

要获取最新版本，请执行下列步骤：

- 1 在 MATLAB 当前文件夹浏览器中，选择要获取的文件夹或文件。如果您选择文件，则无法同时选择文件夹。
- 2 右键点击，然后从上下文菜单中选择**源代码管理 > 获取最新版本**。

MATLAB “获取最新版本” 对话框随即打开，其中列出了您选择的文件或文件夹。



- 3 点击确定。

您现在可以打开文件以进行查看，运行该文件或者签出文件以便进行编辑。

## 删除源代码管理系统中的文件

要删除源代码管理系统中的文件，请执行以下步骤：

- 1 在 MATLAB 当前文件夹浏览器中，选择要删除的文件。
- 2 右键点击，然后从上下文菜单中选择**源代码管理 > 从源代码管理中删除**。

MATLAB 从源代码管理中删除对话框随即打开，其中列出了您选择的文件。



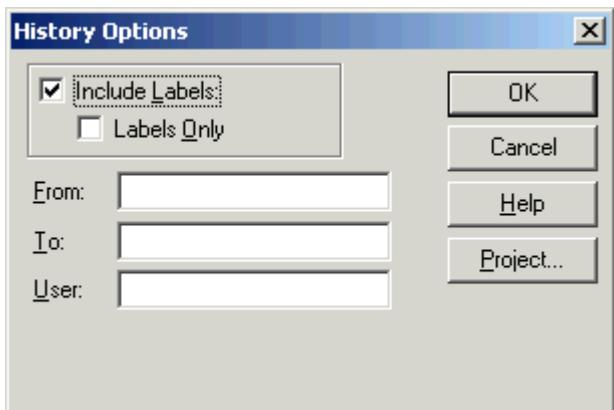
- 3 点击确定。

## 显示文件历史记录

要在源代码管理系统中显示文件的历史记录，请执行下列步骤：

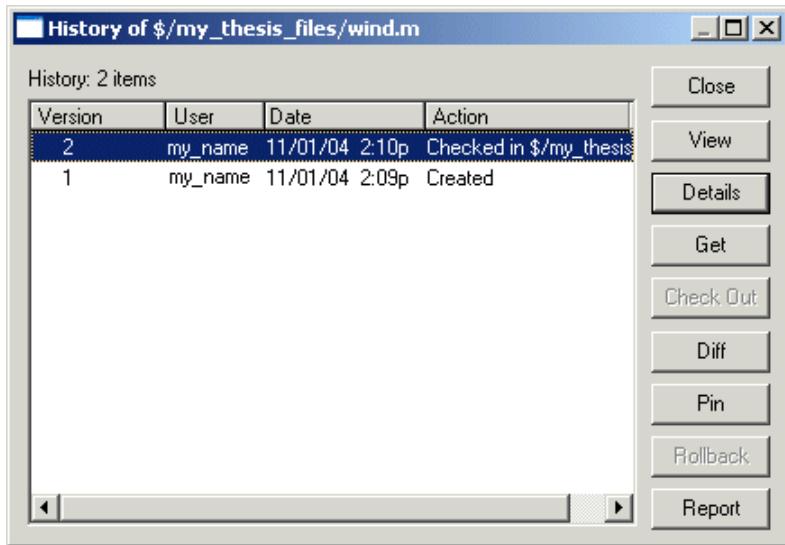
- 1 在 MATLAB 当前文件夹浏览器中，选择您要查看其历史记录的文件。
- 2 右键点击，然后从上下文菜单中选择**源代码管理 > 历史记录**。

随即打开一个特定于您的源代码管理系统的对话框。对于 Microsoft Visual SourceSafe，**历史记录选项**对话框会打开，如下例所示。



- 3 填充该对话框以指定所要求的选定文件的历史记录范围，并点击确定。例如，输入 **my\_name** 作为用户。

显示的历史记录取决于您的源代码管理系统。对于 Microsoft Visual SourceSafe，会打开该文件的**历史记录**对话框，并在源代码管理系统中显示该文件的历史记录。



## 将工作副本与源代码管理中的最新版本进行比较

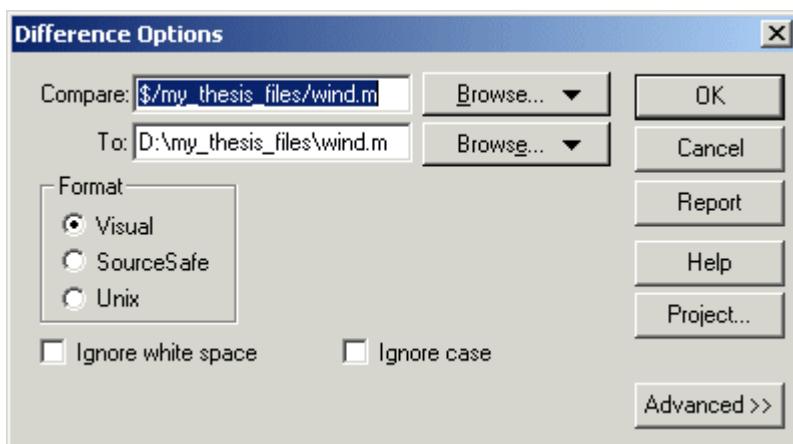
您可以将当前使用的文件版本与源代码管理系统中最新签入的文件版本进行比较。这会高亮显示两个文件的不同之处，并显示自您签出该文件后所做的更改。

要查看不同之处，请执行下列步骤：

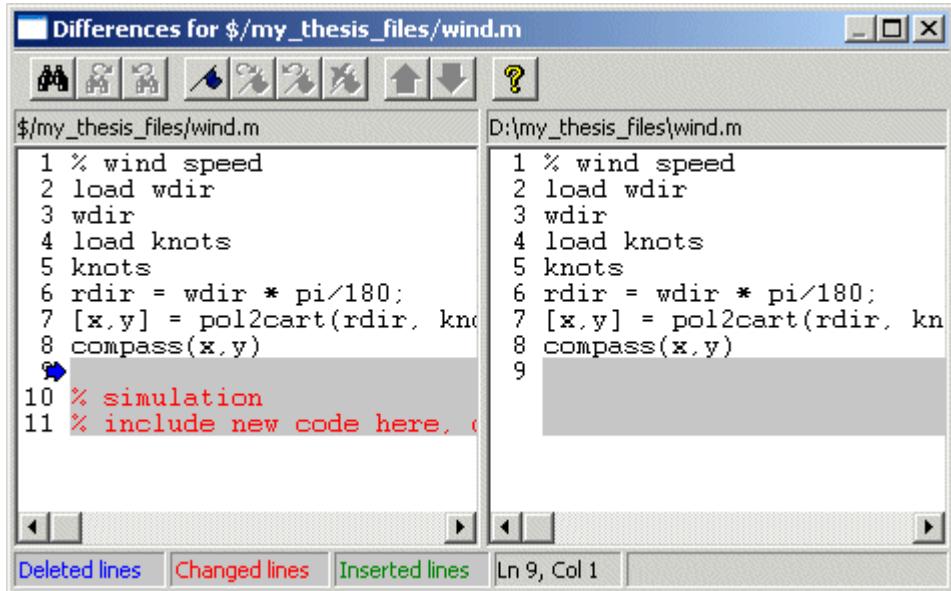
- 1 在 MATLAB 当前文件夹浏览器中，选择您要查看其不同之处的文件。这是已签出和编辑过的文件。
- 2 右键点击，然后从上下文菜单中选择**源代码管理 > 差异**。

随即打开一个特定于您的源代码管理系统的对话框。对于 Microsoft Visual SourceSafe，**差异选项**对话框随即打开。

- 3 查看该对话框中的默认项，作出所有必要的更改，然后点击**确定**。下面的示例适用于 Microsoft Visual SourceSafe。



显示差异的方法取决于您的源代码管理系统。对于 Microsoft Visual SourceSafe，**差异**对话框随即打开。这会高亮显示文件的工作副本与最新签入的文件版本之间的差异。

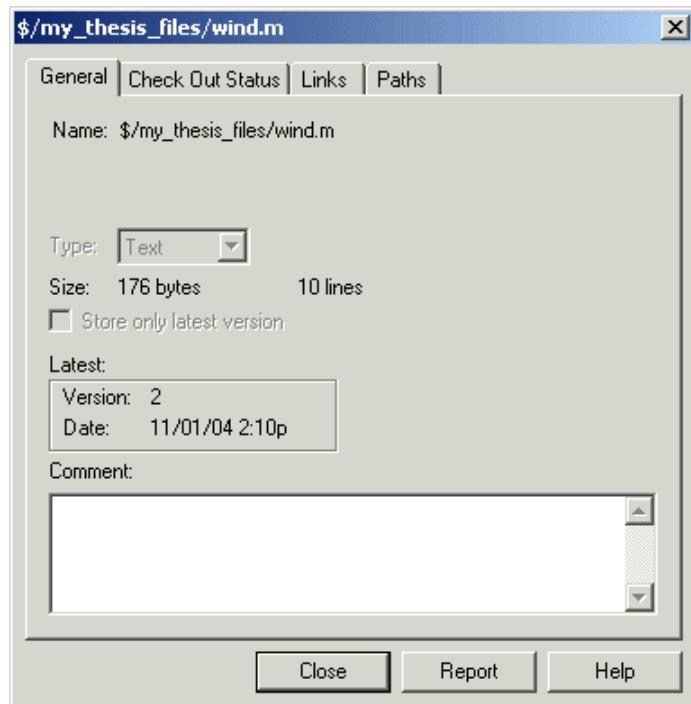


## 查看文件的源代码管理属性

要查看文件的源代码管理属性，请执行下列步骤：

- 1 在 MATLAB 当前文件夹浏览器中，选择您要查看其属性的文件。
- 2 右键点击，然后从上下文菜单中选择**源代码管理 > 属性**。

随即打开一个特定于您的源代码管理系统的对话框。下面的示例显示了 Microsoft Visual SourceSafe 属性对话框。

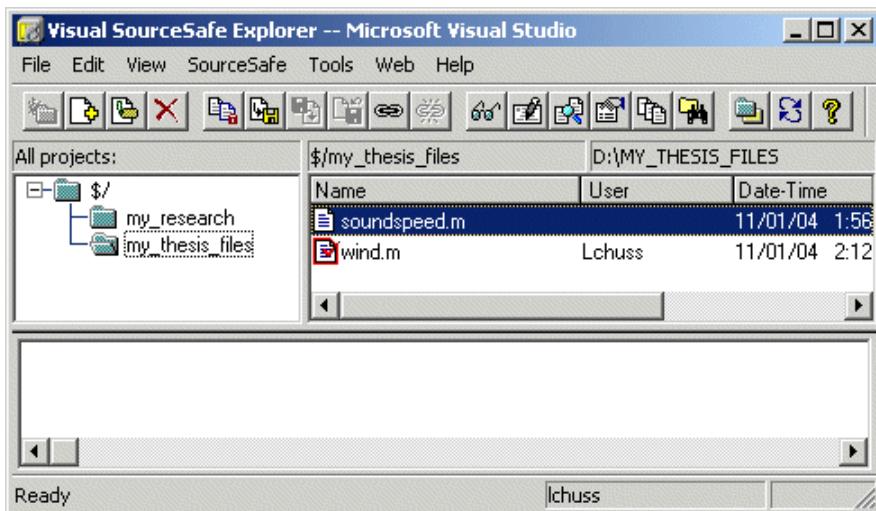


## 启动源代码管理系统

如果源代码管理系统尚未打开，所有 MATLAB 源代码管理操作都会自动启动源代码管理系统以执行相应操作。如果您希望从 MATLAB 启动源代码管理系统而不执行特定源代码管理操作，

- 1 在 MATLAB 当前文件夹浏览器中右键点击任何文件夹或文件
- 2 从上下文菜单中选择**源代码管理 > 启动源代码管理系统**。

您的源代码管理系统的界面随即打开，并在 MATLAB 中显示与当前文件夹关联的源代码管理工程。下面的示例演示了 Microsoft Visual SourceSafe 资源管理器界面。



## 从编辑器访问 MSSCCI 源代码管理

**注意** MSSCCI 支持已删除。将此功能替换为以下选项之一。

- 将作为 MathWorks “源代码管理集成”一部分的源代码管理系统与当前文件夹浏览器配合使用。
  - 使用源代码管理软件开发包为源代码管理创建插件。
  - 使用 MATLAB `system` 函数访问您的源代码管理工具的命令行 API。此选项不会提供与 MATLAB 当前文件夹浏览器菜单或源代码管理状态栏的集成。
- 

您可以在编辑器、Simulink 或 Stateflow 产品中创建或打开文件，并从其**文件 > 源代码管理**菜单而非当前文件夹浏览器中执行大多数源代码管理操作。以下是在您使用编辑器、Simulink 或 Stateflow 时源代码管理接口处理中的一些差异：

- 您一次只能对一个文件执行操作。
- 其中某些对话框在标题栏中具有不同图标。例如，**签出文件**对话框使用 MATLAB 编辑器图标而非 MATLAB 图标。
- 您不能添加新的 **(Untitled)** 文件，但必须首先保存文件。
- 您不能从 Simulink 或 Stateflow 产品注册工程。而要如“向 MATLAB 软件中注册源代码管理工程”（第 33-42 页）中所述使用当前文件夹浏览器注册工程。

# MSSCCI 源代码管理问题故障排除

**注意** MSSCCI 支持已删除。将此功能替换为以下选项之一。

- 将作为 MathWorks “源代码管理集成”一部分的源代码管理系统与当前文件夹浏览器配合使用。
- 使用源代码管理软件开发包为源代码管理创建插件。
- 使用 MATLAB `system` 函数访问您的源代码管理工具的命令行 API。此选项不会提供与 MATLAB 当前文件夹浏览器菜单或源代码管理状态栏的集成。

## 本节内容

“源代码管理错误：提供程序不存在或安装不当”（第 33-53 页）

“针对 @ 字符的限制”（第 33-53 页）

“添加到源代码管理”是唯一可用操作”（第 33-54 页）

“针对源代码管理问题的更多解决方案”（第 33-54 页）

## 源代码管理错误：提供程序不存在或安装不当

在某些情况下，MATLAB 软件可识别您的源代码管理系统，但您不能对 MATLAB 使用源代码管理功能。具体而言，当您选择“预设项”对话框中的 **MATLAB > 常规 > 源代码管理** 时，MATLAB 会列出您的源代码管理系统，但您无法执行任何源代码管理操作。仅列出启动源代码管理系统的项目，当您选择它时，MATLAB 显示以下错误：

`Source control provider is not present or not installed properly.`

出现此错误往往是因为 MATLAB 需要从源代码管理应用程序中获得的注册表项不存在。确保该注册表项存在：

**HKEY\_LOCAL\_MACHINE\SOFTWARE\SourceCodeControlProvider\InstalledSCCProviders**

该注册表项引用类似于以下的另一个注册表项

**HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\SourceSafe\SccServerPath**

该注册表项有一个指向文件系统中的 DLL 文件的路径。确保在该位置存在这个 DLL 文件。如果您不熟悉注册表项，请向您的系统管理员寻求帮助。

如果问题仍未解决，并且您使用 Microsoft Visual SourceSafe，请尝试运行您的源代码管理应用的客户端安装程序。当 SourceSafe 安装在服务器上以供某个组使用时，每个计算机客户端都可以运行安装程序，但这也不必要如此。不过，某些与 SourceSafe 对接的应用程序（包括 MATLAB）需要您运行客户端安装程序。运行客户端安装程序，应该能够解决此问题。

如果问题仍存在，请访问 MATLAB 外部的源代码管理。

## 针对 @ 字符的限制

某些源代码管理系统（例如 Perforce 和 Synergy™）保留 @ 字符。比如，Perforce 将其用作修订设定符。因此，如果您将这些源代码管理系统与名称中包含 @ 字符的 MATLAB 文件和文件夹结合使用，则可能遇到问题。

通过在文件名中引用非标准字符（例如通过某些源代码管理系统允许的转义序列），或许可以摆脱这种限制。请查阅您的源代码管理系统文档或技术支持资源来了解解决方法。

## “添加到源代码管理”是唯一可用操作

要在 Simulink 或 Stateflow 产品中对某文件使用源代码管理功能，必须首先在 MATLAB 中注册该文件的源代码管理工程。如果未在 MATLAB 中注册文件的源代码管理工程，则“预设项”对话框上的所有 **MATLAB > 常规 > 源代码管理** 菜单项都将被禁用，但**添加到源代码管理**除外。您可以选择**添加到源代码管理**，这会在 MATLAB 中注册工程，您也可以使用当前文件夹浏览器注册工程，如“向 MATLAB 软件中注册源代码管理工程”（第 33-42 页）中所述。然后，您可以对该工程（文件夹）中的所有文件执行源代码管理操作。

## 针对源代码管理问题的更多解决方案

MATLAB 与源代码管理系统对接问题的最新解决方案显示在 MathWorks 支持网页上，具体网址为 <https://www.mathworks.com/support/>。搜索与“源代码管理”有关的解决方案和技术说明。

# 单元测试

---

- “使用实时脚本编写测试” (第 34-3 页)
- “编写基于脚本的单元测试” (第 34-5 页)
- “使用局部函数编写基于脚本的测试” (第 34-10 页)
- “扩展基于脚本的测试” (第 34-13 页)
- “在编辑器中运行测试” (第 34-16 页)
- “编写基于函数的单元测试” (第 34-19 页)
- “使用函数编写简单测试用例” (第 34-22 页)
- “使用设置和拆解函数编写测试” (第 34-26 页)
- “扩展基于函数的测试” (第 34-31 页)
- “在 MATLAB 中编写基于类的单元测试” (第 34-35 页)
- “使用类来编写简单测试用例” (第 34-37 页)
- “使用类来编写设置代码和拆解代码” (第 34-41 页)
- “验证、断言及其他验证一览表” (第 34-44 页)
- “标记单元测试” (第 34-46 页)
- “使用共享脚手架编写测试” (第 34-50 页)
- “创建基本自定义脚手架” (第 34-53 页)
- “创建高级自定义脚手架” (第 34-55 页)
- “创建基本参数化测试” (第 34-60 页)
- “创建高级参数化测试” (第 34-64 页)
- “在参数化测试中使用外部参数” (第 34-70 页)
- “创建简单测试套件” (第 34-74 页)
- “为各个工作流运行测试” (第 34-76 页)
- “以编程方式访问测试诊断” (第 34-79 页)
- “向测试运行程序添加插件” (第 34-80 页)
- “编写插件以扩展 TestRunner” (第 34-82 页)
- “创建自定义插件” (第 34-85 页)
- “使用自定义插件并行运行测试” (第 34-90 页)
- “编写用于保存诊断详细信息的插件” (第 34-98 页)
- “用于生成自定义测试输出格式的插件” (第 34-102 页)
- “分析测试用例结果” (第 34-105 页)
- “分析失败的测试结果” (第 34-108 页)
- “重新运行失败的测试” (第 34-110 页)
- “动态筛选的测试” (第 34-113 页)
- “创建自定义约束” (第 34-119 页)
- “创建自定义布尔约束” (第 34-122 页)

- “创建自定义容差” (第 34-125 页)
- “App 测试框架概述” (第 34-129 页)
- “为 App 编写测试” (第 34-133 页)
- “编写使用 App 测试和模拟框架的测试” (第 34-137 页)
- “性能测试框架概述” (第 34-142 页)
- “使用脚本或函数测试性能” (第 34-145 页)
- “使用类测试性能” (第 34-149 页)
- “测量快速执行的测试代码” (第 34-155 页)
- “创建 Mock 对象” (第 34-158 页)
- “指定 Mock 对象行为” (第 34-164 页)
- “验证 Mock 对象交互” (第 34-169 页)

# 使用实时脚本编写测试

此示例说明如何编写实时脚本 'TestRightTriLiveScriptExample mlx' 来对您创建的函数进行测试。该示例函数可计算直角三角形的角度，而您将创建一个基于实时脚本的单元测试来测试该函数。

基于实时脚本的测试必须遵循以下惯例：

- 实时脚本文件的名称必须以单词 'test' 开头或结尾，且不区分大小写。
- 一个单元测试在实时脚本文件中占一节。每个节的题头将成为一个测试元素的名称。如果某节没有题头，则 MATLAB 会为测试分配一个名称。
- 考虑以何种方式运行基于实时脚本的测试。如果您使用实时编辑器中的**运行**按钮来运行测试，则 MATLAB 遇到一次测试失败时，即会停止执行脚本，且不再运行剩余的测试。如果您使用单元测试框架（例如使用 `runtests` 函数）来运行实时脚本，则 MATLAB 遇到一次测试失败后，仍会运行剩余的测试。
- 当实时脚本以测试方式运行时，在一个测试中定义的变量在其他测试中不可访问。同样，这些测试也不能访问在其他工作区中定义的变量。

在此示例之外，在您的当前 MATLAB 文件夹下的文件 `rightTri.m` 中创建一个函数。此函数将三角形的两条边长作为输入，返回所对应直角三角形的三个角的度数。输入的边是三角形两个较短的边，而不是斜边。

```
type rightTri.m
```

```
function angles = rightTri(sides)

A = atand(sides(1)/sides(2));
B = atand(sides(2)/sides(1));
hypotenuse = sides(1)/sind(A);
C = asind(hypotenuse*sind(A)/sides(1));

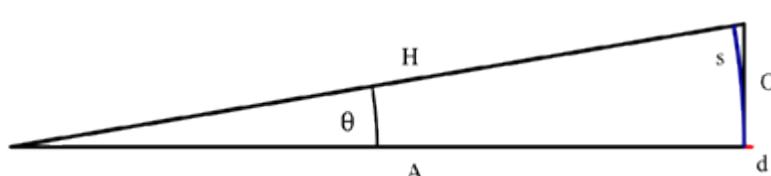
angles = [A B C];

end
```

## 测试：小角度近似

您可以在实时脚本中纳入方程和图像，以帮助记录测试。为小角度近似创建以下测试。通常，当您比较浮点值时，需要指定比较的容差。

`rightTri` 函数应返回与小角度近似一致的值，使得  $\sin(\theta) \approx \theta$ 。



```
angles = rightTri([1 1500]);
smallAngleInRadians = (pi/180)*angles(1); % convert to radians
approx = sin(smallAngleInRadians);
assert(abs(approx-smallAngleInRadians) <= 1e-10, 'Problem with small angle approximation')
```

## 测试：角度求和

$$\sum_k a_k = 180^\circ$$

您可以在同一测试中包含多个 `assert` 语句。但是，如果第一个断言失败，MATLAB 将不会计算剩余的语句。

生成的直角三角形的所有角度之和应始终为 180 度。

```
angles = rightTri([7 9]);
assert(sum(angles) == 180)

angles = rightTri([4 4]);
assert(sum(angles) == 180)

angles = rightTri([2 2*sqrt(3)]);
assert(sum(angles) == 180)
```

### 测试：30-60-90 三角形



测试验证三角形的两条边分别减为 1 和  $\sqrt{3}$ 。在此情况下，各角度分别为  $30^\circ$ ,  $60^\circ$ , and  $90^\circ$ 。

```
tol = 1e-10;
angles = rightTri([2 2*sqrt(3)]);
assert(abs(angles(1)-30)<= tol)
assert(abs(angles(2)-60)<= tol)
assert(abs(angles(3)-90)<= tol)
```

### 测试：等腰三角形

对于等腰三角形，两个非直角都必须为 45 度，否则 `assert` 会引发错误。

测试验证三角形的两条边相等。在这种情况下，对应的角度是相等的。

```
angles = rightTri([4 4]);
assert(angles(1) == 45)
assert(angles(1) == angles(2))
```

要运行您的测试，最佳做法是通过 `runtests` 函数来使用测试框架，而不是使用实时编辑器中的运行按钮。测试框架可提供其他诊断信息。如遇测试失败，该框架将继续运行后续测试，而使用实时编辑器中的运行按钮则不会运行后续测试。例如，要在 MATLAB 命令提示符下运行此测试，请键入 `result = runtests('TestRightTriLiveScriptExample')`。有关详细信息，请参阅 `runtests`。

## 另请参阅

`assert` | `runtests`

## 相关示例

- “编写基于脚本的单元测试”（第 34-5 页）
- “编写基于函数的单元测试”（第 34-19 页）

# 编写基于脚本的单元测试

此示例演示了如何编写脚本，对您创建的函数进行测试。该示例函数可计算直角三角形的角度，您还创建一个基于脚本的单元测试来测试该函数。

## 创建待测试的 rightTri 函数

在您的当前 MATLAB® 文件夹下的 `rightTri.m` 文件中创建以下函数。此函数将三角形的两条边长作为输入，返回所对应直角三角形的三个角的度数。输入的边是三角形两个较短的边，而不是斜边。

```
function angles = rightTri(sides)

A = atand(sides(1)/sides(2));
B = atand(sides(2)/sides(1));
hypotenuse = sides(1)/sind(A);
C = asind(hypotenuse*sind(A)/sides(1));

angles = [A B C];

end
```

## 创建测试脚本

在您的工作文件夹中，创建一个新脚本 `rightTriTest.m`。每个单元测试对 `rightTri` 函数的不同输出进行检查。测试脚本必须遵循以下惯例：

- 脚本文件的名称必须以单词 'test' 开头或结尾，且不区分大小写。
- 将每个单元测试放到脚本文件的单独部分。每个部分以两个百分比符号 (%%) 开始，同一行中紧跟百分比符号的文本将成为测试元素的名称。如果 %% 后面没有文本，则 MATLAB 会为测试分配一个名称。如果 MATLAB 遇到一次测试失败，则仍将运行剩余的测试。
- 在测试脚本中，共享变量部分包含在第一个显式代码节（第一个以 %% 开头的行）之前出现的所有代码。各测试将共享您在这个部分定义的变量。在一个测试内，您可以修改这些变量的值。但在后续测试中，该值会重置为在共享变量部分定义的值。
- 在共享变量部分（第一个代码节）中，为您的测试定义所有必要的先决条件。如果输入或输出不满足此先决条件，MATLAB 不会运行这些测试中的任何一个。MATLAB 将测试标记为失败和不完整。
- 在运行脚本进行测试时，一个测试中定义的变量在其他测试中不可访问，除非这些变量是在共享变量部分（第一个代码节）定义的。同样，这些测试也不能访问在其他工作区中定义的变量。
- 如果脚本文件不包含任何代码节，则 MATLAB 将从脚本文件的全部内容生成为单个测试元素。测试元素的名称与脚本文件名相同。这种情况下，如果 MATLAB 遇到失败的测试，它会停止执行整个脚本。

在 `rightTriTest.m` 中，编写四个测试来测试 `rightTri` 的输出。使用 `assert` 函数测试不同的条件。在共享变量部分，定义四个三角形，并定义 `rightTri` 函数返回直角这一先决条件。

```
% test triangles
tri = [7 9];
triIso = [4 4];
tri306090 = [2 2*sqrt(3)];
triSkewed = [1 1500];

% preconditions
angles = rightTri(tri);
```

```

assert(angles(3) == 90,'Fundamental problem: rightTri not producing right triangle')

%% Test 1: sum of angles
angles = rightTri(tri);
assert(sum(angles) == 180)

angles = rightTri(triIso);
assert(sum(angles) == 180)

angles = rightTri(tri306090);
assert(sum(angles) == 180)

angles = rightTri(triSkewed);
assert(sum(angles) == 180)

%% Test 2: isosceles triangles
angles = rightTri(triIso);
assert(angles(1) == 45)
assert(angles(1) == angles(2))

%% Test 3: 30-60-90 triangle
angles = rightTri(tri306090);
assert(angles(1) == 30)
assert(angles(2) == 60)
assert(angles(3) == 90)

%% Test 4: Small angle approximation
angles = rightTri(triSkewed);
smallAngle = (pi/180)*angles(1); % radians
approx = sin(smallAngle);
assert(approx == smallAngle, 'Problem with small angle approximation')

```

测试 1 测试三角形的角度总和。如果总和不等于 180 度，`assert` 将引发错误。

测试 2 测试如果两条边相等，则对应的角也相等。如果非直角不都等于 45 度，`assert` 函数将引发错误。

测试 3 测试如果三角形的边为 1 和  $\sqrt{3}$ ，则角度分别为 30、60 和 90 度。如果不满足此条件，`assert` 将引发错误。

测试 4 测试小角度近似。小角度近似是指对于小角度而言，以弧度表示的角度正弦值约等于该角度。如果不满足此条件，`assert` 将引发错误。

## 运行测试

执行 `runtests` 函数以运行 `rightTriTest.m` 中的四个测试。`runtests` 函数单独执行每个代码节中的每个测试。如果测试 1 失败，MATLAB 仍会运行剩余的测试。如果您将 `rightTriTest` 作为脚本执行，而不是通过使用 `runtests` 来执行，MATLAB 会在遇到失败断言时停止执行整个脚本。此外，当您使用 `runtests` 函数运行测试时，MATLAB 可提供包含有用信息的测试诊断。

```
result = runtests('rightTriTest');
```

```
Running rightTriTest
```

```
..
```

```
=====
Error occurred in rightTriTest/Test3_30_60_90Triangle and it did not run to completion.
```

```
-----
Error ID:
```

```
-----
'MATLAB:assertion:failed'
-----
Error Details:
-----
Error using rightTriTest (line 31)
Assertion failed.
=====
.
=====
Error occurred in rightTriTest/Test4_SmallAngleApproximation and it did not run to completion.
-----
Error ID:
-----
"
-----
Error Details:
-----
Error using rightTriTest (line 39)
Problem with small angle approximation
=====
.
Done rightTriTest
-----
Failure Summary:

```

Name	Failed	Incomplete	Reason(s)
rightTriTest/Test3_30_60_90Triangle	X	X	Errored.
rightTriTest/Test4_SmallAngleApproximation	X	X	Errored.

**30-60-90** 三角形测试和小角度近似测试在比较浮点数时失败。通常，当您比较浮点值时，需要指定比较的容差。在测试 3 和测试 4 中，MATLAB 在出现失败断言时引发错误，并且不完成测试。因此，测试同时被标记为 **Failed** 和 **Incomplete**。

要提供比 'Assertion failed' (测试 3) 更全面的诊断信息 (**Error Details**)，可考虑向 **assert** 函数传递消息 (如测试 4 中所示)。或者也可以考虑使用基于函数的单元测试。

### 修改测试以使用容差

将 **rightTriTest.m** 另存为 **rightTriTolTest.m**，并修订测试 3 和测试 4 以使用容差。在测试 3 和测试 4 中，并不断言角度等于某个预期值，而是断言实际值与预期值之间的差值小于或等于指定的容差。在测试脚本的共享变量部分定义该容差，以便两个测试均可访问它。

对于基于脚本的单元测试，请手动校验两个值之间的差小于指定容差。但如果您编写基于函数的单元测试，则可以访问内置约束以指定在比较浮点值时的容差。

```
% test triangles
tri = [7 9];
triIso = [4 4];
tri306090 = [2 2*sqrt(3)];
triSkewed = [1 1500];

% Define an absolute tolerance
```

```

tol = 1e-10;

% preconditions
angles = rightTri(tri);
assert(angles(3) == 90, 'Fundamental problem: rightTri not producing right triangle')

%% Test 1: sum of angles
angles = rightTri(tri);
assert(sum(angles) == 180)

angles = rightTri(triIso);
assert(sum(angles) == 180)

angles = rightTri(tri306090);
assert(sum(angles) == 180)

angles = rightTri(triSkewed);
assert(sum(angles) == 180)

%% Test 2: isosceles triangles
angles = rightTri(triIso);
assert(angles(1) == 45)
assert(angles(1) == angles(2))

%% Test 3: 30-60-90 triangle
angles = rightTri(tri306090);
assert(abs(angles(1)-30) <= tol)
assert(abs(angles(2)-60) <= tol)
assert(abs(angles(3)-90) <= tol)

%% Test 4: Small angle approximation
angles = rightTri(triSkewed);
smallAngle = (pi/180)*angles(1); % radians
approx = sin(smallAngle);
assert(abs(approx-smallAngle) <= tol, 'Problem with small angle approximation')

```

重新运行测试。

```

result = runtests('rightTriTolTest');

Running rightTriTolTest
...
Done rightTriTolTest

```

成功通过所有测试。

创建测试结果表格。

```
rt = table(result)
```

```
rt =
```

```
4x6 table
```

Name	Passed	Failed	Incomplete	Duration	Details
------	--------	--------	------------	----------	---------

---

```
{'rightTriTolTest/Test1_SumOfAngles'      } true  false  false  0.033539 {1x1 struct}
{'rightTriTolTest/Test2_IsoscelesTriangles' } true  false  false  0.006427 {1x1 struct}
{'rightTriTolTest/Test3_30_60_90Triangle'   } true  false  false  0.006521 {1x1 struct}
{'rightTriTolTest/Test4_SmallAngleApproximation'} true  false  false  0.006832 {1x1 struct}
```

## 另请参阅

`assert` | `runtests`

## 相关示例

- “使用局部函数编写基于脚本的测试”（第 34-10 页）
- “编写基于函数的单元测试”（第 34-19 页）

## 使用局部函数编写基于脚本的测试

以下示例演示如何编写将局部函数用作辅助函数的基于脚本的测试。示例函数会近似计算角的正弦和余弦。基本脚本的测试将通过使用局部函数检查在容差范围内是否相等来查看近似度。

### 创建要测试的 approxSinCos 函数

在您的当前 MATLAB 文件夹下的 `approxSinCos.m` 文件中创建以下函数。此函数接受以弧度为单位的角，并使用泰勒级数来近似计算角的正弦和余弦。

```
function [sinA,cosA] = approxSinCos(x)
% For a given angle in radians, approximate the sine and cosine of the angle
% using Taylor series.
sinA = x;
cosA = 1;
altSign = -1;
for n = 3:2:26
    sinA = sinA + altSign*(x^n)/factorial(n);
    cosA = cosA + altSign*(x^(n-1))/factorial(n-1);
    altSign = -altSign;
end
```

### 创建测试脚本

在您的当前 MATLAB 文件夹中，创建一个新脚本 `approxSinCosTest.m`。

**注意：**在脚本中包括函数需要安装 MATLAB® R2016b 或更高版本。

```
%% Test 0rad
% Test expected values of 0
[sinApprox,cosApprox] = approxSinCos(0);
assertWithAbsTol(sinApprox,0)
assertWithRelTol(cosApprox,1)

%% Test 2pi
% Test expected values of 2pi
[sinApprox,cosApprox] = approxSinCos(2*pi);
assertWithAbsTol(sinApprox,0)
assertWithRelTol(cosApprox,1)

%% Test pi over 4 equality
% Test sine and cosine of pi/4 are equal
[sinApprox,cosApprox] = approxSinCos(pi/4);
assertWithRelTol(sinApprox,cosApprox,'sine and cosine should be equal')

%% Test matches MATLAB fcn
% Test values of 2pi/3 match MATLAB output for the sin and cos functions
x = 2*pi/3;
[sinApprox,cosApprox] = approxSinCos(x);
assertWithRelTol(sinApprox,sin(x),'sin does not match')
assertWithRelTol(cosApprox,cos(x),'cos does not match')

function assertWithAbsTol(actVal,expVal,varargin)
% Helper function to assert equality within an absolute tolerance.
```

```
% Takes two values and an optional message and compares
% them within an absolute tolerance of 1e-6.
tol = 1e-6;
tf = abs(actVal-expVal) <= tol;
assert(tf, varargin{:});
end

function assertWithRelTol(actVal,expVal,varargin)
% Helper function to assert equality within a relative tolerance.
% Takes two values and an optional message and compares
% them within a relative tolerance of 0.1%.
relTol = 0.001;
tf = abs(expVal - actVal) <= relTol.*abs(expVal);
assert(tf, varargin{:});
end
```

每个单元测试都使用 `assert` 来检查 `approxSinCos` 函数的不同输出。通常，当您比较浮点值时，需要指定比较的容差。局部函数 `assertWithAbsTol` 和 `assertWithRelTol` 为辅助函数，可计算实际值和预期值在指定的绝对容差或相对容差范围内是否相等。

- **Test 0rad** 用于测试 0 弧度的角的计算值和预期值是否位于绝对容差 `1e-6` 或相对容差 0.1% 范围内。通常，您应使用绝对容差来比较接近 0 的值。
- **Test 2pi** 用于测试  $2\pi$  弧度的角计算值和预期值在 `1e-6` 的绝对容差或 0.1% 的相对容差范围内是否相等。
- **Test pi over 4 equality** 用于测试  $\frac{\pi}{4}$  的正弦和余弦在 0.1% 的相对容差范围内是否相等。
- **Test matches MATLAB fcn** 用于测试  $\frac{2\pi}{3}$  的正弦和余弦计算值在 0.1% 的相对容差范围内是否等于 `sin` 和 `cos` 函数中的值。

## 运行测试

执行 `runtests` 函数以运行 `approxSinCosTest.m` 中的四个测试。`runtests` 函数会分别执行每个测试。如遇测试失败，MATLAB 仍将运行剩余的测试。如果您将 `approxSinCosTest` 作为脚本执行，而不是通过使用 `runtests` 来执行，MATLAB 会在遇到失败断言时停止执行整个脚本。此外，当您使用 `runtests` 函数运行测试时，MATLAB 可提供包含有用信息的测试诊断。

```
results = runtests('approxSinCosTest');
```

```
Running approxSinCosTest
```

```
...
```

```
Done approxSinCosTest
```

---

成功通过所有测试。

创建测试结果表格。

```
rt = table(results)
```

```
rt =
```

```
4x6 table
```

Name	Passed	Failed	Incomplete	Duration	Details
------	--------	--------	------------	----------	---------

```
{'approxSinCosTest/Test0rad'}    true  false  false  0.33306 {1x1 struct}
{'approxSinCosTest/Test2pi'}     true  false  false  0.0204 {1x1 struct}
{'approxSinCosTest/TestPiOver4Equality'} true  false  false  0.019276 {1x1 struct}
{'approxSinCosTest/TestMatchesMATLABFcn'} true  false  false  0.040813 {1x1 struct}
```

### 另请参阅

[assert](#) | [runtests](#)

### 相关示例

- “编写基于脚本的单元测试” (第 34-5 页)

### 详细信息

- “向脚本中添加函数” (第 18-13 页)

# 扩展基于脚本的测试

## 本节内容

- “测试套件创建”（第 34-13 页）
- “测试选择”（第 34-13 页）
- “以编程方式访问测试诊断”（第 34-14 页）
- “测试运行程序自定义”（第 34-14 页）

通常，使用基于脚本的测试，您可以创建测试文件并将文件名传递到 `runtests` 函数，而无需显式创建 `Test` 对象套件。如果您创建显式测试套件，则基于脚本的测试中会提供其他功能。这些功能包括选择测试和使用插件自定义测试运行程序。有关其他功能，请考虑使用“基于函数的单元测试”或“基于类的单元测试”。

## 测试套件创建

要从基于脚本的测试直接创建测试套件，请使用 `testsuite` 函数。要更显式地创建测试套件，请使用 `TestSuite` 的 `matlab.unittest.TestSuite.fromFile` 方法。然后，您可以使用 `run` 方法运行这些测试，而不是使用 `runtests` 函数。例如，如果您在文件 `rightTriTolTest.m` 中包含基于脚本的测试，则这三种方法等效。

```
% Implicit test suite
result = runtests('rightTriTolTest.m');

% Explicit test suite
suite = testsuite('rightTriTolTest.m');
result = run(suite);

% Explicit test suite
suite = matlab.unittest.TestSuite.fromFile('rightTriTolTest.m');
result = run(suite);
```

此外，您还可以使用 `matlab.unittest.TestSuite.fromFolder` 方法，根据指定文件夹中的所有测试文件创建测试套件。如果您知道特定测试在基于脚本的测试文件中的名称，则可以使用 `matlab.unittest.TestSuite.fromName` 根据该测试创建测试套件。

## 测试选择

对于显式测试套件，请使用选择器优化您的套件。其中有几个选择器仅适用于基于类的测试，但您可以根据测试名称为您的套件选择测试：

- 在套件生成方法（例如 `matlab.unittest.TestSuite.fromFile`）中使用 'Name' 名称-值对组参数。
- 使用 `selectors` 实例和可选的 `constraints` 实例。

在套件生成方法（如 `matlab.unittest.TestSuite.fromFile`）中使用这些方法，或使用 `selectIf` 方法创建套件并对其进行筛选。例如，在此列表中，`suite` 的四个值等效。

```
import matlab.unittest.selectors.HasName
import matlab.unittest.constraints.ContainsSubstring
import matlab.unittest.TestSuite.fromFile

f = 'rightTriTolTest.m';
selector = HasName(ContainsSubstring('Triangle'));
```

```
% fromFile, name-value pair
suite = TestSuite.fromFile(f,'Name','*Triangle*')

% fromFile, selector
suite = TestSuite.fromFile(f,selector)

% selectIf, name-value pair
fullSuite = TestSuite.fromFile(f);
suite = selectIf(fullSuite,'Name','*Triangle*')

% selectIf, selector
fullSuite = TestSuite.fromFile(f);
suite = selectIf(fullSuite,selector)
```

如果您对选择器或名称-值对组使用其中一个套件创建方法，则测试框架会创建经过筛选的套件。如果您使用 `selectIf` 方法，则测试框架会创建完整的测试套件，然后对套件进行筛选。对于大型测试套件，此方法可能会影响性能。

## 以编程方式访问测试诊断

如果您使用 `runtests` 函数或者 `TestSuite` 或 `TestCase` 的 `run` 方法运行测试，则测试框架会使用 `DiagnosticsRecordingPlugin` 插件记录测试结果的诊断信息。

运行测试后，您可以通过 `TestResult` 上 `Details` 属性中的 `DiagnosticRecord` 字段访问记录的诊断数据。例如，如果您的测试结果存储在变量 `results` 中，可通过调用 `records = result(2).Details.DiagnosticRecord` 来查找套件中第二个测试记录的诊断信息。

记录的诊断信息是 `DiagnosticRecord` 对象。要访问特定测试的特定类型测试诊断信息，请使用 `DiagnosticRecord` 类的 `selectFailed`、`selectPassed`、`selectIncomplete` 和 `selectLogged` 方法。

默认情况下，`DiagnosticsRecordingPlugin` 插件会记录在 `matlab.unittest.Verbose.Terse` 详细级别记录的验证故障和事件。有关详细信息，请参阅 `DiagnosticsRecordingPlugin` 和 `DiagnosticRecord`。

## 测试运行程序自定义

使用 `TestRunner` 对象自定义框架运行测试套件的方式。通过 `TestRunner` 对象，您可以：

- 使用 `withNoPlugins` 方法不在命令行窗口中生成任何输出。
- 使用 `runInParallel` 方法并行运行测试。
- 使用 `addPlugin` 方法向测试运行程序添加插件。

例如，使用测试套件 `suite` 创建一个静默测试运行程序，并使用 `TestRunner` 的 `run` 方法运行测试。

```
runner = matlab.unittest.TestRunner.withNoPlugins;
results = runner.run(suite);
```

使用插件可进一步自定义测试运行程序。例如，您可以重定向输出，确定代码覆盖率，或更改测试运行程序响应警告的方式。有关详细信息，请参阅“向测试运行程序添加插件”（第 34-80 页）和 `plugins` 类。

## 另请参阅

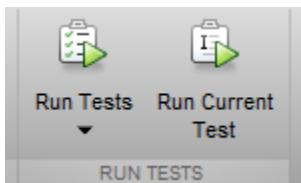
[TestRunner](#) | [TestSuite](#) | [matlab.unittest.constraints](#) | [plugins](#) | [selectors](#)

## 相关示例

- “向测试运行程序添加插件”（第 34-80 页）

## 在编辑器中运行测试

此示例说明在编辑器中工作时如何运行测试。在编辑器中打开基于函数的测试或基于类的测试时，**编辑器**选项卡下显示的是**运行测试**部分，而不是**运行**部分。此部分提供用于在当前文件中运行测试的几个选项。



**运行测试**按钮运行文件中的所有测试。**运行当前测试**按钮在当前光标位置运行测试。

在编辑器中，在名为 `sampleTest.m` 的文件中创建一个测试。保存该测试时，**编辑器**选项卡中的**运行**部分变为**运行测试**。

```
function tests = sampleTest
    tests = functiontests(localfunctions);
end

function testA(testCase)
    verifyEqual(testCase,5,5)
end

function testB(testCase)
    verifyGreaterThanOrEqual(testCase,42,13)
end

function testC(testCase)
    verifySubstring(testCase,'hello, world','llo')
end
```

点击**运行测试**图标。MATLAB 在命令行窗口中显示用于运行测试的命令，然后显示测试输出。MATLAB 运行 `sampleTest.m` 中的所有三个测试。

```
runtests('sampleTest')
```

```
Running sampleTest
```

```
...
```

```
Done sampleTest
```

---

```
ans =
```

```
1×3 TestResult array with properties:
```

```
Name
Passed
Failed
Incomplete
Duration
Details
```

```
Totals:
```

```
3 Passed, 0 Failed, 0 Incomplete.
0.0071673 seconds testing time.
```

在编辑器中，将光标放在 testB 函数中，然后点击运行当前测试图标。MATLAB 只运行 testB。

```
runtests('sampleTest','ProcedureName','testB')

Running sampleTest
.
Done sampleTest
```

---

```
ans =
```

```
TestResult with properties:
```

```
Name: 'sampleTest/testB'
Passed: 1
Failed: 0
Incomplete: 0
Duration: 9.9411e-04
Details: [1x1 struct]
```

```
Totals:
```

```
1 Passed, 0 Failed, 0 Incomplete.
0.00099411 seconds testing time.
```

除运行测试外，您还可以使用运行测试图标下的测试选项自定义测试运行。无论您是运行文件中的所有测试还是仅在光标位置运行测试，MATLAB 都会使用测试选项。

测试选项	说明
清空命令行窗口	在运行测试之前清空命令行窗口。
严格	运行测试时应用严格检查。例如，如果测试发出警告，框架会生成验证失败。 使用此选项运行的测试会将 runtests 的 'Strict' 选项设置为 true。
并行	以并行方式运行测试。此选项仅在安装了 Parallel Computing Toolbox 时才可用。 使用此选项运行的测试会将 runtests 的 'UseParallel' 选项设置为 true。
输出详细信息	控制为测试运行显示的信息量。 例如，使用“输出详细信息”指定为 0：无运行的测试会将 runtests 的 'OutputDetail' 选项设置为 0。
记录级别	显示 log 方法在指定详细级别或更低级别记录的诊断。 例如，使用“记录级别”指定为 3：详细运行的测试会将 runtests 的 'LoggingLevel' 选项设置为 3。

选择一个测试选项时，所做选择在当前 MATLAB 会话的持续时间内保持不变。

**另请参阅**

`runtests`

# 编写基于函数的单元测试

## 本节内容

- “创建测试函数”（第 34-19 页）
- “运行测试”（第 34-21 页）
- “分析结果”（第 34-21 页）

## 创建测试函数

测试函数是一个 MATLAB 文件，其中包含主函数和您的各个局部测试函数。您也可以包括文件脚手架并刷新脚手架函数。文件脚手架包括跨文件中的所有测试共享的设置和拆解函数。这些函数对每个测试文件执行一次。刷新脚手架包括在每个局部测试函数之前和之后执行的设置和拆解函数。

### 创建主函数

主函数将所有局部测试函数收集到一个测试数组中。因为它是主函数，所以函数名对应于您的 .m 文件的名称，并且遵从以 “test”（不区分大小写）开头或结尾的命名约定。在此示例中，MATLAB 文件为 `exampleTest.m`。主函数需要调用 `functiontests` 以生成测试数组 `tests`。使用 `localfunctions` 作为 `functiontests` 输入，以自动生成由文件中所有局部函数的函数句柄组成的元胞数组。这是典型的主函数。

```
function tests = exampleTest
tests = functiontests(localfunctions);
end
```

### 创建局部测试函数

各个测试函数作为局部函数包括在与主（测试生成）函数相同的 MATLAB 文件中。这些测试函数名称必须以不区分大小写的单词 “test” 开头和结尾。其中每个局部测试函数都必须接受单个输入，即函数测试用例对象 `testCase`。单元测试框架自动生成此对象。有关创建测试函数的详细信息，请参阅“使用函数编写简单测试用例”（第 34-22 页）和“验证、断言及其他验证一览表”（第 34-44 页）。这是骨架局部测试函数的典型示例。

```
function testFunctionOne(testCase)
% Test specific code
end

function FunctionTwotest(testCase)
% Test specific code
end
```

### 创建可选脚手架函数

设置和拆解代码也称为测试脚手架函数，用于设置系统的预测试状态并在运行测试后将其恢复为原始状态。这些函数有两种类型：文件脚手架函数和刷新脚手架函数，前者对每个测试文件运行一次，后者在每个局部测试函数之前和之后运行。这些函数不是生成测试必需的。通常，最好使用刷新脚手架而不是文件脚手架来增强单元测试封装。

函数测试用例对象 `testCase` 必须是文件脚手架和刷新脚手架函数的唯一输入。单元测试框架自动生成此对象。`TestCase` 对象是在设置函数、测试函数和拆解函数之间传递信息的一种方法。默认情况下，其 `TestData` 属性是一个 `struct`，其允许轻松添加字段和数据。此测试数据的典型用法包括路径和图形句柄。有关使用 `TestData` 属性的示例，请参阅“使用设置和拆解函数编写测试”（第 34-26 页）。

### 文件脚手架函数

使用文件脚手架函数可跨文件中的所有测试共享设置和拆解函数。文件脚手架函数的名称必须分别为 `setupOnce` 和 `teardownOnce`。这些函数对每个文件执行一次。您可以使用文件脚手架在测试之前设置路径，然后在测试后将其重置为原始路径。这是骨架文件脚手架设置和拆解代码的典型示例。

```
function setupOnce(testCase) % do not change function name
% set a new path, for example
end

function teardownOnce(testCase) % do not change function name
% change back to original path, for example
end
```

### 刷新脚手架函数

使用刷新脚手架函数可设置和拆解每个局部测试函数的状态。这些刷新脚手架函数的名称必须分别为 `setup` 和 `teardown`。您可以使用刷新脚手架在测试之前获取新图窗并在测试之后关闭该图窗。这是骨架测试函数级别设置和拆解代码的典型示例。

```
function setup(testCase) % do not change function name
% open a figure, for example
end

function teardown(testCase) % do not change function name
% close figure, for example
end
```

### 程序列出模板

```
%% Main function to generate tests
function tests = exampleTest
tests = functiontests(localfunctions);
end

%% Test Functions
function testFunctionOne(testCase)
% Test specific code
end

function FunctionTwotest(testCase)
% Test specific code
end

%% Optional file fixtures
function setupOnce(testCase) % do not change function name
% set a new path, for example
end

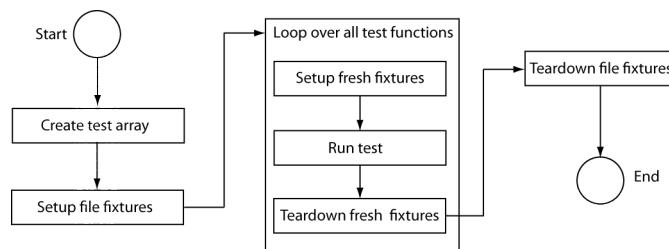
function teardownOnce(testCase) % do not change function name
% change back to original path, for example
end

%% Optional fresh fixtures
function setup(testCase) % do not change function name
% open a figure, for example
end
```

```
function teardown(testCase) % do not change function name
% close figure, for example
end
```

## 运行测试

下图详述了在您运行测试时执行的任务。



要在命令提示符下运行测试，请使用 `runtests` 命令并将 MATLAB 测试文件作为输入。例如：

```
results = runtests('exampleTest.m')
```

您也可以使用 `run` 函数运行测试。

```
results = run(exampleTest)
```

有关运行测试的详细信息，请参阅 `runtests` 参考页和“为各个工作流运行测试”（第 34-76 页）。

## 分析结果

要分析测试结果，请检查 `runtests` 或 `run` 的输出结构体。对于每个测试，结果中包含测试函数的名称（无论测试结果是通过、失败还是未完成）和运行测试所用的时间。有关详细信息，请参阅“分析测试用例结果”（第 34-105 页）和“分析失败的测试结果”（第 34-108 页）。

## 另请参阅

`functiontests` | `localfunctions` | `runtests`

## 相关示例

- “使用函数编写简单测试用例”（第 34-22 页）
- “使用设置和拆解函数编写测试”（第 34-26 页）

## 使用函数编写简单测试用例

此示例演示如何编写 MATLAB 函数 `quadraticSolver.m` 的单元测试。

### 创建 `quadraticSolver.m` 函数

此 MATLAB 函数求二次方程的解。在您的 MATLAB 路径上的文件夹中创建此函数。

```
function roots = quadraticSolver(a, b, c)
% quadraticSolver returns solutions to the
% quadratic equation a*x^2 + b*x + c = 0.

if ~isa(a,'numeric') || ~isa(b,'numeric') || ~isa(c,'numeric')
    error('quadraticSolver:InputMustBeNumeric',...
        'Coefficients must be numeric.');
end

roots(1) = (-b + sqrt(b^2 - 4*a*c)) / (2*a);
roots(2) = (-b - sqrt(b^2 - 4*a*c)) / (2*a);

end
```

### 创建 `solverTest` 函数

在您的 MATLAB 路径上的文件夹中创建此函数。

```
function tests = solverTest
tests = functiontests(localfunctions);
end
```

使用 `localfunctions` 作为输入调用 `functiontests` 可从 `solverTest.m` 文件中的每个局部函数创建一个测试数组。每个测试都是一个局部函数，它遵从在函数名称开头或末尾包含“test”的命名约定。不遵从此约定的局部函数不包括在测试数组中。测试函数必须接受测试框架向其传递函数测试用例对象的单个输入参数。该函数对验证、断言、假设和致命断言使用此对象。它包含一个允许在设置、测试和拆解函数之间传递数据的 `TestData` 结构体。

### 为实数解创建测试函数

创建一个测试函数 `testRealSolution` 来验证 `quadraticSolver` 是否返回实数解的正确值。例如，方程  $x^2 - 3x + 2 = 0$  有实数解  $x = 1$  和  $x = 2$ 。此函数使用此方程的输入调用 `quadraticSolver`。预期的解 `expSolution` 为 `[2,1]`。

使用验证函数 `verifyEqual` 将函数的输出 `actSolution` 与所需的输出 `expSolution` 进行比较。如果验证失败，该框架继续执行测试。通常，在对浮点值使用 `verifyEqual` 时，可指定用于比较的容差。有关详细信息，请参阅 `matlab.unittest.constraints`。

将此函数添加到 `solverTest.m` 文件中。

```
function testRealSolution(testCase)
actSolution = quadraticSolver(1,-3,2);
expSolution = [2 1];
verifyEqual(testCase,actSolution,expSolution)
end
```

## 为虚数解创建测试函数

创建一个测试以验证 `quadraticSolver` 是否返回虚数解的正确值。例如，方程  $x^2 + 2x + 10 = 0$  有虚数解  $x = -1 + 3i$  和  $x = -1 - 3i$ 。通常，在对浮点值使用 `verifyEqual` 时，可指定用于比较的容差。有关详细信息，请参阅 `matlab.unittest.constraints`。

将此函数 `testImaginarySolution` 添加到 `solverTest.m` 文件中。

```
function testImaginarySolution(testCase)
actSolution = quadraticSolver(1,2,10);
expSolution = [-1+3i -1-3i];
verifyEqual(testCase,actSolution,expSolution)
end
```

`solverTest.m` 文件中测试的顺序无关紧要，因为它们是完全独立的测试用例。

## 保存 `solverTest` 函数

下面是完整的 `solverTest.m` 测试文件。将此文件保存在您的 MATLAB 路径上的一个文件夹中。

```
function tests = solverTest
tests = functiontests(localfunctions);
end

function testRealSolution(testCase)
actSolution = quadraticSolver(1,3,2);
expSolution = [2 1];
verifyEqual(testCase,actSolution,expSolution)
end

function testImaginarySolution(testCase)
actSolution = quadraticSolver(1,2,10);
expSolution = [-1+3i -1-3i];
verifyEqual(testCase,actSolution,expSolution)
end
```

## 在 `solverTest` 函数中运行测试

运行测试。

```
results = runtests('solverTest.m')

Running solverTest
..
Done solverTest



---


results =
1x2 TestResult array with properties:

  Name
  Passed
  Failed
  Incomplete
  Duration
```

```
Totals:
  2 Passed, 0 Failed, 0 Incomplete.
  0.19172 seconds testing time.
```

两个测试均通过。

### 在 quadraticSolver.m 中引入错误并运行测试

通过将 `quadraticSolver.m` 中的 `roots` 强制设置为实数来使其中一个测试失败。在结束函数之前，添加以下行：`roots = real(roots);`。（请不要更改 `solverTest.m`。）保存文件并运行测试。

```
results = runtests('solverTest.m')
```

```
Running solverTest
```

```
=====
Verification failed in solverTest/testImaginarySolution.
```

```
-----
Framework Diagnostic:
```

```
verifyEqual failed.
-> Complexity does not match.
```

```
Actual Complexity:
```

```
Real
```

```
Expected Complexity:
```

```
Complex
```

```
Actual Value:
```

```
-1 -1
```

```
Expected Value:
```

```
-1.00000000000000 + 3.00000000000000i -1.00000000000000 - 3.00000000000000i
```

```
-----
Stack Information:
```

```
In C:\work\solverTest.m (testImaginarySolution) at 14
```

```
Done solverTest
```

```
Failure Summary:
```

Name	Failed	Incomplete	Reason(s)
solverTest/testImaginarySolution	X		Failed by verification.

```
results =
```

```
1x2 TestResult array with properties:
```

```
Name
Passed
Failed
Incomplete
Duration
```

```
Totals:
```

```
  1 Passed, 1 Failed, 0 Incomplete.
  0.043751 seconds testing time.
```

虚数测试验证失败。

将 `quadraticSolver.m` 恢复为其以前的状态，通过删除 `roots = real(roots);` 代码更正版本。

## 另请参阅

`matlab.unittest.constraints`

## 详细信息

- “编写基于函数的单元测试”（第 34-19 页）
- “验证、断言及其他验证一览表”（第 34-44 页）

## 使用设置和拆解函数编写测试

以下示例演示如何使用刷新脚手架和文件脚手架为一组 MATLAB® 图窗坐标区属性编写单元测试。

### 创建 axesPropertiesTest 文件

创建一个包含用于测试图窗坐标区属性的主函数的文件，并包括两个测试函数。一个函数用于验证 x 坐标轴范围是否正确，另一个用于验证曲面的面颜色是否正确。

在您的 MATLAB 路径上的一个文件夹中，创建 `axesPropertiesTest.m`。在此文件的主函数中，通过调用 `localfunctions` 函数让 `functiontests` 根据 `axesPropertiesTest.m` 中的每个局部函数创建一个测试数组。

```
% Copyright 2015 The MathWorks, Inc.
```

```
function tests = axesPropertiesTest
tests = functiontests(localfunctions);
end
```

### 创建文件脚手架函数

文件脚手架函数是在您的测试文件中运行一次的设置和拆解代码。这些脚手架跨测试文件共享。在此示例中，文件脚手架函数创建一个临时文件夹并将其设置为当前工作文件夹。它们还创建并保存一个新图窗以进行测试。在测试完成后，该框架恢复原始工作文件夹并删除临时文件夹和保存的图窗。

在此示例中，辅助函数创建一个简单图窗 - 一个红色圆柱。在更实际的方案中，此代码是受测产品的一部分，计算成本很高，因而适宜仅创建图窗一次，并为每个测试函数加载结果的独立副本。但在此示例中，您希望作为 `axesPropertiesTest` 的局部函数创建此辅助函数。请注意，测试数组不包括该函数，这是因为其名称不以 ‘test’ 开头或结尾。

编写一个辅助函数，以创建一个简单的红色圆柱并将其添加为 `axesPropertiesTest` 的局部函数。

```
% Copyright 2015 The MathWorks, Inc.
```

```
function f = createFigure
f = figure;
ax = axes('Parent', f);
cylinder(ax, 10)
h = findobj(ax, 'Type', 'surface');
h.FaceColor = [1 0 0];
end
```

您必须分别命名文件测试脚手架 `setupOnce` 和 `teardownOnce` 的设置和拆解函数。这些函数接受测试框架自动向其传递函数测试用例对象的单个输入参数 `testCase`。此测试用例对象包含一个用于在设置、测试和拆解函数之间传递数据的 `TestData` 结构体。在此示例中，`TestData` 结构体使用分配的字段存储原始路径、临时文件夹名称和图窗文件名。

将设置和拆解函数创建为 `axesPropertiesTest` 的局部函数。

```
% Copyright 2015 The MathWorks, Inc.

function setupOnce(testCase)
% create and change to temporary folder
testCase.TestData.origPath = pwd;
testCase.TestData.tmpFolder = [tmpFolder' datestr(now,30)];
mkdir(testCase.TestData.tmpFolder)
cd(testCase.TestData.tmpFolder)

% create and save a figure
testCase.TestData.figName = 'tmpFig.fig';
aFig = createFigure;
saveas(aFig,testCase.TestData.figName,'fig')
close(aFig)
end

function teardownOnce(testCase)
delete(testCase.TestData.figName)
cd(testCase.TestData.origPath)
rmdir(testCase.TestData.tmpFolder)
end
```

### 创建刷新脚手架函数

刷新脚手架是在文件中的每个测试函数之前和之后运行的函数级别设置和拆解代码。在此示例中，这些函数打开保存的图窗并查找句柄。测试后，该框架关闭图窗。

您必须分别命名刷新脚手架函数 `setup` 和 `teardown`。与文件脚手架函数类似，这些函数采用单个输入参数 `testCase`。在此示例中，这些函数在包括图窗句柄和坐标区句柄的 `TestData` 结构体中创建一个新字段。这样可以在设置、测试和拆解函数之间传递信息。

将设置和拆解函数创建为 `axesPropertiesTest` 的局部函数。打开为每个测试保存的图窗以确保测试独立性。

```
% Copyright 2015 The MathWorks, Inc.

function setup(testCase)
testCase.TestData.Figure = openfig(testCase.TestData.figName);
testCase.TestData.Axes = findobj(testCase.TestData.Figure, ...
    'Type','Axes');
end

function teardown(testCase)
close(testCase.TestData.Figure)
end
```

除了自定义的设置和拆解代码外，单元测试框架还提供某些类用于创建脚手架。有关详细信息，请参阅 `matlab.unittest.fixtures`。

### 创建测试函数

每个测试都是一个局部函数，它遵从在函数名称开头或末尾包含 ‘`test`’ 的命名约定。测试数组不包括不遵从此约定的局部函数。与设置和拆解函数类似，各个测试函数必须接受单个输入参数 `testCase`。将此测试用例对象用于验证、断言、假设和致命断言函数。

`testDefaultXLim` 函数测试验证  $x$  坐标轴范围是否足够大，以便显示圆柱。下限需要小于 -10，上限需要大于 10。这些值来自在辅助函数中生成的图窗 - 一个以原点为中心、单位半径为 10 的圆柱。此测试函数打开在 `setupOnce` 函数中创建并保存的图窗，查询坐标区范围并验证这些范围是否正确。验证函数 `verifyLessThanOrEqual` 和 `verifyGreaterThanOrEqual` 采用在失败时显示的测试用例、实际值、预期值和可选诊断信息作为输入。

创建 `testDefaultXLim` 函数作为 `axesPropertiesTest` 的局部函数。

```
% Copyright 2015 The MathWorks, Inc.
```

```
function testDefaultXLim(testCase)
    xlim = testCase.TestData.Axes.XLim;
    verifyLessThanOrEqual(testCase, xlim(1), -10, ...
        'Minimum x-limit was not small enough');
    verifyGreaterThanOrEqual(testCase, xlim(2), 10, ...
        'Maximum x-limit was not big enough');
end
```

`surfaceColorTest` 函数访问您在 `setupOnce` 函数中创建并保存的图窗。`surfaceColorTest` 查询圆柱的面颜色并验证其是否为红色。红色的 RGB 值为 [1 0 0]。验证函数 `verifyEqual` 采用在失败时显示的测试用例、实际值、预期值和可选诊断信息作为输入。通常，在对浮点值使用 `verifyEqual` 时，可指定用于比较的容差。有关详细信息，请参阅 `matlab.unittest.constraints`。

创建 `surfaceColorTest` 函数作为 `axesPropertiesTest` 的局部函数。

```
% Copyright 2015 The MathWorks, Inc.
```

```
function surfaceColorTest(testCase)
    h = findobj(testCase.TestData.Axes, 'Type', 'surface');
    co = h.FaceColor;
    verifyEqual(testCase, co, [1 0 0], 'FaceColor is incorrect');
end
```

现在，`axesPropertiesTest.m` 文件是完整的，包含主函数、文件脚手架函数、刷新脚手架函数和两个局部测试函数。此时，您可以运行测试。

## 运行测试

下一步是使用 `runtests` 函数运行测试。在此示例中，调用 `runtests` 将执行以下各步骤：

- 1 主函数创建一个测试数组。
- 2 文件脚手架记录工作文件夹，创建一个临时文件夹，将临时文件夹设置为工作文件夹，然后生成并保存图窗。
- 3 刷新脚手架设置将打开已保存的图窗并查找句柄。
- 4 `testDefaultXLim` 测试便会运行。
- 5 刷新脚手架拆解将关闭图窗。
- 6 刷新脚手架设置将打开已保存的图窗并查找句柄。
- 7 `surfaceColorTest` 测试便会运行。

- 8 刷新脚手架拆解将关闭图窗。
- 9 拆解文件脚手架将删除已保存的图窗，重新更改为原始路径并删除临时文件夹。

在命令提示符下，生成并运行测试套件。

```
results = runtests('axesPropertiesTest.m')
```

```
Running axesPropertiesTest
```

```
.. Done axesPropertiesTest
```

```
results =
```

```
1x2 TestResult array with properties:
```

```
Name
Passed
Failed
Incomplete
Duration
Details
```

**Totals:**

```
2 Passed, 0 Failed, 0 Incomplete.
3.3079 seconds testing time.
```

## 创建测试结果表格

要访问表格可用的功能，先从 `TestResult` 对象创建一个表格。

```
rt = table(results)
```

```
rt =
```

```
2x6 table
```

Name	Passed	Failed	Incomplete	Duration	Details
{'axesPropertiesTest/testDefaultXLim'}	true	false	false	2.5109	{1x1 struct}
{'axesPropertiesTest/surfaceColorTest'}	true	false	false	0.797	{1x1 struct}

将测试结果导出为 Excel® 电子表格。

```
writetable(rt,'myTestResults.xls')
```

按持续时间升序排列测试结果。

```
sortrows(rt,'Duration')
```

```
ans =
```

2x6 table

Name	Passed	Failed	Incomplete	Duration	Details
{'axesPropertiesTest/surfaceColorTest'}	true	false	false	0.797	{1x1 struct}
{'axesPropertiesTest/testDefaultXLim'}	true	false	false	2.5109	{1x1 struct}

**另请参阅****matlab.unittest.constraints | matlab.unittest.fixtures****详细信息**

- “编写基于函数的单元测试” (第 34-19 页)
- “验证、断言及其他验证一览表” (第 34-44 页)

# 扩展基于函数的测试

## 本节内容

- “设置和拆解代码的脚手架” (第 34-31 页)
- “测试日志记录和详细程度” (第 34-32 页)
- “测试套件创建” (第 34-32 页)
- “测试选择” (第 34-32 页)
- “测试运行” (第 34-33 页)
- “以编程方式访问测试诊断” (第 34-33 页)
- “测试运行程序自定义” (第 34-33 页)

通常，使用基于函数的测试，您可以创建测试文件并将文件名传递到 `runtests` 函数，而无需显式创建 `Test` 对象套件。不过，如果您创建显式测试套件，则基于函数的测试中会提供其他功能。这些功能包括：

- 测试日志记录和详细程度
- 测试选择
- 用于自定义测试运行程序的插件

有关其他功能，请考虑使用“基于类的单元测试”。

## 设置和拆解代码的脚手架

编写测试时，请使用 `applyFixture` 方法处理设置和拆解代码，以用于以下操作：

- 更改当前工作文件夹
- 向路径中添加文件夹
- 创建临时文件夹
- 取消警告的显示

使用这些脚手架，您就不必在基于函数的测试的 `setupOnce`、`teardownOnce`、`setup` 和 `teardown` 函数中对操作进行手动编码。

例如，如果您手动编写设置和拆解代码，为每个测试设置一个临时文件夹，然后将该文件夹作为当前工作文件夹，则您的 `setup` 和 `teardown` 函数可能类似于如下所示。

```
function setup(testCase)
% store current folder
testCase.TestData.origPath = pwd;

% create temporary folder
testCase.TestData.tmpFolder = ['tmpFolder' datestr(now,30)];
mkdir(testCase.TestData.tmpFolder)

% change to temporary folder
cd(testCase.TestData.tmpFolder)
end

function teardown(testCase)
% change to original folder
cd(testCase.TestData.origPath)
```

```
% delete temporary folder
rmdir(testCase.TestData.tmpFolder)
end
```

然而，您还可以使用脚手架将这两个函数替换为一个经过修改的 `setup` 函数。该脚手架会存储必要信息以还原初始状态并执行拆解操作。

```
function setup(testCase)
% create temporary folder
f = testCase.applyFixture(matlab.unittest.fixtures.TemporaryFolderFixture);

% change to temporary folder
testCase.applyFixture(matlab.unittest.fixtures.CurrentFolderFixture(f.Folder));
end
```

## 测试日志记录和详细程度

您的测试函数可以使用 `log` 方法。默认情况下，测试运行程序会报告使用详细级别 1 (Terse) 记录的统计信息。使用 `matlab.unittest.plugins.LoggingPlugin.withVerbosity` 方法对其他详细级别的消息做出响应。构造一个 `TestRunner` 对象，添加 `LoggingPlugin`，并使用 `run` 方法运行该套件。有关创建测试运行程序的详细信息，请参阅“测试运行程序自定义”（第 34-33 页）。

## 测试套件创建

调用基于函数的测试会返回一套 `Test` 对象。您也可以使用 `testsuite` 函数或 `matlab.unittest.TestSuite.fromFile` 方法。如果您需要特定测试并且知道测试名称，则可以使用 `matlab.unittest.TestSuite.fromName`。如果您要根据特定文件夹中的所有测试来创建套件，则可以使用 `matlab.unittest.TestSuite.fromFolder`。

## 测试选择

对于显式测试套件，请使用选择器优化您的套件。其中有几个选择器仅适用于基于类的测试，但您可以根据测试名称为您的套件选择测试：

- 在套件生成方法（例如 `matlab.unittest.TestSuite.fromFile`）中使用 'Name' 名称-值对组参数。
- 使用 `selectors` 实例和可选的 `constraints` 实例。

在套件生成方法（如 `matlab.unittest.TestSuite.fromFile`）中使用这些方法，或使用 `selectIf` 方法创建套件并对其进行筛选。例如，在此列表中，`suite` 的四个值等效。

```
import matlab.unittest.selectors.HasName
import matlab.unittest.constraints.ContainsSubstring
import matlab.unittest.TestSuite.fromFile

f = 'rightTriTolTest.m';
selector = HasName(ContainsSubstring('Triangle'));

% fromFile, name-value pair
suite = TestSuite.fromFile(f,'Name','*Triangle*')

% fromFile, selector
suite = TestSuite.fromFile(f,selector)

% selectIf, name-value pair
fullSuite = TestSuite.fromFile(f);
```

```

suite = selectIf(fullSuite,'Name','*Triangle*')

% selectIf, selector
fullSuite = TestSuite.fromFile(f);
suite = selectIf(fullSuite,selector)

```

如果您对选择器或名称-值对组使用其中一个套件创建方法，则测试框架会创建经过筛选的套件。如果您使用 `selectIf` 方法，则测试框架会创建完整的测试套件，然后对套件进行筛选。对于大型测试套件，此方法可能会影响性能。

## 测试运行

可以通过多种方式运行基于函数的测试。

运行所有测试	使用函数
文件中	将 <code>runtests</code> 替换为测试文件的名称
在套件中	具有套件的 <code>run</code>
在包含自定义测试运行程序的套件中	<code>run.</code> (请参阅“测试运行程序自定义”(第 34-33 页)。)

有关详细信息，请参阅“为各个工作流运行测试”(第 34-76 页)。

## 以编程方式访问测试诊断

如果您使用 `runtests` 函数或者 `TestSuite` 或 `TestCase` 的 `run` 方法运行测试，则测试框架会使用 `DiagnosticsRecordingPlugin` 插件记录测试结果的诊断信息。

运行测试后，您可以通过 `TestResult` 上 `Details` 属性中的 `DiagnosticRecord` 字段访问记录的诊断数据。例如，如果您的测试结果存储在变量 `results` 中，可通过调用 `records = result(2).Details.DiagnosticRecord` 来查找套件中第二个测试记录的诊断信息。

记录的诊断信息是 `DiagnosticRecord` 对象。要访问特定测试的特定类型测试诊断信息，请使用 `DiagnosticRecord` 类的 `selectFailed`、`selectPassed`、`selectIncomplete` 和 `selectLogged` 方法。

默认情况下，`DiagnosticsRecordingPlugin` 插件会记录在 `matlab.unittest.Verbose.Terse` 详细级别记录的验证故障和事件。有关详细信息，请参阅 `DiagnosticsRecordingPlugin` 和 `DiagnosticRecord`。

## 测试运行程序自定义

使用 `TestRunner` 对象自定义框架运行测试套件的方式。通过 `TestRunner` 对象，您可以：

- 使用 `withNoPlugins` 方法不在命令行窗口中生成任何输出。
- 使用 `runInParallel` 方法并行运行测试。
- 使用 `addPlugin` 方法向测试运行程序添加插件。

例如，使用测试套件 `suite` 创建一个静默测试运行程序，并使用 `TestRunner` 的 `run` 方法运行测试。

```

runner = matlab.unittest.TestRunner.withNoPlugins;
results = runner.run(suite);

```

使用插件可进一步自定义测试运行程序。例如，您可以重定向输出，确定代码覆盖率，或更改测试运行程序响应警告的方式。有关详细信息，请参阅“向测试运行程序添加插件”（第 34-80 页）和 **plugins** 类。

### 另请参阅

[matlab.unittest.TestCase](#) | [matlab.unittest.TestSuite](#) | [matlab.unittest.constraints](#) |  
[matlab.unittest.diagnostics](#) | [matlab.unittest.qualifications](#) | [matlab.unittest.selectors](#)

### 相关示例

- “为各个工作流运行测试”（第 34-76 页）
- “向测试运行程序添加插件”（第 34-80 页）

# 在 MATLAB 中编写基于类的单元测试

要测试 MATLAB 程序，可使用验证机制来编写单元测试，这些验证是用于测试值和响应失败的方法。

## 测试类定义

测试类必须从 `matlab.unittest.TestCase` 继承并包含具有 `Test` 属性的 `methods` 代码块。`methods` 块包含函数，其中每个函数是一个单元测试。以下是一个常规的基本类定义。

```
%% Test Class Definition
classdef MyComponentTest < matlab.unittest.TestCase

%% Test Method Block
methods (Test)
    % includes unit test functions
end
end
```

## 单元测试

单元测试是确定软件单元正确性的方法。每个单元测试都包含在方法块内。函数必须接受 `TestCase` 实例作为输入。

```
%% Test Class Definition
classdef MyComponentTest < matlab.unittest.TestCase

%% Test Method Block
methods (Test)

    %% Test Function
    function testASolution(testCase)
        %% Exercise function under test
        % act = the value from the function under test

        %% Verify using test qualification
        % exp = your expected value
        % testCase.<qualification method>(act,exp);
    end
end
end
```

验证是用于测试值并响应失败的方法。下表列出了验证的类型。

验证	使用此验证在不引发异常的条件下生成和记录失败。将继续运行完成其余测试。	<code>matlab.unittest.qualifications.Verifiable</code>
假设	使用此验证机制确保仅当满足特定先决条件时才运行测试。但其实在不满足此先决条件的情况下执行测试也不会导致测试失败。发生假设失败时，测试框架将测试标记为已滤除。	<code>matlab.unittest.qualifications.Assumable</code>

断言	使用此验证机制确保满足当前测试的先决条件。	<code>matlab.unittest.qualifications.Assertable</code>
致命断言	当断言点处的失败使得当前测试方法的其余部分无效或者状态不可恢复时，使用此验证机制。	<code>matlab.unittest.qualifications.FatalAssertable</code>

MATLAB 单元测试框架为每类验证机制提供了约 25 种验证方法。例如，使用 `verifyClass` 或 `assertClass` 测试某值是否属于预期的类，可以使用 `assumeTrue` 或 `fatalAssertTrue` 测试实际值是否为 `true`。有关验证方法的汇总，请参阅“验证、断言及其他验证一览表”（第 34-44 页）。

通常，每个单元测试函数都会通过执行您要测试的代码获取一个实际值，并定义与之关联的预期值。例如，如果您要测试 `plus` 函数，实际值可能是 `plus(2,3)`，预期值是 5。在测试函数内，您将实际值和预期值传递给验证方法。例如：

```
testCase.verifyEqual(plus(2,3),5)
```

有关基本单元测试的示例，请参阅“使用类来编写简单测试用例”（第 34-37 页）。

## 高级测试类的其他功能

MATLAB 单元测试框架提供了用于编写更多高级测试类的一些功能：

- 设置和拆解方法块，用于隐式设置系统的预测试状态并在运行测试后将其恢复为原始状态。有关设置和拆解代码的测试类示例，请参阅“使用类来编写设置代码和拆解代码”（第 34-41 页）。
- 高级验证功能，包括实际值代理、测试诊断和约束接口。有关详细信息，请参阅 `matlab.unittest.constraints` 和 `matlab.unittest.diagnostics`。
- 参数化测试，用于在指定的参数列表上合并和执行测试。有关详细信息，请参阅“创建基本参数化测试”（第 34-60 页）和“创建高级参数化测试”（第 34-64 页）。
- 即用型脚手架，用于处理常用测试动作的安装和拆解以及在类之间共享脚手架。有关详细信息，请参阅 `matlab.unittest.fixtures` 和“使用共享脚手架编写测试”（第 34-50 页）。
- 创建自定义测试脚手架的能力。有关详细信息，请参阅“创建基本自定义脚手架”（第 34-53 页）和“创建高级自定义脚手架”（第 34-55 页）。

## 另请参阅

### 相关示例

- “使用类来编写简单测试用例”（第 34-37 页）
- “使用类来编写设置代码和拆解代码”（第 34-41 页）
- “创建简单测试套件”（第 34-74 页）
- “为各个工作流运行测试”（第 34-76 页）
- “分析测试用例结果”（第 34-105 页）
- “分析失败的测试结果”（第 34-108 页）

# 使用类来编写简单测试用例

此示例说明如何编写 MATLAB® 函数 `quadraticSolver.m` 的单元测试。

## 创建 `quadraticSolver.m` 函数

以下 MATLAB 函数求二次方程的解。在您的 MATLAB 路径上的文件夹中创建此函数。

```
function roots = quadraticSolver(a, b, c)
% quadraticSolver returns solutions to the
% quadratic equation a*x^2 + b*x + c = 0.

if ~isa(a,'numeric') || ~isa(b,'numeric') || ~isa(c,'numeric')
    error('quadraticSolver:InputMustBeNumeric',...
        'Coefficients must be numeric.');
end

roots(1) = (-b + sqrt(b^2 - 4*a*c)) / (2*a);
roots(2) = (-b - sqrt(b^2 - 4*a*c)) / (2*a);

end
```

## 创建 `SolverTest` 类定义

要使用 `matlab.unittest` 框架，请以测试用例（从 `matlab.unittest.TestCase` 派生的一个类）的形式编写 MATLAB 函数（测试）。

创建一个子类 `SolverTest`。

```
% Copyright 2015 The MathWorks, Inc.

classdef SolverTest < matlab.unittest.TestCase

    methods (Test)

        end

    end
```

以下步骤显示如何创建特定测试。使用 `(Test)` 属性将这些测试放在 `methods` 块内。

### 创建实数解的测试方法

创建一个测试方法 `testRealSolution` 以验证 `quadraticSolver` 是否返回实数解的正确值。例如，方程  $x^2 - 3x + 2 = 0$  有实数解  $x = 1$  和  $x = 2$ 。此方法使用此方程的输入调用 `quadraticSolver`。解 `expSolution` 是 `[2,1]`。

使用 `matlab.unittest.TestCase` 方法 `verifyEqual` 将函数的输出 `actSolution` 与所需的输出 `expSolution` 进行比较。如果验证失败，则继续执行测试。

```
% Copyright 2015 The MathWorks, Inc.

function testRealSolution(testCase)
    actSolution = quadraticSolver(1,-3,2);
    expSolution = [2,1];
    testCase.verifyEqual(actSolution,expSolution)
end
```

将此函数添加到 **methods (Test)** 块内。

### 创建虚数解的测试方法

创建一个测试以验证 **quadraticSolver** 是否返回虚数解的正确值。例如，方程  $x^2 - 2x + 10 = 0$  有虚数解  $x = -1 + 3i$  和  $x = -1 - 3i$ 。将此函数 **testImaginarySolution** 添加到 **methods (Test)** 块内。

```
% Copyright 2015 The MathWorks, Inc.

function testImaginarySolution(testCase)
    actSolution = quadraticSolver(1,2,10);
    expSolution = [-1+3i, -1-3i];
    testCase.verifyEqual(actSolution,expSolution)
end
```

测试在该块中的顺序无关紧要。

### 保存类定义

以下是完整的 **SolverTest** 类定义。将此文件保存在您的 MATLAB 路径上的一个文件夹中。

```
classdef SolverTest < matlab.unittest.TestCase
    % SolverTest tests solutions to the quadratic equation
    % a*x^2 + b*x + c = 0

    methods (Test)
        function testRealSolution(testCase)
            actSolution = quadraticSolver(1,-3,2);
            expSolution = [2,1];
            testCase.verifyEqual(actSolution,expSolution);
        end
        function testImaginarySolution(testCase)
            actSolution = quadraticSolver(1,2,10);
            expSolution = [-1+3i, -1-3i];
            testCase.verifyEqual(actSolution,expSolution);
        end
    end
end
```

### 运行 SolverTest 测试用例中的测试

运行 **SolverTest** 类定义文件中的所有测试。

```
testCase = SolverTest;
res = run(testCase)

Running SolverTest
..
Done SolverTest

_____

res =
1x2 TestResult array with properties:

Name
Passed
Failed
Incomplete
Duration
Details

Totals:
2 Passed, 0 Failed, 0 Incomplete.
0.55494 seconds testing time.
```

## 运行单个测试方法

运行单个测试 `testRealSolution`:

```
testCase = SolverTest;
res = run(testCase,'testRealSolution')

Running SolverTest
.
Done SolverTest

_____

res =
TestResult with properties:

Name: 'SolverTest/testRealSolution'
Passed: 1
Failed: 0
Incomplete: 0
Duration: 0.0120
Details: [1x1 struct]

Totals:
1 Passed, 0 Failed, 0 Incomplete.
```

0.011966 seconds testing time.

## 另请参阅

### 相关示例

- “在 MATLAB 中编写基于类的单元测试” (第 34-35 页)
- “使用类来编写设置代码和拆解代码” (第 34-41 页)
- “分析测试用例结果” (第 34-105 页)
- “创建简单测试套件” (第 34-74 页)

# 使用类来编写设置代码和拆解代码

## 本节内容

- “测试脚手架”（第 34-41 页）
- “包含方法级设置代码的测试用例”（第 34-41 页）
- “包含类级设置代码的测试用例”（第 34-42 页）

## 测试脚手架

测试脚手架是设置和拆解代码，用于设置系统的预测试状态并在运行测试后将其恢复为原始状态。设置和拆解方法在 **TestCase** 类中由以下方法属性定义：

- **TestMethodSetup** 和 **TestMethodTeardown** 方法在每个测试方法之前和之后运行。
- **TestClassSetup** 和 **TestClassTeardown** 方法在测试用例中的所有测试方法之前和之后运行。

该测试框架保证在子类中执行 **TestMethodSetup** 和 **TestClassSetup** 方法之前先执行超类的对应方法。

测试作者最好使用 **addTeardown** 方法从 **TestMethodSetup** 和 **TestClassSetup** 块中执行所有拆解活动，而不是在 **TestMethodTeardown** 和 **TestClassTeardown** 块中实现相应的拆解方法。这样可保证拆解按与设置相反的顺序执行，还可确保测试内容满足异常安全条件。

## 包含方法级设置代码的测试用例

以下测试用例 **FigurePropertiesTest** 包含方法级设置代码。**TestMethodSetup** 方法在运行每个测试之前创建一个图窗，**TestMethodTeardown** 在运行测试之后关闭图窗。如前所述，您应该尽量使用 **addTeardown** 方法来定义拆解活动。但为了直观易懂，以下示例显示了 **TestMethodTeardown** 块的实现。

```
classdef FigurePropertiesTest < matlab.unittest.TestCase

    properties
        TestFigure
    end

    methods(TestMethodSetup)
        function createFigure(testCase)
            % comment
            testCase.TestFigure = figure;
        end
    end

    methods(TestMethodTeardown)
        function closeFigure(testCase)
            close(testCase.TestFigure)
        end
    end

    methods(Test)
        function defaultCurrentPoint(testCase)
```

```

cp = testCase.TestFigure.CurrentPoint;
testCase.verifyEqual(cp, [0 0], ...
    'Default current point is incorrect')
end

function defaultCurrentObject(testCase)
    import matlab.unittest.constraints.IsEmpty

    co = testCase.TestFigure.CurrentObject;
    testCase.verifyThat(co, IsEmpty, ...
        'Default current object should be empty')
end
end

```

## 包含类级设置代码的测试用例

以下测试用例 BankAccountTest 包含类级设置代码。

要设置 BankAccountTest (用于测试“开发类 - 典型工作流”中所述的 BankAccount 类示例)，请添加一个 TestClassSetup 方法 addBankAccountClassToPath。此方法添加指向 BankAccount 示例文件的路径。通常，您使用 PathFixture 设置路径。为了直观易懂，以下示例采用手动方式执行设置和拆解活动。

```

classdef BankAccountTest < matlab.unittest.TestCase
    % Tests the BankAccount class.

    methods (TestClassSetup)
        function addBankAccountClassToPath(testCase)
            p = path;
            testCase.addTeardown(@path,p);
            addpath(fullfile(matlabroot,'help','techdoc','matlab_oop',...
                'examples')));
        end
    end

    methods (Test)
        function testConstructor(testCase)
            b = BankAccount(1234, 100);
            testCase.verifyEqual(b.AccountNumber, 1234, ...
                'Constructor failed to correctly set account number');
            testCase.verifyEqual(b.AccountBalance, 100, ...
                'Constructor failed to correctly set account balance');
        end

        function testConstructorNotEnoughInputs(testCase)
            import matlab.unittest.constraints.Throws;
            testCase.verifyThat(@()BankAccount, ...
                Throws('MATLAB:minrhs'));
        end

        function testDeposit(testCase)
            b = BankAccount(1234, 100);
            b.deposit(25);
            testCase.verifyEqual(b.AccountBalance, 125);
        end
    end

```

```
end

function testWithdraw(testCase)
    b = BankAccount(1234, 100);
    b.withdraw(25);
    testCase.verifyEqual(b.AccountBalance, 75);
end

function testNotifyInsufficientFunds(testCase)
    callbackExecuted = false;
    function testCallback(~,~)
        callbackExecuted = true;
    end

    b = BankAccount(1234, 100);
    b.addListener('InsufficientFunds', @testCallback);

    b.withdraw(50);
    testCase.assertFalse(callbackExecuted, ...
        'The callback should not have executed yet');
    b.withdraw(60);
    testCase.assertTrue(callbackExecuted, ...
        'The listener callback should have fired');
end
end
end
```

## 另请参阅

[addTeardown | matlab.unittest.TestCase](#)

## 相关示例

- “在 MATLAB 中编写基于类的单元测试”（第 34-35 页）
- “使用类来编写简单测试用例”（第 34-37 页）

## 验证、断言及其他验证一览表

测试值和响应失败有四种类型的验证：验证、假设、断言和致命断言。

- 验证 - 产生并记录失败而不引发异常，这意味着其余测试会一直运行直至完成。
- 假设 - 确保仅在满足某些先决条件时且此类事件不会生成测试失败时才运行测试。发生假设失败时，测试框架将测试标记为已滤除。
- 断言 - 确保满足当前测试的先决条件。
- 致命断言 - 当断言点处的失败使得当前测试方法的其余部分无效或者状态不可恢复时，使用此验证。

测试类型	验证	假设	断言	致命断言
值为 true。	verifyTrue	assumeTrue	assertTrue	fatalAssertTrue
值为 false。	verifyFalse	assumeFalse	assertFalse	fatalAssertFalse
值等于指定值。	verifyEqual	assumeEqual	assertEquals	fatalAssertEqual
值不等于指定值。	verifyNotEqual	assumeNotEqual	assertNotEqual	fatalAssertNotEqual
两个值是同一实例的句柄。	verifySameHandle	assumeSameHandle	assertSameHandle	fatalAssertSameHandle
值不是指定实例的句柄。	verifyNotSameHandle	assumeNotSameHandle	assertNotSameHandle	fatalAssertNotSameHandle
函数在计算时返回 true。	verifyReturnsTrue	assumeReturnsTrue	assertReturnsTrue	fatalAssertReturnsTrue
测试产生无条件失败。	verifyFail	assumeFail	assertFail	fatalAssertFail
值满足指定约束。	verifyThat	assumeThat	assertThat	fatalAssertThat
值大于指定值。	verifyGreaterThan	assumeGreaterThan	assertGreaterThan	fatalAssertGreaterThan
值大于或等于指定值。	verifyGreaterThanOrEqualTo	assumeGreaterThanOrEqualTo	assertGreaterThanOrEqualTo	fatalAssertGreaterThanOrEqualTo
值小于指定值。	verifyLessThan	assumeLessThan	assertLessThan	fatalAssertLessThan
值小于或等于指定值。	verifyLessThanOrEqualTo	assumeLessThanOrEqualTo	assertLessThanOrEqualTo	fatalAssertLessThanOrEqualTo
值是指定的确切类。	verifyClass	assumeClass	assertClass	fatalAssertClass
值是指定类型的对象。	verifyInstanceOf	assumeInstanceOf	assertInstanceOf	fatalAssertInstanceOf
值为空。	verifyEmpty	assumeEmpty	assertEmpty	fatalAssertEmpty
值不为空。	verifyNotEmpty	assumeNotEmpty	assertNotEmpty	fatalAssertNotEmpty
值具有指定大小。	verifySize	assumeSize	assertSize	fatalAssertSize
值具有指定长度。	verifyLength	assumeLength	assertLength	fatalAssertLength
值具有指定的元素数。	verifyNumElements	assumeNumElements	assertNumElements	fatalAssertNumElements
字符串包含指定字符串。	verifySubstring	assumeSubstring	assertSubstring	fatalAssertSubstring

测试类型	验证	假设	断言	致命断言
文本与指定的正则表达式匹配。	<code>verifyMatches</code>	<code>assumeMatches</code>	<code>assertMatches</code>	<code>fatalAssertMatches</code>
函数引发指定异常。	<code>verifyError</code>	<code>assumeError</code>	<code>assertError</code>	<code>fatalAssertError</code>
函数引发指定警告。	<code>verifyWarning</code>	<code>assumeWarning</code>	<code>assertWarning</code>	<code>fatalAssertWarning</code>
函数未发出警告。	<code>verifyWarningFree</code>	<code>assumeWarningFree</code>	<code>assertWarningFree</code>	<code>fatalAssertWarningFree</code>

## 另请参阅

[Assertable](#) | [Assumable](#) | [FatalAssertable](#) | [Verifiable](#) | `matlab.unittest.qualifications`

## 标记单元测试

您可以使用测试标记将测试划分为多个类别，然后运行具有指定标记的测试。典型的测试标记可标识特定功能或描述测试类型。

### 标记测试

要定义测试标记，请使用有意义的字符向量构成的元胞数组，或使用字符串数组。例如，`TestTags = {'Unit'}` 或 `TestTags = ['Unit','FeatureA']`。

- 要标记单独的测试，请使用 `TestTags` 方法属性。
- 要标记一个类中的所有测试，请使用 `TestTags` 类属性。如果您在超类中使用 `TestTags` 类属性，则子类中的测试会继承这些标记。

此示例测试类 `ExampleTagTest` 使用 `TestTags` 方法属性来标记单个测试。

```
classdef ExampleTagTest < matlab.unittest.TestCase
    methods (Test)
        function testA (testCase)
            % test code
            end
    end
    methods (Test, TestTags = {'Unit'})
        function testB (testCase)
            % test code
            end
        function testC (testCase)
            % test code
            end
    end
    methods (Test, TestTags = {'Unit','FeatureA'})
        function testD (testCase)
            % test code
            end
    end
    methods (Test, TestTags = {'System','FeatureA'})
        function testE (testCase)
            % test code
            end
    end
end
```

类 `ExampleTagTest` 中的若干测试会被标记。例如，使用 '`Unit`' 和 '`FeatureA`' 标记 `testD`。未标记其中一个测试 `testA`。

此示例测试类 `ExampleTagClassTest` 使用 `TestTags` 类属性来标记该类中的所有测试，并使用 `TestTags` 方法属性将标记添加到单个测试。

```
classdef (TestTags = {'FeatureB'}) ...
    ExampleTagClassTest < matlab.unittest.TestCase
    methods (Test)
        function testF (testCase)
            % test code
            end
    end
```

```

methods (Test, TestTags = {'FeatureC','System'})
    function testG (testCase)
        % test code
    end
end
methods (Test, TestTags = {'System','FeatureA'})
    function testH (testCase)
        % test code
    end
end
end

```

使用 'FeatureB' 对类 `ExampleTagClassTest` 中的每个测试进行标记。此外，使用多种标记（包括 'FeatureA'、'FeatureC' 和 'System'）来标记单独的测试。

## 选择并运行测试

选择并运行标记的测试有三种方式：

- “使用 `runtests` 运行选定的测试”（第 34-47 页）
- “使用 `TestSuite` 方法选择测试”（第 34-48 页）
- “使用 `HasTag` 选择器选择测试”（第 34-48 页）

### 使用 `runtests` 运行选定的测试

使用 `runtests` 函数选择并运行测试，而不需要显式创建测试套件。选择并运行 `ExampleTagTest` 和 `ExampleTagClassTest` 中所有包含 'FeatureA' 标记的测试。

```
results = runtests({'ExampleTagTest','ExampleTagClassTest'},'Tag','FeatureA');
```

Running ExampleTagTest

..  
Done ExampleTagTest

Running ExampleTagClassTest

.  
Done ExampleTagClassTest

`runtests` 选择并运行了三个测试。

在表中显示结果。

```
table(results)
```

ans =

3×6 table

Name	Passed	Failed	Incomplete	Duration	Details
'ExampleTagTest/testE'	true	false	false	0.00039529	[1×1 struct]
'ExampleTagTest/testD'	true	false	false	0.00045658	[1×1 struct]
'ExampleTagClassTest/testH'	true	false	false	0.00043899	[1×1 struct]

选定的测试为 ExampleTagTest 中的 testE 和 testD 以及 ExampleTagClassTest 中的 testH。

### 使用 TestSuite 方法选择测试

为 ExampleTagTest 类中使用 'FeatureA' 标记的测试创建测试套件。

```
import matlab.unittest.TestSuite
sA = TestSuite.fromClass(?ExampleTagTest,'Tag','FeatureA');
```

为 ExampleTagClassTest 类中使用 'FeatureC' 标记的测试创建测试套件。

```
sB = TestSuite.fromFile('ExampleTagClassTest.m','Tag','FeatureC');
```

串联套件并查看测试名称。

```
suite = [sA sB];
{suite.Name}'
```

ans =

3×1 cell array

'ExampleTagTest/testE'  
 'ExampleTagTest/testD'  
 'ExampleTagClassTest/testG'

### 使用 HasTag 选择器选择测试

为 ExampleTagTest 和 ExampleTagClassTest 类中的所有测试创建测试套件。

```
import matlab.unittest.selectors.HasTag
sA = TestSuite.fromClass(?ExampleTagTest);
sB = TestSuite.fromFile('ExampleTagClassTest.m');
suite = [sA sB];
```

选择所有不含标记的测试。

```
s1 = suite.selectIf(~HasTag)
```

s1 =

Test with properties:

Name: 'ExampleTagTest/testA'  
 ProcedureName: 'testA'  
 TestClass: "ExampleTagTest"  
 BaseFolder: 'C:\work'  
 Parameterization: [0×0 matlab.unittest.parameters.EmptyParameter]  
 SharedTestFixtures: [0×0 matlab.unittest.fixtures.EmptyFixture]  
 Tags: {1×0 cell}

Tests Include:

0 Parameterizations, 0 Shared Test Fixture Classes, 0 Tags.

选择所有包含 'Unit' 标记的测试并显示其名称。

```
s2 = suite.selectIf(HasTag('Unit'));
{s2.Name}'
```

```
ans =  
3×1 cell array  
'ExampleTagTest/testD'  
'ExampleTagTest/testB'  
'ExampleTagTest/testC'
```

使用约束选择所有包含 'FeatureB' 或 'System' 标记的测试。

```
import matlab.unittest.constraints.IsEqualTo  
constraint = IsEqualTo('FeatureB') | IsEqualTo('System');  
s3 = suite.selectIf(HasTag(constraint));  
{s3.Name}'  
  
ans =  
4×1 cell array  
'ExampleTagTest/testE'  
'ExampleTagClassTest/testH'  
'ExampleTagClassTest/testG'  
'ExampleTagClassTest/testF'
```

## 另请参阅

[matlab.unittest.TestCase](#) | [matlab.unittest.TestSuite](#) | [matlab.unittest.constraints](#) |  
[matlab.unittest.selectors.HasTag](#) | [runtests](#)

## 使用共享脚手架编写测试

本示例显示如何使用共享脚手架创建测试。您可以使用 `TestCase` 类的 `SharedTestFixtures` 属性跨测试类共享测试脚手架。为了举例说明此属性，请在您的当前工作文件夹的子目录中创建多个测试类。测试方法仅在高级别显示。

此示例中使用的两个测试类测试 `DocPolynom` 类和 `BankAccount` 类。您可以在 MATLAB 中访问这两个类，但您必须将其添加到 MATLAB 路径。路径脚手架将目录添加到当前路径，运行测试并从该路径中删除该目录。因为两个类都需要添加到路径，所以这些测试使用共享脚手架。

### 为 `DocPolynom` 类创建测试

为 `DocPolynom` 类创建一个测试文件。通过为 `TestCase` 指定 `SharedTestFixtures` 属性并传入 `PathFixture` 来创建共享脚手架。

#### `DocPolynomTest` 类定义文件

```
classdef (SharedTestFixtures={matlab.unittest.fixtures.PathFixture( ...
    fullfile(matlabroot,'help','techdoc','matlab_oop','examples'))}) ...
    DocPolynomTest < matlab.unittest.TestCase
    % Tests the DocPolynom class.

    properties
        msgEqn = 'Equation under test: ';
    end

    methods (Test)
        function testConstructor(testCase)
            p = DocPolynom([1, 0, 1]);
            testCase.verifyClass(p, ?DocPolynom)
        end

        function testAddition(testCase)
            p1 = DocPolynom([1, 0, 1]);
            p2 = DocPolynom([5, 2]);

            actual = p1 + p2;
            expected = DocPolynom([1, 5, 3]);

            msg = [testCase.msgEqn, ...
                '(x^2 + 1) + (5*x + 2) = x^2 + 5*x + 3'];
            testCase.verifyEqual(actual, expected, msg)
        end

        function testMultiplication(testCase)
            p1 = DocPolynom([1, 0, 3]);
            p2 = DocPolynom([5, 2]);

            actual = p1 * p2;
            expected = DocPolynom([5, 2, 15, 6]);

            msg = [testCase.msgEqn, ...
                '(x^2 + 3) * (5*x + 2) = 5*x^3 + 2*x^2 + 15*x + 6'];
            testCase.verifyEqual(actual, expected, msg)
        end
    end
```

```
    end
end
```

### 为 BankAccount 类创建测试

针对 BankAccount 类创建一个测试文件。通过指定 TestCase 的 SharedTestFixtures 属性并传入 PathFixture 来创建共享脚手架。

### BankAccountTest 类定义文件

```
classdef (SharedTestFixtures={matlab.unittest.fixtures.PathFixture( ...
    fullfile(matlabroot, 'help', 'techdoc', 'matlab_oop', ...
    'examples'))}) BankAccountTest < matlab.unittest.TestCase
% Tests the BankAccount class.

methods (Test)
    function testConstructor(testCase)
        b = BankAccount(1234, 100);
        testCase.verifyEqual(b.AccountNumber, 1234, ...
            'Constructor failed to correctly set account number')
        testCase.verifyEqual(b.AccountBalance, 100, ...
            'Constructor failed to correctly set account balance')
    end

    function testConstructorNotEnoughInputs(testCase)
        import matlab.unittest.constraints.Throws
        testCase.verifyThat(@()BankAccount, ...
            Throws('MATLAB:minrhs'))
    end

    function testDeposit(testCase)
        b = BankAccount(1234, 100);
        b.deposit(25)
        testCase.verifyEqual(b.AccountBalance, 125)
    end

    function testWithdraw(testCase)
        b = BankAccount(1234, 100);
        b.withdraw(25)
        testCase.verifyEqual(b.AccountBalance, 75)
    end

    function testNotifyInsufficientFunds(testCase)
        callbackExecuted = false;
        function testCallback(~,~)
            callbackExecuted = true;
        end

        b = BankAccount(1234, 100);
        b.addlistener('InsufficientFunds', @testCallback);

        b.withdraw(50)
        testCase.assertFalse(callbackExecuted, ...
            'The callback should not have executed yet')
        b.withdraw(60)
        testCase.verifyTrue(callbackExecuted, ...
            'The listener callback should have fired')
    end
end
```

```
    end  
end
```

### 编译测试套件

类 DocPolynomTest.m 和 BankAccountTest.m 在您的工作目录中。从您的当前工作目录创建一个测试套件。如果您有其他测试，它们会在您使用 TestSuite.fromFolder 方法时包括在该套件中。在命令提示符下创建测试套件。

```
import matlab.unittest.TestSuite;  
suiteFolder = TestSuite.fromFolder(pwd);
```

### 运行测试

在命令提示符下，运行测试套件中的测试。

```
result = run(suiteFolder);
```

Setting up PathFixture.

Description: Adds 'C:\Program Files\MATLAB\R2013b\help\techdoc\matlab\_oop\examples' to the path.

---

Running BankAccountTest

.....

Done BankAccountTest

---

Running DocPolynomTest

...

Done DocPolynomTest

---

Tearing down PathFixture.

Description: Restores the path to its previous state.

---

测试框架设置测试脚手架，在每个文件中运行所有测试，然后拆解脚手架。如果使用 TestClassSetup 方法设置和拆解路径脚手架，将设置并拆解该脚手架两次 - 每个测试文件各一次。

### 另请参阅

[PathFixture](#) | [TestCase](#) | [matlab.unittest.fixtures](#)

# 创建基本自定义脚手架

此示例演示如何创建基本自定义脚手架，以便将显示格式更改为十六进制表示。此示例还演示使用该脚手架测试一个将一列数字显示为文本的函数。测试完成后，该框架将显示格式恢复为其预测试状态。

## 创建 FormatHexFixture 类定义

在您的工作文件夹下的文件中，创建一个从 `matlab.unittest.fixtures.Fixture` 类继承的新类 `FormatHexFixture`。因为我们希望该脚手架恢复 MATLAB 显示格式的预测试状态，所以创建一个 `OriginalFormat` 属性来跟踪原始显示格式。

```
classdef FormatHexFixture < matlab.unittest.fixtures.Fixture
    properties (Access=private)
        OriginalFormat
    end
```

## 实现设置和拆解方法

`Fixture` 类的子类必须实现 `setup` 方法。使用此方法记录预测试显示格式，并将该格式设置为 `'hex'`。使用 `teardown` 方法恢复原始显示格式。在 `FormatHexFixture.m` 文件的 `methods` 块中定义 `setup` 和 `teardown` 方法。

```
methods
    function setup(fixture)
        fixture.OriginalFormat = get(0, 'Format');
        set(0, 'Format', 'hex')
    end
    function teardown(fixture)
        set(0, 'Format', fixture.OriginalFormat)
    end
end
end
```

## 应用自定义脚手架

在您的工作文件夹下的文件中，创建以下测试类 `SampleTest.m`。

```
classdef SampleTest < matlab.unittest.TestCase
    methods (Test)
        function test1(testCase)
            testCase.applyFixture(FormatHexFixture);
            actStr = getColumnForDisplay([1;2;3], 'Small Integers');
            expStr = ['Small Integers '
                      '3ff0000000000000'
                      '4000000000000000'
                      '4008000000000000'];
            testCase.verifyEqual(actStr, expStr)
        end
    end
end

function str = getColumnForDisplay(values, title)
elements = cell(numel(values)+1, 1);
elements{1} = title;
for idx = 1:numel(values)
    elements{idx+1} = displayNumber(values(idx));
end
```

```
str = char(elements);
end

function str = displayNumber(n)
str = strtrim(evalc('disp(n);'));
end
```

该测试应用自定义脚手架并验证是否按预期显示十六进制表示形式的列。

在命令提示符下运行测试。

```
run(SampleTest);

Running SampleTest
.
Done SampleTest
```

---

### 另请参阅

`matlab.unittest.fixtures.Fixture`

### 相关示例

- “创建高级自定义脚手架” (第 34-55 页)
- “使用共享脚手架编写测试” (第 34-50 页)

# 创建高级自定义脚手架

此示例演示如何创建设置环境变量的自定义脚手架。在测试之前，该脚手架将保存当前 `UserName` 变量。

## 创建 `UserNameEnvironmentVariableFixture` 类定义

在您的工作文件夹下的文件中，创建一个从 `matlab.unittest.fixtures.Fixture` 类继承的新类 `UserNameEnvironmentVariableFixture`。因为您希望向该脚手架传递用户名，所以创建一个 `UserName` 属性来在不同方法之间传递数据。

```
classdef UserNameEnvironmentVariableFixture < ...
    matlab.unittest.fixtures.Fixture

    properties (SetAccess=private)
        UserName
    end
```

## 定义脚手架构造函数

在 `UserNameEnvironmentVariableFixture.m` 文件的 `methods` 块中，创建一个构造函数方法来验证输入并定义 `SetupDescription`。让构造函数接受字符串并设置该脚手架的 `UserName` 属性。

```
methods
    function fixture = UserNameEnvironmentVariableFixture(name)
        validateattributes(name, {'char'}, {'row'}, ", 'UserName')
        fixture.UserName = name;
        fixture.SetupDescription = sprintf( ...
            'Set the UserName environment variable to "%s"!', ...
            fixture.UserName);
    end
```

## 实现设置方法

`Fixture` 类的子类必须实现 `setup` 方法。使用此方法保存原始 `UserName` 变量。此方法还定义 `TeardownDescription` 并注册拆解任务：测试后将 `UserName` 设置为原始状态。

在 `UserNameEnvironmentVariableFixture.m` 文件的 `methods` 块中定义 `setup` 方法。

```
function setup(fixture)
    originalUserName = getenv('UserName');
    fixture.assertNotEmpty(originalUserName, ...
        'An existing UserName environment variable must be defined.')
    fixture.addTeardown(@setenv, 'UserName', originalUserName)
    fixture.TeardownDescription = sprintf(...
        'Restored the UserName environment variable to "%s"!', ...
        originalUserName);
    setenv('UserName', fixture.UserName)
end
end
```

## 实现 `isCompatible` 方法

如果构造函数可配置，派生自 `Fixture` 的类必须实现 `isCompatible` 方法。因为您可以通过构造函数配置 `UserName` 属性，所以 `UserNameEnvironmentVariableFixture` 必须实现 `isCompatible`。

`isCompatible` 方法是通过同一类的两个实例调用的。本例中，它是通过 `UserNameEnvironmentVariableFixture` 的两个实例调用的。如果这两个实例的 `UserName` 属性相等，则测试框架将这两个实例视为兼容的。

在 `UserNameEnvironmentVariableFixture.m` 内的新 `methods` 块中，定义一个返回逻辑值 1 (`true`) 或逻辑值 0 (`false`) 的 `isCompatible` 方法。

```

methods (Access=protected)
    function bool = isCompatible(fixture, other)
        bool = strcmp(fixture.UserName, other.UserName);
    end
end

```

### 脚手架类定义总结

以下是 `UserNameEnvironmentVariableFixture.m` 的完整内容。

```

classdef UserNameEnvironmentVariableFixture < ...
    matlab.unittest.fixture

    properties (SetAccess=private)
        UserName
    end

    methods
        function fixture = UserNameEnvironmentVariableFixture(name)
            validateattributes(name, {'char'}, {'row'}, ',UserName')
            fixture.UserName = name;
            fixture.SetupDescription = sprintf( ...
                'Set the UserName environment variable to "%s".',...
                fixture.UserName);
        end

        function setup(fixture)
            originalUserName = getenv('UserName');
            fixture.assertNotEmpty(originalUserName, ...
                'An existing UserName environment variable must be defined.')
            fixture.addTeardown(@setenv, 'UserName', originalUserName)
            fixture.TeardownDescription = sprintf(... ...
                'Restored the UserName environment variable to "%s".',...
                originalUserName);
            setenv('UserName', fixture.UserName)
        end
    end

    methods (Access=protected)
        function bool = isCompatible(fixture, other)
            bool = strcmp(fixture.UserName, other.UserName);
        end
    end
end

```

### 将自定义脚手架应用于单个测试类

在您的工作文件夹下的文件中，创建以下测试类 `ExampleTest.m`。

```

classdef ExampleTest < matlab.unittest.TestCase
    methods(TestMethodSetup)
        function mySetup(testCase)
            testCase.applyFixture(... ...
                UserNameEnvironmentVariableFixture('David'));
        end
    end

    methods (Test)
        function t1(~)
            fprintf(1, 'Current UserName: "%s", getenv("UserName")')
        end
    end
end

```

该测试对 `ExampleTest` 类中的每个测试使用 `UserNameEnvironmentVariableFixture`。

在命令提示符下运行测试。

```

run(ExampleTest);

Running ExampleTest
Current UserName: "David".
Done ExampleTest

```

### 将自定义脚手架应用为共享脚手架

在您的工作文件夹中，使用共享脚手架创建三个测试类。使用共享脚手架允许跨类共享 **UserNameEnvironmentVariableFixture**。

按如下所示创建 **testA.m**。

```

classdef (SharedTestFixtures=...
    UserNameEnvironmentVariableFixture('David')) ...
    testA < matlab.unittest.TestCase
methods (Test)
    function t1(~)
        fprintf(1, 'Current UserName: "%s", getenv('UserName'))
    end
end
end

```

按如下所示创建 **testB.m**。

```

classdef (SharedTestFixtures=...
    UserNameEnvironmentVariableFixture('Andy')) ...
    testB < matlab.unittest.TestCase
methods (Test)
    function t1(~)
        fprintf(1, 'Current UserName: "%s", getenv('UserName'))
    end
end
end

```

按如下所示创建 **testC.m**。

```

classdef (SharedTestFixtures=...
    UserNameEnvironmentVariableFixture('Andy')) ...
    testC < matlab.unittest.TestCase
methods (Test)
    function t1(~)
        fprintf(1, 'Current UserName: "%s", getenv('UserName'))
    end
end
end

```

在命令提示符下运行这些测试。

```
runtests({'testA','testB','testC'});
```

```

Setting up UserNameEnvironmentVariableFixture
Done setting up UserNameEnvironmentVariableFixture: Set the UserName environment variable to "David".

```

---

```

Running testA
Current UserName: "David".
Done testA

```

---

Tearing down UserNameEnvironmentVariableFixture

Done tearing down UserNameEnvironmentVariableFixture: Restored the UserName environment variable to "K

---

Setting up UserNameEnvironmentVariableFixture

Done setting up UserNameEnvironmentVariableFixture: Set the UserName environment variable to "Andy".

---

Running testB

Current UserName: "Andy".

Done testB

---

Running testC

Current UserName: "Andy".

Done testC

---

Tearing down UserNameEnvironmentVariableFixture

Done tearing down UserNameEnvironmentVariableFixture: Restored the UserName environment variable to "K

---

请记住，如果脚手架的 `UserName` 属性匹配，则它们是兼容的。`testA` 和 `testB` 中的测试使用了不兼容的共享脚手架，因为 '`David`' 不等于 '`Andy`'。因此，该框架在对 `testA` 和 `testB` 的各次调用中调用脚手架的 `teardown` 和 `setup` 方法。不过，`testC` 中的共享测试脚手架与 `testB` 中的脚手架兼容，因此该框架在 `testC` 之前不重复脚手架拆解和设置。

### 在设置方法中调用 `addTeardown` 的替代方法

在 `setup` 方法中使用 `addTeardown` 方法的替代方法是实现单独的 `teardown` 方法。实现 `UserNameEnvironmentVariableFixture.m` 中的以下 `setup` 和 `teardown` 方法，而不是上述的 `setup` 方法。

### 替代 `UserNameEnvironmentVariableFixture` 类定义

```
classdef UserNameEnvironmentVariableFixture < ...
    matlab.unittest.fixtures.Fixture

    properties (Access=private)
        OriginalUser
    end
    properties (SetAccess=private)
        UserName
    end

    methods
        function fixture = UserNameEnvironmentVariableFixture(name)
            validateattributes(name, {'char'}, {'row'}, ",'UserName'")
            fixture.UserName = name;
            fixture.SetupDescription = sprintf( ...
                'Set the UserName environment variable to "%s".',...
                fixture.UserName);
        end

        function setup(fixture)
            fixture.OriginalUser = getenv('UserName');
            fixture.assertNotEmpty(fixture.OriginalUser, ...
                'An existing UserName environment variable must be defined.')
            setenv('UserName', fixture.UserName)
        end
    end
```

```
function teardown(fixture)
    fixture.TeardownDescription = sprintf(...%
        'Restored the %s environment variable to "%s"!',...
        fixture.OriginalUser);
    setenv('UserName', fixture.OriginalUser)

    end
end

methods (Access=protected)
    function bool = isCompatible(fixture, other)
        bool = strcmp(fixture.UserName, other.UserName);
    end
end
end
```

`setup` 方法不包含对 `addTeardown` 的调用或 `TeardownDescription` 的定义。这些任务转移到了 `teardown` 方法。替代类定义包含一个额外属性 `OriginalUser`，该属性允许在各方法之间传递信息。

## 另请参阅

`matlab.unittest.fixtures.Fixture`

## 相关示例

- “[创建基本自定义脚手架](#)”（第 34-53 页）
- “[使用共享脚手架编写测试](#)”（第 34-50 页）

## 创建基本参数化测试

此示例演示如何创建基本参数化测试。

### 创建要测试的函数

在您的工作文件夹下，于文件 `sierpinsk.m` 中创建一个函数。此函数返回一个表示 Sierpinski 地毯分形图像的矩阵。它采用分形级别和可选数据类型作为输入。

```
function carpet = sierpinsk(nLevels,classname)
if nargin == 1
    classname = 'single';
end

mSize = 3^nLevels;
carpet = ones(mSize,classname);

cutCarpet(1,1,mSize,nLevels) % begin recursion

function cutCarpet(x,y,s,cL)
if cL
    ss = s/3; % define subsize
    for lx = 0:2
        for ly = 0:2
            if lx == 1 && ly == 1
                % remove center square
                carpet(x+ss:x+2*ss-1,y+ss:y+2*ss-1) = 0;
            else
                % recurse
                cutCarpet(x + lx*ss, y + ly*ss, ss, cL-1)
            end
        end
    end
end
end
end
```

### 创建 `TestCarpet` 测试类

在您的工作文件夹下的文件中，创建一个新类 `TestCarpet` 以测试 `sierpinsk` 函数。

```
classdef TestCarpet < matlab.unittest.TestCase
```

### 定义属性块

定义用于参数化测试的属性。在 `TestCarpet` 类中，使用 `TestParameter` 特性在一个属性块中定义这些属性。

```
properties (TestParameter)
    type = {'single','double','uint16'};
    level = struct('small', 2,'medium', 4, 'large', 6);
    side = struct('small', 9, 'medium', 81,'large', 729);
end
```

`type` 属性包含您要测试的不同数据类型。`level` 属性包含您要测试的不同分形级别。`side` 属性包含 Sierpinski 地毯矩阵的行数和列数并且对应于 `level` 属性。要为每个参数化值提供有意义的名称，请将 `level` 和 `side` 定义为结构体。

## 定义测试方法块

在 TestCarpet 类中定义以下测试方法。

```
methods (Test)
    function testRemainPixels(testCase, level)
        % expected number pixels equal to 1
        expPixelCount = 8^level;
        % actual number pixels equal to 1
        actPixels = find(sierpinski(level));
        testCase.verifyNumElements(actPixels,expPixelCount)
    end

    function testClass(testCase, type, level)
        testCase.verifyClass(...,
            sierpinski(level,type), type);
    end

    function testDefaultL1Output(testCase)
        exp = single([1 1 1; 1 0 1; 1 1 1]);
        testCase.verifyEqual(sierpinski(1), exp)
    end
end
```

`testRemainPixels` 方法通过验证特定级别的非零像素数是否与预期相同，来测试 `sierpinski` 函数的输出。此方法使用 `level` 属性，因此产生三个测试元素 - `level` 中的每个值各一个。`testClass` 方法使用 `type` 和 `level` 属性的每个组合测试从 `sierpinski` 函数输出的类。此方法产生九个测试元素。`testDefaultL1Output` 测试方法不使用 `TestParameter` 属性，因此不会被参数化。该测试方法验证 1 级矩阵是否包含预期值。因为该测试方法未被参数化，所以它产生一个测试元素。

在上面的测试方法中，您未定义 `Test` 方法块的 `ParameterCombination` 属性。该属性默认为 `'exhaustive'`。测试框架对测试参数的每个组合调用给定测试方法一次。

## 使用 `ParameterCombination` 属性定义测试方法块

在 `TestCarpet` 类中定义以下测试方法以确保 `sierpinski` 函数的矩阵输出具有正确数目的元素。将 `ParameterCombination` 属性设置为 `'sequential'`。

```
methods (Test, ParameterCombination='sequential')
    function testNumel(testCase, level, side)
        import matlab.unittest.constraints.HasElementCount
        testCase.verifyThat(sierpinski(level), ...
            HasElementCount(side^2))
    end
end
end
```

其 `ParameterCombination` 属性设置为 `'sequential'` 的测试方法对该参数的每个相应值调用一次。属性 `level` 和 `side` 必须具有相同数目的值。因为这些属性中的每个都有三个值，所以 `testNumel` 方法被调用三次。

## TestCarpet 类定义总结

`TestCarpet.m` 的完整内容如下。

```
classdef TestCarpet < matlab.unittest.TestCase
```

```

properties (TestParameter)
    type = {'single','double','uint16'};
    level = struct('small', 2,'medium', 4, 'large', 6);
    side = struct('small', 9, 'medium', 81,'large', 729);
end

methods (Test)
    function testRemainPixels(testCase, level)
        % expected number pixels equal to 1
        expPixelCount = 8^level;
        % actual number pixels equal to 1
        actPixels = find(sierpinski(level));
        testCase.verifyNumElements(actPixels,expPixelCount)
    end

    function testClass(testCase, type, level)
        testCase.verifyClass...
            sierpinski(level,type), type)
    end

    function testDefaultL1Output(testCase)
        exp = single([1 1 1; 1 0 1; 1 1 1]);
        testCase.verifyEqual(sierpinski(1), exp)
    end
end

methods (Test, ParameterCombination='sequential')
    function testNumel(testCase, level, side)
        import matlab.unittest.constraints.HasElementCount
        testCase.verifyThat(sierpinski(level),...
            HasElementCount(side^2))
    end
    end
end

```

### 运行所有测试

在命令提示符下，基于 `TestCarpet.m` 创建一个套件。

```

suite = matlab.unittest.TestSuite.fromFile('TestCarpet.m');
{suite.Name}

ans =

```

```

'TestCarpet/testNumel(level=small,side=small)'
'TestCarpet/testNumel(level=medium,side=medium)'
'TestCarpet/testNumel(level=large,side=large)'
'TestCarpet/testRemainPixels(level=small)'
'TestCarpet/testRemainPixels(level=medium)'
'TestCarpet/testRemainPixels(level=large)'
'TestCarpet/testClass(type=single,level=small)'
'TestCarpet/testClass(type=single,level=medium)'
'TestCarpet/testClass(type=single,level=large)'
'TestCarpet/testClass(type=double,level=small)'
'TestCarpet/testClass(type=double,level=medium)'
'TestCarpet/testClass(type=double,level=large)'
'TestCarpet/testClass(type=uint16,level=small)'
'TestCarpet/testClass(type=uint16,level=medium)'

```

```
'TestCarpet/testClass(type=uint16,level=large)'
'TestCarpet/testDefaultL1Output'
```

该套件有 16 个测试元素。该元素的 Name 指示任何参数化。

```
suite.run;

Running TestCarpet
.....
.....
Done TestCarpet
```

### 运行 level 参数属性命名为 small 的测试

使用 TestSuite 的 selectIf 方法选择使用特定参数化的测试元素。选择所有在 level 参数属性列表中使用参数名称 small 的测试元素。

```
s1 = suite.selectIf('ParameterName','small');
{s1.Name}

ans =

'TestCarpet/testNumel(level=small,side=small)'
'TestCarpet/testRemainPixels(level=small)'
'TestCarpet/testClass(type=single,level=small)'
'TestCarpet/testClass(type=double,level=small)'
'TestCarpet/testClass(type=uint16,level=small)'
```

该套件有五个元素。

```
s1.run;

Running TestCarpet
.....
Done TestCarpet
```

或者，直接从 TestSuite 的 fromFile 方法创建同一测试套件。

```
import matlab.unittest.selectors.HasParameter
s1 = matlab.unittest.TestSuite.fromFile("TestCarpet.m',...
    HasParameter('Name','small'));
```

## 另请参阅

[matlab.unittest.TestCase](#) | [matlab.unittest.selectors.HasParameter](#) | [selectIf](#)

## 相关示例

- “[创建高级参数化测试](#)”（第 34-64 页）

## 创建高级参数化测试

此示例演示如何创建在 `TestClassSetup`、`TestMethodSetup` 和 `Test methods` 块中被参数化的测试。该示例测试类测试随机数生成器。

### 测试概述

`TestRand` 测试类在三个不同级别被参数化。

参数化级别	参数化定义		可访问的参数化属性
	方法属性	属性特性	
测试级别	<code>Test</code>	<code>TestParameter</code>	<code>TestParameter</code> 、 <code>MethodSetupParameter</code> 和 <code>ClassSetupParameter</code>
方法设置级别	<code>TestMethodSetup</code>	<code>MethodSetupParameter</code>	<code>MethodSetupParameter</code> 和 <code>ClassSetupParameter</code>
类设置级别	<code>TestClassSetup</code>	<code>ClassSetupParameter</code>	<code>ClassSetupParameter</code>

在每个测试级别，您可以使用 `ParameterCombination` 方法属性指定测试参数化。

ParameterCombination 属性	方法调用
'exhaustive' (默认值)	对所有的参数组合调用方法。如何您不指定 <code>ParameterCombination</code> 属性，则测试框架使用此默认组合。
'sequential'	通过每个参数中的相应值调用方法。每个参数必须包含相同数量的值。
'pairwise'	至少对每对参数值调用一次方法。虽然测试框架可保证至少为每对值创建一次测试，但您不应依赖该大小、顺序或特定组的测试套件元素。

例如，使用组合的方法属性 `TestMethodSetup`, `ParameterCombination='sequential'` 指定 `MethodSetupParameter` 属性块中定义的方法设置级别参数的顺序组合。

对于此示例，类设置级别参数化定义随机数生成器的类型。方法设置级别参数化定义随机数生成器的种子，测试级别参数化定义随机数输出的数据类型和大小。

### 创建 `TestRand` 测试类

在您的工作文件夹下的文件中，创建一个从 `matlab.unittest.TestCase` 继承的类。该类测试随机数生成的各个方面。

```
classdef TestRand < matlab.unittest.TestCase
```

#### 定义属性块

定义用于参数化测试的属性。每个 `properties` 块对应于特定级别的参数化。

```
properties (ClassSetupParameter)
    generator = {'twister','combRecursive','multFibonacci'};
```

```

end

properties (MethodSetupParameter)
  seed = {0, 123, 4294967295};
end

properties (TestParameter)
  dim1 = struct('small', 1,'medium', 2, 'large', 3);
  dim2 = struct('small', 2,'medium', 3, 'large', 4);
  dim3 = struct('small', 3,'medium', 4, 'large', 5);
  type = {'single','double'};
end

```

## 定义测试类和测试方法设置方法

定义测试类和测试方法级别的设置方法。这些方法注册初始随机数生成器状态。该框架运行测试后，这些方法将恢复原始状态。ClassSetup 方法定义随机数生成器的类型，TestMethodSetup 为生成器提供种子。

```

methods (TestClassSetup)
  function ClassSetup(testCase, generator)
    orig = rng;
    testCase.addTeardown(@rng, orig)
    rng(0, generator)
  end
end

methods (TestMethodSetup)
  function MethodSetup(testCase, seed)
    orig = rng;
    testCase.addTeardown(@rng, orig)
    rng(seed)
  end
end

```

## 定义顺序参数化的测试方法

使用 Test 和 ParameterCombination='sequential' 属性定义 methods 块。测试框架对每个对应的属性值调用这些方法一次。

```

methods (Test, ParameterCombination='sequential')
  function testSize(testCase,dim1,dim2,dim3)
    testCase.verifySize(rand(dim1,dim2,dim3),[dim1 dim2 dim3])
  end
end

```

该方法可用于测试 dim1、dim2 和 dim3 中的每个对应的参数的输出的大小。例如，要测试所有 'medium' 值，请使用 testCase.verifySize(rand(2,3,4),[2 3 4]);。对于给定的 TestClassSetup 和 TestMethodSetup 参数化，该框架调用 testSize 方法三次 - 'small'、'medium' 和 'large' 值各一次。

## 定义成对的参数化测试方法

使用 Test 和 ParameterCombination='pairwise' 属性定义 methods 块。测试框架对每对属性值至少调用这些方法一次。

```

methods (Test, ParameterCombination='pairwise')
  function testRepeatable(testCase,dim1,dim2,dim3)

```

```

state = rng;
firstRun = rand(dim1,dim2,dim3);
rng(state)
secondRun = rand(dim1,dim2,dim3);
testCase.verifyEqual(firstRun,secondRun)
end
end

```

测试方法验证随机数生成器结果是否可重复。对于给定的 `TestClassSetup` 和 `TestMethodSetup` 参数化，该框架调用 `testRepeatable` 方法 10 次以确保测试每对 `dim1`、`dim2` 和 `dim3`。但是，如果参数组合属性是 `Exhaustive`，该框架调用该方法  $3^3=27$  次。

### 定义 `Exhaustive` 参数化测试方法

使用 `Test` 属性或未定义的参数组合定义 `methods` 块。该参数组合默认为 `exhaustive`。测试框架对每个属性值组合调用这些方法一次。

```

methods (Test)
    function testClass(testCase,dim1,dim2,type)
        testCase.verifyClass(rand(dim1,dim2,type), type)
    end
end

```

该测试方法验证从 `rand` 输出的类是否与预期的类相同。对于给定的 `TestClassSetup` 和 `TestMethodSetup` 参数化，该框架调用 `testClass` 方法  $3 \times 3 \times 2 = 18$  次以确保测试每个 `dim1`、`dim2` 和 `type` 组合。

### TestRand 类定义总结

```

classdef TestRand < matlab.unittest.TestCase
    properties (ClassSetupParameter)
        generator = {'twister','combRecursive','multFibonacci'};
    end

    properties (MethodSetupParameter)
        seed = {0, 123, 4294967295};
    end

    properties (TestParameter)
        dim1 = struct('small', 1,'medium', 2, 'large', 3);
        dim2 = struct('small', 2,'medium', 3, 'large', 4);
        dim3 = struct('small', 3,'medium', 4, 'large', 5);
        type = {'single','double'};
    end

    methods (TestClassSetup)
        function ClassSetup(testCase, generator)
            orig = rng;
            testCase.addTeardown(@rng, orig)
            rng(0, generator)
        end
    end

    methods (TestMethodSetup)
        function MethodSetup(testCase, seed)
            orig = rng;
            testCase.addTeardown(@rng, orig)
            rng(seed)
        end
    end

```

```

    end
end

methods (Test, ParameterCombination='sequential')
    function testSize(testCase,dim1,dim2,dim3)
        testCase.verifySize(rand(dim1,dim2,dim3),[dim1 dim2 dim3])
    end
end

methods (Test, ParameterCombination='pairwise')
    function testRepeatable(testCase,dim1,dim2,dim3)
        state = rng;
        firstRun = rand(dim1,dim2,dim3);
        rng(state)
        secondRun = rand(dim1,dim2,dim3);
        testCase.verifyEqual(firstRun,secondRun);
    end
end

methods (Test)
    function testClass(testCase,dim1,dim2,type)
        testCase.verifyClass(rand(dim1,dim2,type), type)
    end
end
end

```

### 从所有测试创建套件

在命令提示符下，基于 `TestRand.m` 类创建一个套件。

```

suite = matlab.unittest.TestSuite.fromClass(?TestRand)
suite =

```

`1×279 Test array with properties:`

Name
ProcedureName
TestClass
BaseFolder
Parameterization
SharedTestFixture
Tags

Tests Include:

17 Unique Parameterizations, 0 Shared Test Fixture Classes, 0 Tags.

该测试套件包含 279 个测试元素。对于给定的 `TestClassSetup` 和 `TestMethodSetup` 参数化，该框架创建  $3+10+18=31$  个测试元素。这 31 个元素被调用三次 - 每个 `TestMethodSetup` 参数化一次，从而为每个 `TestClassSetup` 参数化生成  $3*31=93$  个测试元素。存在三个 `TestClassSetup` 参数化，生成总计  $3*93=279$  个测试元素。

检查第一个测试元素的名称。

```
suite(1).Name
```

```
ans =
```

```
'TestRand[generator=twister]/[seed=value1]testClass(dim1=small,dim2=small,type=single)'
```

每个元素的名称是根据以下项的组合构造的：

- 测试类： **TestRand**
- 类设置属性和属性名： **[generator=twister]**
- 方法设置属性和属性名： **[seed=value1]**
- 测试方法名称： **testClass**
- 测试方法属性和属性名称： **(dim1=small,dim2=small,type=single)**

**seed** 属性的名称并不是特别有意义 (**value1**)。测试框架提供此名称是因为 **seed** 属性值是数字。要使名称更有意义，请使用说明性更强的字段名称将 **seed** 属性定义为结构体。

### 使用选择器从类运行套件

在命令提示符下，创建一个选择器以选择用于测试 'twister' 生成器的 'single' 精度的测试元素。省略使用具有 'large' 名称的属性的测试元素。

```
import matlab.unittest.selectors.HasParameter
s = HasParameter('Property','generator', 'Name','twister') & ...
    HasParameter('Property','type', 'Name','single') & ...
    ~HasParameter('Name','large');

suite2 = matlab.unittest.TestSuite.fromClass(?TestRand,s)

suite2 =
1x12 Test array with properties:

    Name
    ProcedureName
    TestClass
    BaseFolder
    Parameterization
    SharedTestFixtures
    Tags

Tests Include:
    9 Unique Parameterizations, 0 Shared Test Fixture Classes, 0 Tags.
```

如果您首先生成完整套件，请使用 **selectIf** 方法构造与上面相同的测试套件。

```
suite = matlab.unittest.TestSuite.fromClass(?TestRand);
suite2 = selectIf(suite,s);
```

运行测试套件。

```
suite2.run;

Running TestRand
.....
..
Done TestRand
```

### 使用选择器从方法运行套件

在命令提示符下，创建一个选择器，以便省略使用具有 'large' 或 'medium' 名称的属性的测试元素。将结果限制为来自 **testRepeatable** 方法的测试元素。

```

import matlab.unittest.selectors.HasParameter
s = ~(HasParameter('Name','large') | HasParameter('Name','medium'));

suite3 = matlab.unittest.TestSuite.fromMethod(?TestRand,'testRepeatable',s);
{suite3.Name}

ans =
9×1 cell array

'TestRand[generator=twister]/[seed=value1]testRepeatable(dim1=small,dim2=small,dim3=small)'
'TestRand[generator=twister]/[seed=value2]testRepeatable(dim1=small,dim2=small,dim3=small)'
'TestRand[generator=twister]/[seed=value3]testRepeatable(dim1=small,dim2=small,dim3=small)'
'TestRand[generator=combRecursive]/[seed=value1]testRepeatable(dim1=small,dim2=small,dim3=small)'
'TestRand[generator=combRecursive]/[seed=value2]testRepeatable(dim1=small,dim2=small,dim3=small)'
'TestRand[generator=combRecursive]/[seed=value3]testRepeatable(dim1=small,dim2=small,dim3=small)'
'TestRand[generator=multFibonacci]/[seed=value1]testRepeatable(dim1=small,dim2=small,dim3=small)'
'TestRand[generator=multFibonacci]/[seed=value2]testRepeatable(dim1=small,dim2=small,dim3=small)'
'TestRand[generator=multFibonacci]/[seed=value3]testRepeatable(dim1=small,dim2=small,dim3=small)'

```

运行测试套件。

```
suite3.run;
```

Running TestRand

.....

Done TestRand

### 运行所有双精度测试

在命令提示符下，通过 `TestRand.m` 运行所有使用参数名称 '`double`' 的测试元素。

```
runtests('TestRand','ParameterName','double');
```

Running TestRand

.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.

Done TestRand

### 另请参阅

[matlab.unittest.TestCase](#) | [matlab.unittest.TestSuite](#) | [matlab.unittest.selectors](#)

### 相关示例

- “[创建基本参数化测试](#)”（第 34-60 页）

## 在参数化测试中使用外部参数

您可以将变量输入插入到基于类的现有测试中。要提供在测试文件外定义并且应该由测试迭代使用（通过参数化测试）的测试数据，请创建一个 **Parameter** 并使用 'ExternalParameters' 选项作为 **TestSuite** 创建方法，例如 **TestSuite.fromClass**。

创建以下函数进行测试。该函数接受数组，对数组进行向量化，删除 0、**Nan** 和 **Inf**，然后对数组进行排序。

```
function Y = cleanData(X)
    Y = X(:); % Vectorize array
    Y = rmmissing(Y); % Remove NaN
    % Remove 0 and Inf
    idx = (Y==0 | Y==Inf);
    Y = Y(~idx);
    % If array is empty, set to eps
    if isempty(Y)
        Y = eps;
    end
    Y = sort(Y); % Sort vector
end
```

为 **cleanData** 函数创建以下参数化测试。该测试对 **properties** 模块中定义的两个数据集重复使用四种测试方法。

```
classdef TestClean < matlab.unittest.TestCase
    properties (TestParameter)
        Data = struct('clean',[5 3 9;1 42 5;32 5 2],...
            'needsCleaning',[1 13;NaN 0;Inf 42]);
    end
    methods (Test)
        function classCheck(testCase,Data)
            act = cleanData(Data);
            testCase.assertClass(act,'double')
        end
        function sortCheck(testCase,Data)
            act = cleanData(Data);
            testCase.verifyTrue(issorted(act))
        end
        function finiteCheck(testCase,Data)
            import matlab.unittest.constraints.IsFinite
            act = cleanData(Data);
            testCase.verifyThat(act,IsFinite)
        end
        function noZeroCheck(testCase,Data)
            import matlab.unittest.constraints.EveryElementOf
            import matlab.unittest.constraints.AreEqual
            act = cleanData(Data);
            testCase.verifyThat(EveryElementOf(act),~IsEqualTo(0))
        end
    end
end
```

创建并运行参数化测试套件。查看结果。该框架使用测试文件中定义的数据运行八个参数化测试。

```

import matlab.unittest.TestSuite
suite1 = TestSuite.fromClass(?TestClean);
results = suite1.run;
table(results)

Running TestClean
.....
Done TestClean
_____

ans =
8×6 table

    Name      Passed  Failed  Incomplete  Duration  Details
    _____  _____  _____  _____  _____  _____
    'TestClean/classCheck(Data=clean)'    true    false    false    0.27887 [1×1 struct]
    'TestClean/classCheck(Data=needsCleaning)'  true    false    false    0.0069881 [1×1 struct]
    'TestClean/sortCheck(Data=clean)'    true    false    false    0.0046028 [1×1 struct]
    'TestClean/sortCheck(Data=needsCleaning)'  true    false    false    0.0035363 [1×1 struct]
    'TestClean/finiteCheck(Data=clean)'   true    false    false    0.023965 [1×1 struct]
    'TestClean/finiteCheck(Data=needsCleaning)' true    false    false    0.0024557 [1×1 struct]
    'TestClean/noZeroCheck(Data=clean)'   true    false    false    0.7077 [1×1 struct]
    'TestClean/noZeroCheck(Data=needsCleaning)' true    false    false    0.0093186 [1×1 struct]

```

在测试文件外部创建数据集。

```
A = [NaN 2 0;1 Inf 3];
```

使用外部数据集创建参数。`fromData` 方法接受来自 `TestClean` 中 `properties` 模块的参数化属性的名称，以及元胞数组（或结构体）形式的新数据。

```

import matlab.unittest.parameters.Parameter
newData = {A};
param = Parameter.fromData('Data',newData);

```

创建新测试套件并查看套件元素名称。`fromClass` 方法接受新参数。

```

suite2 = TestSuite.fromClass(?TestClean,'ExternalParameters',param);
{suite2.Name}

```

```
ans =
```

```
4×1 cell array
```

```
{"TestClean/classCheck(Data=value1#ext)" }
 {"TestClean/sortCheck(Data=value1#ext)" }
 {"TestClean/finiteCheck(Data=value1#ext)" }
 {"TestClean/noZeroCheck(Data=value1#ext)" }
```

该框架使用外部参数创建四个套件元素。由于参数定义为元胞数组，因此 MATLAB 生成参数名称 (`value1`)。此外，它还在参数名称的后面追加字符 `#ext`，表示该参数是在外部定义的。

要指定有意义的参数名称（而不是 `valueN`），请使用结构体定义参数。查看套件元素名称并运行测试。

```

newData = struct('commandLineData',A);
param = Parameter.fromData('Data',newData);
suite2 = TestSuite.fromClass(?TestClean,'ExternalParameters',param);
{suite2.Name}
results = suite2.run;

ans =
4×1 cell array

{'TestClean/classCheck(Data=commandLineData#ext)' }
{'TestClean/sortCheck(Data=commandLineData#ext)' }
{'TestClean/finiteCheck(Data=commandLineData#ext)'}
{'TestClean/noZeroCheck(Data=commandLineData#ext)'}

Running TestClean
.....
Done TestClean

```

---

再创建一个数据集，并将其存储在 ASCII 分隔的文件中。

```

B = rand(3);
B(2,4) = 0;
dlmwrite('myFile.dat',B)
clear B

```

使用存储的数据集和 A 创建参数。

```

newData = struct('commandLineData',A,'storedData',dlmread('myFile.dat'));
param2 = Parameter.fromData('Data',newData);
suite3 = TestSuite.fromClass(?TestClean,'ExternalParameters',param2);

```

要使用测试文件中定义的参数以及在外部定义的参数运行测试，请串联测试套件。查看套件元素名称并运行测试。

```

suite = [suite1 suite3];
{suite.Name}
results = suite.run;

```

```

ans =
16×1 cell array

{'TestClean/classCheck(Data=clean)'      }
{'TestClean/classCheck(Data=needsCleaning)'   }
{'TestClean/sortCheck(Data=clean)'      }
{'TestClean/sortCheck(Data=needsCleaning)'   }
{'TestClean/finiteCheck(Data=clean)'     }
{'TestClean/finiteCheck(Data=needsCleaning)'  }
{'TestClean/noZeroCheck(Data=clean)'    }
{'TestClean/noZeroCheck(Data=needsCleaning)'  }
{'TestClean/classCheck(Data=commandLineData#ext)' }
{'TestClean/classCheck(Data=storedData#ext)'   }
{'TestClean/sortCheck(Data=commandLineData#ext)' }
{'TestClean/sortCheck(Data=storedData#ext)'   }


```

```
{"TestClean/finiteCheck(Data=commandLineData#ext)"}
 {"TestClean/finiteCheck(Data=storedData#ext)"  }
 {"TestClean/noZeroCheck(Data=commandLineData#ext)"}
 {"TestClean/noZeroCheck(Data=storedData#ext)"  }
```

Running TestClean

.....  
Done TestClean

Running TestClean

.....  
Done TestClean

## 另请参阅

[matlab.unittest.TestSuite](#) | [matlab.unittest.parameters.Parameter.fromData](#)

## 相关示例

- “[创建基本参数化测试](#)” (第 34-60 页)
- “[创建高级参数化测试](#)” (第 34-64 页)

## 创建简单测试套件

此示例演示如何使用 SolverTest 测试用例将测试合并到测试套件中。使用 `matlab.unittest.TestSuite` 类中的静态 `from*` 方法为您的测试组合创建套件，无论它们是按数据包和类还是文件和文件夹（或同时按这两项）组织。

### 创建二次求解器函数

创建以下函数来对您的工作文件夹下的文件 `quadraticSolver.m` 中二次方程的根求解。

```
function roots = quadraticSolver(a, b, c)
% quadraticSolver returns solutions to the
% quadratic equation a*x^2 + b*x + c = 0.

if ~isa(a,'numeric') || ~isa(b,'numeric') || ~isa(c,'numeric')
    error('quadraticSolver:InputMustBeNumeric',...
        'Coefficients must be numeric.');
end

roots(1) = (-b + sqrt(b^2 - 4*a*c)) / (2*a);
roots(2) = (-b - sqrt(b^2 - 4*a*c)) / (2*a);

end
```

### 为二次求解器函数创建测试

在您的工作文件夹下的文件 `SolverTest.m` 中，创建以下测试类。

```
classdef SolverTest < matlab.unittest.TestCase
    % SolverTest tests solutions to the quadratic equation
    % a*x^2 + b*x + c = 0

    methods (Test)
        function testRealSolution(testCase)
            actSolution = quadraticSolver(1,-3,2);
            expSolution = [2,1];
            testCase.verifyEqual(actSolution,expSolution);
        end
        function testImaginarySolution(testCase)
            actSolution = quadraticSolver(1,2,10);
            expSolution = [-1+3i, -1-3i];
            testCase.verifyEqual(actSolution,expSolution);
        end
    end
end
```

### 导入 TestSuite 类

在命令提示符下，将 `matlab.unittest.TestSuite` 类添加到当前导入列表。

```
import matlab.unittest.TestSuite
```

确保 `SolverTest` 类定义文件在您的 MATLAB 路径上。

### 从 SolverTest 类创建套件

`fromClass` 方法从 `SolverTest` 类中的所有 `Test` 方法创建一个套件。

```
suiteClass = TestSuite.fromClass(?SolverTest);
result = run(suiteClass);
```

### 从 SolverTest 类定义文件创建套件

**fromFile** 方法使用标识该类的文件的名称创建一个套件。

```
suiteFile = TestSuite.fromFile('SolverTest.m');
result = run(suiteFile);
```

### 从当前文件夹中的所有测试用例文件创建套件

**fromFolder** 方法从指定文件夹中的所有测试用例文件创建一个套件。例如，以下文件在当前文件夹中：

- BankAccountTest.m
- DocPolynomTest.m
- FigurePropertiesTest.m
- IsSupportedTest.m
- SolverTest.m

```
suiteFolder = TestSuite.fromFolder(pwd);
result = run(suiteFolder);
```

### 从单个测试方法创建套件

**fromMethod** 方法从单个测试方法创建套件。

```
suiteMethod = TestSuite.fromMethod(?SolverTest,'testRealSolution')
result = run(suiteMethod);
```

## 另请参阅

TestSuite

## 相关示例

- “使用类来编写简单测试用例”（第 34-37 页）

## 为各个工作流运行测试

### 本节内容

- “设置示例测试”（第 34-76 页）
- “在类或函数中运行所有测试”（第 34-76 页）
- “在类或函数中运行单个测试”（第 34-76 页）
- “按名称运行测试套件”（第 34-77 页）
- “从测试数组运行测试套件”（第 34-77 页）
- “使用自定义测试运行程序运行测试”（第 34-78 页）

### 设置示例测试

要了解运行测试的不同方法，请在您的当前工作文件夹中创建一个基于类的测试和一个基于函数的测试。对于基于类的测试文件，请使用 `matlab.unittest.qualifications.Verifiable` 示例中提供的 `DocPolynomTest` 示例测试。对于基于函数的测试文件，请使用“使用设置和拆解函数编写测试”（第 34-26 页）示例中提供的 `axesPropertiesTest` 示例测试。

### 在类或函数中运行所有测试

使用 `TestCase` 类的 `run` 方法可直接运行包含在单个测试文件中的多个测试。当直接运行测试时，您无需显式创建 `Test` 数组。

```
% Directly run a single file of class-based tests
results1 = run(DocPolynomTest);

% Directly run a single file of function-based tests
results2 = run(axesPropertiesTest);
```

您还可以将测试文件输出分配给变量并使用函数形式或圆点表示法运行测试。

```
% Create Test or TestCase objects
t1 = DocPolynomTest; % TestCase object from class-based test
t2 = axesPropertiesTest; % Test object from function-based test

% Run tests using functional form
results1 = run(t1);
results2 = run(t2);

% Run tests using dot notation
results1 = t1.run;
results2 = t2.run;
```

您也可以使用 `runtests` 或从编辑器中运行包含在单个文件中的多个测试。

### 在类或函数中运行单个测试

通过将测试方法指定为 `run` 方法的输入参数从基于类的测试文件中运行单个测试。例如，仅从文件 `DocPolynomTest` 运行测试 `testMultiplication`。

```
results1 = run(DocPolynomTest,'testMultiplication');
```

基于函数的测试文件返回 `Test` 对象数组而非单个 `TestCase` 对象。您可以通过对该数组进行索引来运行特定测试。不过，您必须检查测试数组中的 `Name` 字段以确保您运行正确的测试。例如，仅从文件 `axesPropertiesTest` 运行测试 `surfaceColorTest`。

```
t2 = axesPropertiesTest; % Test object from function-based test
t2(:).Name
```

`ans =`

```
axesPropertiesTest/testDefaultXLim
```

`ans =`

```
axesPropertiesTest/surfaceColorTest
```

`surfaceColorTest` 测试对应于数组中的第二个元素。

仅运行 `surfaceColorTest` 测试。

```
results2 = t2(2).run; % or results2 = run(t2(2));
```

您也可以从编辑器中运行单个测试。

## 按名称运行测试套件

您可以一起运行一组或一套测试。要使用 `runtests` 运行测试套件，该套件应定义为一个字符串元胞数组，表示测试文件、测试类、包含测试的包或包含测试的文件夹。

```
suite = {'axesPropertiesTest','DocPolynomTest'};
runtests(suite);
```

使用 `pwd` 作为 `runtests` 函数的输入运行当前文件夹中的所有测试。

```
runtests(pwd);
```

您也可以显式创建 `Test` 数组并使用 `run` 方法运行它们。

## 从测试数组运行测试套件

您可以显式创建 `Test` 数组并使用 `TestSuite` 类中的 `run` 方法运行它们。使用此方法，您可以显式定义 `TestSuite` 对象，因此可检查内容。`runtests` 函数不返回 `TestSuite` 对象。

```
import matlab.unittest.TestSuite
s1 = TestSuite.fromClass('DocPolynomTest');
s2 = TestSuite.fromFile('axesPropertiesTest.m');

% generate test suite and then run
fullSuite = [s1 s2];
result = run(fullSuite);
```

因为该套件是显式定义的，所以您可以轻松地对它执行更多分析，例如重新运行失败的测试。

```
failedTests = fullSuite([result.Failed]);
result2 = run(failedTests);
```

## 使用自定义测试运行程序运行测试

您可以通过定义自定义测试运行程序并添加插件来详细说明测试。TestRunner 类的 run 方法作用于 TestSuite 对象。

```
import matlab.unittest.TestRunner
import matlab.unittest.TestSuite
import matlab.unittest.plugins.TestRunProgressPlugin

% Generate TestSuite.
s1 = TestSuite.fromClass(?DocPolynomTest);
s2 = TestSuite.fromFile('axesPropertiesTest.m');
suite = [s1 s2];

% Create silent test runner.
runner = TestRunner.withNoPlugins;

% Add plugin to display test progress.
runner.addPlugin(TestRunProgressPlugin.withVerbosity(2))

% Run tests using customized runner.
result = run(runner,[suite]);
```

### 另请参阅

run | run | run | runtests

### 详细信息

- “在编辑器中运行测试” (第 34-16 页)

# 以编程方式访问测试诊断

如果您使用 `runtests` 函数或者 `TestSuite` 或 `TestCase` 的 `run` 方法运行测试，则测试框架会使用 `DiagnosticsRecordingPlugin` 插件记录测试结果的诊断信息。

运行测试后，您可以通过 `TestResult` 上 `Details` 属性中的 `DiagnosticRecord` 字段访问记录的诊断数据。例如，如果您的测试结果存储在变量 `results` 中，可通过调用 `records = result(2).Details.DiagnosticRecord` 来查找套件中第二个测试记录的诊断信息。

记录的诊断信息是 `DiagnosticRecord` 对象。要访问特定测试的特定类型测试诊断信息，请使用 `DiagnosticRecord` 类的 `selectFailed`、`selectPassed`、`selectIncomplete` 和 `selectLogged` 方法。

默认情况下，`DiagnosticsRecordingPlugin` 插件会记录在 `Terse` 级别记录的验证故障和事件。要配置此插件以记录传递诊断信息或在不同详细级别记录的其他消息，请配置 `DiagnosticsRecordingPlugin` 实例并将其添加到测试运行程序中。

## 另请参阅

`matlab.unittest.TestResult` | `matlab.unittest.plugins.DiagnosticsRecordingPlugin` |  
`matlab.unittest.plugins.diagnosticrecord.DiagnosticRecord`

## 相关示例

- “向测试运行程序添加插件”（第 34-80 页）

## 向测试运行程序添加插件

此示例演示如何向测试运行程序添加插件。`matlab.unittest.plugins.TestRunProgressPlugin` 显示有关测试用例的进度消息。此插件是 `matlab.unittest` 包的一部分。MATLAB® 将其用于默认测试运行程序。

### 为 BankAccount 类创建测试

在您的工作文件夹下的文件中，为 `BankAccount` 类创建一个测试文件。

```
type BankAccountTest.m

classdef BankAccountTest < matlab.unittest.TestCase
    % Tests the BankAccount class.

    methods (TestClassSetup)
        function addBankAccountClassToPath(testCase)
            p = path;
            testCase.addTeardown(@path,p);
            addpath(fullfile(matlabroot,'help','techdoc','matlab_oop',...
                'examples'));
        end
    end

    methods (Test)
        function testConstructor(testCase)
            b = BankAccount(1234, 100);
            testCase.verifyEqual(b.AccountNumber, 1234, ...
                'Constructor failed to correctly set account number');
            testCase.verifyEqual(b.AccountBalance, 100, ...
                'Constructor failed to correctly set account balance');
        end

        function testConstructorNotEnoughInputs(testCase)
            import matlab.unittest.constraints.Throws;
            testCase.verifyThat(@()BankAccount, ...
                Throws('MATLAB:minrhs'));
        end

        function testDeposit(testCase)
            b = BankAccount(1234, 100);
            b.deposit(25);
            testCase.verifyEqual(b.AccountBalance, 125);
        end

        function testWithdraw(testCase)
            b = BankAccount(1234, 100);
            b.withdraw(25);
            testCase.verifyEqual(b.AccountBalance, 75);
        end

        function testNotifyInsufficientFunds(testCase)
            callbackExecuted = false;
            function testCallback(~,~)
                callbackExecuted = true;
            end
        end
    end
end
```

```

b = BankAccount(1234, 100);
b.addlistener('InsufficientFunds', @testCallback);

b.withdraw(50);
testCase.assertFalse(callbackExecuted, ...
    'The callback should not have executed yet');
b.withdraw(60);
testCase.assertTrue(callbackExecuted, ...
    'The listener callback should have fired');

end
end
end

```

### 创建测试套件

在命令提示符处，基于 `BankAccountTest` 测试用例创建一个测试套件 `ts`。

```
ts = matlab.unittest.TestSuite.fromClass(?BankAccountTest);
```

### 显示不含任何插件的结果

创建一个不含任何插件的测试运行程序。

```
runner = matlab.unittest.TestRunner.withNoPlugins;
res = runner.run(ts);
```

不显示任何输出。

### 自定义测试运行程序

添加自定义插件 `TestRunProgressPlugin`。

```
import matlab.unittest.plugins.TestRunProgressPlugin
runner.addPlugin(TestRunProgressPlugin.withVerbosity(2))
res = runner.run(ts);
```

```
Running BankAccountTest
.....
Done BankAccountTest
```

---

MATLAB 显示有关 `BankAccountTest` 的进度消息。

### 另请参阅

`matlab.unittest.plugins`

## 编写插件以扩展 TestRunner

### 本节内容

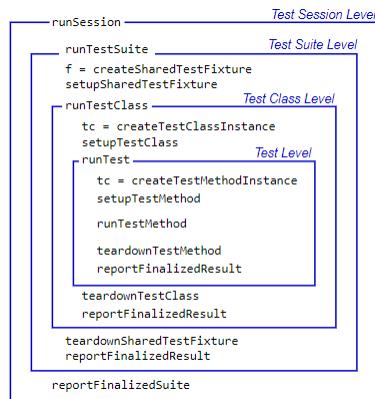
- “自定义插件概述”（第 34-82 页）
- “扩展测试会话级别插件方法”（第 34-82 页）
- “扩展测试套件级别插件方法”（第 34-83 页）
- “扩展测试类级别插件方法”（第 34-83 页）
- “扩展测试级别插件方法”（第 34-83 页）

### 自定义插件概述

**TestRunnerPlugin** 方法有四个级别：测试会话、测试套件、测试类和测试。在每个级别，您都可以实现方法来扩展测试运行。此外，您可以在测试套件、测试类和测试级别实现方法，以扩展测试或测试脚手架的创建、设置和拆解。

在测试套件、测试类和测试级别，**TestRunner** 使用 **reportFinalizedResult** 方法报告最终测试结果。当没有其余的测试内容可以修改测试结果时，表示测试结果已最终确定。**TestRunner** 会确定它是否在每个级别都调用 **reportFinalizedResult** 方法。在测试会话级别，**TestRunner** 使用 **reportFinalizedSuite** 方法在测试套件最终确定后报告测试结果。

**TestRunner** 运行不同方法，如下图所示。



创建方法是唯一包含输出参数的 **TestRunnerPlugin** 方法集。通常，可扩展创建方法以在对应级别侦听源自测试内容的各种事件。因为 **TestCase** 和 **Fixture** 实例均继承自 **handle** 类，所以可以使用 **addlistener** 方法添加侦听程序。设置、运行和拆解测试内容的方法可扩展 **TestRunner** 评估测试内容的方式。

### 扩展测试会话级别插件方法

测试会话级别的 **TestRunnerPlugin** 方法可扩展传递给 **TestRunner** 的测试套件的运行和报告。下列方法属于 **runSession** 方法的范围。

此级别的运行方法 **runTestSuite** 可扩展测试框架传递给 **TestRunner** 的整个 **TestSuite** 数组的一部分的运行。**reportFinalizedSuite** 方法可扩展 **runTestSuite** 最终确定的测试套件的报告。

## 扩展测试套件级别插件方法

测试套件级别的 TestRunnerPlugin 方法扩展共享测试脚手架的创建、设置、运行和拆解。下列方法属于 runTestSuite 方法的范围。

方法类型	测试级别处于 runTestSuite 范围内
创建方法	<code>createSharedTestFixture</code>
设置方法	<code>setupSharedTestFixture</code>
运行方法	<code>runTestClass</code>
拆解方法	<code>teardownSharedTestFixture</code>

在此级别，`createSharedTestFixture` 方法是唯一带输出参数的插件方法。它为测试类所需的每个共享脚手架返回 `Fixture` 实例。这些脚手架实例可通过 `TestCase` 的 `getSharedTestFixtures` 方法提供给测试。

此级别的运行方法 `runTestClass` 扩展属于同一测试类的测试或同一基于函数的测试的运行，并纳入为测试类级别插件方法描述的功能。

## 扩展测试类级别插件方法

测试类级别的 TestRunnerPlugin 方法可扩展属于同一测试类或同一基于函数的测试的测试套件元素的创建、设置、运行和拆解。这些方法适用于 TestRunner 运行的整个 TestSuite 数组的一部分。

方法类型	测试级别处于 runTestClass 范围内
创建方法	<code>createClassInstance</code>
设置方法	<code>setupTestClass</code>
运行方法	<code>runTest</code>
拆解方法	<code>teardownTestClass</code>

在此级别，`createClassInstance` 方法是唯一带输出参数的插件方法。它返回在类级别创建的 `TestCase` 实例。对于每个类，测试框架向任何带有 `TestClassSetup` 或 `TestClassTeardown` 属性的方法传递该实例。

如果某测试类设置包含带 `ClassSetupParameter` 特性的属性，则它已被参数化。本例中，测试框架按照类设置参数化的规定评估 `setupTestClass` 和 `teardownTestClass` 方法很多次。

此级别的运行方法 `runTest` 可扩展单个 TestSuite 元素的运行，并纳入为测试级别插件方法描述的功能。

测试框架在 `runTestClass` 方法范围内对测试类级别的方法进行计算。如果 `TestClassSetup` 代码成功完成，则它为 `TestSuite` 数组中的每个元素调用 `runTest` 方法一次。每个 `TestClassSetup` 参数化调用创建、设置和拆解方法一次。

## 扩展测试级别插件方法

测试级别的 TestRunnerPlugin 方法可扩展单个测试套件元素的创建、设置、运行和拆解。单个 Test 元素包括一个测试方法，如果测试被参数化，则包括测试参数化的一个实例。

方法类型	测试级别处于 runTest 范围内
创建方法	<code>createTestMethodInstance</code>
设置方法	<code>setupTestMethod</code>
运行方法	<code>runTestMethod</code>
拆解方法	<code>teardownTestMethod</code>

在此级别，`createTestMethodInstance` 方法是唯一带输出参数的插件方法。它返回为每个 Test 元素创建的 TestCase 实例。测试框架将每个实例传入相应的 Test 方法，以及任何带有 `TestMethodSetup` 或 `TestMethodTeardown` 属性的方法。

测试框架在 `runTest` 方法范围内对测试级别的方法进行计算。框架完成所有 `TestMethodSetup` 工作后，它为每个 Test 元素调用此级别的插件方法一次。

## 另请参阅

`addlistener` | `matlab.unittest.TestCase` | `matlab.unittest.TestRunner` |  
`matlab.unittest.TestSuite` | `matlab.unittest.fixtures.Fixture` |  
`matlab.unittest.plugins.OutputStream` | `matlab.unittest.plugins.Parallelizable` |  
`matlab.unittest.plugins.TestRunnerPlugin`

## 相关示例

- “创建自定义插件”（第 34-85 页）
- “使用自定义插件并行运行测试”（第 34-90 页）
- “用于生成自定义测试输出格式的插件”（第 34-102 页）
- “编写用于保存诊断详细信息的插件”（第 34-98 页）

# 创建自定义插件

此示例演示如何创建一个自定义插件，以便统计在 `TestRunner` 运行测试套件时的通过和失败断言数。该插件在测试结束时输出简短摘要。为了扩展 `TestRunner`，该插件会创建 `matlab.unittest.plugins.TestRunnerPlugin` 类的子类并覆盖所选方法。

## 创建 AssertionCountingPlugin 类

在当前文件夹中的文件中，创建自定义插件类 `AssertionCountingPlugin`，该类继承自 `TestRunnerPlugin` 类。有关 `AssertionCountingPlugin` 的完整代码，请参阅 `AssertionCountingPlugin` 类定义总结（第 34-0 页）。

要跟踪通过和失败断言数，请编写 `properties` 代码块以定义两个只读属性 `NumPassingAssertions` 和 `NumFailingAssertions`。

```
properties (SetAccess = private)
    NumPassingAssertions
    NumFailingAssertions
end
```

## 扩展 TestSuite 的运行

编写具有 `protected` 访问权限的 `methods` 代码块以实现 `runTestSuite` 方法。

```
methods (Access = protected)
    function runTestSuite(plugin, pluginData)
        suiteSize = numel(pluginData.TestSuite);
        fprintf('## Running a total of %d tests\n', suiteSize)

        plugin.NumPassingAssertions = 0;
        plugin.NumFailingAssertions = 0;

        runTestSuite@matlab.unittest.plugins.TestRunnerPlugin(...%
            plugin, pluginData);

        fprintf('## Done running tests\n')
        plugin.printAssertionSummary()
    end
end
```

测试框架对此方法进行一次计算。它显示关于测试总数的信息，初始化插件用于生成文本输出的属性，并调用超类方法。在框架完成对超类方法的计算后，`runTestSuite` 方法通过调用辅助函数方法 `printAssertionSummary`（请参阅 `printAssertionSummary` 定义（第 34-0 页））来显示断言计数摘要。

## 扩展共享测试脚手架和 TestCase 实例的创建

将侦听程序添加到 `AssertionPassed` 和 `AssertionFailed` 事件以统计断言数。要添加这些侦听程序，请扩展测试框架用于创建测试内容的方法。测试内容包括每个 `Test` 元素的 `TestCase` 实例，`TestClassSetup` 和 `TestClassTeardown` 方法的类级别 `TestCase` 实例，以及在 `TestCase` 类具有 `SharedTestFixture` 属性时使用的 `Fixture` 实例。

在覆盖创建方法时调用相应的超类方法。这些创建方法返回测试框架针对其各自上下文创建的内容。当使用 `incrementPassingAssertionsCount` 和 `incrementFailingAssertionsCount` 辅助函数方法（第 34-0 页）实现上述方法时，将插件所需的侦听程序添加到返回的 `Fixture` 或 `TestCase` 实例。

将这些创建方法添加到具有 `protected` 访问权限的 `methods` 代码块。

```

methods (Access = protected)
  function fixture = createSharedTestFixture(plugin, pluginData)
    fixture = createSharedTestFixture@...
    matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

    fixture.addlistener('AssertionPassed', ...
      @(~,~)plugin.incrementPassingAssertionsCount);
    fixture.addlistener('AssertionFailed', ...
      @(~,~)plugin.incrementFailingAssertionsCount);
  end

  function testCase = createTestClassInstance(plugin, pluginData)
    testCase = createTestClassInstance@...
    matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

    testCase.addlistener('AssertionPassed', ...
      @(~,~)plugin.incrementPassingAssertionsCount);
    testCase.addlistener('AssertionFailed', ...
      @(~,~)plugin.incrementFailingAssertionsCount);
  end

  function testCase = createTestMethodInstance(plugin, pluginData)
    testCase = createTestMethodInstance@...
    matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

    testCase.addlistener('AssertionPassed', ...
      @(~,~)plugin.incrementPassingAssertionsCount);
    testCase.addlistener('AssertionFailed', ...
      @(~,~)plugin.incrementFailingAssertionsCount);
  end
end

```

### 扩展单个测试套件元素的运行

扩展 `runTest` 以便在运行时显示每个测试的名称。将此方法添加到具有 `protected` 访问权限的 `methods` 代码块。像所有插件方法一样，`runTest` 方法要求您调用对应的超类方法。

```

methods (Access = protected)
  function runTest(plugin, pluginData)
    fprintf('## Running test: %s\n', pluginData.Name)

    runTest@matlab.unittest.plugins.TestRunnerPlugin(...,
      plugin, pluginData);
  end
end

```

### 定义辅助函数方法

在具有 `private` 访问权限的 `methods` 块中，定义三个辅助函数方法。这些方法累加通过或失败断言数，并输出断言计数摘要。

```

methods (Access = private)
  function incrementPassingAssertionsCount(plugin)
    plugin.NumPassingAssertions = plugin.NumPassingAssertions + 1;
  end

  function incrementFailingAssertionsCount(plugin)
    plugin.NumFailingAssertions = plugin.NumFailingAssertions + 1;
  end

```

```

end

function printAssertionSummary(plugin)
    fprintf('%s\n', repmat('_', 1, 30))
    fprintf('Total Assertions: %d\n', ...
        plugin.NumPassingAssertions + plugin.NumFailingAssertions)
    fprintf('\t%d Passed, %d Failed\n', ...
        plugin.NumPassingAssertions, plugin.NumFailingAssertions)
end
end

```

### AssertionCountingPlugin 类定义总结

以下代码提供 AssertionCountingPlugin 的完整内容。

```

classdef AssertionCountingPlugin < ...
    matlab.unittest.plugins.TestRunnerPlugin

    properties (SetAccess = private)
        NumPassingAssertions
        NumFailingAssertions
    end

    methods (Access = protected)
        function runTestSuite(plugin, pluginData)
            suiteSize = numel(pluginData.TestSuite);
            fprintf("## Running a total of %d tests\n", suiteSize)

            plugin.NumPassingAssertions = 0;
            plugin.NumFailingAssertions = 0;

            runTestSuite@matlab.unittest.plugins.TestRunnerPlugin(...%
                plugin, pluginData);

            fprintf("## Done running tests\n")
            plugin.printAssertionSummary()
        end

        function fixture = createSharedTestFixture(plugin, pluginData)
            fixture = createSharedTestFixture@...
                matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

            fixture.addlistener('AssertionPassed', ...
                @(~,~)plugin.incrementPassingAssertionsCount);
            fixture.addlistener('AssertionFailed', ...
                @(~,~)plugin.incrementFailingAssertionsCount);
        end

        function testCase = createTestClassInstance(plugin, pluginData)
            testCase = createTestClassInstance@...
                matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

            testCase.addlistener('AssertionPassed', ...
                @(~,~)plugin.incrementPassingAssertionsCount);
            testCase.addlistener('AssertionFailed', ...
                @(~,~)plugin.incrementFailingAssertionsCount);
        end

        function testCase = createTestMethodInstance(plugin, pluginData)
            testCase = createTestMethodInstance@...
                matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

            testCase.addlistener('AssertionPassed', ...
                @(~,~)plugin.incrementPassingAssertionsCount);
            testCase.addlistener('AssertionFailed', ...
                @(~,~)plugin.incrementFailingAssertionsCount);
        end

        function runTest(plugin, pluginData)

```

```

fprintf("## Running test: %s\n", pluginData.Name)

runTest@matlab.unittest.plugins.TestRunnerPlugin...
    plugin, pluginData);
end
end

methods (Access = private)
    function incrementPassingAssertionsCount(plugin)
        plugin.NumPassingAssertions = plugin.NumPassingAssertions + 1;
    end

    function incrementFailingAssertionsCount(plugin)
        plugin.NumFailingAssertions = plugin.NumFailingAssertions + 1;
    end

    function printAssertionSummary(plugin)
        fprintf("%s\n", repmat(' ', 1, 30))
        fprintf('Total Assertions: %d\n', ...
            plugin.NumPassingAssertions + plugin.NumFailingAssertions)
        fprintf('\t%d Passed, %d Failed\n', ...
            plugin.NumPassingAssertions, plugin.NumFailingAssertions)
    end
end
end

```

### 创建示例测试类

在当前文件夹中，创建一个名为 `ExampleTest.m` 的文件，其中包含以下测试类。

```

classdef ExampleTest < matlab.unittest.TestCase
    methods(Test)
        function testOne(testCase) % Test fails
            testCase.assertEqual(5, 4)
        end
        function testTwo(testCase) % Test passes
            testCase.verifyEqual(5, 5)
        end
        function testThree(testCase) % Test passes
            testCase.assertEqual(7*2, 14)
        end
    end
end

```

### 将插件添加到 TestRunner 并运行测试

在命令提示符下，基于 `ExampleTest` 类创建测试套件。

```

import matlab.unittest.TestSuite
import matlab.unittest.TestRunner

suite = TestSuite.fromClass(?ExampleTest);

```

创建一个不含插件的 `TestRunner` 实例。此代码创建一个静默运行程序，以便您控制安装的插件。

```
runner = TestRunner.withNoPlugins;
```

运行测试。

```
result = runner.run(suite);
```

将 `AssertionCountingPlugin` 添加到运行程序并运行测试。

```
runner.addPlugin(AssertionCountingPlugin)
result = runner.run(suite);

## Running a total of 3 tests
### Running test: ExampleTest/testOne
### Running test: ExampleTest/testTwo
### Running test: ExampleTest/testThree
## Done running tests

Total Assertions: 2
  1 Passed, 1 Failed
```

## 另请参阅

[addlistener](#) | [matlab.unittest.TestCase](#) | [matlab.unittest.TestRunner](#) |  
[matlab.unittest.fixtures.Fixture](#) | [matlab.unittest.plugins.OutputStream](#) |  
[matlab.unittest.plugins.TestRunnerPlugin](#)

## 相关示例

- “编写插件以扩展 TestRunner” (第 34-82 页)
- “使用自定义插件并行运行测试” (第 34-90 页)
- “编写用于保存诊断详细信息的插件” (第 34-98 页)

## 使用自定义插件并行运行测试

此示例说明如何创建一个支持并行运行测试的自定义插件。该自定义插件对测试套件的通过和失败断言数进行计数。为了扩展 `TestRunner`, 该插件会覆盖 `matlab.unittest.plugins.TestRunnerPlugin` 类的所选方法。此外, 为了支持并行运行测试, 该插件会创建 `matlab.unittest.plugins.Parallelizable` 接口的子类。要并行运行测试, 您需要 Parallel Computing Toolbox。

### 创建插件类

在当前文件夹中的文件中, 创建可并行化的插件类 `AssertionCountingPlugin`, 它同时从继承自 `TestRunnerPlugin` 和 `Parallelizable` 类。有关 `AssertionCountingPlugin` 的完整代码, 请参阅插件类定义总结 (第 34-0 页)。

要跟踪通过和失败的断言数, 请在 `properties` 块中定义四个只读属性。当前并行池中的每个 MATLAB 工作进程使用 `NumPassingAssertions` 和 `NumFailingAssertions` 来跟踪运行 `TestSuite` 数组的一部分时的通过和失败断言数。MATLAB 客户端使用 `FinalizedNumPassingAssertions` 和 `FinalizedNumFailingAssertions` 汇总来自不同工作进程的结果, 并在测试会话结束时报告通过和失败断言的总数。

```
properties (SetAccess = private)
    NumPassingAssertions
    NumFailingAssertions
    FinalizedNumPassingAssertions
    FinalizedNumFailingAssertions
end
```

### 扩展测试会话的运行

要扩展整个 `TestSuite` 数组的运行, 请在具有 `protected` 访问权限的 `methods` 块中覆盖 `TestRunnerPlugin` 的 `runSession` 方法。测试框架在客户端对此方法进行一次计算。

```
methods (Access = protected)
function runSession(plugin, pluginData)
    suiteSize = numel(pluginData.TestSuite);
    fprintf('## Running a total of %d tests\n', suiteSize);
    plugin.FinalizedNumPassingAssertions = 0;
    plugin.FinalizedNumFailingAssertions = 0;

    runSession@matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

    fprintf('## Done running tests\n')
    plugin.printAssertionSummary()
end
end
```

`runSession` 显示关于 `Test` 元素总数的信息, 初始化插件用于生成文本输出的属性, 并调用超类方法来触发整个测试运行。在框架完成对超类方法的计算后, `runSession` 通过调用辅助函数方法 `printAssertionSummary` (请参阅定义辅助函数方法 (第 34-0 页)) 显示断言计数摘要。

### 扩展共享测试脚手架和 `TestCase` 实例的创建

将侦听程序添加到 `AssertionPassed` 和 `AssertionFailed` 事件以统计断言数。要添加这些侦听程序, 请扩展测试框架用于创建测试内容的方法。测试内容包括每个 `Test` 元素的 `TestCase` 实例, `TestClassSetup` 和 `TestClassTeardown` 方法代码块的类级别 `TestCase` 实例, 以及当 `TestCase` 类具有 `SharedTestFixture` 属性时使用的 `Fixture` 实例。

在覆盖创建方法时调用相应的超类方法。这些创建方法返回测试框架针对其各自上下文创建的内容。当使用 `incrementPassingAssertionsCount` 和 `incrementFailingAssertionsCount` 辅助函数方法（第 34-0 页）实现上述方法时，将插件所需的侦听程序添加到返回的 `Fixture` 或 `TestCase` 实例。

将这些创建方法添加到具有 `protected` 访问权限的 `methods` 代码块。

```
methods (Access = protected)
function fixture = createSharedTestFixture(plugin, pluginData)
    fixture = createSharedTestFixture@...
    matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

    fixture.addlistener('AssertionPassed', ...
        @(~,~)plugin.incrementPassingAssertionsCount);
    fixture.addlistener('AssertionFailed', ...
        @(~,~)plugin.incrementFailingAssertionsCount);
end

function testCase = createTestClassInstance(plugin, pluginData)
    testCase = createTestClassInstance@...
    matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

    testCase.addlistener('AssertionPassed', ...
        @(~,~)plugin.incrementPassingAssertionsCount);
    testCase.addlistener('AssertionFailed', ...
        @(~,~)plugin.incrementFailingAssertionsCount);
end

function testCase = createTestMethodInstance(plugin, pluginData)
    testCase = createTestMethodInstance@...
    matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

    testCase.addlistener('AssertionPassed', ...
        @(~,~)plugin.incrementPassingAssertionsCount);
    testCase.addlistener('AssertionFailed', ...
        @(~,~)plugin.incrementFailingAssertionsCount);
end
end
```

### 扩展测试套件部分的运行

测试框架将整个 `TestSuite` 数组分成不同的组，并将它们分配给工作进程进行处理。每个工作进程可以运行一个或多个测试套件部分。要自定义工作进程的行为，请在具有 `protected` 访问权限的 `methods` 块中覆盖 `TestRunnerPlugin` 的 `runTestSuite` 方法。

扩展 `TestRunner` 以显示单个工作进程运行的每个测试组的标识符以及该组中 `Test` 元素的数量。此外，将通过和失败断言数存储在一个缓冲区中，以便客户端可以检索这些值来产生最终结果。像所有插件方法一样，`runTestSuite` 方法要求您在适当的时机调用对应的超类方法。在本例中，请在初始化属性之后和存储工作进程数据之前调用超类方法。测试框架在工作进程上计算 `runTestSuite` 的次数等于测试套件部分的个数。

```
methods (Access = protected)
function runTestSuite(plugin, pluginData)
    suiteSize = numel(pluginData.TestSuite);
    groupNumber = pluginData.Group;
    fprintf('## Running a total of %d tests in group %d\n', ...
        suiteSize, groupNumber);
    plugin.NumPassingAssertions = 0;
```

```

plugin.NumFailingAssertions = 0;

runTestSuite@matlab.unittest.plugins.TestRunnerPlugin(
    plugin, pluginData);

assertionStruct = struct('Passing', plugin.NumPassingAssertions, ...
    'Failing', plugin.NumFailingAssertions);
plugin.storeIn(pluginData.CommunicationBuffer, assertionStruct);
end
end

```

为了存储特定于测试的数据，`runTestSuite` 的实现包含对 `Parallelizable` 接口的 `storeIn` 方法的调用。当工作进程必须向客户端报告时，请使用 `storeIn` 和 `retrieveFrom`。在此示例中，在从超类方法返回后，`NumPassingAssertions` 和 `NumFailingAssertions` 包含对应于一组测试的通过和失败断言数。由于 `storeIn` 只以一个输入参数的形式接受工作进程数据，因此 `assertionStruct` 使用两个字段对断言计数进行分组。

### 扩展最终测试套件部分的报告

扩展 `reportFinalizedSuite` 以通过检索每个最终测试套件部分的测试数据来汇总断言计数。要检索为一个测试套件部分存储的 `assertionStruct`，请在 `reportFinalizedSuite` 的范围内调用 `retrieveFrom` 方法。将字段值添加到对应的类属性，并调用超类方法。测试框架在客户端上计算此方法的次数等于测试套件部分的个数。

```

methods (Access = protected)
    function reportFinalizedSuite(plugin, pluginData)
        assertionStruct = plugin.retrieveFrom(pluginData.CommunicationBuffer);
        plugin.FinalizedNumPassingAssertions = ...
            plugin.FinalizedNumPassingAssertions + assertionStruct.Passing;
        plugin.FinalizedNumFailingAssertions = ...
            plugin.FinalizedNumFailingAssertions + assertionStruct.Failing;

        reportFinalizedSuite@matlab.unittest.plugins.TestRunnerPlugin(
            plugin, pluginData);
    end
end

```

### 定义辅助函数方法

在具有 `private` 访问权限的 `methods` 块中，定义三个辅助函数方法。这些方法累加每个运行测试套件部分中的通过或失败断言数，并输出断言计数摘要。

```

methods (Access = private)
    function incrementPassingAssertionsCount(plugin)
        plugin.NumPassingAssertions = plugin.NumPassingAssertions + 1;
    end

    function incrementFailingAssertionsCount(plugin)
        plugin.NumFailingAssertions = plugin.NumFailingAssertions + 1;
    end

    function printAssertionSummary(plugin)
        fprintf('%s\n', repmat('_', 1, 30))
        fprintf('Total Assertions: %d\n', plugin.FinalizedNumPassingAssertions + ...
            plugin.FinalizedNumFailingAssertions)
        fprintf('\t%d Passed, %d Failed\n', plugin.FinalizedNumPassingAssertions, ...
            plugin.FinalizedNumFailingAssertions)
    end

```

```
    end
end
```

## 插件类定义总结

以下代码提供 AssertionCountingPlugin 的完整内容。

```
classdef AssertionCountingPlugin < ...
    matlab.unittest.plugins.TestRunnerPlugin & ...
    matlab.unittest.plugins.Parallelizable

    properties (SetAccess = private)
        NumPassingAssertions
        NumFailingAssertions
        FinalizedNumPassingAssertions
        FinalizedNumFailingAssertions
    end

    methods (Access = protected)
        function runSession(plugin, pluginData)
            suiteSize = numel(pluginData.TestSuite);
            fprintf("## Running a total of %d tests\n\n", suiteSize);
            plugin.FinalizedNumPassingAssertions = 0;
            plugin.FinalizedNumFailingAssertions = 0;

            runSession@matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

            fprintf("## Done running tests\n")
            plugin.printAssertionSummary()
        end

        function fixture = createSharedTestFixture(plugin, pluginData)
            fixture = createSharedTestFixture@...
                matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

            fixture.addlistener('AssertionPassed', ...
                @(~,~)plugin.incrementPassingAssertionsCount);
            fixture.addlistener('AssertionFailed', ...
                @(~,~)plugin.incrementFailingAssertionsCount);
        end

        function testCase = createTestClassInstance(plugin, pluginData)
            testCase = createTestClassInstance@...
                matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

            testCase.addlistener('AssertionPassed', ...
                @(~,~)plugin.incrementPassingAssertionsCount);
            testCase.addlistener('AssertionFailed', ...
                @(~,~)plugin.incrementFailingAssertionsCount);
        end

        function testCase = createTestMethodInstance(plugin, pluginData)
            testCase = createTestMethodInstance@...
                matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

            testCase.addlistener('AssertionPassed', ...
                @(~,~)plugin.incrementPassingAssertionsCount);
            testCase.addlistener('AssertionFailed', ...
                @(~,~)plugin.incrementFailingAssertionsCount);
        end

        function runTestSuite(plugin, pluginData)
            suiteSize = numel(pluginData.TestSuite);
            groupNumber = pluginData.Group;
            fprintf("## Running a total of %d tests in group %d\n", ...
                suiteSize, groupNumber);
            plugin.NumPassingAssertions = 0;
            plugin.NumFailingAssertions = 0;

            runTestSuite@matlab.unittest.plugins.TestRunnerPlugin(...
                plugin, pluginData);
        end
    end
end
```

```

assertionStruct = struct('Passing', plugin.NumPassingAssertions, ...
    'Failing', plugin.NumFailingAssertions);
plugin.storeIn(pluginData.CommunicationBuffer, assertionStruct);
end

function reportFinalizedSuite(plugin, pluginData)
    assertionStruct = plugin.retrieveFrom(pluginData.CommunicationBuffer);
    plugin.FinalizedNumPassingAssertions = ...
        plugin.FinalizedNumPassingAssertions + assertionStruct.Passing;
    plugin.FinalizedNumFailingAssertions = ...
        plugin.FinalizedNumFailingAssertions + assertionStruct.Failing;

    reportFinalizedSuite@matlab.unittest.plugins.TestRunnerPlugin(... ...
        plugin, pluginData);
end
end

methods (Access = private)
    function incrementPassingAssertionsCount(plugin)
        plugin.NumPassingAssertions = plugin.NumPassingAssertions + 1;
    end

    function incrementFailingAssertionsCount(plugin)
        plugin.NumFailingAssertions = plugin.NumFailingAssertions + 1;
    end

    function printAssertionSummary(plugin)
        fprintf("%s\n", repmat('_', 1, 30))
        fprintf('Total Assertions: %d\n', plugin.FinalizedNumPassingAssertions + ...
            plugin.FinalizedNumFailingAssertions)
        fprintf("\t%d Passed, %d Failed\n", plugin.FinalizedNumPassingAssertions, ...
            plugin.FinalizedNumFailingAssertions)
    end
end
end

```

### 创建示例测试类

在您的当前文件夹中，创建一个名为 `ExampleTest.m` 的文件，其中包含以下参数化测试类。该类产生 300 个 Test 元素，其中 100 个是断言测试，用于比较 1 和 10 之间的伪随机整数。

```

classdef ExampleTest < matlab.unittest.TestCase

    properties (TestParameter)
        num1 = repmat({@()randi(10)}, 1, 10);
        num2 = repmat({@()randi(10)}, 1, 10);
    end

    methods(Test)
        function testAssert(testCase, num1, num2)
            testCase.assertNotEqual(num1(), num2())
        end
        function testVerify(testCase, num1, num2)
            testCase.verifyNotEqual(num1(), num2())
        end
        function testAssume(testCase, num1, num2)
            testCase.assumeNotEqual(num1(), num2())
        end
    end
end

```

### 将插件添加到 TestRunner 并运行测试

在命令提示符下，基于 `ExampleTest` 类创建测试套件。

```
import matlab.unittest.TestSuite
import matlab.unittest.TestRunner

suite = TestSuite.fromClass(?ExampleTest);
```

创建一个没有插件的 `TestRunner` 实例。此代码创建一个静默运行程序，以便您控制安装的插件。

```
runner = TestRunner.withNoPlugins;
```

将 `AssertionCountingPlugin` 添加到运行程序并以并行方式运行测试。如果对运行程序调用 `run` 方法，您还可以在串行模式下运行相同的测试。

```
runner.addPlugin(AssertionCountingPlugin)
result = runner.runInParallel(suite);
```

```
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 6).
## Running a total of 300 tests
```

```
Split tests into 18 groups and running them on 6 workers.
```

```
-----
```

```
Finished Group 6
```

```
-----
```

```
### Running a total of 18 tests in group 6
```

```
-----
```

```
Finished Group 1
```

```
-----
```

```
### Running a total of 20 tests in group 1
```

```
-----
```

```
Finished Group 2
```

```
-----
```

```
### Running a total of 20 tests in group 2
```

```
-----
```

```
Finished Group 3
```

```
-----
```

```
### Running a total of 19 tests in group 3
```

```
-----
```

```
Finished Group 4
```

```
-----
```

```
### Running a total of 19 tests in group 4
```

```
-----
```

```
Finished Group 5
```

```
-----
```

```
### Running a total of 18 tests in group 5
```

```
-----
```

```
Finished Group 7
```

```
-----
```

```
### Running a total of 18 tests in group 7
```

```
-----
```

```
Finished Group 8
```

```
-----
```

#### Running a total of 17 tests in group 8

-----  
Finished Group 9  
-----

#### Running a total of 17 tests in group 9

-----  
Finished Group 10  
-----

#### Running a total of 17 tests in group 10

-----  
Finished Group 11  
-----

#### Running a total of 16 tests in group 11

-----  
Finished Group 12  
-----

#### Running a total of 16 tests in group 12

-----  
Finished Group 15  
-----

#### Running a total of 15 tests in group 15

-----  
Finished Group 14  
-----

#### Running a total of 15 tests in group 14

-----  
Finished Group 17  
-----

#### Running a total of 14 tests in group 17

-----  
Finished Group 16  
-----

#### Running a total of 14 tests in group 16

-----  
Finished Group 13  
-----

#### Running a total of 15 tests in group 13

-----  
Finished Group 18  
-----

#### Running a total of 12 tests in group 18

## Done running tests

---

```
Total Assertions: 100  
 88 Passed, 12 Failed
```

## 另请参阅

[addlistener](#) | [matlab.unittest.TestCase](#) | [matlab.unittest.TestResult](#) |  
[matlab.unittest.TestRunner](#) | [matlab.unittest.TestSuite](#) | [matlab.unittest.fixtures.Fixture](#) |  
[matlab.unittest.plugins.Parallelizable](#) | [matlab.unittest.plugins.TestRunnerPlugin](#) |  
[runInParallel](#)

## 相关示例

- “编写插件以扩展 TestRunner” (第 34-82 页)
- “创建自定义插件” (第 34-85 页)

## 编写用于保存诊断详细信息的插件

本示例演示如何创建自定义插件来保存诊断详细信息。该插件将侦听测试失败并保存诊断信息，以便您在框架完成测试后访问此信息。

### 创建插件

在工作文件夹下的文件中，创建从 `matlab.unittest.plugins.TestRunnerPlugin` 类继承的 `myPlugin` 类。在插件类中：

- 定义插件的 `FailedTestData` 属性，用于存储来自失败测试的信息。
- 覆盖 `TestRunnerPlugin` 的默认 `createTestMethodInstance` 方法，以侦听断言、致命断言和验证失败，以及记录相关信息。
- 覆盖 `TestRunnerPlugin` 的默认 `runTestSuite` 方法，以初始化 `FailedTestData` 属性值。如果您没有初始化该属性值，则在每次使用同一测试运行程序运行测试时，失败测试信息都会附加到 `FailedTestData` 属性。
- 定义辅助函数 `recordData`，以将关于测试失败的信息保存为表。

该插件保存包含在 `PluginData` 和 `QualificationEventData` 对象中的信息。它还会保存失败类型和时间戳。

```
classdef DiagnosticRecorderPlugin < matlab.unittest.plugins.TestRunnerPlugin

    properties
        FailedTestData
    end

    methods (Access = protected)
        function runTestSuite(plugin, pluginData)
            plugin.FailedTestData = [];
            runTestSuite@...
            matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);
        end

        function testCase = createTestMethodInstance(plugin, pluginData)
            testCase = createTestMethodInstance@...
            matlab.unittest.plugins.TestRunnerPlugin(plugin, pluginData);

            testName = pluginData.Name;
            testCase.addlistener('AssertionFailed', ...
                @(~,event)plugin.recordData(event,testName, 'Assertion'));
            testCase.addlistener('FatalAssertionFailed', ...
                @(~,event)plugin.recordData(event,testName, 'Fatal Assertion'));
            testCase.addlistener('VerificationFailed', ...
                @(~,event)plugin.recordData(event,testName, 'Verification'));
        end
    end

    methods (Access = private)
        function recordData(plugin,eventData,name,failureType)
            s.Name = {name};
            s.Type = {failureType};
            if isempty(eventData.TestDiagnosticResult)
                s.TestDiagnostics = 'TestDiagnostics not provided';
            else

```

```

    s.TestDiagnostics = eventData.TestDiagnosticResult;
  end
  s.FrameworkDiagnostics = eventData.FrameworkDiagnosticResult;
  s.Stack = eventData.Stack;
  s.Timestamp = datetime;

  plugin.FailedTestData = [plugin.FailedTestData; struct2table(s)];
end
end
end

```

### 创建测试类

在您的工作文件夹中，创建包含以下测试类的文件 `ExampleTest.m`。

```

classdef ExampleTest < matlab.unittest.TestCase
  methods(Test)
    function testOne(testCase)
      testCase.assertGreater(5,10)
    end
    function testTwo(testCase)
      wrongAnswer = 'wrong';
      testCase.verifyEmpty(wrongAnswer,'Not Empty')
      testCase.verifyClass(wrongAnswer,'double','Not double')
    end

    function testThree(testCase)
      testCase.assertEqual(7*2,13,'Values not equal')
    end
    function testFour(testCase)
      testCase.fatalAssertEqual(3+2,6);
    end
  end
end

```

`testFour` 中的致命断言失败会导致框架停止运行并引发错误。在此示例中，没有后续测试。如果有后续测试，框架将不会运行该测试。

### 将插件添加到测试运行程序并运行测试

在命令提示符处，基于 `ExampleTest` 类创建测试套件，并创建测试运行程序。

```

import matlab.unittest.TestSuite
import matlab.unittest.TestRunner

suite = TestSuite.fromClass(?ExampleTest);
runner = TestRunner.withNoPlugins;

```

创建一个 `myPlugin` 实例并将其添加到测试运行程序。运行测试。

```

p = DiagnosticRecorderPlugin;
runner.addPlugin(p)
result = runner.run(suite);

```

```

Error using ExampleTest/testFour (line 16)
Fatal assertion failed.

```

对于失败的致命断言，框架会引发错误，并且测试运行程序不会返回 `TestResult` 对象。但是，`DiagnosticRecorderPlugin` 会存储关于该测试及之前测试的信息以及失败断言。

## 检查诊断信息

在命令提示符下查看关于失败测试的信息。该信息保存在插件的 `FailedTestData` 属性中。

```
T = p.FailedTestData
```

```
T =
```

```
5×6 table
```

Name	Type	TestDiagnostics			
'ExampleTest/testOne'	'Assertion'	'TestDiagnostics not provided'	'assertGreaterThan failed.'	'The value must be greater than'	
'ExampleTest/testTwo'	'Verification'	'Not Empty'	'verifyEmpty failed.'	'The value must be empty.'	'The value has a'
'ExampleTest/testTwo'	'Verification'	'Not double'	'verifyClass failed.'	'The value's class is incorrect.'	'Actual Class'
'ExampleTest/testThree'	'Assertion'	'Values not equal'	'assertEqual failed.'	'The values are not equal using "isequaln".'	
'ExampleTest/testFour'	'Fatal Assertion'	'TestDiagnostics not provided'	'fatalAssertEqual failed.'	'The values are not equal using "	

有很多选项可用于对这些信息进行存档或后处理。例如，您可以将变量保存为 MAT 文件或使用 `writetable` 将该表写出为各种文件类型，例如 .txt、.csv 或 .xls。

查看第三项测试失败的堆栈信息

```
T.Stack(3)
```

```
ans =
```

```
struct with fields:
```

```
file: 'C:\Work\ExampleTest.m'
name: 'ExampleTest.testTwo'
line: 9
```

显示框架针对第五项测试失败显示的诊断信息。

```
celldisp(T.FrameworkDiagnostics(5))
```

```
ans{1} =
```

```
fatalAssertEqual failed.
--> The values are not equal using "isequaln".
--> Failure table:
```

Actual	Expected	Error	RelativeError
5	6	-1	-0.1666666666666667

Actual Value:

5

Expected Value:

6

## 另请参阅

`addlistener` | `matlab.unittest.TestCase` | `matlab.unittest.TestRunner` |  
`matlab.unittest.plugins.TestRunnerPlugin`

## 相关示例

- “编写插件以扩展 TestRunner” (第 34-82 页)
- “创建自定义插件” (第 34-85 页)
- “用于生成自定义测试输出格式的插件” (第 34-102 页)

## 用于生成自定义测试输出格式的插件

本示例演示如何创建一个插件，该插件使用自定义格式将最终测试结果写入到输出流。

### 创建插件

在工作文件夹下的文件中，创建从 `matlab.unittest.plugins.TestRunnerPlugin` 类继承的 `ExampleCustomPlugin` 类。在插件类中：

- 定义插件的 `Stream` 属性，用于存储 `OutputStream` 实例。默认情况下，该插件会写入标准输出。
- 覆盖 `TestRunnerPlugin` 的默认 `runTestSuite` 方法，以输出指示测试运行程序正运行新测试会话的文本。如果您要写入单个日志文件，此信息特别有用，因为它可让您区分测试运行。
- 覆盖 `TestRunnerPlugin` 的默认 `reportFinalizedResult` 方法，以将最终测试结果写入输出流。您可以修改 `print` 方法，以便以适用于您的测试日志或连续集成系统的方式输出测试结果。

```
classdef ExampleCustomPlugin < matlab.unittest.plugins.TestRunnerPlugin
    properties (Access=private)
        Stream
    end

    methods
        function p = ExampleCustomPlugin(stream)
            if ~nargin
                stream = matlab.unittest.plugins.ToStandardOutput;
            end
            validateattributes(stream, ...
                {'matlab.unittest.plugins.OutputStream'}, {})
            p.Stream = stream;
        end
    end

    methods (Access=protected)
        function runTestSuite(plugin,pluginData)
            plugin.Stream.print('\n-- NEW TEST SESSION at %s --\n',...
                char(datetime))
            runTestSuite@...
            matlab.unittest.plugins.TestRunnerPlugin(plugin,pluginData);
        end

        function reportFinalizedResult(plugin,pluginData)
            thisResult = pluginData.TestResult;
            if thisResult.Passed
                status = 'PASSED';
            elseif thisResult.Failed
                status = 'FAILED';
            elseif thisResult.Incomplete
                status = 'SKIPPED';
            end
            plugin.Stream.print(... 
                '## YPS Company - Test %s ## - %s in %f seconds.\n',...
                status, thisResult.Name, thisResult.Duration)

            reportFinalizedResult@...
            matlab.unittest.plugins.TestRunnerPlugin(plugin,pluginData)
        end
    end
```

```

end
end

```

### 创建测试类

在您的工作文件夹中，创建包含以下测试类的文件 `ExampleTest.m`。在此测试类中，其中两个测试会通过，其他测试会导致验证或假设失败。

```

classdef ExampleTest < matlab.unittest.TestCase
methods(Test)
    function testOne(testCase)
        testCase.assertGreater(5,1)
    end
    function testTwo(testCase)
        wrongAnswer = 'wrong';
        testCase.verifyEmpty(wrongAnswer,'Not Empty')
        testCase.verifyClass(wrongAnswer,'double','Not double')
    end
    function testThree(testCase)
        testCase.assertEqual(7*2,13,'Values not equal')
    end
    function testFour(testCase)
        testCase.verifyEqual(3+2,5);
    end
end
end

```

### 将插件添加到测试运行程序并运行测试

在命令提示符处，基于 `ExampleTest` 类创建测试套件，并创建测试运行程序。

```

import matlab.unittest.TestSuite
import matlab.unittest.TestRunner

suite = TestSuite.fromClass(?ExampleTest);
runner = TestRunner.withNoPlugins;

```

创建一个 `ExampleCustomPlugin` 实例并将其添加到测试运行程序。运行测试。

```

import matlab.unittest.plugins.ToFile
fname = 'YPS_test_results.txt';
p = ExampleCustomPlugin(ToFile(fname));

runner.addPlugin(p)
result = runner.run(suite);

```

查看输出文件的内容。

```
type(fname)
```

```

--- NEW TEST SESSION at 26-Jan-2015 10:41:24 ---
### YPS Company - Test PASSED ### - ExampleTest/testOne in 0.123284 seconds.
### YPS Company - Test FAILED ### - ExampleTest/testTwo in 0.090363 seconds.
### YPS Company - Test SKIPPED ### - ExampleTest/testThree in 0.518044 seconds.
### YPS Company - Test PASSED ### - ExampleTest/testFour in 0.020599 seconds.

```

使用同一测试运行程序重新运行 `Incomplete` 测试。查看输出文件的内容。

```
suiteFiltered = suite([result.Incomplete]);
runner.run(suiteFiltered);

type(fname)

-- NEW TEST SESSION at 26-Jan-2015 10:41:24 --
### YPS Company - Test PASSED ### - ExampleTest/testOne in 0.123284 seconds.
### YPS Company - Test FAILED ### - ExampleTest/testTwo in 0.090363 seconds.
### YPS Company - Test SKIPPED ### - ExampleTest/testThree in 0.518044 seconds.
### YPS Company - Test PASSED ### - ExampleTest/testFour in 0.020599 seconds.

-- NEW TEST SESSION at 26-Jan-2015 10:41:58 --
### YPS Company - Test SKIPPED ### - ExampleTest/testThree in 0.007892 seconds.
```

### 另请参阅

[ToFile](#) | [ToStandardOutput](#) | [matlab.unittest.plugins.OutputStream](#) |  
[matlab.unittest.plugins.TestRunnerPlugin](#)

### 相关示例

- “编写插件以扩展 TestRunner” (第 34-82 页)
- “编写用于保存诊断详细信息的插件” (第 34-98 页)

# 分析测试用例结果

此示例说明如何分析从 `SolverTest` 测试用例创建的测试运行程序返回的信息。

## 创建二次求解器函数

创建以下函数来对您的工作文件夹下的文件 `quadraticSolver.m` 中二次方程的根求解。

```
type quadraticSolver.m

function roots = quadraticSolver(a, b, c)
% quadraticSolver returns solutions to the
% quadratic equation a*x^2 + b*x + c = 0.

if ~isa(a,'numeric') || ~isa(b,'numeric') || ~isa(c,'numeric')
    error('quadraticSolver:InputMustBeNumeric',...
        'Coefficients must be numeric.');
end

roots(1) = (-b + sqrt(b^2 - 4*a*c)) / (2*a);
roots(2) = (-b - sqrt(b^2 - 4*a*c)) / (2*a);

end
```

## 为二次求解器函数创建测试

在您的工作文件夹下的文件 `SolverTest.m` 中，创建以下测试类。

```
type SolverTest.m

classdef SolverTest < matlab.unittest.TestCase
    % SolverTest tests solutions to the quadratic equation
    % a*x^2 + b*x + c = 0

    methods (Test)
        function testRealSolution(testCase)
            actSolution = quadraticSolver(1,-3,2);
            expSolution = [2,1];
            testCase.verifyEqual(actSolution,expSolution);
        end
        function testImaginarySolution(testCase)
            actSolution = quadraticSolver(1,2,10);
            expSolution = [-1+3i, -1-3i];
            testCase.verifyEqual(actSolution,expSolution);
        end
    end
end

end
```

## 运行 SolverTest 测试用例

创建一个测试套件 `quadTests`。

```
quadTests = matlab.unittest.TestSuite.fromClass(?SolverTest);
result = run(quadTests);
```

Running SolverTest

..

```
Done SolverTest
```

所有测试都已通过。

### 浏览输出参数 result

输出参数 `result` 是一个 `matlab.unittest.TestResult` 对象。它包含 `SolverTest` 中的两个测试的信息。

`whos result`

Name	Size	Bytes	Class	Attributes
result	1x2	4864	matlab.unittest.TestResult	

### 显示一个测试的信息

要查看一个值的信息，请键入：

`result(1)`

```
ans =
TestResult with properties:
```

```
Name: 'SolverTest/testRealSolution'
Passed: 1
Failed: 0
Incomplete: 0
Duration: 0.4514
Details: [1x1 struct]
```

Totals:

```
1 Passed, 0 Failed, 0 Incomplete.
0.45139 seconds testing time.
```

### 创建测试结果表格

要访问表格可用的功能，先从 `TestResult` 对象创建一个表格。

`rt = table(result)`

Name	Passed	Failed	Incomplete	Duration	Details
{'SolverTest/testRealSolution'}	true	false	false	0.45139	{1x1 struct}
{'SolverTest/testImaginarySolution'}	true	false	false	0.011828	{1x1 struct}

按持续时间对测试结果排序。

`sortrows(rt,'Duration')`

Name	Passed	Failed	Incomplete	Duration	Details
{'SolverTest/testRealSolution'}	true	false	false	0.45139	{1x1 struct}
{'SolverTest/testImaginarySolution'}	true	false	false	0.011828	{1x1 struct}

```
{'SolverTest/testImaginarySolution'}  true    false    false    0.011828  {1x1 struct}  
{'SolverTest/testRealSolution'}   true    false    false    0.45139   {1x1 struct}
```

将测试结果导出到 CSV 文件。

```
writetable(rt,'myTestResults.csv','QuoteStrings',true)
```

## 另请参阅

### 相关示例

- “使用类来编写简单测试用例”（第 34-37 页）

## 分析失败的测试结果

此示例演示如何识别和重新运行失败的测试。

### 创建不正确的测试方法

使用 SolverTest 测试用例，添加一个方法 testBadRealSolution。此测试（基于 testRealSolution）使用输入 1,3,2 调用 quadraticSolver，但根据不正确的解 [2,1] 测试结果。

```
function testBadRealSolution(testCase)
    actSolution = quadraticSolver(1,3,2);
    expSolution = [2,1];
    testCase.verifyEqual(actSolution,expSolution)
end
```

### 运行新测试套件

保存更新的 SolverTest 类定义并重新运行测试。

```
quadTests = matlab.unittest.TestSuite.fromClass(?SolverTest);
result1 = run(quadTests);
```

Running SolverTest

..

=====

Verification failed in SolverTest/testBadRealSolution.

-----

Framework Diagnostic:

-----

verifyEqual failed.  
 --> The values are not equal using "isequaln".  
 --> Failure table:

Index	Actual	Expected	Error	RelativeError
1	-1	2	-3	-1.5
2	-2	1	-3	-3

-----

Actual Value:

-1 -2

Expected Value:

2 1

-----

Stack Information:

-----

In C:\work\SolverTest.m (SolverTest.testBadRealSolution) at 19

-----

. Done SolverTest

-----

Failure Summary:

Name	Failed	Incomplete	Reason(s)
------	--------	------------	-----------

SolverTest/testBadRealSolution	X		Failed by verification.
--------------------------------	---	--	-------------------------

## 分析结果

输出告知您 SolverTest/testBadRealSolution 失败。从 Framework Diagnostic，您可以看到以下内容：

```
Actual Value:  
  -1  -2  
Expected Value:  
  2   1
```

此时，您必须决定错误是存在于 quadraticSolver 中还是 expSolution 的值中。

## 更正错误

在 testBadRealSolution 中编辑 expSolution 的值：

```
expSolution = [-1 -2];
```

## 重新运行测试

保存 SolverTest 并仅重新运行已失败的测试。

```
failedTests = quadTests([result1.Failed]);  
result2 = run(failedTests)
```

```
Running SolverTest
```

```
.
```

```
Done SolverTest
```

---

```
result2 =
```

```
TestResult with properties:
```

```
  Name: 'SolverTest/testBadRealSolution'  
  Passed: 1  
  Failed: 0  
Incomplete: 0  
  Duration: 0.0108  
  Details: [1x1 struct]
```

```
Totals:
```

```
  1 Passed, 0 Failed, 0 Incomplete.  
  0.010813 seconds testing time.
```

您也可以使用测试结果中的 ([rerun](#)) 链接重新运行失败的测试。

## 另请参阅

### 详细信息

- “重新运行失败的测试”（第 34-110 页）

## 重新运行失败的测试

如果测试失败是由不正确或不完整的代码引起的，则快速、方便地重新运行失败的测试很有用。当您运行一个测试套件时，测试结果包含有关该测试套件和测试运行程序的信息。如果结果中有测试失败，则 MATLAB 显示的测试结果中会有用于重新运行失败测试的链接。

Totals:

1 Passed, 1 Failed ([rerun](#)), 0 Incomplete.  
0.25382 seconds testing time.

通过此链接，您可以修改测试代码或被测代码，并快速重新运行失败的测试。但是，如果您对测试类进行了结构性更改，则使用重新运行链接不会反映所做更改。结构性更改包括添加、删除或重命名测试方法，以及修改测试参数属性及其值。在这种情况下，请重新创建整个测试套件以反映更改。

在您的当前工作文件夹中创建以下函数。该函数旨在计算平方和平方根。但是，在此示例中，函数计算的是值的立方而不是平方。

```
function [x,y] = exampleFunction(n)
    validateattributes(n,{'numeric'},{'scalar'})

    x = n^3; % square (incorrect code, should be n^2)
    y = sqrt(n); % square root
end
```

在文件 `exampleTest.m` 中创建以下测试。

```
function tests = exampleTest
    tests = functiontests(localfunctions);
end

function testSquare(testCase)
    [sqrVal,sqrRootVal] = exampleFunction(3);
    verifyEqual(testCase,sqrVal,9);
end

function testSquareRoot(testCase)
    [sqrVal,sqrRootVal] = exampleFunction(100);
    verifyEqual(testCase,sqrRootVal,10);
end
```

创建一个测试套件并运行测试。`testSquare` 测试失败，因为 `exampleFunction` 的实现不正确。

```
suite = testsuite('ExampleTest.m');
results = run(suite)
```

Running exampleTest

=====

Verification failed in exampleTest/testSquare.

-----

Framework Diagnostic:

-----

verifyEqual failed.  
-> The values are not equal using "isequaln".  
-> Failure table:

Actual	Expected	Error	RelativeError
--------	----------	-------	---------------

27	9	18	2
----	---	----	---

Actual Value:

```

27
Expected Value:
9

-----
Stack Information:
=====
In C:\Work\exampleTest.m (testSquare) at 7
=====
.. Done exampleTest

-----
Failure Summary:
=====
Name      Failed Incomplete Reason(s)
=====
exampleTest/testSquare  X      Failed by verification.

results =
1x2 TestResult array with properties:
=====
Name
Passed
Failed
Incomplete
Duration
Details

Totals:
1 Passed, 1 Failed (rerun), 0 Incomplete.
0.24851 seconds testing time.

```

更新 `exampleFunction` 中的代码以修复代码编写错误。

```

function [x,y] = exampleFunction(n)
    validateattributes(n,'numeric','scalar')

    x = n^2;    % square
    y = sqrt(n); % square root
end

```

点击命令行窗口中的 ([rerun](#)) 链接以重新运行失败的测试。如果存储测试结果的变量被覆盖，则您不能重新运行失败的测试。如果链接不再位于命令行窗口中，您可以在提示符下键入 `results` 以查看该链接。

Running exampleTest

.  
Done exampleTest

`ans =`

TestResult with properties:

```

Name: 'exampleTest/testSquare'
Passed: 1
Failed: 0
Incomplete: 0
Duration: 0.0034
Details: [1x1 struct]

```

Totals:

1 Passed, 0 Failed, 0 Incomplete.  
0.0033903 seconds testing time.

MATLAB 将与您重新运行的测试相关联的 `TestResult` 数组存储在 `ans` 变量中。`results` 是  $1 \times 2$  数组，其中包含 `exampleTest.m` 中的所有测试；`ans` 是  $1 \times 1$  数组，其中包含来自一个失败测试的重新运行结果。

### whos

```
Name      Size      Bytes Class          Attributes
ans      1x1        664  matlab.unittest.TestResult
results  1x2       1344  matlab.unittest.TestResult
suite    1x2        96   matlab.unittest.Test
```

要以编程方式重新运行失败的测试，请对 `TestResult` 对象使用 `Failed` 属性以创建并运行一个已筛选测试套件。

```
failedTests = suite([results.Failed]);
result2 = run(failedTests);
```

Running exampleTest

.

Done exampleTest

---

为确保所有通过的测试继续通过，请重新运行完整测试套件。

## 另请参阅

### 详细信息

- “分析失败的测试结果”（第 34-108 页）

# 动态筛选的测试

## 本节内容

- “测试方法”（第 34-113 页）
- “方法设置和拆解代码”（第 34-115 页）
- “类设置和拆解代码”（第 34-117 页）

假设失败会生成已筛选的测试。在 `matlab.unittest.TestResult` 类中，此类测试标记为 **Incomplete**。

因为通过使用假设筛选测试内容不会产生测试失败，所以这可能生成无效的测试代码。要避免此结果，需要监视已筛选的测试。

## 测试方法

如果在带有 `Test` 属性的 `TestCase` 方法内部遇到假设失败，整个方法将标记为已筛选，但 MATLAB 会运行后续 `Test` 方法。

下面的类包含 `Test` 块中的其中一个方法的假设失败。

```
classdef ExampleTest < matlab.unittest.TestCase
methods(Test)
    function testA(testCase)
        testCase.verifyTrue(true)
    end
    function testB(testCase)
        testCase.assertEqual(0,1)
        % remaining test code is not exercised
    end
    function testC(testCase)
        testCase.verifyFalse(true)
    end
end
end
```

因为 `testB` 方法包含假设失败，所以当您运行测试时，测试框架筛选该测试并将其标记为未完成。在 `testB` 中的假设失败之后，测试框架会继续执行 `testC`，其中包含验证失败。

```
ts = matlab.unittest.TestSuite.fromClass(?ExampleTest);
res = ts.run;
```

Running ExampleTest

=====
ExampleTest/testB was filtered.

Details

=====
Verification failed in ExampleTest/testC.

=====
Framework Diagnostic:

=====
verifyFalse failed.

```
--> The value must evaluate to "false".
```

Actual logical:

```
1
```

---

Stack Information:

---

```
In C:\work\ExampleTest.m (ExampleTest.testC) at 11
```

---

```
. Done ExampleTest
```

---

Failure Summary:

Name	Failed	Incomplete	Reason(s)
ExampleTest/testB	X		Filtered by assumption.
ExampleTest/testC	X		Failed by verification.

如果您检查 **TestResult**, 则会注意到有一个通过的测试, 一个失败的测试和一个由于假设失败而未完成的测试。

```
res
```

```
res =
```

1×3 TestResult array with properties:

```
Name  
Passed  
Failed  
Incomplete  
Duration  
Details
```

Totals:

```
1 Passed, 1 Failed, 1 Incomplete.  
2.4807 seconds testing time.
```

测试框架跟踪未完成的测试, 以便监视已筛选测试中是否有未执行的测试代码。您可以在 **TestResult** 对象中查看有关这些测试的信息。

```
res([res.Incomplete])
```

```
ans =
```

TestResult with properties:

```
Name: 'ExampleTest/testB'  
Passed: 0  
Failed: 0  
Incomplete: 1  
Duration: 2.2578  
Details: [1×1 struct]
```

```
Totals:
  0 Passed, 0 Failed, 1 Incomplete.
  2.2578 seconds testing time.
```

要仅从已筛选测试创建经过修改的测试套件，请从原始测试套件选择未完成的测试。

```
tsFiltered = ts([res.Incomplete])

tsFiltered =

Test with properties:
  Name: 'ExampleTest/testB'
  ProcedureName: 'testB'
  TestClass: "ExampleTest"
  BaseFolder: 'C:\work'
  Parameterization: [0×0 matlab.unittest.parameters.EmptyParameter]
  SharedTestFixtures: [0×0 matlab.unittest.fixtures.EmptyFixture]
  Tags: {1×0 cell}
```

```
Tests Include:
  0 Parameterizations, 0 Shared Test Fixture Classes, 0 Tags.
```

## 方法设置和拆解代码

如果在带有 `TestMethodSetup` 属性的 `TestCase` 方法内遇到假设失败，MATLAB 会筛选本应为该实例运行的方法。如果某测试使用 `TestMethodSetup` 块中的假设，则考虑使用 `TestClassSetup` 块中的假设，这同样会筛选该类中的所有 `Test` 方法，但不够详尽而效果更高。

`ExampleTest.m` 中的以下 `TestMethodSetup` 块中的一个方法包含假设失败。

```
methods(TestMethodSetup)
  function setupMethod1(testCase)
    testCase.assertEqual(1,0)
    % remaining test code is not exercised
  end
  function setupMethod2(testCase)
    disp('* Running setupMethod2 *')
    testCase.assertEqual(1,1)
  end
end
```

## 更新的 ExampleTest 类定义

```
classdef ExampleTest < matlab.unittest.TestCase
  methods(TestMethodSetup)
    function setupMethod1(testCase)
      testCase.assertEqual(1,0)
      % remaining test code is not exercised
    end
    function setupMethod2(testCase)
      disp('* Running setupMethod2 *')
      testCase.assertEqual(1,1)
    end
  end

  methods(Test)
    function testA(testCase)
```

```

    testCase.verifyTrue(true)
end
function testB(testCase)
    testCase.assertEqual(0,1)
    % remaining test code is not exercised
end
function testC(testCase)
    testCase.verifyFalse(true)
end
end
end

```

当您运行测试时，您会看到该框架执行完 `TestMethodSetup` 块中不包含假设失败的所有方法，并将 `Test` 块中的所有方法标记为未完成。

```
ts = matlab.unittest.TestSuite.fromClass(?ExampleTest);
res = ts.run;
```

Running ExampleTest

---

=====  
ExampleTest/testA was filtered.

Details

---

=====  
\* Running setupMethod2 \*

.

---

=====  
ExampleTest/testB was filtered.

Details

---

=====  
\* Running setupMethod2 \*

.

---

=====  
ExampleTest/testC was filtered.

Details

---

=====  
\* Running setupMethod2 \*

.

Done ExampleTest

---

Failure Summary:

Name	Failed	Incomplete	Reason(s)
ExampleTest/testA	X		Filtered by assumption.
ExampleTest/testB	X		Filtered by assumption.
ExampleTest/testC	X		Filtered by assumption.

`Test` 方法未更改，但由于 `TestMethodSetup` 块中存在假设失败，所有 3 个均已筛选。测试框架执行 `TestMethodSetup` 块中不含假设失败的方法，例如 `setupMethod2`。和预期一样，测试框架执行 `setupMethod2` 3 次，每个 `Test` 方法之前各一次。

## 类设置和拆解代码

如果在带有 `TestClassSetup` 或 `TestClassTeardown` 属性的 `TestCase` 方法内遇到假设失败，MATLAB 会筛选整个 `TestCase` 类。

`ExampleTest.m` 中的以下 `TestClassSetup` 块中的方法包含假设失败。

```
methods(TestClassSetup)
    function setupClass(testCase)
        testCase.assertEqual(1,0)
        % remaining test code is not exercised
    end
end
```

### 更新的 ExampleTest 类定义

```
classdef ExampleTest < matlab.unittest.TestCase
    methods(TestClassSetup)
        function setupClass(testCase)
            testCase.assertEqual(1,0)
            % remaining test code is not exercised
        end
    end

    methodsTestMethodSetup)
        function setupMethod1(testCase)
            testCase.assertEqual(1,0)
            % remaining test code is not exercised
        end
        function setupMethod2(testCase)
            disp('* Running setupMethod2 *')
            testCase.assertEqual(1,1)
        end
    end

    methods(Test)
        function testA(testCase)
            testCase.verifyTrue(true)
        end
        function testB(testCase)
            testCase.assertEqual(0,1)
            % remaining test code is not exercised
        end
        function testC(testCase)
            testCase.verifyFalse(true)
        end
    end
end
```

当您运行测试时，您会看到该框架不执行 `TestMethodSetup` 或 `Test` 中的任何方法。

```
ts = matlab.unittest.TestSuite.fromClass(?ExampleTest);
res = ts.run;
```

Running ExampleTest

---

All tests in ExampleTest were filtered.

Details

---

Done ExampleTest

---

Failure Summary:

Name	Failed	Incomplete	Reason(s)
ExampleTest/testA	X		Filtered by assumption.
ExampleTest/testB	X		Filtered by assumption.
ExampleTest/testC	X		Filtered by assumption.

**Test** 和 **TestMethodSetup** 方法未更改，但由于 **TestClassSetup** 块中存在假设失败，所有内容均已筛选。

**另请参阅**

[TestCase](#) | [TestResult](#) | [matlab.unittest.qualifications.Assumable](#)

## 创建自定义约束

此示例演示如何创建一个自定义约束，该约束确定给定值的大小是否与预期值的大小相同。

从 `matlab.unittest.constraints.Constraint` 类派生以创建一个名为 `HasSameSizeAs` 的类，将其置于当前文件夹。类构造函数比较实际大小与预期大小，并接受具有预期大小的值。该值存储在 `ValueWithExpectedSize` 属性中。由于 `Constraint` 实现最好是不可变的，因此请将属性 `SetAccess` 特性设置为 `immutable`。

```
classdef HasSameSizeAs < matlab.unittest.constraints.Constraint

    properties(SetAccess = immutable)
        ValueWithExpectedSize
    end

    methods
        function constraint = HasSameSizeAs(value)
            constraint.ValueWithExpectedSize = value;
        end
    end

end
```

在具有 `private` 访问权限的 `methods` 块中，定义辅助函数方法 `sizeMatchesExpected`，该方法确定实际值和预期值是否具有相同的大小。该方法由其他约束方法调用。

```
methods(Access = private)
    function bool = sizeMatchesExpected(constraint,actual)
        bool = isequal(size(actual),size(constraint.ValueWithExpectedSize));
    end
end
```

从 `matlab.unittest.constraints.Constraint` 类派生的类必须覆盖 `satisfiedBy` 方法。此方法必须包含比较逻辑并返回一个逻辑值。

在 `methods` 块中，通过调用辅助函数方法实现 `satisfiedBy`。如果实际大小和预期大小相等，该方法返回 `true`。

```
methods
    function bool = satisfiedBy(constraint,actual)
        bool = constraint.sizeMatchesExpected(actual);
    end
end
```

从 `matlab.unittest.constraints.Constraint` 类派生的类还必须覆盖 `getDiagnosticFor` 方法。此方法必须针对约束计算实际值并提供 `Diagnostic` 对象。在此示例中，`getDiagnosticFor` 返回 `StringDiagnostic` 对象。

```
methods
    function diag = getDiagnosticFor(constraint,actual)
        import matlab.unittest.diagnostics.StringDiagnostic
        if constraint.sizeMatchesExpected(actual)
            diag = StringDiagnostic('HasSameSizeAs passed.');
        else
            diag = StringDiagnostic(sprintf...
                'HasSameSizeAs failed.\nActual Size: [%s]\nExpectedSize: [%s]',...
                int2str(size(actual)),...)
```

```

        int2str(size(constraint.ValueWithExpectedSize)));
    end
end
end

```

### HasSameSizeAs 类定义总结

以下代码提供 HasSameSizeAs 的完整内容。

```

classdef HasSameSizeAs < matlab.unittest.constraints.Constraint

properties(SetAccess = immutable)
    ValueWithExpectedSize
end

methods
    function constraint = HasSameSizeAs(value)
        constraint.ValueWithExpectedSize = value;
    end

    function bool = satisfiedBy(constraint,actual)
        bool = constraint.sizeMatchesExpected(actual);
    end

    function diag = getDiagnosticFor(constraint,actual)
        import matlab.unittest.diagnostics.StringDiagnostic
        if constraint.sizeMatchesExpected(actual)
            diag = StringDiagnostic('HasSameSizeAs passed.');
        else
            diag = StringDiagnostic(sprintf',...
                'HasSameSizeAs failed.\nActual Size: [%s]\nExpectedSize: [%s]',...
                int2str(size(actual)),...
                int2str(size(constraint.ValueWithExpectedSize))));
        end
    end
end
end

methods(Access = private)
    function bool = sizeMatchesExpected(constraint,actual)
        bool = isequal(size(actual),size(constraint.ValueWithExpectedSize));
    end
end
end

```

在命令提示符处，创建测试用例以执行交互式测试。

```

import matlab.unittest.TestCase
testCase = TestCase.forInteractiveUse;

```

测试一个通过的用例。

```

testCase.verifyThat(zeros(5),HasSameSizeAs(repmat(1,5)))

```

Verification passed.

测试一个失败的用例。

```

testCase.verifyThat(zeros(5),HasSameSizeAs(ones(1,5)))

```

Verification failed.

-----  
Framework Diagnostic:  
-----

HasSameSizeAs failed.

Actual Size: [5 5]  
ExpectedSize: [1 5]

## 另请参阅

[getDiagnosticFor](#) | [matlab.unittest.constraints.Constraint](#) | [satisfiedBy](#)

## 相关示例

- “[创建自定义布尔约束](#)” (第 34-122 页)

## 创建自定义布尔约束

此示例演示如何创建一个自定义布尔约束，该约束确定给定值的大小是否与预期值的大小相同。

从 `matlab.unittest.constraints.BooleanConstraint` 类派生以创建一个名为 `HasSameSizeAs` 的类，将其置于当前文件夹。类构造函数比较实际大小与预期大小，并接受具有预期大小的值。该值存储在 `ValueWithExpectedSize` 属性中。由于 `BooleanConstraint` 实现最好是不可变的，因此请将属性 `SetAccess` 特性设置为 `immutable`。

```
classdef HasSameSizeAs < matlab.unittest.constraints.BooleanConstraint

    properties(SetAccess = immutable)
        ValueWithExpectedSize
    end

    methods
        function constraint = HasSameSizeAs(value)
            constraint.ValueWithExpectedSize = value;
        end
    end

end
```

在具有 `private` 访问权限的 `methods` 块中，定义辅助函数方法 `sizeMatchesExpected`，该方法确定实际值和预期值是否具有相同的大小。该方法由其他约束方法调用。

```
methods(Access = private)
    function bool = sizeMatchesExpected(constraint,actual)
        bool = isequal(size(actual),size(constraint.ValueWithExpectedSize));
    end
end
```

`matlab.unittest.constraints.BooleanConstraint` 类是 `matlab.unittest.constraints.Constraint` 类的子类。因此，从 `BooleanConstraint` 类派生的类必须覆盖 `Constraint` 类的方法。在 `methods` 块内，覆盖 `satisfiedBy` 和 `getDiagnosticFor` 方法。`satisfiedBy` 实现必须包含比较逻辑并返回一个逻辑值。`getDiagnosticFor` 实现必须针对约束对实际值进行计算，并提供 `Diagnostic` 对象。在此示例中，`getDiagnosticFor` 返回 `StringDiagnostic` 对象。

```
methods
    function bool = satisfiedBy(constraint,actual)
        bool = constraint.sizeMatchesExpected(actual);
    end

    function diag = getDiagnosticFor(constraint,actual)
        import matlab.unittest.diagnostics.StringDiagnostic
        if constraint.sizeMatchesExpected(actual)
            diag = StringDiagnostic('HasSameSizeAs passed.');
        else
            diag = StringDiagnostic(sprintf(
                'HasSameSizeAs failed.\nActual Size: [%s]\nExpectedSize: [%s]',...
                int2str(size(actual)),...
                int2str(size(constraint.ValueWithExpectedSize))));
        end
    end
end
```

派生自 `BooleanConstraint` 的类必须实现 `getNegativeDiagnosticFor` 方法。当约束取反时，此方法必须提供 `Diagnostic` 对象。

在具有 `protected` 访问权限的 `methods` 块中覆盖 `getNegativeDiagnosticFor`。

```

methods(Access = protected)
    function diag = getNegativeDiagnosticFor(constraint,actual)
        import matlab.unittest.diagnostics.StringDiagnostic
        if constraint.sizeMatchesExpected(actual)
            diag = StringDiagnostic(sprintf(...,
                ['Negated HasSameSizeAs failed.\nSize [%s] of',...
                'Actual Value and Expected Value were the same',...
                'but should not have been.',int2str(size(actual))]));
        else
            diag = StringDiagnostic('Negated HasSameSizeAs passed.');
        end
    end
end

```

为实现必需方法，约束继承相应的 `and`、`or` 和 `not` 重载，以便与其他 `BooleanConstraint` 对象进行组合或求反。

## HasSameSizeAs 类定义总结

以下代码提供 `HasSameSizeAs` 的完整内容。

```

classdef HasSameSizeAs < matlab.unittest.constraints.BooleanConstraint

properties(SetAccess = immutable)
    ValueWithExpectedSize
end

methods
    function constraint = HasSameSizeAs(value)
        constraint.ValueWithExpectedSize = value;
    end

    function bool = satisfiedBy(constraint,actual)
        bool = constraint.sizeMatchesExpected(actual);
    end

    function diag = getDiagnosticFor(constraint,actual)
        import matlab.unittest.diagnostics.StringDiagnostic

        if constraint.sizeMatchesExpected(actual)
            diag = StringDiagnostic('HasSameSizeAs passed.');
        else
            diag = StringDiagnostic(sprintf(...,
                ['HasSameSizeAs failed.\nActual Size: [%s]\nExpectedSize: [%s]',...
                int2str(size(actual)),...,int2str(size(constraint.ValueWithExpectedSize))]);
        end
    end
end

methods(Access = protected)
    function diag = getNegativeDiagnosticFor(constraint,actual)
        import matlab.unittest.diagnostics.StringDiagnostic
        if constraint.sizeMatchesExpected(actual)
            diag = StringDiagnostic(sprintf(...,
                ['Negated HasSameSizeAs failed.\nSize [%s] of',...
                'Actual Value and Expected Value were the same',...
                'but should not have been.',int2str(size(actual))]));
        else
            diag = StringDiagnostic('Negated HasSameSizeAs passed.');
        end
    end
end

methods(Access = private)
    function bool = sizeMatchesExpected(constraint,actual)
        bool = isequal(size(actual),size(constraint.ValueWithExpectedSize));
    end
end

end

```

在命令提示符处，创建测试用例以执行交互式测试。

```
import matlab.unittest.TestCase
import matlab.unittest.constraints.HasLength
testCase = TestCase.forInteractiveUse;
```

测试一个通过的用例。

```
testCase.verifyThat(zeros(5),HasLength(5) | ~HasSameSizeAs(repmat(1,5)))
```

```
Verification passed.
```

测试通过，原因是 or 条件之一 HasLength(5) 为 true。

测试一个失败的用例。

```
testCase.verifyThat(zeros(5),HasLength(5) & ~HasSameSizeAs(repmat(1,5)))
```

```
Verification failed.
```

```
-----  
Framework Diagnostic:
```

```
AndConstraint failed.
```

```
--> + [First Condition]:
```

```
    | HasLength passed.
```

```
    |
```

```
    | Actual Value:
```

```
    |   0   0   0   0   0  
    |   0   0   0   0   0  
    |   0   0   0   0   0  
    |   0   0   0   0   0  
    |   0   0   0   0   0
```

```
    | Expected Length:
```

```
    |   5
```

```
--> AND
```

```
+ [Second Condition]:
```

```
    | Negated HasSameSizeAs failed.
```

```
    | Size [5 5] of Actual Value and Expected Value were the same but should not have been.
```

测试失败，原因是 and 条件之一 ~HasSameSizeAs(repmat(1,5)) 为 false。

## 另请参阅

[getDiagnosticFor](#) | [getNegativeDiagnosticFor](#) |  
[matlab.unittest.constraints.BooleanConstraint](#) | [matlab.unittest.constraints.Constraint](#) |  
[satisfiedBy](#)

## 相关示例

- “[创建自定义约束](#)” (第 34-119 页)

## 创建自定义容差

此示例演示如何创建自定义容差以确定两个 DNA 序列是否具有指定容差范围内的 Hamming 距离。对于相同长度的两个 DNA 序列，Hamming 距离是一个序列的核苷酸（字母）与另一个序列的核苷酸（字母）之间相差的位置数。

在工作文件夹下的文件 DNA.m 中，为 DNA 序列创建一个简单类。

```
classdef DNA
    properties(SetAccess=immutable)
        Sequence
    end

    methods
        function dna = DNA(sequence)
            validLetters = ...
                sequence == 'A' | ...
                sequence == 'C' | ...
                sequence == 'T' | ...
                sequence == 'G';
            if ~all(validLetters())
                error('Sequence contained a letter not found in DNA.')
            end
            dna.Sequence = sequence;
        end
    end
end
```

在工作文件夹下的文件中，创建一个容差类，这样可以测试 DNA 序列是否处于指定的 Hamming 距离内。构造函数要求 Value 属性定义最大 Hamming 距离。

```
classdef HammingDistance < matlab.unittest.constraints.Tolerance
    properties
        Value
    end

    methods
        function tolerance = HammingDistance(value)
            tolerance.Value = value;
        end
    end
end
```

在包含 HammingDistance 类定义的 methods 块中，包含以下方法，这样容差支持 DNA 对象。容差类必须实现 supports 方法。

```
methods
    function tf = supports(~, value)
        tf = isa(value, 'DNA');
    end
end
```

在包含 HammingDistance 类定义的 methods 块中，包含返回 true 或 false 的以下方法。容差类必须实现 satisfiedBy 方法。测试框架将利用此方法确定两个值是否处于容差范围之内。

```
methods
    function tf = satisfiedBy(tolerance, actual, expected)
```

```

if ~isSameSize(actual.Sequence, expected.Sequence)
    tf = false;
    return
end
tf = hammingDistance(actual.Sequence,expected.Sequence) <= tolerance.Value;
end
end

```

在 HammingDistance.m 文件中，在 classdef 块的外部定义以下辅助函数。如果两个 DNA 序列的大小相同并且 hammingDistance 函数返回两个序列之间的 Hamming 距离，则 isSameSize 函数返回 true。

```

function tf = isSameSize(str1, str2)
tf = isequal(size(str1), size(str2));
end

function distance = hammingDistance(str1, str2)
distance = nnz(str1 ~= str2);
end

```

该函数返回一个 Diagnostic 对象和相关比较信息。在包含 HammingDistance 类定义的 methods 块中，包含返回 StringDiagnostic 的以下方法。容差类必须实现 getDiagnosticFor 方法。

```

methods
    function diag = getDiagnosticFor(tolerance, actual, expected)
        import matlab.unittest.diagnostics.StringDiagnostic

        if ~isSameSize(actual.Sequence, expected.Sequence)
            str = 'The DNA sequences must be the same length.';
        else
            str = sprintf('%s%d.\n%s%d.', ...
                'The DNA sequences have a Hamming distance of ', ...
                hammingDistance(actual.Sequence, expected.Sequence), ...
                'The allowable distance is ', ...
                tolerance.Value);
        end
        diag = StringDiagnostic(str);
    end
end

```

### HammingDistance 类定义总结

```

classdef HammingDistance < matlab.unittest.constraints.Tolerance
    properties
        Value
    end

    methods
        function tolerance = HammingDistance(value)
            tolerance.Value = value;
        end

        function tf = supports(~, value)
            tf = isa(value, 'DNA');
        end

        function tf = satisfiedBy(tolerance, actual, expected)
            if ~isSameSize(actual.Sequence, expected.Sequence)

```

```

tf = false;
return
end
tf = hammingDistance(actual.Sequence,expected.Sequence) <= tolerance.Value;
end

function diag = getDiagnosticFor(tolerance, actual, expected)
import matlab.unittest.diagnostics.StringDiagnostic

if ~isSameSize(actual.Sequence, expected.Sequence)
    str = 'The DNA sequences must be the same length.';
else
    str = sprintf('%s%d.\n%s%d.', ...
        'The DNA sequences have a Hamming distance of ', ...
        hammingDistance(actual.Sequence, expected.Sequence), ...
        'The allowable distance is ', ...
        tolerance.Value);
end
diag = StringDiagnostic(str);
end
end
end

function tf = isSameSize(str1, str2)
tf = isequal(size(str1), size(str2));
end

function distance = hammingDistance(str1, str2)
distance = nnz(str1 ~= str2);
end

```

在命令提示符处，创建一个供交互测试的 TestCase。

```

import matlab.unittest.TestCase
import matlab.unittest.constraints.AreEqual

testCase = TestCase.forInteractiveUse;

```

创建两个 DNA 对象。

```

sampleA = DNA('ACCTGAGTA');
sampleB = DNA('ACCACAGTA');

```

验证 DNA 序列是否彼此相等。

```

testCase.verifyThat(sampleA, isEqualTo(sampleB))

```

Interactive verification failed.

---

Framework Diagnostic:

---

```

IsEqualTo failed.
-> ObjectComparator failed.
--> The objects are not equal using "isequal".

```

Actual Object:

DNA with properties:

Sequence: 'ACCTGAGTA'  
Expected Object:  
DNA with properties:

Sequence: 'ACCACAGTA'

验证在 Hamming 距离为 1 的范围内 DNA 序列是否彼此相等。

```
testCase.verifyThat(sampleA, isEqualTo(sampleB,...  
    'Within', HammingDistance(1)))
```

Interactive verification failed.

---

Framework Diagnostic:

---

isEqualTo failed.  
-> ObjectComparator failed.  
--> The objects are not equal using "isequal".  
--> The DNA sequences have a Hamming distance of 2.  
The allowable distance is 1.

Actual Object:  
DNA with properties:

Sequence: 'ACCTGAGTA'  
Expected Object:  
DNA with properties:

Sequence: 'ACCACAGTA'

该序列在容差为 1 的范围内彼此不相等。测试框架通过 `getDiagnosticFor` 方法显示其他诊断。

验证在 Hamming 距离为 2 的范围内 DNA 序列是否彼此相等。

```
testCase.verifyThat(sampleA, isEqualTo(sampleB,...  
    'Within', HammingDistance(2)))
```

Interactive verification passed.

### 另请参阅

`matlab.unittest.constraints.Tolerance`

# App 测试框架概述

使用 MATLAB App 测试框架来测试通过 App 设计工具创建的 App，或测试使用 `uifigure` 函数以编程方式构建的 App。您可以使用 App 测试框架来编写测试类，这些测试类以编程方式对 UI 组件执行手势（如按下按钮或拖动滑块）操作并验证 App 的行为。

## App 测试

**测试创建** - 基于类的测试可以通过子类化 `matlab.uitest.TestCase` 来使用 App 测试框架。

`matlab.uitest.TestCase` 是 `matlab.unittest.TestCase` 的一个子类，因此您的测试可以使用单元测试框架的功能，例如验证、脚手架和插件。要在命令提示符下试用 App 测试框架，请使用 `matlab.uitest.TestCase.forInteractiveUse` 创建一个测试用例实例。

**测试内容** - 通常，App 的测试以编程方式使用 `matlab.uitest.TestCase` 的某一手势方法（如 `press` 或 `drag`）与 App 组件进行交互，然后对结果执行验证。例如，测试可能会按下一个复选框并验证其他复选框已禁用。它也可能在文本框中键入数字，然后验证 App 是否正确计算出结果。这些类型的测试要求您对被测 App 的属性有一定了解。要验证按钮按下操作，您必须知道 MATLAB 将按钮的状态存储在 App 对象中的哪个位置。要验证计算结果，您必须知道如何在 App 中访问结果。

**测试清理** - 最好包含用于在测试后删除 App 的拆解操作。通常，测试方法使用 `matlab.unittest.TestCase` 的 `addTeardown` 方法添加此操作。

**App 锁定** - 当 App 测试创建图窗时，框架会立即锁定图窗以防止与组件的外部交互。如果您在命令提示符下创建用于试验的 `matlab.uitest.TestCase.forInteractiveUse` 实例，则 App 测试框架不会锁定 UI 组件。

要解锁图窗以进行调试，请使用 `matlab.uitest.unlock` 函数。

## UI 组件的手势支持

`matlab.uitest.TestCase` 的手势方法支持各种 UI 组件。

组件	<code>matlab.uitest.TestCase</code> 手势方法				
	<code>press</code>	<code>choose</code>	<code>drag</code>	<code>type</code>	<code>hover</code>
坐标区	✗				✗
按钮	✗				
按钮组		✗			
复选框	✗	✗			
日期选择器				✗	
分档旋钮		✗			
下拉列表		✗		✗	
编辑字段（数值、文本）				✗	
图像	✗				
旋钮		✗	✗		
列表框		✗			
菜单	✗				

极坐标区	✗				✗
单选按钮	✗	✗			
滑块		✗	✗		
微调器	✗			✗	
状态按钮	✗	✗			
开关 (跷板、滑块、拨动)	✗	✗			
制表符		✗			
选项卡组		✗			
文本区域				✗	
切换按钮	✗	✗			
树节点		✗			
UI 坐标区	✗				✗
UI 图窗	✗				✗

## 为 App 编写测试

此示例说明如何为 App 编写测试，该 App 提供用于更改绘图的样本大小和颜色图的选项。要以编程方式与 App 交互并验证结果，请结合使用单元测试框架和 App 测试框架。

要在测试之前了解此 App 的属性，请在命令提示符下创建该 App 的实例。

```
app = ConfigurePlotAppExample;
```

此步骤对于测试不是必需的，但是了解 App 测试使用的属性会很有帮助。例如，使用 `app.UpdatePlotButton` 访问 App 对象内的 **Update Plot** 按钮。

创建一个从 `matlab.unittest.TestCase` 继承的测试类。

```
classdef testConfigurePlotAppExample < matlab.unittest.TestCase
    methods (Test)
        end
    end
```

创建一个测试方法 `test_SampleSize` 以测试样本大小。该测试方法修改样本大小、更新绘图并验证曲面使用了指定的样本大小。测试完成后，对 `addTeardown` 的调用会删除该 App。

```
classdef testConfigurePlotAppExample < matlab.unittest.TestCase
    methods (Test)
        function test_SampleSize(testCase)
            app = ConfigurePlotAppExample;
            testCase.addTeardown(@delete,app);

            testCase.type(app.SampleSizeEditField,12);
            testCase.press(app.UpdatePlotButton);

            ax = app.UIAxes;
```

```

    surfaceObj = ax.Children;
    testCase.verifySize(surfaceObj.ZData,[12 12]);
end
end

end

```

创建另一个测试方法 `test_Colormap` 来测试颜色图。该测试方法选择一个颜色图、更新绘图并验证该绘图使用了指定的颜色图。完整的代码现在如下所示。

```

classdef testConfigurePlotAppExample < matlab.unittest.TestCase

methods (Test)
    function test_SampleSize(testCase)
        app = ConfigurePlotAppExample;
        testCase.addTeardown(@delete,app);

        testCase.type(app.SampleSizeEditField,12);
        testCase.press(app.UpdatePlotButton);

        ax = app.UIAxes;
        surfaceObj = ax.Children;
        testCase.verifySize(surfaceObj.ZData,[12 12]);
    end

    function test_Colormap(testCase)
        app = ConfigurePlotAppExample;
        testCase.addTeardown(@delete,app);

        testCase.choose(app.ColormapDropDown,'Winter');
        testCase.press(app.UpdatePlotButton);

        expectedMap = winter;
        ax = app.UIAxes;
        testCase.verifyEqual(ax.Colormap,expectedMap);
    end
end

```

在命令提示符下运行这些测试。

```

results = runtests('testConfigurePlotAppExample')

Running testConfigurePlotAppExample
..
Done testConfigurePlotAppExample

```

---

```
results =
```

```
1×2 TestResult array with properties:
```

```
Name
Passed
Failed
Incomplete
```

Duration  
Details

Totals:

2 Passed, 0 Failed, 0 Incomplete.

4.7551 seconds testing time.

### 另请参阅

`matlab.uitest.TestCase`

### 详细信息

- “为 App 编写测试” (第 34-133 页)
- “编写使用 App 测试和模拟框架的测试” (第 34-137 页)
- “App 设计工具组件”

# 为 App 编写测试

此示例说明如何为通过 App 设计工具创建的 App 编写测试。要以编程方式与 App 交互并验证结果，请使用 App 测试框架和单元测试框架。

要在测试之前了解此 App 的属性，请在命令提示符下创建该 App 的实例。

```
app = PatientsDisplay;
```

此步骤对于测试不是必需的，但是了解 App 测试使用的属性会很有帮助。例如，使用 `app.BloodPressureSwitch` 访问 App 对象内的 **Blood Pressure** 开关。

创建一个从 `matlab.unittest.TestCase` 继承的测试类。要测试选项卡切换功能，请创建一个测试方法 `test_tab`。该测试方法选择 **Data** 选项卡，然后验证所选的选项卡是否具有正确的标题。

`TestMethodSetup` 方法会为每个测试创建一个 App，并在测试完成后将其删除。

```
classdef TestPatientsDisplay < matlab.unittest.TestCase
    properties
        App
    end

    methods (TestMethodSetup)
        function launchApp(testCase)
            testCase.App = PatientsDisplay;
            testCase.addTeardown(@delete,testCase.App);
        end
    end

    methods (Test)
        function test_tab(testCase)
            % Choose Data Tab
            dataTab = testCase.App.Tab2;
            testCase.choose(dataTab)

            % Verify Data Tab is selected
            testCase.verifyEqual(testCase.App.TabGroup.SelectedTab.Title,'Data')
        end
    end
end
```

创建一个用于测试各种绘图选项的 `test_plottingOptions` 方法。该测试方法按下 **Histogram** 单选按钮并验证 **x** 标签已更改。然后，它会更改 **Bin Width** 滑块并验证 bin 的数量。

```
classdef TestPatientsDisplay < matlab.unittest.TestCase
    properties
        App
    end

    methods (TestMethodSetup)
        function launchApp(testCase)
            testCase.App = PatientsDisplay;
            testCase.addTeardown(@delete,testCase.App);
        end
    end

    methods (Test)
        function test_plottingOptions(testCase)
            % Press the histogram radio button
            testCase.press(testCase.App.HistogramButton)

            % Verify xlabel updated from 'Weight' to 'Systolic'
            testCase.verifyEqual(testCase.App.UIAxes.XLabel.String,'Systolic')

            % Change the Bin Width to 9
            testCase.choose(testCase.App.BinWidthSlider,9)

            % Verify the number of bins is now 4
            testCase.verifyEqual(testCase.App.UIAxes.Children.NumBins,4)
        end

        function test_tab(testCase) ...
```

```
    end
end
```

创建一个用于测试血压数据和显示的 `test_bloodPressure` 方法。该测试方法验证 y 轴标签和散点的值。然后它切换到 `Diastolic` 读数，并再次验证标签和数据。

```
classdef TestPatientsDisplay < matlab.unittest.TestCase
properties
    App
end

methods (TestMethodSetup)
    function launchApp(testCase)
        testCase.App = PatientsDisplay;
        testCase.addTeardown(@delete,testCase.App);
    end
end

methods (Test)
    function test_bloodPressure(testCase)
        % Extract blood pressure data from app
        t = cell2table(testCase.App.Tab2.Children.Data);
        t.Var2 = categorical(t.Var2);
        allMales = t(t.Var2==['Male']);%
        maleDiastolicData = allMales.Var7';
        maleSystolicData = allMales.Var8';

        % Verify ylabel and that male Systolic data shows
        ax = testCase.App.UIAxes;
        testCase.verifyEqual(ax.YLabel.String,'Systolic')
        testCase.verifyEqual(ax.Children.YData,maleSystolicData)

        % Switch to 'Diastolic' reading
        testCase.choose(testCase.App.BloodPressureSwitch,'Diastolic')

        % Verify ylabel changed and male Diastolic data shows
        testCase.verifyEqual(ax.YLabel.String,'Diastolic')
        testCase.verifyEqual(ax.Children.YData,maleDiastolicData);
    end

    function test_plottingOptions(testCase) ...
    function test_tab(testCase) ...
end
end
```

创建一个用于测试性别数据和显示的 `test_gender` 方法。该测试方法验证男性散点的数量，然后启用相应复选框以包含女性数据。它验证绘制了两个数据集，并且女性数据的颜色为红色。最后，它清除男性数据复选框并验证绘图数据集和散点的数量。此测试失败，因为有 53 个女性散点而不是 50 个。要在测试失败时进行屏幕截图，请使用 `ScreenshotDiagnostic` 并配合 `onFailure` 方法。

```
classdef TestPatientsDisplay < matlab.unittest.TestCase
properties
    App
end

methods (TestMethodSetup)
    function launchApp(testCase)
        testCase.App = PatientsDisplay;
        testCase.addTeardown(@delete,testCase.App);
    end
end

methods (Test)
    function test_gender(testCase)
        import matlab.unittest.diagnostics.ScreenshotDiagnostic
        testCase.onFailure(ScreenshotDiagnostic);

        % Verify 47 male scatter points
        ax = testCase.App.UIAxes;
        testCase.verifyNumElements(ax.Children.XData,47);

        % Enable the checkbox for female data
        testCase.choose(testCase.App.FemaleCheckBox);

        % Verify two data sets display and the female data is red
        testCase.assertNumElements(ax.Children,2);
    end
end
```

```

testCase.verifyEqual(ax.Children(1).CData,[1 0 0]);

% Disable the male data
testCase.choose(testCase.App.MaleCheckBox,false);

% Verify one data set displays and number of scatter points
testCase.verifyNumElements(ax.Children,1);
testCase.verifyNumElements(ax.Children.XData,50);
end

function test_bloodPressure(testCase)
    % Extract blood pressure data from app
    t = cell2table(testCase.App.Tab2.Children.Data);
    t.Var2 = categorical(t.Var2);
    allMales = t(t.Var2=='Male,:);
    maleDiastolicData = allMales.Var7';
    maleSystolicData = allMales.Var8';

    % Verify ylabel and that male Systolic data shows
    ax = testCase.App.UIAxes;
    testCase.verifyEqual(ax.YLabel.String,'Systolic')
    testCase.verifyEqual(ax.Children.YData,maleSystolicData)

    % Switch to 'Diastolic' reading
    testCase.choose(testCase.App.BloodPressureSwitch,'Diastolic')

    % Verify ylabel changed and male Diastolic data shows
    testCase.verifyEqual(ax.YLabel.String,'Diastolic')
    testCase.verifyEqual(ax.Children.YData,maleDiastolicData);
end

function test_plottingOptions(testCase)
    % Press the histogram radio button
    testCase.press(testCase.App.HistogramButton)

    % Verify xlabel updated from 'Weight' to 'Systolic'
    testCase.verifyEqual(testCase.App.UIAxes.XLabel.String,'Systolic')

    % Change the Bin Width to 9
    testCase.choose(testCase.App.BinWidthSlider,9)

    % Verify the number of bins is now 4
    testCase.verifyEqual(testCase.App.UIAxes.Children.NumBins,4)
end

function test_tab(testCase)
    % Choose Data Tab
    dataTab = testCase.App.Tab2;
    testCase.choose(dataTab)

    % Verify Data Tab is selected
    testCase.verifyEqual(testCase.App.TabGroup.SelectedTab.Title,'Data')
end
end

```

运行测试。

```
results = runtests('TestPatientsDisplay');
```

```
Running TestPatientsDisplay
```

```
=====
Verification failed in TestPatientsDisplay/test_gender.
```

```
-----
Framework Diagnostic:
```

```
verifyNumElements failed.  
-> The value did not have the correct number of elements.
```

```
Actual Number of Elements:
```

```
53
```

```
Expected Number of Elements:
```

```
50
```

```
Actual Value:
```

Columns 1 through 49

131 133 119 142 142 132 128 137 129 131 133 117 137 146 123 143 114 126 137 138 137 118 128 135 121 136 135 1

Columns 50 through 53

141 129 124 134

---

Additional Diagnostic:

---

Screenshot captured to:

→ C:\Temp\54fd8dc0-0637-4926-9c4f-f217fe195fe1\Screenshot\_daba8870-adb3-4a1c-ba11-df3d9b51d36f.png

---

Stack Information:

---

In C:\Work\TestPatientsDisplay.m (TestPatientsDisplay.test\_gender) at 34

---

....

Done TestPatientsDisplay

---

Failure Summary:

Name	Failed	Incomplete	Reason(s)
TestPatientsDisplay/test_gender	X		Failed by verification.

### 另请参阅

**matlab.uitest.TestCase**

### 详细信息

- “App 测试框架概述” (第 34-129 页)
- “编写使用 App 测试和模拟框架的测试” (第 34-137 页)

# 编写使用 App 测试和模拟框架的测试

此示例说明如何编写使用 App 测试框架和模拟框架的测试。该 App 包含一个文件选择对话框和一个指示所选文件的标签。要以编程方式测试该 App，请使用一个 mock 对象来定义文件选择器的行为。

## 创建 App

在当前工作文件夹中创建 `launchApp` App。该 App 允许用户选择一个输入文件并在 App 中显示文件的名称。文件选择对话框是一个等待用户输入的阻断型模态对话框。

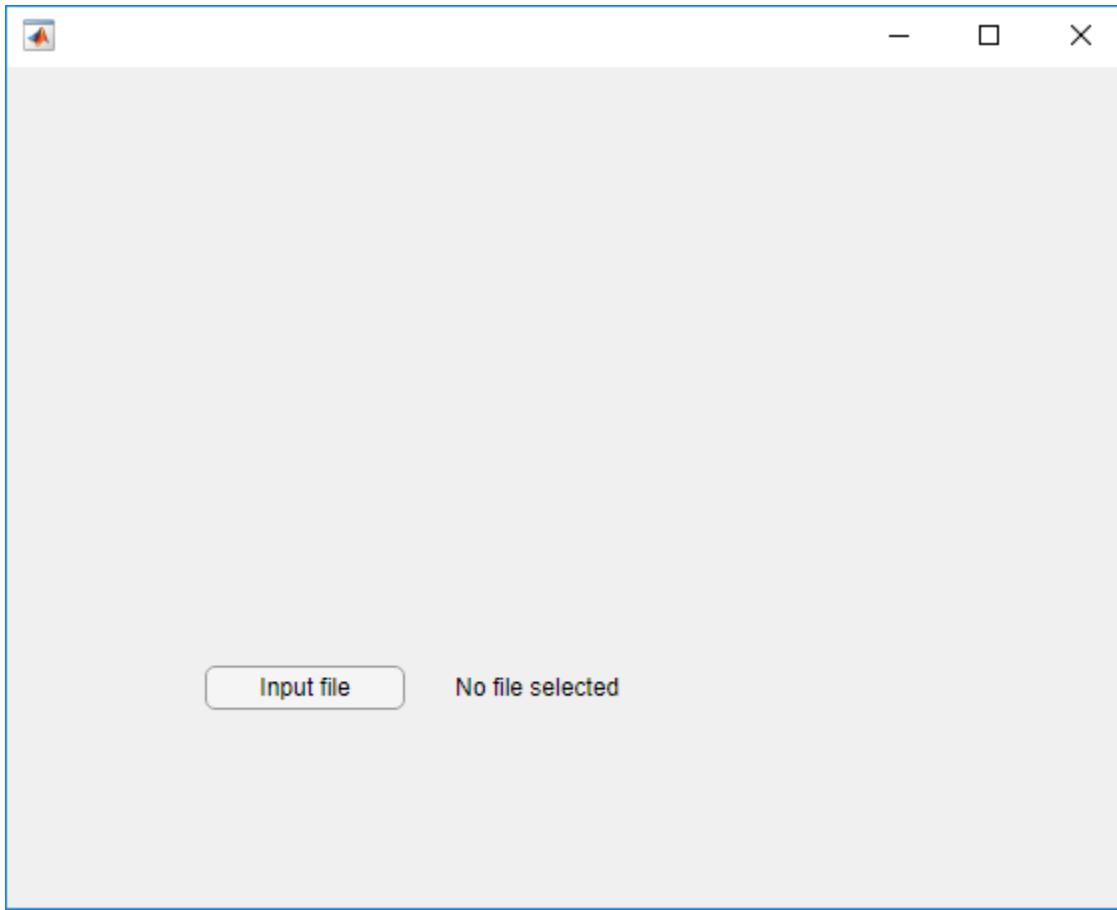
```
function app = launchApp
    f = uifigure;
    button = uibutton(f,'Text','Input file');
    button.ButtonPushedFcn = @(src,evt)pickFile;
    label = uilabel(f,'Text','No file selected');
    label.Position(1) = button.Position(1) + button.Position(3) + 25;
    label.Position(3) = 200;

    % Add components to an App struct for output
    app.UIFigure = f;
    app.Button = button;
    app.Label = label;

    function file = pickFile()
        [file,folder,status] = uigetfile('*.*');
        if status
            label.Text = file;
        end
    end
end

要在测试之前了解此 App 的属性，请在命令提示符下创建该 App 的实例。此步骤对于测试不是必需的，但是了解 App 测试使用的属性会很有帮助。例如，使用 app.Button 访问 App 对象内的 Input file 按钮。
```

```
app = launchApp;
```



## 在手动干预下测试 App

在不使用 mock 的情况下创建 `LaunchAppTest` 类。该测试假定文件 `input2.txt` 存在于当前工作文件夹中。如果它不存在，请创建它。该测试以编程方式按下 **Input file** 按钮，并验证标签是否匹配 '`input2.txt`'。您必须手动选择文件。

```
classdef LaunchAppTest < matlab.unittest.TestCase
    properties
        TestFile = 'input2.txt';
    end
    methods(TestClassSetup)
        function checkTestFiles(tc)
            import matlab.unittest.constraints.IsFalse
            tc.assertThat(tc.TestFile,IsFalse)
        end
    end
    methods (Test)
        function testInputButton(tc)
            app = launchApp;
            tc.addTeardown(@close,app.UIFigure);
            tc.press(app.Button);
            tc.verifyEqual(app.Label.Text,tc.TestFile)
        end
    end
```

```

    end
end
end

```

运行测试。当文件选择对话框出现时，选择 `input2.txt` 以允许 MATLAB 继续测试。选择任何其他文件会导致测试失败。

```

results = runtests('LaunchAppTest');

Running LaunchAppTest
.
Done LaunchAppTest

```

## 创建全自动测试

要测试 App 而无需手动干预，请使用模拟框架。将 App 修改为接受文件选择服务，而不是在 App 中实现该服务（依赖项注入）。

使用 `Abstract` 方法创建一个 `FileChooser` 服务，用于实现文件选择功能。

```

classdef FileChooser
    % Interface to choose a file
    methods (Abstract)
        [file, folder, status] = chooseFile(chooser, varargin)
    end
end

```

创建一个默认 `FileChooser`，它使用 `uigetfile` 函数进行文件选择。

```

classdef DefaultFileChooser < FileChooser
    methods
        function [file, folder, status] = chooseFile(chooser, varargin)
            [file, folder, status] = uigetfile(varargin{:});
        end
    end
end

```

将 App 更改为接受可选的 `FileChooser` 对象。当在没有输入的情况下调用时，该 App 使用 `DefaultFileChooser` 的实例。

```

function app = launchApp(fileChooser)
    if nargin==0
        fileChooser = DefaultFileChooser;
    end
    f = uifigure;
    button = uibutton(f,'Text','Input file');
    button.ButtonPushedFcn = @(src,evt)pickFile(fileChooser);
    label = uilabel(f,'Text','No file selected');
    label.Position(1) = button.Position(1) + button.Position(3) + 25;
    label.Position(3) = 200;

    % Add components to an App struct for output
    app.UIFigure = f;
    app.Button = button;
    app.Label = label;

```

```

function file = pickFile(fileChooser)
    [file, folder, status] = fileChooser.chooseFile('*.*');
    if status
        label.Text = file;
    end
end
end

```

对 `LaunchAppTest` 进行以下修改。

- 将测试更改为同时从 `matlab.uitest.TestCase` 和 `matlab.mock.TestCase` 继承。
- 删除 `properties` 代码块和 `TestClassSetup` 代码块。由于 mock 定义 `chooseFile` 方法调用的相应输出，因此测试不依赖于外部文件的存在。
- 更改 `testInputButton` 测试方法，以便它执行下列操作。
  - 创建 `FileChooser` 的一个 mock 对象。
  - 定义 mock 行为，以便在使用输入 `'*.*'` 调用 `chooseFile` 方法时，输出为测试文件名 (`'input2.txt'`)、当前工作文件夹和选定的筛选器索引 1。这些输出类似于 `uigetfile` 函数的输出。
  - 按下按钮并验证所选文件名称。这些步骤与原始测试中的步骤相同，但 mock 会分配输出值，因此您无需与 App 交互即可继续测试。
- 要测试 **Cancel** 按钮，请添加测试方法 `testInputButton_Cancel`，以便它执行下列操作。
  - 创建 `FileChooser` 的一个 mock 对象。
  - 定义 mock 行为，以便在使用输入 `'*.*'` 调用 `chooseFile` 方法时，输出为测试文件名 (`'input2.txt'`)、当前工作文件夹和选定的筛选器索引 0。这些输出类似于在用户选择一个文件然后选择取消时 `uigetfile` 函数的输出。
  - 按下按钮并验证测试调用 `chooseFile` 方法，并且标签指示没有选择文件。

```

classdef LaunchAppTest < matlab.uitest.TestCase & matlab.mock.TestCase
methods (Test)
    function testInputButton(tc)
        import matlab.mock.actions.AssignOutputs
        fname = 'myFile.txt';

        [mockChooser, behavior] = tc.createMock(?FileChooser);
        when(behavior.chooseFile('*.*'), AssignOutputs(fname, pwd, 1))

        app = launchApp(mockChooser);
        tc.addTeardown(@close, app.UIFigure);

        tc.press(app.Button);

        tc.verifyEqual(app.Label.Text, fname);
    end

    function testInputButton_Cancel(tc)
        import matlab.mock.actions.AssignOutputs

        [mockChooser, behavior] = tc.createMock(?FileChooser);
        when(behavior.chooseFile('*.*'), AssignOutputs('myFile.txt', pwd, 0))

        app = launchApp(mockChooser);
        tc.addTeardown(@close, app.UIFigure);

        tc.press(app.Button);

        tc.verifyCalled(behavior.chooseFile('*.*'));
        tc.verifyEqual(app.Label.Text, 'No file selected');
    end
end

```

运行测试。测试运行完成，无需手动选择文件。

```
results = runtests('LaunchAppTest');

Running LaunchAppTest
..
Done LaunchAppTest
```

---

## 另请参阅

[matlab.mock.TestCase](#) | [matlab.uitest.TestCase](#)

## 详细信息

- “App 测试框架概述” (第 34-129 页)
- “为 App 编写测试” (第 34-133 页)
- “创建 Mock 对象” (第 34-158 页)

## 性能测试框架概述

本节内容
“确定已测量代码的范围”（第 34-142 页）
“计时试验的类型”（第 34-142 页）
“编写具有测量范围的性能测试”（第 34-143 页）
“运行性能测试”（第 34-143 页）
“理解无效测试结果”（第 34-144 页）

性能测试接口利用脚本、函数和基于类的单元测试接口。您可以在性能测试中执行验证，以确保在度量代码性能时功能行为正确。此外，您还可以将性能测试作为标准回归测试来运行，以确保代码更改不会中断性能测试。

### 确定已测量代码的范围

此表介绍为不同类型的测试测量的代码。

测试类型	测量内容	排除项
基于脚本	每节脚本中的代码	<ul style="list-style-type: none"> <li>共享变量节中的代码</li> <li>测量的框架开销估计值</li> </ul>
基于函数	每个测试函数中的代码	<ul style="list-style-type: none"> <li>以下函数中的代码：<code>setup</code>、<code>setupOnce</code>、<code>teardown</code> 和 <code>teardownOnce</code></li> <li>测量的框架开销估计值</li> </ul>
基于类	每个带 <code>Test</code> 属性标记的方法中的代码	<ul style="list-style-type: none"> <li>带有以下属性的方法中的代码：<code>TestMethodSetup</code>、<code>TestMethodTeardown</code>、<code>TestClassSetup</code> 和 <code>TestClassTeardown</code></li> <li>共享脚手架设置和拆解</li> <li>测量的框架开销估计值</li> </ul>
基于类，从 <code>matlab.perftest.TestCase</code> 派生并使用 <code>startMeasuring</code> 和 <code>stopMeasuring</code> 方法	每个带有 <code>Test</code> 属性的方法中 <code>startMeasuring</code> 和 <code>stopMeasuring</code> 调用之间的代码	<ul style="list-style-type: none"> <li><code>startMeasuring/stopMeasuring</code> 边界外的代码</li> <li>测量的框架开销估计值</li> </ul>
基于类，从 <code>matlab.perftest.TestCase</code> 派生并使用 <code>keepMeasuring</code> 方法	每个带有 <code>Test</code> 属性标记的方法中的每个 <code>keepMeasuring-while</code> 循环中的代码	<ul style="list-style-type: none"> <li><code>keepMeasuring-while</code> 边界外的代码</li> <li>测量的框架开销估计值</li> </ul>

### 计时试验的类型

您可以创建两种类型的计时试验。

- 不定次计时试验会采集不定个数的测量值，以达到指定的误差界限和置信水平。不定次计时试验用于定义测量样本的统计目标。使用 `runperf` 函数或 `TimeExperiment` 类的 `limitingSamplingError` 静态方法生成此试验。

- 定次计时试验会采集固定个数的测量值。定次计时试验可用于测量代码的首次成本或明确控制样本大小。使用 `TimeExperiment` 类的 `withFixedSampleSize` 静态方法生成此试验。

此表汇总了不定次计时试验和定次计时试验的差别。

	不定次计时试验	定次计时试验
预备测量数	默认为 4，但可通过 <code>TimeExperiment.limitingSamplingError</code> 配置	默认为 0，但可通过 <code>TimeExperiment.withFixedSampleSize</code> 配置
样本数量	默认介于 4 和 256 之间，但可通过 <code>TimeExperiment.limitingSamplingError</code> 配置	在试验构造期间定义
相对误差界限	默认为 5%，但可通过 <code>TimeExperiment.limitingSamplingError</code> 配置	不适用
置信水平	默认为 95%，但可通过 <code>TimeExperiment.limitingSamplingError</code> 配置	不适用
无效测试结果的框架行为	停止测量测试并转至下一个	采集指定数量的样本

## 编写具有测量范围的性能测试

如果基于类的测试派生自 `matlab.perftest.TestCase` 而非 `matlab.unittest.TestCase`，则可以多次使用 `startMeasuring` 与 `stopMeasuring` 或 `keepMeasuring` 方法，以定义性能测试测量边界。如果测试方法多次调用 `startMeasuring`、`stopMeasuring` 和 `keepMeasuring`，则性能框架会对测量值进行累加并求和。性能框架不支持嵌套的测量边界。如果您在 `Test` 方法中未能正确使用这些方法，且以 `TimeExperiment` 方式运行测试，则框架会将测量值标记为无效。另外，您仍可将这些性能测试作为单元测试运行。有关详细信息，请参阅“使用类测试性能”（第 34-149 页）。

## 运行性能测试

运行性能测试的方法有两种：

- 使用 `runperf` 函数运行测试。此函数使用不定数目的测量值获得 0.95 置信水平的相对误差界限为 0.05 的样本均值。它会运行这些测试四次来预备该代码，并运行 4 到 256 次来采集符合统计目标的测量值。
- 使用 `TestSuite` 类中的 `testsuite` 函数或方法生成显式测试套件，然后创建并运行计时试验。
  - 使用 `TimeExperiment` 类的 `withFixedSampleSize` 方法构造采集固定个数的测量值的计时试验。您可以指定固定数量的预备测量值和固定数量的样本。
  - 使用 `TimeExperiment` 类的 `limitingSamplingError` 方法构造一个具有指定统计目标（例如误差界限和置信水平）的计时试验。此外，您还可以指定预备测量值的数目以及最小和最大样本数。

您可以将性能测试作为回归测试运行。有关详细信息，请参阅“为各个工作流运行测试”（第 34-76 页）。

## 理解无效测试结果

在某些情况下，测试结果的 `MeasurementResult` 标为无效。当性能测试框架将 `MeasurementResult` 的 `Valid` 属性设置为 `false` 时，测试结果将标为无效。如果您的测试失败或被筛选掉，则会发生这种失效。另外，如果您的测试未正确使用 `matlab.perftest.TestCase` 的 `startMeasuring` 和 `stopMeasuring` 方法，则该测试的 `MeasurementResult` 将标为无效。

当性能测试框架遇到无效测试结果时，它的行为方式会有所不同，具体取决于计时试验的类型：

- 如果您创建的是不定次计时试验，则该框架会停止对该测试进行测量并移至下一个测试。
- 如果您创建的是定次计时试验，则该框架会继续采集指定数量的样本。

## 另请参阅

`matlab.perftest.TimeExperiment` | `matlab.unittest.measurement.MeasurementResult` |  
`runperf` | `testsuite`

## 相关示例

- “使用脚本或函数测试性能”（第 34-145 页）
- “使用类测试性能”（第 34-149 页）

# 使用脚本或函数测试性能

以下示例演示如何创建基于脚本或函数的性能测试，使用四种不同方法计算向量预分配耗用的时间。

## 编写性能测试

在您的当前工作文件夹下的文件中创建一个性能测试 `preallocationTest.m`。在此示例中，您可以选择使用以下基于脚本的测试或基于函数的测试。此示例中的输出适用于基于函数的测试。如果您使用基于脚本的测试，则测试名称会有所不同。

基于脚本的性能测试	基于函数的性能测试
<pre>vectorSize = 1e7;  %% Ones Function x = ones(1,vectorSize);  %% Indexing With Variable id = 1:vectorSize; x(id) = 1;  %% Indexing On LHS x(1:vectorSize) = 1;  %% For Loop for i=1:vectorSize     x(i) = 1; end</pre>	<pre>function tests = preallocationTest tests = functiontests(localfunctions); end  function testOnes(testCase) vectorSize = getSize(); x = ones(1,vectorSize()); end  function testIndexingWithVariable(testCase) vectorSize = getSize(); id = 1:vectorSize; x(id) = 1; end  function testIndexingOnLHS(testCase) vectorSize = getSize(); x(1:vectorSize) = 1; end  function testForLoop(testCase) vectorSize = getSize(); for i=1:vectorSize     x(i) = 1; end end  function vectorSize = getSize() vectorSize = 1e7; end</pre>

## 运行性能测试

使用 `runperf` 运行性能测试。

```
results = runperf('preallocationTest.m')
```

```
Running preallocationTest
```

```
.....
```

```
Done preallocationTest
```

---

```
results =
```

```
1×4 TimeResult array with properties:
```

```
Name
Valid
Samples
TestActivity
```

Totals:

```
4 Valid, 0 Invalid.
10.2561 seconds testing time.
```

**results** 变量是  $1 \times 4$  TimeResult 数组。数组中的每个元素对应于 **preallocationTest.m** 的代码节中定义的一个测试。

### 显示测试结果

显示第二个测试的测量结果。您的结果可能有所不同。

**results(2)**

```
ans =
```

TimeResult with properties:

```
Name: 'preallocationTest/testIndexingWithVariable'
Valid: 1
Samples: [4x7 table]
TestActivity: [8x12 table]
```

Totals:

```
1 Valid, 0 Invalid.
1.2274 seconds testing time.
```

如 **TestActivity** 属性的大小所示，性能测试框架采集了 8 个测量值。此测量值个数包括四个测量值来预备代码。**Samples** 属性会排除预备测量值。

显示第二个测试的样本测量值。

**results(2).Samples**

```
>> results(2).Samples
```

```
ans =
```

4×7 table

Name	MeasuredTime	Timestamp	Host	Platform	Version
preallocationTest/testIndexingWithVariable	0.15271	24-Jun-2019 16:13:33	MY-HOSTNAME	win64	9
preallocationTest/testIndexingWithVariable	0.15285	24-Jun-2019 16:13:33	MY-HOSTNAME	win64	9
preallocationTest/testIndexingWithVariable	0.15266	24-Jun-2019 16:13:33	MY-HOSTNAME	win64	9
preallocationTest/testIndexingWithVariable	0.15539	24-Jun-2019 16:13:34	MY-HOSTNAME	win64	9

### 计算单个测试元素的统计信息

显示第二个测试的测量时间均值。要排除在预备运行中采集的数据，请使用 **Samples** 字段中的值。

```
sampleTimes = results(2).Samples.MeasuredTime;
meanTest2 = mean(sampleTimes)
```

```
meanTest2 =
```

```
0.1534
```

性能测试框架为第二个测试采集了 4 个样本测量值。该测试的平均运行时间为 **0.1534** 秒。

### 计算所有测试元素的统计信息

确定所有测试元素的平均时间。**preallocationTest** 测试包括四种不同方法，用于分配由 1 组成的向量。比较每个方法（测试元素）的时间。

因为性能测试框架为每个测试元素返回一个 **Samples** 表，并将所有这些表串联成一个表。然后按测试元素 **Name** 对行分组，并计算每个组的平均 **MeasuredTime**。

```
fullTable = vertcat(results.Samples);
summaryStats = varfun(@mean,fullTable,...  
    'InputVariables','MeasuredTime','GroupingVariables','Name')
```

```
summaryStats =
```

```
4×3 table
```

Name	GroupCount	mean_MeasuredTime
preallocationTest/testOnes	4	0.041072
preallocationTest/testIndexingWithVariable	4	0.1534
preallocationTest/testIndexingOnLHS	4	0.04677
preallocationTest/testForLoop	4	1.0343

### 更改统计目标并重新运行测试

通过构造和运行计时试验来更改 **runperf** 函数定义的统计目标。构造所采集测量值的样本均值达到 97% 置信水平的 8% 相对误差界限目标的计时试验。

构造一个显式测试套件。

```
suite = testsuite('preallocationTest');
```

构造一个采集不定测量值样本数的计时试验，并运行这些测试。

```
import matlab.perftest.TimeExperiment
experiment = TimeExperiment.limitingSamplingError('NumWarmups',2,...  
    'RelativeMarginOfError',0.08, 'ConfidenceLevel', 0.97);
resultsTE = run(experiment,suite);
```

```
Running preallocationTest
```

```
.....
```

```
Done preallocationTest
```

计算所有测试元素的统计信息。

```
fullTableTE = vertcat(resultsTE.Samples);
summaryStatsTE = varfun(@mean,fullTableTE,...  
    'InputVariables','MeasuredTime','GroupingVariables','Name')
```

```
summaryStatsTE =
```

4×3 table

Name	GroupCount	mean_MeasuredTime
preallocationTest/testOnes	4	0.040484
preallocationTest/testIndexingWithVariable	4	0.15187
preallocationTest/testIndexingOnLHS	4	0.046224
preallocationTest/testForLoop	4	1.0262

**另请参阅**

[matlab.perftest.TimeExperiment](#) | [matlab.perftest.TimeResult](#) |  
[matlab.unittest.measurement.DefaultMeasurementResult](#) | [runperf](#) | [testsuite](#)

# 使用类测试性能

此示例说明如何为 `fprintf` 函数创建性能测试和回归测试。

## 编写性能测试

考虑以下单元（回归）测试。您可以使用 `runperf('fprintfTest')` 而非 `runtests('fprintfTest')` 将此测试作为性能测试运行。

```
classdef fprintfTest < matlab.unittest.TestCase
    properties
        file
        fid
    end
    methods(TestMethodSetup)
        function openFile(testCase)
            testCase.file = tempname;
            testCase.fid = fopen(testCase.file,'w');
            testCase.assertNotEqual(testCase.fid,-1,'IO Problem')

            testCase.addTeardown(@delete,testCase.file);
            testCase.addTeardown(@fclose,testCase.fid);
        end
    end

    methods(Test)
        function testPrintingToFile(testCase)
            textToWrite = repmat('abcdef',1,5000000);
            fprintf(testCase.fid,'%s',textToWrite);
            testCase.verifyEqual(fileread(testCase.file),textToWrite)
        end

        function testBytesToFile(testCase)
            textToWrite = repmat('tests_',1,5000000);
            nbytes = fprintf(testCase.fid,'%s',textToWrite);
            testCase.verifyEqual(nbytes,length(textToWrite))
        end
    end
end
```

测量的时间不包括打开和关闭文件的时间或断言，因为这些活动发生在 `TestMethodSetup` 块（而非 `Test` 块）内。但是，测量的时间包括执行验证的时间。最佳做法是测量更准确的性能边界。

在您的当前工作文件夹下的文件中创建一个性能测试 `fprintfTest.m`。此测试类似于包含以下修改的回归测试：

- 该测试派生自 `matlab.perftest.TestCase` 而非 `matlab.unittest.TestCase`。
- 该测试会调用 `startMeasuring` 和 `stopMeasuring` 方法来创建一个围绕 `fprintf` 函数调用过程的范围。

```
classdef fprintfTest < matlab.perftest.TestCase
    properties
        file
        fid
    end
    methods(TestMethodSetup)
        function openFile(testCase)
```

```

testCase.file = tempname;
testCase.fid = fopen(testCase.file,'w');
testCase.assertEqual(testCase.fid,-1,'IO Problem')

testCase.addTeardown(@delete,testCase.file);
testCase.addTeardown(@fclose,testCase.fid);
end
end

methods(Test)
function testPrintingToFile(testCase)
textToWrite = repmat('abcdef',1,5000000);

testCase.startMeasuring();
fprintf(testCase.fid,"%s",textToWrite);
testCase.stopMeasuring();

testCase.verifyEqual(fileread(testCase.file),textToWrite)
end

function testBytesToFile(testCase)
textToWrite = repmat('tests_',1,5000000);

testCase.startMeasuring();
nbytes = fprintf(testCase.fid,"%s",textToWrite);
testCase.stopMeasuring();

testCase.verifyEqual(nbytes,length(textToWrite))
end
end
end

```

此性能测试的测量时间仅包括对 `fprintf` 的调用，并且测试框架仍会计算验证。

### 运行性能测试

运行性能测试。根据系统的不同，您可能会看到系统发出警告，提示性能测试框架运行了最大次数的测试，但未达到 0.95 置信水平的 0.05 的相对误差界限目标。

```
results = runperf('fprintfTest')
```

```
Running fprintfTest
```

```
.....
```

```
Done fprintfTest
```

```
results =
```

```
1x2 TimeResult array with properties:
```

Name
Valid
Samples
TestActivity

```
Totals:
```

```
2 Valid, 0 Invalid.  
4.1417 seconds testing time.
```

**results** 变量是  $1 \times 2$  **TimeResult** 数组。数组中的每个元素都对应于测试文件中定义的一个测试。

### 显示测试结果

显示第一个测试的测量结果。您的结果可能有所不同。

```
results(1)
```

```
ans =
```

```
TimeResult with properties:
```

```
Name: 'fprintfTest/testPrintingToFile'  
Valid: 1  
Samples: [4x7 table]  
TestActivity: [8x12 table]
```

Totals:

```
1 Valid, 0 Invalid.  
2.7124 seconds testing time.
```

如 **TestActivity** 属性的大小所示，性能测试框架采集了 8 个测量值。此数字包括 4 个用于预备代码的测量值。**Samples** 属性会排除预备测量值。

显示第一个测试的测量值样本。

```
results(1).Samples
```

```
ans =
```

```
4x7 table
```

Name	MeasuredTime	Timestamp	Host	Platform	Version
fprintfTest/testPrintingToFile	0.067729	24-Jun-2019 16:22:09	MY-HOSTNAME	win64	9.7.0.1141441 (
fprintfTest/testPrintingToFile	0.067513	24-Jun-2019 16:22:09	MY-HOSTNAME	win64	9.7.0.1141441 (
fprintfTest/testPrintingToFile	0.068737	24-Jun-2019 16:22:09	MY-HOSTNAME	win64	9.7.0.1141441 (
fprintfTest/testPrintingToFile	0.068576	24-Jun-2019 16:22:10	MY-HOSTNAME	win64	9.7.0.1141441 (

### 计算单个测试元素的统计信息

显示首个测试的测量时间均值。要排除在预备运行中采集的数据，请使用 **Samples** 字段中的值。

```
sampleTimes = results(1).Samples.MeasuredTime;  
meanTest = mean(sampleTimes)
```

```
meanTest =
```

```
0.0681
```

### 计算所有测试元素的统计信息

确定所有测试元素的平均时间。**fprintfTest** 测试包括两个不同的方法。比较每个方法（测试元素）的时间。

因为性能测试框架为每个测试元素返回一个 `Samples` 表，并将所有这些表串联成一个表。然后按测试元素 `Name` 对行分组，并计算每个组的平均 `MeasuredTime`。

```
fullTable = vertcat(results.Samples);
summaryStats = varfun(@mean,fullTable, ...
    'InputVariables','MeasuredTime','GroupingVariables','Name')

summaryStats =
2×3 table

    Name        GroupCount    mean_MeasuredTime
    ____        _____         _____
    fprintfTest/testPrintingToFile      4          0.068139
    fprintfTest/testBytesToFile        9          0.071595
```

两种测试方法均会将同量数据写入文件中。因此，均值间的部分差异归因于带有输出参数调用 `fprintf` 函数。

### 更改统计目标并重新运行测试

通过构造和运行计时试验来更改 `runperf` 函数定义的统计目标。构造所采集测量值的样本均值达到 97% 置信水平的 3% 相对误差界限目标的计时试验。采集 4 个预备测量值和最多 16 个样本测量值。

构造一个显式测试套件。

```
suite = testsuite('fprintfTest');
```

构造一个采集不定测量值样本数的计时试验，并运行这些测试。

```
import matlab.perftest.TimeExperiment
experiment = TimeExperiment.limitingSamplingError('NumWarmups',4, ...
    'MaxSamples',16,'RelativeMarginOfError',0.03,'ConfidenceLevel',0.97);
resultsTE = run(experiment,suite);
```

```
Running fprintfTest
..... .......Warning: Target Relative Margin of Error not met after running the MaxSamples for fprintfTest/test
.....
Done fprintfTest
```

在此示例输出中，性能测试框架采用指定的最大样本数无法达到更严格的统计目标。您的结果可能有所不同。

计算所有测试元素的统计信息。

```
fullTableTE = vertcat(resultsTE.Samples);
summaryStatsTE = varfun(@mean,fullTableTE, ...
    'InputVariables','MeasuredTime','GroupingVariables','Name')

summaryStatsTE =
2×3 table
```

Name	GroupCount	mean_MeasuredTime
fprintfTest/testPrintingToFile	16	0.069482
fprintfTest/testBytesToFile	4	0.067902

将最大样本数增加到 32 并重新运行计时试验。

```
experiment = TimeExperiment.limitingSamplingError('NumWarmups',4,...  
    'MaxSamples',32,'RelativeMarginOfError',0.03,'ConfidenceLevel',0.97);  
resultsTE = run(experiment,suite);
```

Running fprintfTest

.....  
Done fprintfTest

计算所有测试元素的统计信息。

```
fullTableTE = vertcat(resultsTE.Samples);  
summaryStatsTE = varfun(@mean,fullTableTE,...  
    'InputVariables','MeasuredTime','GroupingVariables','Name')  
summaryStatsTE =
```

2×3 table

Name	GroupCount	mean_MeasuredTime
fprintfTest/testPrintingToFile	4	0.067228
fprintfTest/testBytesToFile	4	0.067766

测试框架使用 4 个样本达到两个测试的统计目标。

### 测量首次成本

启动一个新 MATLAB 会话。新会话可确保 MATLAB 尚未运行测试中包含的代码。

创建并运行一个不用预备测量值，仅采用一个样本测量值的定次计时试验，测量代码的首次成本。

构造一个显式测试套件。因为您是在测量函数的首次成本，所以请运行单个测试。要运行多个测试，请保存相应结果并在测试间启动一个新 MATLAB 会话。

```
suite = testsuite('fprintfTest/testPrintingToFile');
```

构造并运行此计时试验。

```
import matlab.perftest.TimeExperiment  
experiment = TimeExperiment.withFixedSampleSize(1);  
results = run(experiment,suite);
```

Running fprintfTest

.  
Done fprintfTest

显示结果。观察 TestActivity 表以确保没有预备样本。

```
fullTable = results.TestActivity
```

fullTable =

1×12 table

Name	Passed	Failed	Incomplete	MeasuredTime	Objective	Timestamp	Host	Platform	Version
fprintfTest/testPrintingToFile	true	false	false	0.071754	sample	24-Jun-2019 16:31:27	MY-HOSTNAME	win64	9.7.0.1141441 (R2019b) P

性能测试框架为每个测试采集一个样本。

**另请参阅**

[matlab.perftest.TestCase](#) | [matlab.perftest.TimeExperiment](#) | [matlab.perftest.TimeResult](#) |  
[matlab.unittest.measurement.DefaultMeasurementResult](#) | [runperf](#) | [testsuite](#)

## 测量快速执行的测试代码

对于执行速度太快以至于 MATLAB 不能准确测量的性能测试，将通过假设失败将其滤除。使用 **keepMeasuring** 方法，测试框架可以通过自动确定代码迭代次数并测量平均性能来显著加快代码测量速度。

在您的当前工作文件夹中创建一个基于类的测试，**PreallocationTest.m**，用于对不同的预分配方法进行比较。由于测试方法中包括验证，因此可以使用 **startMeasuring** 和 **stopMeasuring** 方法来定义要测量的代码的边界。

```
classdef PreallocationTest < matlab.perftest.TestCase
methods(Test)
    function testOnes(testCase)
        testCase.startMeasuring
        x = ones(1,1e5);
        testCase.stopMeasuring
        testCase.verifyEqual(size(x),[1 1e5])
    end

    function testIndexingWithVariable(testCase)
        import matlab.unittest.constraints.IsSameSetAs
        testCase.startMeasuring
        id = 1:1e5;
        x(id) = 1;
        testCase.stopMeasuring
        testCase.verifyThat(x,IsSameSetAs(1))
    end

    function testIndexingOnLHS(testCase)
        import matlab.unittest.constraints.EveryElementOf
        import matlab.unittest.constraints.AreEqual
        testCase.startMeasuring
        x(1:1e5) = 1;
        testCase.stopMeasuring
        testCase.verifyThat(EveryElementOf(x),IsEqualTo(1))
    end

    function testForLoop(testCase)
        testCase.startMeasuring
        for i=1:1e5
            x(i) = 1;
        end
        testCase.stopMeasuring
        testCase.verifyNumElements(x,1e5)
    end
end
```

运行 **PreallocationTest** 作为性能测试。滤除两个测试，因为测量结果太接近框架的精度。

```
results = runperf('PreallocationTest');

Running PreallocationTest
.....
=====
PreallocationTest/testOnes was filtered.
    Test Diagnostic: The MeasuredTime should not be too close to the precision of the framework.
Details
=====
```

```

...
=====
PreallocationTest/testIndexingOnLHS was filtered.
  Test Diagnostic: The MeasuredTime should not be too close to the precision of the framework.
Details
=====
...
Done PreallocationTest

Failure Summary:

```

Name	Failed	Incomplete	Reason(s)
PreallocationTest/testOnes	X		Filtered by assumption.
PreallocationTest/testIndexingOnLHS	X		Filtered by assumption.

要指示框架自动遍历已测量的代码并计算测量结果平均值，请修改 `PreallocationTest` 以使用 `keepMeasuring-while` 循环，而不是使用 `startMeasuring` 和 `stopMeasuring`。

```

classdef PreallocationTest < matlab.perftest.TestCase
methods(Test)
    function testOnes(testCase)
        while(testCase.keepMeasuring)
            x = ones(1,1e5);
        end
        testCase.verifyEqual(size(x),[1 1e5])
    end

    function testIndexingWithVariable(testCase)
        import matlab.unittest.constraints.IssameSetAs
        while(testCase.keepMeasuring)
            id = 1:1e5;
            x(id) = 1;
        end
        testCase.verifyThat(x,IssameSetAs(1))
    end

    function testIndexingOnLHS(testCase)
        import matlab.unittest.constraints.EveryElementOf
        import matlab.unittest.constraints.IisEqualTo
        while(testCase.keepMeasuring)
            x(1:1e5) = 1;
        end
        testCase.verifyThat(EveryElementOf(x),IsEqualTo(1))
    end

    function testForLoop(testCase)
        while(testCase.keepMeasuring)
            for i=1:1e5
                x(i) = 1;
            end
        end
        testCase.verifyNumElements(x,1e5)
    end
end

```

重新运行测试。所有测试全部完成。

```
results = runperf('PreallocationTest');
```

```
Running PreallocationTest
```

```
..... Done PreallocationTest
```

查看结果。

```
sampleSummary(results)
```

```
ans =
```

```
4×7 table
```

Name	SampleSize	Mean	StandardDeviation	Min	Median	Max
PreallocationTest/testOnes	4	3.0804e-05	1.8337e-07	3.0577e-05	3.0843e-05	3.0953e-05
PreallocationTest/testIndexingWithVariable	4	0.00044536	1.7788e-05	0.00042912	0.00044396	0.00046441
PreallocationTest/testIndexingOnLHS	4	5.6352e-05	1.8863e-06	5.5108e-05	5.5598e-05	5.9102e-05
PreallocationTest/testForLoop	4	0.0097656	0.00018202	0.0096181	0.0097065	0.010031

## 另请参阅

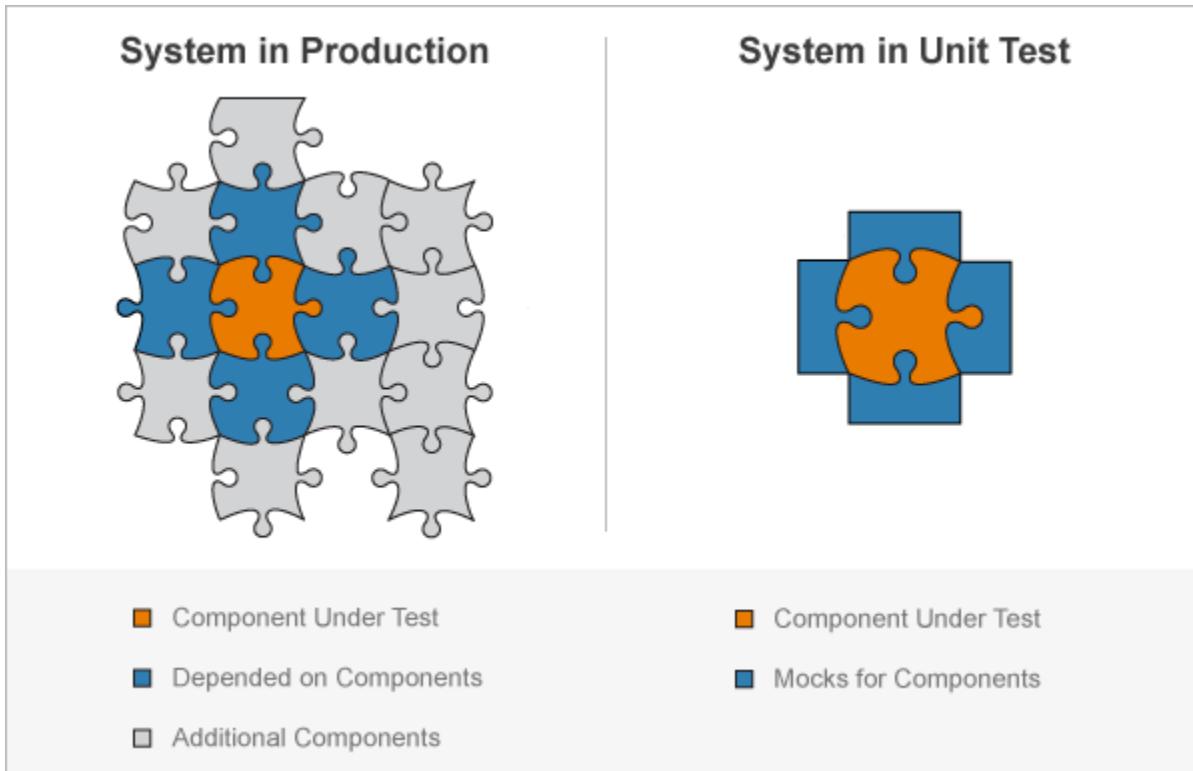
[keepMeasuring](#) | [runperf](#)

## 相关示例

- “性能测试框架概述” (第 34-142 页)
- “使用类测试性能” (第 34-149 页)

## 创建 Mock 对象

进行单元测试时，您经常要单独测试完整系统的一部分，将其与所依赖的组件隔离。要测试该系统的一部分，我们可以代入 mock 对象来替代所依赖的组件。mock 对象至少实现与生产对象相同的接口的一部分，但其实现方式通常很简单、高效、可预测和可控制。当您使用模拟框架时，受测组件并不清楚其协作对象是 "real" 对象还是 mock 对象。



例如，假定您想测试一种用于购买股票的算法，但不想测试整个系统。您可以使用一个 mock 对象来替换股票价格查找功能，使用另一个 mock 对象来验证交易者是否购买了股票。您所测试的算法不知道它是在操作 mock 对象，因此您可以在与系统的其余部分隔离情况下测试算法。

通过 mock 对象，您可以定义行为（称为 stubbing 的进程）。例如，可以指定某个对象生成对查询的预定义响应。此外，还可以拦截并记住从受测组件发送到 mock 对象的消息（称为 spying 的进程）。例如，可以验证是否调用了特定方法或设置了某个属性。

测试处于隔离状态的组件的典型工作流如下：

- 1 为所依赖的组件创建 mock。
- 2 定义这些 mock 的行为。例如，定义模拟方法或属性（使用一组特定的输入进行调用时）返回的输出。
- 3 测试所关注的组件。
- 4 验证所关注的组件与模拟组件之间的交互。例如，验证是否使用特定输入调用了模拟方法或是否设置了某个属性。

## 依赖的组件

在此示例中，受测组件是一个简单的日间交易算法。它是您要独立于其他组件单独测试的系统部分。日间交易算法有两个依赖项：检索股票价格数据的数据服务以及股票购买代理。

在您的当前工作文件夹下的文件 `DataService.m` 中，创建一个包含 `lookupPrice` 方法的抽象类。

```
classdef DataService
    methods (Abstract,Static)
        price = lookupPrice(ticker,date)
    end
end
```

在生产代码中，可能有多个属于 `DataService` 类的具体实现，例如 `BloombergDataService` 类。该类使用 Datafeed Toolbox™。但是，由于我们为 `DataService` 类创建 mock，因此您不需要安装该工具箱即可运行交易算法测试。

```
classdef BloombergDataService < DataService
    methods (Static)
        function price = lookupPrice(ticker,date)
            % This method assumes you have installed and configured the
            % Bloomberg software.
            conn = blp;
            data = history(conn,ticker,'LAST_PRICE',date-1,date);
            price = data(end);
            close(conn)
        end
    end
end
```

在此示例中，假定代理组件尚未开发。实现该组件后，它便会带有一个 `buy` 方法，用于接受股票代码和要购买的指定股数，并返回状态代码。代理组件的 mock 使用隐式接口，不从超类派生。

## 受测组件

在您的当前工作文件夹下的文件 `trader.m` 中，创建一个简单的日间交易算法。`trader` 函数接受以下项目作为输入：用于查找股票价格的数据服务对象、用于定义股票购买方式的代理对象、股票代码以及要购买的股数。如果昨天的价格低于两天前的价格，则指示代理购买指定的股数。

```
function trader(dataService,broker,ticker,numShares)
    yesterday = datetime('yesterday');
    priceYesterday = dataService.lookupPrice(ticker,yesterday);
    price2DaysAgo = dataService.lookupPrice(ticker,yesterday-days(1));

    if priceYesterday < price2DaysAgo
        broker.buy(ticker,numShares);
    end
end
```

## Mock 对象和行为对象

mock 对象是超类所指定接口的抽象方法和属性的实现。您也可以在没有超类的情况下构造 mock，在这种情况下，mock 具有一个隐式接口。mock 对象会为受测组件执行操作，例如调用方法或访问属性。

当您创建 mock 时，还会创建一个关联的行为对象。行为对象与 mock 对象定义的方法相同，用于控制 mock 行为。使用行为对象可定义 mock 操作并验证交互。例如，使用它来定义模拟方法返回的值或验证是否已访问某个属性。

在命令提示符下，创建一个 mock 测试用例以供交互使用。此示例后面将介绍在测试类中而非在命令提示符下使用 mock。

```
import matlab.mock.TestCase
testCase = TestCase.forInteractiveUse;
```

### 创建 Stub 以定义行为

为数据服务依赖项创建一个 mock 并检验其方法。数据服务 mock 会返回预定义的值，并替换用于提供实际股票价格的服务的实现。因此，它展示了上桩行为。

```
[stubDataService,dataServiceBehavior] = createMock(testCase,?DataService);
methods(stubDataService)
```

Methods for class matlab.mock.classes.DataServiceMock:

Static methods:

`lookupPrice`

在 `DataService` 类中，`lookupPrice` 方法是抽象和静态的。模拟框架会将该方法作为具体和静态的方法进行实现。

定义数据服务 mock 的行为。对于股票代码 "FOO"，它会将昨天的价格返回为 \$123，而昨天之前的任何价格都为 \$234。因此，根据 `trader` 函数的要求，代理将始终购买股票 "FOO"。对于股票代码 "BAR"，它会将昨天的价格返回为 \$765，而昨天之前的任何价格都为 \$543。因此，代理将从不会购买股票 "BAR"。

```
import matlab.unittest.constraints.IsLessThan
yesterday = datetime('yesterday');

testCase.assignOutputsWhen(dataServiceBehavior.lookupPrice...
    "FOO",yesterday),123);
testCase.assignOutputsWhen(dataServiceBehavior.lookupPrice...
    "FOO",IsLessThan(yesterday)),234);

testCase.assignOutputsWhen(dataServiceBehavior.lookupPrice...
    "BAR",yesterday),765);
testCase.assignOutputsWhen(dataServiceBehavior.lookupPrice...
    "BAR",IsLessThan(yesterday)),543);
```

您现在可以调用所模拟的 `lookupPrice` 方法。

```
p1 = stubDataService.lookupPrice("FOO",yesterday)
p2 = stubDataService.lookupPrice("BAR",yesterday-days(5))
```

```
p1 =
```

```
123
```

```
p2 =
```

```
543
```

虽然 `testCase` 的 `assignOutputsWhen` 方法便于指定行为，但如果您使用 `AssignOutputs` 操作，则有更多功能可供使用。有关详细信息，请参阅“指定 Mock 对象行为”（第 34-164 页）。

### 创建 Spy 以拦截消息

为代理依赖项创建一个 `mock` 并检验其方法。由于代理 `mock` 用于验证与受测组件（`trader` 函数）的交互，因此它可展示出 spying 行为。代理 `mock` 具有一个隐式接口。虽然 `buy` 方法当前未实现，但您可以创建一个具有该方法的 `mock`。

```
[spyBroker,brokerBehavior] = createMock(testCase,'AddedMethods',{'buy'});
methods(spyBroker)
```

Methods for class `matlab.mock.classes.Mock`:

```
buy
```

调用该 `mock` 的 `buy` 方法。默认情况下，它会返回空值。

```
s1 = spyBroker.buy
s2 = spyBroker.buy("inputs",[13 42])
```

```
s1 =
```

```
[]
```

```
s2 =
```

```
[]
```

由于 `trader` 函数不会使用状态返回代码，因此返回空值这一默认 `mock` 行为是可接受的。代理 `mock` 是一个纯粹的 spy，不需要实现任何上桩行为。

### 调用受测组件

调用 `trader` 函数。除了股票代码和要购买的股数之外，`trader` 函数还接受数据服务和代理作为输入。传入 `spyBroker` 和 `stubDataService` `mock`，而不是传入实际的数据服务和代理对象。

```
trader(stubDataService,spyBroker,"FOO",100)
trader(stubDataService,spyBroker,"FOO",75)
trader(stubDataService,spyBroker,"BAR",100)
```

## 验证函数交互

使用代理行为对象 (spy) 来验证 `trader` 函数是否按预期调用 `buy` 方法。

使用 `TestCase.verifyCalled` 方法来验证 `trader` 函数是否指示 `buy` 方法购买 100 股 FOO 股票。

```
import matlab.mock.constraints.WasCalled;
testCase.verifyCalled(brokerBehavior.buy("FOO",100))
```

Interactive verification passed.

验证 FOO 股票是否被购买了两次，而不管指定的股数如何。虽然 `verifyCalled` 方法便于指定行为，但如果使用 `WasCalled` 约束，则有更多功能可供使用。例如，您可以验证模拟方法是否被调用了指定的次数。

```
import matlab.unittest.constraints.IsAnything
testCase.verifyThat(brokerBehavior.buy("FOO",IsAnything), ...
    WasCalled('WithCount',2))
```

Interactive verification passed.

验证是否未调用 `buy` 方法来请求购买 100 股 BAR 股票。

```
testCase.verifyNotCalled(brokerBehavior.buy("BAR",100))
```

Interactive verification passed.

尽管调用了 `trader` 函数来请求购买 100 股 BAR 股票，但该 stub 为 BAR 定义了昨天的价格，以返回一个高于昨天之前的所有日期的值。因此，代理将从不会购买股票 "BAR"。

## trader 函数的测试类

交互测试用例便于在命令提示符下进行试验。但是，通常会在测试类中创建和使用 mock。在您的当前工作文件夹下的文件中，创建以下包含此示例中的交互测试的测试类。

```
classdef TraderTest < matlab.mock.TestCase
methods(Test)
    function buysStockWhenDrops(testCase)
        import matlab.unittest.constraints.IsLessThan
        import matlab.unittest.constraints.IsAnything
        import matlab.mock.constraints.WasCalled
        yesterday = datetime('yesterday');

        % Create mocks
        [stubDataService,dataServiceBehavior] = createMock(testCase, ...
            ?DataService);
        [spyBroker,brokerBehavior] = createMock(testCase, ...
            'AddedMethods',{'buy'});

        % Set up behavior
        testCase.assignOutputsWhen(dataServiceBehavior.lookupPrice(...
            "FOO",yesterday),123);
        testCase.assignOutputsWhen(dataServiceBehavior.lookupPrice(...
            "FOO",IsLessThan(yesterday)),234);

        % Call function under test
        trader(stubDataService,spyBroker,"FOO",100)
    end
end
```

```

trader(stubDataService,spyBroker,"FOO",75)

% Verify interactions
testCase.verifyCalled(brokerBehavior.buy("FOO",100))
testCase.verifyThat(brokerBehavior.buy("FOO",IsAnything), ...
    WasCalled('WithCount',2))
end
function doesNotBuyStockWhenIncreases(testCase)
    import matlab.unittest.constraints.IsLessThan
    yesterday = datetime('yesterday');

    % Create mocks
    [stubDataService,dataServiceBehavior] = createMock(testCase, ...
        ?DataService);
    [spyBroker,brokerBehavior] = createMock(testCase, ...
        'AddedMethods',{'buy'});

    % Set up behavior
    testCase.assignOutputsWhen(dataServiceBehavior.lookupPrice(...
        "BAR",yesterday),765);
    testCase.assignOutputsWhen(dataServiceBehavior.lookupPrice(...
        "BAR",IsLessThan(yesterday)),543);

    % Call function under test
    trader(stubDataService,spyBroker,"BAR",100)

    % Verify interactions
    testCase.verifyNotCalled(brokerBehavior.buy("BAR",100))
end
end
end

```

运行测试并查看结果表。

```

results = runtests('TraderTest');
table(results)

Running TraderTest
..
Done TraderTest
_____
ans =
2×6 table
Name      Passed Failed Incomplete Duration Details
_____
'TraderTest/buysStockWhenDrops'    true   false   false    0.13921 [1×1 struct]
'TraderTest/doesNotBuyStockWhenIncreases' true   false   false    0.066186 [1×1 struct]

```

## 另请参阅

## 指定 Mock 对象行为

### 本节内容

- “定义 Mock 方法的行为” (第 34-164 页)
- “定义 Mock 属性的行为” (第 34-165 页)
- “定义重复行为和后续行为” (第 34-166 页)
- “行为汇总” (第 34-167 页)

当您创建 mock 时，还会创建一个用于控制 mock 行为的关联行为对象。使用此对象可定义 mock 方法和属性的行为 (stub)。有关创建 mock 的详细信息，请参阅“[创建 Mock 对象](#)” (第 34-158 页)。

mock 对象是超类所指定接口的抽象方法和属性的实现。您也可以在没有超类的情况下构造 mock，在这种情况下，mock 具有一个隐式接口。

创建一个具有隐式接口的 mock。该接口包含 Name 和 ID 属性以及 findUser 方法，后者接受标识符并返回名称。虽然该接口当前未实现，但您可以创建一个具有该接口的 mock。

```
testCase = matlab.mock.TestCase.forInteractiveUse;
[mock,behaviorObj] = testCase.createMock('AddedProperties',...
{'Name','ID'},'AddedMethods',{'findUser'});
```

### 定义 Mock 方法的行为

您可以指定某个 mock 方法返回特定的值或在不同情况下引发异常。

指定 findUser 方法在使用任何输入调用时返回 "Unknown"。默认情况下，当您调用 findUser 方法时，MATLAB 将返回一个空数组。

- `assignOutputsWhen` 方法用于定义调用该方法时的返回值。
- 模拟方法调用 (`behaviorObj.findUser`) 用于隐式创建一个 `MethodCallBehavior` 对象。
- `MethodCallBehavior` 对象的 `withAnyInputs` 方法用于指定行为适用于具有任何数量的输入以及任何值的方法调用。

```
testCase.assignOutputsWhen(withAnyInputs(behaviorObj.findUser),"Unknown")
n = mock.findUser(1)
```

```
n =
"Unknown"
```

指定当输入值为 1701 时 mock 方法返回 "Jim"。只有对于输入值 1701，此行为才会取代返回值 "Unknown"，因为它是在该设定后定义的。

```
testCase.assignOutputsWhen(behaviorObj.findUser(1701),"Jim")
n = mock.findUser(1701)

n =
"Jim"
```

指定当 findUser 方法仅使用该对象作为输入来调用时，mock 方法返回 "Unspecified ID"。 `MethodCallBehavior` 对象的 `withExactInputs` 方法用于指定该行为应用于将该对象作为唯一输入值的方法调用。

```
testCase.assignOutputsWhen(withExactInputs(behaviorObj.findUser), ...
    "Unspecified ID")
n = mock.findUser % equivalent to n = findUser(mock)

n =
```

"Unspecified ID"

您可以使用 `matlab.unittest.constraints` 包中的类来帮助定义行为。指定 `findUser` 在使用大于 5000 的 ID 调用时引发异常。

```
import matlab.unittest.constraints.IsGreaterThanOrEqual
testCase.throwExceptionWhen(behaviorObj.findUser(IsGreaterThanOrEqual(5000)));
n = mock.findUser(5001)
```

Error using  
`matlab.mock.internal.MockContext/createMockObject/mockMethodCallback` (line 323)  
The following method call was specified to throw an exception:  
`findUser([1×1 matlab.mock.classes.Mock], 5001)`

您可以根据方法调用中请求的输出数目来定义行为。如果方法调用请求两个输出值，则返回 "???" 作为名称以及返回 -1 作为 ID。

```
testCase.assignOutputsWhen(withNargout(2, ...
    withAnyInputs(behaviorObj.findUser)), "???", -1)
[n,id] = mock.findUser(13)
```

n =

"???"

id =

-1

## 定义 Mock 属性的行为

访问某个 mock 属性时，您可以指定该属性返回特定的或所存储的属性值。设置此值时，可以指定该 mock 何时存储属性值。此外，还可以定义测试框架针对 mock 属性设置或访问活动何时引发异常。

定义 mock 属性的行为时，请记住在命令行窗口中显示属性值为一种属性访问 (get) 操作。

与定义 mock 方法的行为类似，定义 mock 属性的行为需要使用 `PropertyBehavior` 类的实例。当您访问 mock 属性时，该框架会返回此类的实例。要定义访问行为，请通过调用 `PropertyBehavior` 类的 `get` 方法来使用 `PropertyGetBehavior` 的实例。要定义设置行为，请通过调用 `PropertyBehavior` 类的 `set` 或 `setToValue` 方法来使用 `PropertySetBehavior` 的实例。

指定当 `Name` 属性设置为任何值时测试框架引发异常。

- `throwExceptionWhen` 方法用于指示该框架对指定行为引发异常。
- 访问行为对象 `PropertyBehavior` 类 (`behaviorObj.Name`) 的属性用于创建一个 `PropertyBehavior` 类实例。
- 调用 `PropertyBehavior` 类的 `set` 方法用于创建 `PropertySetBehavior`。

```
testCase.throwExceptionWhen(set(behaviorObj.Name))
mock.Name = "Sue";
```

```
Error using matlab.mock.internal.MockContext/createMockObject/mockPropertySetCallback (line 368)
The following property set was specified to throw an exception:
<Mock>.Name = "Sue"
```

允许该 mock 在属性设置为 "David" 时存储值。

```
testCase.storeValueWhen(setToValue(behaviorObj.Name,"David"));
mock.Name = "David"
```

```
mock =
```

Mock with properties:

```
Name: "David"
ID: []
```

## 定义重复行为和后续行为

`matlab.mock.TestCase` 方法便于定义行为。但是，如果您改用 `matlab.mock.actions` 包中的类，则有更多功能可供使用。通过这些类，您可以定义重复同一操作多次的行为并指定后续操作。要定义重复或后续行为，请将 `matlab.mock.actions` 包中某个类的实例传递给该行为类的 `when` 方法。

将值 1138 赋给 ID 属性，然后在发生属性访问时引发异常。

```
import matlab.mock.actions.AssignOutputs
import matlab.mock.actions.ThrowException
when(get(behaviorObj.ID),then(AssignOutputs(1138),ThrowException))
id = mock.ID
id = mock.ID

id =
```

```
1138
```

```
Error using matlab.mock.internal.MockContext/createMockObject/mockPropertyGetCallback (line 346)
The following property access was specified to throw an exception:
<Mock>.ID
```

依次为 ID 属性指定值 1138 和 237。然后，在发生属性访问时引发异常。每次调用 `then` 方法最多接受两个操作。要指定更多后续操作，请多次调用 `then`。

```
when(get(behaviorObj.ID),then(AssignOutputs(1138), ...
    then(AssignOutputs(237),ThrowException)))
id = mock.ID
id = mock.ID
id = mock.ID

id =
```

```
1138
```

```
id =
```

```
237
```

```
Error using matlab.mock.internal.MockContext/createMockObject/mockPropertyGetCallback (line 346)
```

The following property access was specified to throw an exception:  
 <Mock>.ID

如果该对象是唯一的输入值，请指定 `findUser` 函数返回值 "Phil" 两次。

```
when(withExactInputs(behaviorObj.findUser),repeat(2,AssignOutputs("Phil")))
n = mock.findUser
n = mock.findUser

n =
"Phil"
```

n =

"Phil"

第三次调用该函数。如果您重复执行某一操作，并且随后不调用 `then` 方法，则该 mock 会继续返回重复的值。

```
n = mock.findUser
n =
"Phil"
```

定义设置 Name 的值时的行为。前两次引发异常，然后存储该值。

```
import matlab.mock.actions.StoreValue
when(set(behaviorObj.Name),then(repeat(2,ThrowException),StoreValue))
mock.Name = "John"
```

Error using matlab.mock.internal.MockContext/createMockObject/mockPropertySetCallback (line 368)  
 The following property set was specified to throw an exception:

<Mock>.Name = "John"

`mock.Name = "Penny"`

Error using matlab.mock.internal.MockContext/createMockObject/mockPropertySetCallback (line 368)  
 The following property set was specified to throw an exception:

<Mock>.Name = "Penny"

`mock.Name = "Tommy"`

`mock =`

Mock with properties:

Name: "Tommy"

## 行为汇总

行为	TestCase 方法	matlab.mock.Actions 类 (允许定义重复行为和后续行为)
在发生方法调用和属性访问时返回指定的值。	<code>assignOutputsWhen</code>	<code>AssignOutputs</code>

行为	TestCase 方法	matlab.mock.Actions 类 (允许定义重复行为和后续行为)
访问属性时返回所存储的值。	returnStoredValueWhen	ReturnStoredValue
设置属性时存储值。	storeValueWhen	StoreValue
调用方法或者设置或访问属性时引发异常。	throwExceptionWhen	ThrowException

## 另请参阅

[matlab.mock.TestCase](#) | [matlab.mock.actions.AssignOutputs](#) |  
[matlab.mock.actions.ReturnStoredValue](#) | [matlab.mock.actions.StoreValue](#) |  
[matlab.mock.actions.ThrowException](#)

## 相关示例

- “创建 Mock 对象” (第 34-158 页)

## 验证 Mock 对象交互

当您创建 mock 时，还会创建一个用于控制 mock 行为的关联行为对象。使用此对象可访问所拦截的从受测组件发送到 mock 对象的消息（称为 spying 的进程）。有关创建 mock 的详细信息，请参阅“[创建 Mock 对象](#)”（第 34-158 页）。

在模拟框架中，验证为用于测试与该对象的交互的函数。有四种验证类型：

- 确认 - 在不引发异常的情况下产生并记录失败。由于确认不会引发异常，因此即使出现确认失败的情形，依然会完成所有的测试内容。通常，确认是单元测试的主要验证类型，因为这些确认一般不要求提前从测试中退出。使用其他验证类型来测试是否违反先决条件或测试安装是否正确。
- 假设 - 确保测试环境满足先决条件，不会导致测试失败。假设失败会导致测试被滤除，且测试框架会将这些测试标记为未完成。
- 断言 - 确保失败情况会使当前测试内容的剩余部分失效，但不会阻止后续测试方法正常执行。断言点处的失败会将当前测试方法标记为失败且未完成。
- 致命断言 - 在失败时中止测试会话。当失败涉及根本以致没必要继续测试时，这种验证会很有用。当脚手架拆解未能正确还原 MATLAB 状态，适合中止测试并启动一个新会话时，这些验证也很有用。

mock 对象是超类所指定接口的抽象方法和属性的实现。您也可以在没有超类的情况下构造 mock，在这种情况下，mock 具有一个隐式接口。为骰子类创建一个具有隐式接口的 mock。该接口包含 `Color` 和 `NumSides` 属性以及 `roll` 方法，后者接受骰子数并返回一个值。虽然该接口当前未实现，但您可以创建一个具有该接口的 mock。

```
testCase = matlab.mock.TestCase.forInteractiveUse;
[mock,behaviorObj] = testCase.createMock('AddedProperties',...
{'NumSides','Color'},'AddedMethods',{'roll'});
```

## 验证 Mock 方法交互

由于 mock 会记录向其发送的交互，因此您可以验证是否调用了 mock 方法。掷一个骰子。

```
val = mock.roll(1);
```

验证是否使用了 1 个骰子调用 `roll` 方法。

```
testCase.verifyCalled(behaviorObj.roll(1))
```

Interactive verification passed.

验证是否使用了 3 个骰子调用 `roll` 方法。此测试将失败。

```
testCase.verifyCalled(behaviorObj.roll(3), ...
'roll method should have been called with input 3.')
```

Interactive verification failed.

Test Diagnostic:

roll method should have been called with input 3.

Framework Diagnostic:

verifyCalled failed.

```
--> Method 'roll' was not called with the specified signature.  
--> Observed method call(s) with any signature:  
    out = roll([1×1 matlab.mock.classes.Mock], 1)
```

```
Specified method call:  
MethodCallBehavior  
[...] = roll(<Mock>, 3)
```

验证是否未使用 2 个骰子调用 roll 方法。

```
testCase.verifyNotCalled(behaviorObj.roll(2))
```

Interactive verification passed.

由于 MethodCallBehavior 类的 withAnyInputs、withExactInputs 和 withNargout 方法会返回 MethodCallBehavior 对象，因此您可以在验证中使用这些方法。验证是否使用了任何输入至少调用一次 roll 方法。

```
testCase.verifyCalled(withAnyInputs(behaviorObj.roll))
```

Interactive verification passed.

验证是否未使用 2 个输出和任何输入调用 roll 方法。

```
testCase.verifyNotCalled(withNargout(2,withAnyInputs(behaviorObj.roll)))
```

Interactive verification passed.

## 验证 Mock 属性交互

与方法调用类似，mock 会记录属性设置和访问操作。设置骰子的颜色。

```
mock.Color = "red"
```

```
mock =
```

Mock with properties:

```
NumSides: []  
Color: "red"
```

验证是否设置了颜色。

```
testCase.verifySet(behaviorObj.Color)
```

Interactive verification passed.

验证是否对颜色进行了访问。此测试将通过，因为在 MATLAB 显示该对象时会隐式访问属性。

```
testCase.verifyAccessed(behaviorObj.Color)
```

Interactive verification passed.

断言未设置面数。

```
testCase.assertNotSet(behaviorObj.NumSides)
```

Interactive assertion passed.

## 使用 Mock 对象约束

`matlab.mock.TestCase` 方法便于监视 mock 交互。但是，如果您改用 `matlab.mock.constraints` 包中的类，则有更多功能可供使用。要使用约束，请将行为对象和约束传递给 `verifyThat`、`assumeThat`、`assertThat` 或 `fatalAssertThat` 方法。

创建一个新的 mock 对象。

```
testCase = matlab.mock.TestCase.forInteractiveUse;
[mock,behaviorObj] = testCase.createMock('AddedProperties',...
    {'NumSides','Color'},'AddedMethods',{'roll'});
```

掷 2 个骰子。然后，通过约束来验证是否使用了两个骰子至少调用一次 `roll` 方法。

```
val = mock.roll(2);
```

```
import matlab.mock.constraints.WasCalled
testCase.verifyThat(behaviorObj.roll(2),WasCalled)
```

Interactive verification passed.

掷一个骰子。然后，验证是否使用了任何输入至少调用两次 `roll` 方法。

```
val = mock.roll(1);
```

```
testCase.verifyThat(withAnyInputs(behaviorObj.roll), ...
    WasCalled('WithCount',2))
```

Interactive verification passed.

验证是否未访问 `NumSides`。

```
import matlab.mock.constraints.WasAccessed
testCase.verifyThat(behaviorObj.NumSides,~WasAccessed)
```

Interactive verification passed.

设置骰子的颜色。然后，验证是否将该属性设置了一次。

```
mock.Color = "blue";
```

```
import matlab.mock.constraints.WasSet
testCase.verifyThat(behaviorObj.Color,WasSet('WithCount',1))
```

Interactive verification passed.

访问 `Color` 属性。然后，验证该属性是否未正好访问一次。此测试将失败。

```
c = mock.Color
```

```
testCase.verifyThat(behaviorObj.Color,~WasAccessed('WithCount',1))
```

```
c =
```

```
"blue"
```

Interactive verification failed.

---

Framework Diagnostic:

Negated WasAccessed failed.

-> Property 'Color' was accessed the prohibited number of times.

Actual property access count:

1

Prohibited property access count:

1

Specified property access:

PropertyGetBehavior  
<Mock>.Color

设置面数。然后，验证面数是否设置为 22。

```
mock.NumSides = 22;
testCase.verifyThat(behaviorObj.NumSides,WasSet('ToValue',22))
```

Interactive verification passed.

使用 `matlab.unittest.constraints` 包中的约束来断言骰子的面数未被设置为超过 20。此测试将失败。

```
import matlab.unittest.constraints.IsLessThanOrEqualTo
testCase.verifyThat(behaviorObj.NumSides, ...
    WasSet('ToValue',IsLessThanOrEqualTo(20)))
```

Interactive verification failed.

---

Framework Diagnostic:

WasSet failed.

-> Property 'NumSides' was not set to the specified value.

-> Observed property set(s) to any value:

<Mock>.NumSides = 22

Specified property set:

PropertySetBehavior  
<Mock>.NumSides = <IsLessThanOrEqualTo constraint>

## 验证概述

验证类型	TestCase 方法	matlab.mock.constraints 类	
		使用 matlab.unittest.TestCase 方法	通过 matlab.mock.constraints 类
已调用方法	verifyCalled 或 verifyNotCalled	verifyThat	WasCalled 或 Occurred
	assumeCalled 或 assumeNotCalled	assumeThat	

验证类型	TestCase 方法	matlab.mock.constraints 类	
		使用 matlab.unittest.TestCase 方法	通过 matlab.mock.constraints 类
	assertCalled 或 assertNotCalled	assertThat	
	fatalAssertCalled 或 fatalAssertNotCalled	fatalAssertThat	
方法已被调用特定次数	不适用	verifyThat、assumeThat、assertThat 或 fatalAssertThat	WasCalled
已访问属性	verifyAccessed 或 verifyNotAccessed	verifyThat	WasAccessed 或 Occurred
	assumeAccessed 或 assumeNotAccessed	assumeThat	
	assertAccessed 或 assertNotAccessed	assertThat	
	fatalAssertAccessed 或 fatalAssertNotAccessed	fatalAssertThat	
属性已被访问特定次数	不适用	verifyThat、assumeThat、assertThat 或 fatalAssertThat	WasAccessed
已设置属性	verifySet 或 verifyNotSet	verifyThat	WasSet 或 Occurred
	assumeSet 或 assumeNotSet	assumeThat	
	assertSet 或 assertNotSet	assertThat	
	fatalAssertSet 或 fatalAssertNotSet	fatalAssertThat	
属性已被设置特定次数	不适用	verifyThat、assumeThat、assertThat 或 fatalAssertThat	WasSet
属性已被设置为特定值	不适用	verifyThat、assumeThat、assertThat 或 fatalAssertThat	WasSet 或 Occurred
按特定顺序调用方法并访问或设置属性	不适用	verifyThat、assumeThat、assertThat 或 fatalAssertThat	Occurred

## 另请参阅



# System object 用法和编写

---

- “何谓 System object? ” (第 35-2 页)
- “System object 与 MATLAB 函数” (第 35-5 页)
- “使用 System object 在 MATLAB 中进行系统设计” (第 35-7 页)
- “定义基本 System object” (第 35-11 页)
- “更改输入数目” (第 35-13 页)
- “验证属性和输入值” (第 35-16 页)
- “初始化属性并设置一次性计算” (第 35-18 页)
- “构造时设置属性值” (第 35-20 页)
- “重置算法并释放资源” (第 35-22 页)
- “定义属性特性” (第 35-24 页)
- “隐藏非活动属性” (第 35-26 页)
- “将属性值限制为有限列表” (第 35-28 页)
- “处理调整后的属性” (第 35-32 页)
- “定义复合的 System object” (第 35-34 页)
- “定义有限源对象” (第 35-36 页)
- “保存和加载 System object” (第 35-38 页)
- “定义 System object 信息” (第 35-41 页)
- “处理输入设定更改” (第 35-43 页)
- “调用序列摘要” (第 35-45 页)
- “详细的调用序列” (第 35-48 页)
- “定义 System object 的技巧” (第 35-50 页)
- “使用 MATLAB 编辑器插入 System object 代码” (第 35-52 页)
- “分析 System object 代码” (第 35-57 页)
- “在 System object 中使用全局变量” (第 35-59 页)

## 何谓 System object?

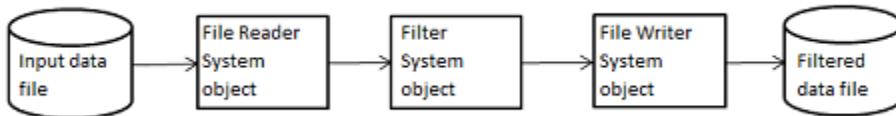
### 本节内容

[“运行 System object”](#) (第 35-3 页)

[“System object 函数”](#) (第 35-3 页)

System object 是一种专用的 MATLAB 对象。许多工具箱中都包含 System object。System object 专为实现和仿真输入随时间变化的动态系统而设计。许多信号处理、通信和控制系统都是动态的。在动态系统中，输出信号的值同时取决于输入信号的瞬时值以及系统的过往行为。System object 使用内部状态来存储下一个计算步骤中使用的系统过往行为。因此，System object 非常适用于分段处理大型数据流的迭代计算，例如视频和音频处理系统。这种处理流化数据的功能具有不必在内存中保存大量数据的优点。采用流化数据，您还可以使用可高效利用循环的简化程序。

例如，您可以在系统中使用 System object，以便从某个文件中读取数据、对该数据进行滤波，然后将滤波后的输出写入其他文件。通常，每次循环迭代中都会将指定数量的数据传递给滤波器。文件读取器对象使用状态来跟踪在文件中开始下一次数据读取的位置。同样，文件写入器对象会跟踪其最后将数据写入输出文件的位置，以使数据不会被覆盖。滤波器对象保留其自身的内部状态，以确保滤波正常执行。下图表示系统的单个循环。



这些优点使得 System object 适用于处理流化数据。

许多 System object 支持：

- 定点算术运算（需要 Fixed-Point Designer™ 许可证）
- C 代码生成（需要 MATLAB Coder™ 或 Simulink Coder 许可证）
- HDL 代码生成（需要 HDL Coder™ 许可证）
- 可执行文件或共享库生成（需要 MATLAB Compiler 许可证）

**注意** 查看产品文档以确认要使用的特定 System object 对定点、代码生成和 MATLAB Compiler 的支持。

System object 至少使用两个命令来处理数据：

- 创建对象（如 `fft256 = dsp.FFT`）
- 通过对象运行数据（如 `fft256(x)`）

通过将创建与执行分离，您可以创建多个持久的可重用对象，并且每个对象具有不同的设置。使用此方法可避免重复进行输入确认和验证、便于在编程循环中使用并提高整体性能。相比较而言，MATLAB 函数必须在您每次调用时对参数进行验证。

除了系统工具箱提供的 System object 以外，您还可以创建自己的 System object。请参阅“[创建 System object](#)”。

## 运行 System object

要运行某个 System object 并执行其算法定义的操作，您可以调用该对象，就好像它是一个函数一样。例如，要创建一个 FFT 对象（该对象使用 `dsp.FFT` System object、将长度指定为 1024 并命名为 `dft`），请使用：

```
dft = dsp.FFT('FFTLengthSource','Property','FFTLength',1024);
```

要使用输入 `x` 运行此对象，请使用：

```
dft(x);
```

如果您在不带任何输入参数的情况下运行 System object，则必须包含空的圆括号。例如，`asyobj()`。

当您运行某个 System object 时，该对象还会执行与数据处理有关的其他重要任务，例如初始化和处理对象状态。

---

**注意** 运行 System object 的替代方法是使用 `step` 函数。例如，对于使用 `dft = dsp.FFT` 创建的对象，您可以通过 `step(dft,x)` 来运行该对象。

---

## System object 函数

在创建 System object 之后，可以使用各种对象函数来处理该对象的数据，或获取有关该对象的信息。使用函数的语法为 `<object function name>(<system object name>)`，加上可能的额外输入参数。例如，对于 `txfourier = dsp.FFT`（其中 `txfourier` 为所分配的名称），可使用 `reset(txfourier)` 来调用 `reset` 函数。

### 常见对象函数

所有 System object 都支持以下对象函数。如果某个函数不适用于特定对象，调用该函数不会对此对象有任何作用。

函数	说明
<code>运行对象函数，或 <b>step</b></code>	<p>运行相应用对象以使用该对象定义的算法来处理数据。</p> <p>示例：对于 <code>dft = dsp.FFT;</code> 对象，通过以下方式运行对象：</p> <ul style="list-style-type: none"> <li>• <code>y = dft(x)</code></li> <li>• <code>y = step(dft,x)</code></li> </ul> <p>在此处理过程中，该对象会根据需要初始化资源、返回输出并更新对象状态。在执行过程中，您只能更改可调属性。运行 System object 的两种方法都会返回常规的 MATLAB 变量。</p>
<code>release</code>	释放资源并允许更改 System object 属性值以及 System object 正在使用时受限的附加特性。
<code>reset</code>	将 System object 重置为该对象的初始值。
<code>nargin</code>	返回 System object 算法定义接受的输入数目。如果算法定义中包含 <code>varargin</code> ，则 <code>nargin</code> 输出为负数。
<code>nargout</code>	返回 System object 算法定义接受的输出数目。如果算法定义中包含 <code>varargout</code> ，则 <code>nargout</code> 输出为负数。

函数	说明
<b>getDiscreteState</b>	返回对象离散状态的结构体。如果对象没有任何离散状态， <b>getDiscreteState</b> 将返回一个空结构体。
<b>clone</b>	使用相同的属性值创建另一个具有同一类型的对象
<b>isLocked</b>	返回一个逻辑值，指示对象是否已被调用并且尚未对该对象调用 <b>release</b> 。
<b>isDone</b>	仅适用于从 <code>matlab.system.mixin.FiniteSource</code> 继承的源对象。返回一个指示是否已到达数据文件的结尾的逻辑值。如果特定对象没有数据结尾功能，此函数值始终返回 <code>false</code> 。
<b>info</b>	返回一个包含有关对象的特征信息的结构体。该结构体的字段因对象而异。如果特定对象没有特征信息，则该结构体为空。

## 另请参阅

`matlab.System`

## 相关示例

- “System object 与 MATLAB 函数”（第 35-5 页）
- “使用 System object 在 MATLAB 中进行系统设计”（第 35-7 页）
- “System Design in Simulink Using System Objects”（Simulink）

# System object 与 MATLAB 函数

## 本节内容

- “System object 与 MATLAB 函数”（第 35-5 页）
- “使用仅包含 MATLAB 函数的代码处理音频数据”（第 35-5 页）
- “使用 System object 处理音频数据”（第 35-6 页）

## System object 与 MATLAB 函数

许多 System object 都具有对应的 MATLAB 函数。对于简单的一次性计算，使用 MATLAB 函数。但是，如果您需要设计和模拟一个带有许多组件的系统，请使用 System object。如果您的计算需要管理内部状态、包含随时间而变化的输入或者要处理大型数据流，则使用 System object 也是适合的。

仅使用 MATLAB 函数来构建具有不同执行阶段和内部状态的动态系统将需要进行复杂的编程。您需要编写代码以初始化系统、验证数据、管理内部状态以及重置和终止系统。System object 可在执行期间自动完成其中许多管理操作。通过将程序中的 System object 与其他 MATLAB 函数组合使用，您可以简化代码并提高效率。

## 使用仅包含 MATLAB 函数的代码处理音频数据

此示例说明如何编写仅包含 MATLAB® 函数的代码来读取音频数据。

该代码会从文件中读取音频数据、对音频数据进行滤波，并播放滤波后的音频数据。音频数据是按帧读取的。该代码与下一示例中的 System object 代码生成相同的结果，您可对方法进行比较。

找到源音频文件。

```
fname = 'speech_dft_8kHz.wav';
```

从源文件中获取样本总数和采样率。

```
audioInfo = audioinfo(fname);
maxSamples = audioInfo.TotalSamples;
fs = audioInfo.SampleRate;
```

定义要使用的滤波器。

```
b = fir1(160,.15);
```

初始化滤波器状态。

```
z = zeros(1,numel(b)-1);
```

定义要一次处理的音频数据量，然后初始化 while 循环索引。

```
frameSize = 1024;
nIdx = 1;
```

定义 while 循环以处理音频数据。

```
while nIdx <= maxSamples(1)-frameSize+1
    audio = audioread(fname,[nIdx nIdx+frameSize-1]);
```

```
[y,z] = filter(b,1, audio, z);
sound(y,fs);
nIdx = nIdx+frameSize;
end
```

该循环使用显式的索引和状态管理，此方法可能很繁琐且容易出错。您必须详尽了解各种状态，例如大小和数据类型。这种仅包含 MATLAB 的代码存在的另一个问题是，声音函数未设计为实时运行。生成的音频断断续续，几乎听不见。

## 使用 System object 处理音频数据

此示例说明如何编写 System object 代码来读取音频数据。

该代码使用 DSP System Toolbox™ 软件中的 System object 来从文件中读取音频数据、对音频数据进行滤波，然后播放滤波后的音频数据。该代码与前面显示的 MATLAB® 代码生成相同的结果，允许您对方法进行比较。

找到源音频文件。

```
fname = "speech_dft_8kHz.wav";
```

定义 System object 以读取文件。

```
audioIn = dsp.AudioFileReader(fname,'OutputDataType','single');
```

定义 System object 以进行数据滤波。

```
filtLP = dsp.FIRFilter('Numerator',fir1(160,.15));
```

定义 System object 以播放滤波后的音频数据。

```
audioOut = audioDeviceWriter('SampleRate',audioIn.SampleRate);
```

定义 while 循环以处理音频数据。

```
while ~isDone(audioIn)
    audio = audioIn(); % Read audio source file
    y = filtLP(audio); % Filter the data
    audioOut(y); % Play the filtered data
end
```

此 System object 代码避免了仅使用 MATLAB 代码存在的问题。在不需要使用显式索引的情况下，文件读取器对象即可管理数据帧大小，而滤波器可管理状态。音频设备写入器对象会播放每个经过处理的音频帧。

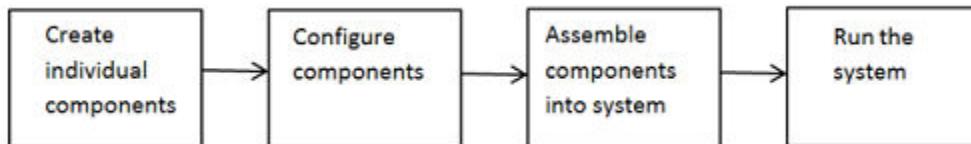
# 使用 System object 在 MATLAB 中进行系统设计

## 本节内容

- “在 MATLAB 中进行系统设计和仿真”（第 35-7 页）
- “创建单个组件”（第 35-7 页）
- “配置组件”（第 35-8 页）
- “同时创建和配置组件”（第 35-8 页）
- “将组件组合到系统中”（第 35-9 页）
- “运行系统”（第 35-9 页）
- “重新配置对象”（第 35-10 页）

## 在 MATLAB 中进行系统设计和仿真

通过 System object，您可以在 MATLAB 中进行系统设计和仿真。您可按下图所示在 MATLAB 中使用 System object。



- 1 “创建单个组件”（第 35-7 页） - 创建要在系统中使用的 System object。“创建单个组件”（第 35-7 页）。除了系统工具箱提供的 System object 以外，您还可以创建您自己的 System object。请参阅“创建 System object”。
- 2 “配置组件”（第 35-8 页） - 如有必要，可更改这些对象的属性值以便为您的特定系统建立模型。所有 System object 属性都具有默认值，您可以使用这些默认值而不必更改它们。请参阅“配置组件”（第 35-8 页）。
- 3 “将组件组合到系统中”（第 35-9 页） - 编写一个包含这些 System object 的 MATLAB 程序，并使用 MATLAB 变量作为输入和输出来连接这些对象以对系统进行仿真。请参阅“连接 System object”（第 35-9 页）。
- 4 “运行系统”（第 35-9 页） - 运行您的程序。当系统正在运行时，您可以更改可调属性。请参阅“运行系统”（第 35-9 页）和“重新配置对象”（第 35-10 页）。

## 创建单个组件

此部分中的示例说明如何使用软件中预定义的 System object。如果您使用某个函数来创建并使用 System object，请通过条件代码指定对象创建。条件化创建可以防止在循环中调用该函数时出现错误。您也可以创建您自己的 System object，请参阅“创建 System object”。

此部分说明如何使用 DSP System Toolbox™ 和 Audio Toolbox™ 中的预定义组件来设置您的系统：

- `dsp.AudioFileReader` - 读取音频数据文件
- `dsp.FIRFilter` - 对音频数据进行滤波
- `audioDeviceWriter` - 播放滤波后的音频数据

首先，使用默认的属性设置创建组件对象。

```
audioIn = dsp.AudioFileReader;
filtLP = dsp.FIRFilter;
audioOut = audioDeviceWriter;
```

## 配置组件

### 何时配置组件

如果您在创建对象时未设置其属性并且不想使用默认值，则必须显式设置这些属性。某些属性允许您在系统正在运行时更改其值。有关信息，请参阅“重新配置对象”（第 35-10 页）。

大多数属性都是相互独立的。但是，某些 System object 属性会启用或禁用其他属性，或者限制其他属性的值。为避免出现错误或警告，您应先设置控制属性，然后再设置从属属性。

### 显示组件属性值

要显示某个对象的当前属性值，请在命令行中键入该对象的句柄名称（如 `audioIn`）。要显示特定属性的值，请键入 `objectHandle.propertyName`（如 `audioIn.FileName`）。

### 配置组件属性值

此部分说明如何通过设置组件对象的属性来为您的系统配置组件。

如果您在配置组件之前已单独创建了组件，请按照此过程操作。您也可以同时创建和配置组件，如后面的示例所述。

对于文件读取器对象，指定要读取的文件并设置输出数据类型。

对于滤波器对象，使用 `fir1` 函数（用于指定低通滤波器阶数和截止频率）指定滤波器分子系数。

对于音频设备写入器对象，指定采样率。在本示例中，使用相同的采样率作为输入数据。

```
audioIn.Filename = "speech_dft_8kHz.wav";
audioIn.OutputDataType = "single";
filtLP.Numerator = fir1(160,.15);
audioOut.SampleRate = audioIn.SampleRate;
```

## 同时创建和配置组件

此示例说明如何同时创建 System object 组件和配置所需属性。使用 'Name', Value 对组参数指定每个属性。

创建文件读取器对象、指定要读取的文件并设置输出数据类型。

```
audioIn = dsp.AudioFileReader("speech_dft_8kHz.wav',...
    'OutputDataType','single');
```

创建滤波器对象，然后使用 `fir1` 函数指定滤波器分子。指定 `fir1` 函数的低通滤波器阶数和截止频率。

```
filtLP = dsp.FIRFilter('Numerator',fir1(160,.15));
```

创建音频播放器对象，然后将采样率设置为与输入数据相同的采样率。

```
audioOut = audioDeviceWriter('SampleRate',audioIn.SampleRate);
```

## 将组件组合到系统中

### 连接 System object

当您确定了所需组件并已创建和配置 System object 之后，请组装您的系统。您可以像其他 MATLAB 变量一样使用 System object，并将它们包含在 MATLAB 代码中。您可以在 System object 中传递 MATLAB 变量。

使用 System object 和使用函数的主要区别在于 System object 使用一个两步骤的过程。首先，创建相应对象并设置其参数，然后运行该对象。运行该对象会将其初始化并控制系统的数据流和状态管理。通常在代码循环中调用 System object。

您可以使用某个对象的输出作为另一个对象的输入。对于某些 System object，您可使用这些对象的属性来更改输入或输出。要验证是否正在使用合适数目的输入和输出，可以对任何 System object 使用 `nargin` 和 `nargout`。有关所有可用的 System object 函数的信息，请参阅“System object 函数”（第 35-3 页）。

### 连接系统中的组件

此部分演示了如何将组件连接在一起以读取、滤波和播放音频数据文件。while 循环使用 `isDone` 函数来完整读取整个文件。

```
while ~isDone(audioIn)
    audio = audioIn(); % Read audio source file
    y = filtLP(audio); % Filter the data
    audioOut(y); % Play the filtered data
end
```

## 运行系统

通过在命令行下直接键入或运行包含程序的文件来运行代码。当您运行系统代码时，数据是通过您的对象来处理的。

### 当系统正在运行时您无法更改的内容

首次调用 System object 时会初始化并运行该对象。当 System object 已开始处理数据时，您不能更改不可调属性。

根据 System object 的不同，其他设定也可能受到限制：

- 输入大小
- 输入复/实性
- 输入数据类型
- 可调属性数据类型
- 离散状态数据类型

如果 System object 作者对这些设定进行了限制，则您在 System object 使用期间尝试更改这些设定会出错。

## 重新配置对象

### 更改属性

当 System object 已开始处理数据时，您不能更改不可调属性。您可以对任何 System object 使用 `isLocked` 以验证对象是否正在处理数据。处理完成后，您可以使用 `release` 函数释放资源并允许更改不可调属性。

某些对象属性为可调属性，这使您能够对它们进行更改，即使对象在使用中也是如此。大多数 System object 属性是不可调属性。请参阅对象的参考页以确定单个属性是否为可调属性。

### 更改输入复/实性、维度或数据类型

在对象使用期间，在您调用算法后，某些 System object 不允许更改输入复/实性、大小或数据类型。如果 System object 对这些设定进行了限制，您可以调用 `release` 来更改这些设定。调用 `release` 还会重置 System object 的其他方面，例如状态和离散状态。

### 更改系统中的可调属性

此示例说明如何在代码运行时通过修改滤波器对象的 `Numerator` 属性，将滤波器类型更改为高通滤波器。更改将在下次调用该对象时生效。

```
reset(audioIn);% Reset audio file
Wn = [0.05,0.1,0.15,0.2];
for x=1:4000
    Wn_X = ceil(x/1000);
    filtLP.Numerator = fir1(160,Wn(Wn_X),'high');
    audio = audioIn(); % Read audio source file
    y = filtLP(audio); % Filter the data
    audioOut(y); % Play the filtered data
end
```

# 定义基本 System object

此示例说明如何创建一个将数值以 1 为单位递增的基本 System object。该示例中使用的类定义文件包含定义 System object 所需的最少元素。

## 创建 System object 类

您可以创建并编辑 MAT 文件，或使用 MATLAB 编辑器创建 System object。以下示例介绍了如何在 MATLAB 编辑器中使用新建菜单。

- 1 在 MATLAB 的“编辑器”选项卡中，选择新建 > **System object** > **基本**。此时将打开一个简单的 System object 模板。
- 2 从 `matlab.System` 为您的对象划分子类。在您的文件的第一行中将 `Untitled` 替换为 `AddOne`。

```
classdef AddOne < matlab.System
```

System object 由基类 `matlab.System` 构成，并且可能包括一个或多个 mixin 类。您可以在类定义文件的第一行中指定基类和 mixin 类。

- 3 保存该文件并将其命名为 `AddOne.m`。

## 定义算法

`stepImpl` 方法包含在您运行对象时要执行的算法。定义此方法以使其包含您希望 System object 执行的操作。

- 1 在所创建的基本 System object 中，检查 `stepImpl` 方法模板。

```
methods (Access = protected)
    function y = stepImpl(obj,u)
        % Implement algorithm. Calculate y as a function of input u and
        % discrete states.
        y = u;
    end
end
```

`stepImpl` 方法的访问权限始终设置为 `protected`，因为该方法为用户无法直接调用或运行的内部方法。

所有方法（静态方法除外）都要求将 System object 句柄作为第一个输入参数。MATLAB 编辑器插入的默认值为 `obj`。您可以对 System object 句柄使用任何名称。

默认情况下，输入数目和输出数目都为 1。可以使用输入/输出来添加输入和输出。您也可以使用不同数目的输入或输出，请参阅“更改输入数目”（第 35-13 页）。

或者，如果您通过编辑 MAT 文件创建 System object，则可以使用插入方法 > 实现算法来添加 `stepImpl` 方法。

- 2 在 `stepImpl` 方法中更改计算以将 1 添加到 `u` 的值中。

```
methods (Access = protected)

    function y = stepImpl(~,u)
        y = u + 1;
    end
```

**提示** 您可以使用波浪号 (~) 来指示函数中不使用对象句柄，而不必传入对象句柄。使用波浪号而不是对象句柄可防止出现有关未使用的变量的警告。

- 3 删除默认情况下包含在基本模板中的未使用的方法。

您可以修改这些方法以添加更多 System object 操作和属性。您也可以不进行任何更改，System object 仍将按预期运行。

现在，类定义文件包含此 System object 所需的全部代码。

```
classdef AddOne < matlab.System
% ADDONE Compute an output value one greater than the input value

% All methods occur inside a methods declaration.
% The stepImpl method has protected access
methods (Access = protected)

    function y = stepImpl(~,u)
        y = u + 1;
    end
end
end
```

### 另请参阅

[getNumInputsImpl](#) | [getNumOutputsImpl](#) | [matlab.System](#) | [stepImpl](#)

### 相关示例

- “更改输入数目” (第 35-13 页)
- “在 MATLAB 中进行系统设计和仿真” (第 35-7 页)

## 更改输入数目

此示例说明如何在使用和不使用 `getNumInputsImpl` 的情况下设置 System object™ 的输入数目。

如果您的输入或输出数目可变，并且打算在 Simulink® 中使用 System object，则必须在类定义中包含 `getNumInputsImpl` 或 `getNumOutputsImpl` 方法。

这些示例说明了更改输入数目的情况。如果要更改输出数目，这些原则同样适用。

就像对所有 System object Impl 方法一样，您应始终将 `getNumInputsImpl` 和 `getNumOutputsImpl` 方法的访问权限设置为 `protected`，因为它们是不能直接调用的内部方法。

### 最多允许三个输入

此示例说明如何编写允许输入数目可变的 System object。

通过添加处理一个、两个或三个输入的代码，将 `stepImpl` 方法更新为最多可接受三个输入。如果您仅在 MATLAB 中使用此 System object，则不需要 `getNumInputsImpl` 和 `getNumOutputsImpl`。

### 完整的类定义

```
classdef AddTogether < matlab.System
    % Add inputs together

    methods (Access = protected)
        function y = stepImpl(~,x1,x2,x3)
            switch nargin
                case 2
                    y = x1;
                case 3
                    y = x1 + x2;
                case 4
                    y = x1 + x2 + x3;
                otherwise
                    y = [];
            end
        end
    end
end
```

使用一个、两个和三个输入运行此 System object。

```
addObj = AddTogether;
addObj(2)
```

```
ans =
```

```
2
```

```
addObj(2,3)
```

```
ans =
```

5

```
addObj(2,3,4)
```

```
ans =
```

```
9
```

### 使用属性控制输入和输出的数目

此示例说明如何编写允许在运行之前更改输入和输出数目的 System object。如果要将您的 System object 包含在 Simulink 中，请使用此方法：

- 添加一个不可调属性 `NumInputs` 来控制输入的数目。
- 实现关联的 `getNumInputsImpl` 方法，以指定输入的数目。

### 完整的类定义

```
classdef AddTogether2 < matlab.System
    % Add inputs together. The number of inputs is controlled by the
    % nontunable property |NumInputs|.

    properties (Nontunable)
        NumInputs = 3; % Default value
    end
    methods (Access = protected)
        function y = stepImpl(obj,x1,x2,x3)
            switch obj.NumInputs
                case 1
                    y = x1;
                case 2
                    y = x1 + x2;
                case 3
                    y = x1 + x2 + x3;
                otherwise
                    y = [];
            end
        end
        function validatePropertiesImpl(obj)
            if ((obj.NumInputs < 1) || ...
                (obj.NumInputs > 3))
                error("Only 1, 2, or 3 inputs allowed.");
            end
        end
    end
    function numIn = getNumInputsImpl(obj)
        numIn = obj.NumInputs;
    end
end
end
```

使用一个、两个和三个输入运行此 System object。

```
addObj = AddTogether2;
addObj.NumInputs = 1;
addObj(2)
```

ans =

2

```
release(addObj);
addObj.NumInputs = 2;
addObj(2,3)
```

ans =

5

```
release(addObj);
addObj.NumInputs = 3;
addObj(2,3,4)
```

ans =

9

## 另请参阅

[getNumInputsImpl](#) | [getNumOutputsImpl](#)

## 相关示例

- “验证属性和输入值” (第 35-16 页)
- “定义基本 System object” (第 35-11 页)
- “在方法定义中使用 ~ 作为输入参数” (第 35-50 页)

## 验证属性和输入值

此示例说明如何验证为 System object 提供的输入和属性值是否有效。

### 验证单个属性

要独立于其他属性来验证一个属性值，请使用 MATLAB 类属性验证。此示例说明如何指定一个逻辑属性、一个正整数属性和一个必须为三个值之一的字符串属性。

```
properties
    UseIncrement (1,1) logical = false
    WrapValue (1,1) {mustBePositive, mustBeInteger} = 1
    Color (1,1) string {mustBeMember(Color, ["red","green","blue"])} = "red"
end
```

### 验证相互依赖的属性

要验证两个或多个相互依赖的属性的值，请使用 `validatePropertiesImpl`。此示例说明如何编写 `validatePropertiesImpl` 以验证逻辑属性 (`UseIncrement`) 为 `true` 并且 `WrapValue` 的值大于 `Increment`。

```
methods (Access = protected)
    function validatePropertiesImpl(obj)
        if obj.UseIncrement && obj.WrapValue > obj.Increment
            error("Wrap value must be less than increment value");
        end
    end
end
```

### 验证输入

要验证输入值，请使用 `validateInputsImpl` 方法。此示例说明如何验证第一个输入是否为数值。

```
methods (Access = protected)
    function validateInputsImpl(~,x)
        if ~isnumeric(x)
            error("Input must be numeric");
        end
    end
end
```

### 完整的类示例

此示例是一个完整的 System object，它显示每种验证语法的示例。

```
classdef AddOne < matlab.System
% ADDONE Compute an output value by incrementing the input value

% All properties occur inside a properties declaration.
% These properties have public access (the default)
properties
    UseIncrement (1,1) logical = false
    WrapValue (1,1) {mustBePositive, mustBeInteger} = 10
    Increment (1,1) {mustBePositive, mustBeInteger} = 1
end
```

```
methods (Access = protected)
    function validatePropertiesImpl(obj)
        if obj.UseIncrement && obj.WrapValue > obj.Increment
            error("Wrap value must be less than increment value");
        end
    end

    % Validate the inputs to the object
    function validateInputsImpl(~,x)
        if ~isnumeric(x)
            error("Input must be numeric");
        end
    end

    function out = stepImpl(obj,in)
        if obj.UseIncrement
            out = in + obj.Increment;
        else
            out = in + 1;
        end
    end
end
end
```

## 另请参阅

[validateInputsImpl](#) | [validatePropertiesImpl](#)

## 相关示例

- “定义基本 System object”（第 35-11 页）
- “更改输入复/实性、维度或数据类型”（第 35-10 页）
- “调用序列摘要”（第 35-45 页）
- “属性 set 方法”
- “在方法定义中使用 ~ 作为输入参数”（第 35-50 页）

## 初始化属性并设置一次性计算

此示例说明如何编写代码以初始化并设置 System object。

在此示例中，通过打开相应文件以使 System object 写入该文件来分配文件资源。您可以在设置时一次性执行这些初始化任务，而不是在每次运行对象时执行。

### 定义要初始化的公共属性

在此示例中，您会定义公共的 **Filename** 属性，并将该属性的值指定为不可调的字符串向量 **default.bin**。调用 **setup** 方法之后，用户便无法更改不可调属性。

```
properties (Nontunable)
    Filename = "default.bin"
end
```

### 定义要初始化的私有属性

用户无法直接访问私有属性，而只能通过 System object 的方法来访问。在此示例中，您将 **pFileID** 属性定义为私有属性。此外，还将此属性定义为隐藏属性，以指示它是一个内部属性，从不会对用户显示。

```
properties (Hidden,Access = private)
    pFileID;
end
```

### 定义设置

您可以使用 **setupImpl** 方法来执行设置和初始化任务。您应在 **setupImpl** 方法中包含只想一次性执行的代码。当您首次运行对象时，**setupImpl** 方法将被调用一次。在此示例中，您通过打开文件写入二进制数据来分配文件资源。

```
methods
    function setupImpl(obj)
        obj.pFileID = fopen(obj.Filename,"wb");
        if obj.pFileID < 0
            error("Opening the file failed");
        end
    end
end
```

虽然不属于设置过程，但您应在您的代码使用完这些文件后关闭它们。您可使用 **releaseImpl** 方法来释放资源。

### 包含初始化和设置过程的完整类定义文件

```
classdef MyFile < matlab.System
% MyFile write numbers to a file

    % These properties are nontunable. They cannot be changed
    % after the setup method has been called or the object
    % is running.
    properties (Nontunable)
        Filename = "default.bin" % the name of the file to create
    end

    % These properties are private. Customers can only access
    % these properties through methods on this object
```

```

properties (Hidden,Access = private)
    pFileID; % The identifier of the file to open
end

methods (Access = protected)
    % In setup allocate any resources, which in this case
    % means opening the file.
    function setupImpl(obj)
        obj.pFileID = fopen(obj.Filename,'wb');
        if obj.pFileID < 0
            error("Opening the file failed");
        end
    end

    % This System
    % object™ writes the input to the file.
    function stepImpl(obj,data)
        fwrite(obj.pFileID,data);
    end

    % Use release to close the file to prevent the
    % file handle from being left open.
    function releaseImpl(obj)
        fclose(obj.pFileID);
    end
end

```

## 另请参阅

[matlab.System Constructor](#) | [releaseImpl](#) | [setupImpl](#) | [stepImpl](#)

## 相关示例

- “[释放 System object 资源](#)” (第 35-22 页)
- “[定义属性特性](#)” (第 35-24 页)
- “[调用序列摘要](#)” (第 35-45 页)

## 构造时设置属性值

此示例说明如何定义 System object 构造函数并允许其接受名称-值属性对组作为输入。

### 将属性设置为使用名称-值对组输入

定义 System object 构造函数，该函数是一种与类（此示例中为 MyFile）同名的方法。在该方法内，使用 `setProperties` 方法来设置所有公共属性，使它们可在用户构造对象时用于输入。`nargin` 是一个 MATLAB 函数，用于确定输入参数的数目。`varargin` 指示对象的所有公共属性。

```
methods
    function obj = MyFile(varargin)
        setProperties(obj,nargin,varargin{:});
    end
end
```

### 包含构造函数设置的完整类定义文件

```
classdef MyFile < matlab.System
% MyFile write numbers to a file

    % These properties are nontunable. They cannot be changed
    % after the setup method has been called or while the
    % object is running.
    properties (Nontunable)
        Filename ="default.bin" % the name of the file to create
        Access = 'wb' % The file access character vector (write, binary)
    end

    % These properties are private. Customers can only access
    % these properties through methods on this object
    properties (Hidden,Access = private)
        pFileID; % The identifier of the file to open
    end

    methods
        % You call setProperties in the constructor to let
        % a user specify public properties of object as
        % name-value pairs.
        function obj = MyFile(varargin)
            setProperties(obj,nargin,varargin{:});
        end
    end

    methods (Access = protected)
        % In setup allocate any resources, which in this case is
        % opening the file.
        function setupImpl(obj)
            obj.pFileID = fopen(obj.Filename,obj.Access);
            if obj.pFileID < 0
                error("Opening the file failed");
            end
        end

        % This System object writes the input to the file.
        function stepImpl(obj,data)
            fwrite(obj.pFileID,data);
        end
    end
end
```

```
end

% Use release to close the file to prevent the
% file handle from being left open.
function releaseImpl(obj)
    fclose(obj.pFileID);
end
end
end
```

## 另请参阅

[nargin | setProperties](#)

## 相关示例

- “[定义属性特性](#)” (第 35-24 页)
- “[释放 System object 资源](#)” (第 35-22 页)

## 重置算法并释放资源

### 本节内容

- “重置算法状态” (第 35-22 页)
- “释放 System object 资源” (第 35-22 页)

### 重置算法状态

当用户对 System object 调用 `reset` 时，将调用内部 `resetImpl` 方法。在此示例中，`pCount` 是 Counter System object 的内部计数器属性。当用户调用 `reset` 时，`pCount` 重置为 0。

```
classdef Counter < matlab.System
% Counter System object that increments a counter

    properties (Access = private)
        pCount
    end

    methods (Access = protected)
        % Increment the counter and return
        % its value as an output
        function c = stepImpl(obj)
            obj.pCount = obj.pCount + 1;
            c = obj.pCount;
        end

        % Reset the counter to zero.
        function resetImpl(obj)
            obj.pCount = 0;
        end
    end
end
```

### 释放 System object 资源

当对 System object 调用 `release` 时，如果以前调用过 `step` 或 `setup`，则调用内部 `releaseImpl` 方法（请参阅“调用序列摘要”（第 35-45 页））。此示例说明如何实现某些方法来释放由 System object 分配和使用的资源。这些资源包括分配的内存和用于读取或写入的文件。

此方法允许您清除白板图窗窗口中的坐标区，同时使图窗保持打开状态。

```
function releaseImpl(obj)
    cla(Whiteboard.getWhiteboard());
    hold on
end
```

有关 Whiteboard System object 的完整定义，请参阅“创建 Whiteboard System object”（第 35-28 页）。

### 另请参阅

`releaseImpl` | `resetImpl`

## 详细信息

- “调用序列摘要” (第 35-45 页)
- “初始化属性并设置一次性计算” (第 35-18 页)

## 定义属性特性

属性特性用于添加属性详细信息，可为属性提供控制层。除了 MATLAB 属性特性和属性验证，System object 还可以使用 **Nontunable** 或 **DiscreteState**。要指定多个特性，请用逗号分隔它们。

### 将属性指定为不可调属性

默认情况下，所有属性均为可调的，这意味着属性值可以随时更改。

当算法依赖于数据处理开始后始终恒定的值时，则要为属性应用 **Nontunable** 特性。将属性定义为不可调属性后，不需要检查或响应可变值，从而可以提高算法的效率。对于代码生成，将某个属性定义为不可调属性可以优化与该属性关联的内存。您应该将影响输入或输出端口数量的所有属性都定义为不可调属性。

使用 System object 时，只能在调用对象之前或调用 **release** 函数之后更改不可调属性。例如，您将 **InitialValue** 属性定义为 **nontunable**，并将其值设置为 0。

```
properties (Nontunable)
    InitialValue = 0;
end
```

### 将属性指定为离散状态属性

如果您的算法使用可保留状态的属性，则可以为这些属性分配 **DiscreteState** 特性。当用户调用 **getDiscreteState** 时，具有此特性的属性将通过 **getDiscreteStateImpl** 显示其状态值。以下限制适用于带有 **DiscreteState** 特性的属性：

- 数值、逻辑值或 fi 值，但非定标的双精度 fi 值
- 没有以下任何特性：**Nontunable**、**Dependent**、**Abstract**、**Constant**。
- 无默认值
- 不可公开设置
- 默认情况下 **GetAccess = Public**
- 当调用 System object 时，只能通过 **setupImpl** 设置值。（请参阅“调用序列摘要”（第 35-45 页）。）

例如，您将 **Count** 属性定义为离散状态：

```
properties (DiscreteState)
    Count;
end
```

### 具有各种属性特性的类示例

此示例包含两个不可调属性、一个离散状态属性以及一个用于设置属性特性的 MATLAB 类属性验证。

```
classdef Counter < matlab.System
% Counter Increment a counter to a maximum value

% These properties are nontunable. They cannot be changed
% after the setup method has been called or while the
% object is running.
properties (Nontunable)
    % The initial value of the counter
    initialValue = 0
    % The maximum value of the counter, must be a positive integer scalar
    maxValue (1, 1) {mustBePositive, mustBeInteger} = 3
```

```

end

properties
    % Whether to increment the counter, must be a logical scalar
    Increment (1, 1) logical = true
end

properties (DiscreteState)
    % Count state variable
    Count
end

methods (Access = protected)
    % Increment the counter and return its value
    % as an output

    function c = stepImpl(obj)
        if obj.Increment && (obj.Count < obj.MaxValue)
            obj.Count = obj.Count + 1;
        else
            disp(['Max count, ' num2str(obj.MaxValue) ', reached'])
        end
        c = obj.Count;
    end

    % Setup the Count state variable
    function setupImpl(obj)
        obj.Count = 0;
    end

    % Reset the counter to one.
    function resetImpl(obj)
        obj.Count = obj.InitialValue;
    end
end
end

```

## 另请参阅

### 详细信息

- “类属性”
- “属性特性”
- “当系统正在运行时您无法更改的内容”（第 35-9 页）
- “调用序列摘要”（第 35-45 页）

## 隐藏非活动属性

要仅显示活动的 System object 属性，请使用 `isInactivePropertyImpl` 方法。此方法用于指定某个属性是否为非活动属性。非活动属性是不因其他属性的值而影响 System object 的属性。当您将某个属性传递给 `isInactiveProperty` 方法，而此方法返回 `true` 时，则该属性为非活动属性，不会在调用 `disp` 函数时显示。

### 指定非活动属性

此示例使用 `isInactiveProperty` 方法来检查从属属性的值。对于此 System object，如果 `UseRandomInitialValue` 属性设置为 `true`，则 `InitialValue` 属性是无关的。此 `isInactiveProperty` 方法会检查该情况，如果 `UseRandomInitialValue` 为 `true`，则返回 `true` 以隐藏非活动的 `InitialValue` 属性。

```
methods (Access = protected)
    function flag = isInactivePropertyImpl(obj,propertyName)
        if strcmp(propertyName,'InitialValue')
            flag = obj.UseRandomInitialValue;
        else
            flag = false;
        end
    end
end
```

### 包含非活动属性方法的完整类定义文件

```
classdef Counter < matlab.System
    % Counter Increment a counter

    % These properties are nontunable. They cannot be changed
    % after the setup method has been called or when the
    % object is running.
    properties (Nontunable)
        % Allow the user to set the initial value
        UseRandomInitialValue = true
        InitialValue = 0
    end

    % The private count variable, which is tunable by default
    properties (Access = private)
        pCount
    end

    methods (Access = protected)
        % Increment the counter and return its value
        % as an output
        function c = stepImpl(obj)
            obj.pCount = obj.pCount + 1;
            c = obj.pCount;
        end

        % Reset the counter to either a random value or the initial
        % value.
        function resetImpl(obj)
            if obj.UseRandomInitialValue
```

```
obj.pCount = rand();
else
    obj.pCount = obj.InitialValue;
end
end

% This method controls visibility of the object's properties
function flag = isInactivePropertyImpl(obj,propertyName)
if strcmp(propertyName,'InitialValue')
    flag = obj.UseRandomInitialValue;
else
    flag = false;
end
end
end
```

## 另请参阅

[isInactivePropertyImpl](#)

## 将属性值限制为有限列表

如果要创建具有一组有限的可接受值的 System object 属性，您可以使用枚举或属性验证。

对于在 Simulink 的 MATLAB System 模块中使用的 System object，您可以使用枚举或属性验证。如果使用枚举，枚举也可以从 `Simulink.IntEnumType` 派生。您可以使用此类型的枚举将自定义头等属性添加到 MATLAB System 模块的输入或输出。请参阅“在 Simulink 模型中使用枚举数据”(Simulink)。

### 使用 `mustBeMember` 进行属性验证

要使用属性验证来限制属性值，您可以使用 `mustBeMember` 验证函数。

以下示例定义一个 `Style` 属性，其值可以是 `solid`、`dash` 或 `dot`。默认值为 `solid`。(1,1) 将属性定义为标量。

```
properties
    Style (1,1) string {mustBeMember(Style, ["solid","dash","dot"])} = "solid";
end
```

### 枚举属性

要在 System object 中使用枚举数据，您应在 System object 类定义中引用枚举作为属性，并在单独的类定义文件中定义该枚举类。

要创建枚举属性，您需要：

- 设置为枚举类的 System object 属性。
- 关联的枚举类定义，用于为属性定义所有可能的值。

此示例为 System object 定义颜色枚举属性。枚举类 `ColorValues` 的定义如下：

```
classdef ColorValues < int32
    enumeration
        blue (0)
        red (1)
        green (2)
    end
end
```

`ColorValues` 类继承自 `int32`，以实现代码生成兼容性。枚举值必须是有效的 MATLAB 标识符（第 1-5 页）。

在 System object 中，`Color` 属性定义为 `ColorValues` 对象，`blue` 为默认值。(1,1) 将 `Color` 属性定义为标量：

```
properties
    Color (1, 1) ColorValues = ColorValues.blue
end
```

### 创建 Whiteboard System object

此示例说明 Whiteboard System object 的类定义、两种类型的有限列表属性以及如何使用该对象。每次运行 Whiteboard 对象时，它都会在白板上绘制一条线。

## Whiteboard System object 的定义

```

type('Whiteboard.m')

classdef Whiteboard < matlab.System
    % Whiteboard Draw lines on a figure window
    %

    properties(Nontunable)
        Color (1, 1) ColorValues = ColorValues.blue
        Style (1,1) string {mustBeMember(Style, ["solid","dash","dot"])} = "solid";
    end

    methods (Access = protected)
        function stepImpl(obj)
            h = Whiteboard.getWhiteboard();
            switch obj.Style
                case "solid"
                    linestyle = "-";
                case "dash"
                    linestyle = "--";
                case "dot"
                    linestyle = ":";
            end
            plot(h, randn([2,1]), randn([2,1]), ...
                  "Color",string(obj.Color), "LineStyle",linestyle);
        end

        function releaseImpl(~)
            cla(Whiteboard.getWhiteboard());
            hold on
        end
    end

    methods (Static)
        function a = getWhiteboard()
            h = findobj('tag','whiteboard');
            if isempty(h)
                h = figure('tag','whiteboard');
                hold on
            end
            a = gca;
        end
    end
end

```

构造 System object。

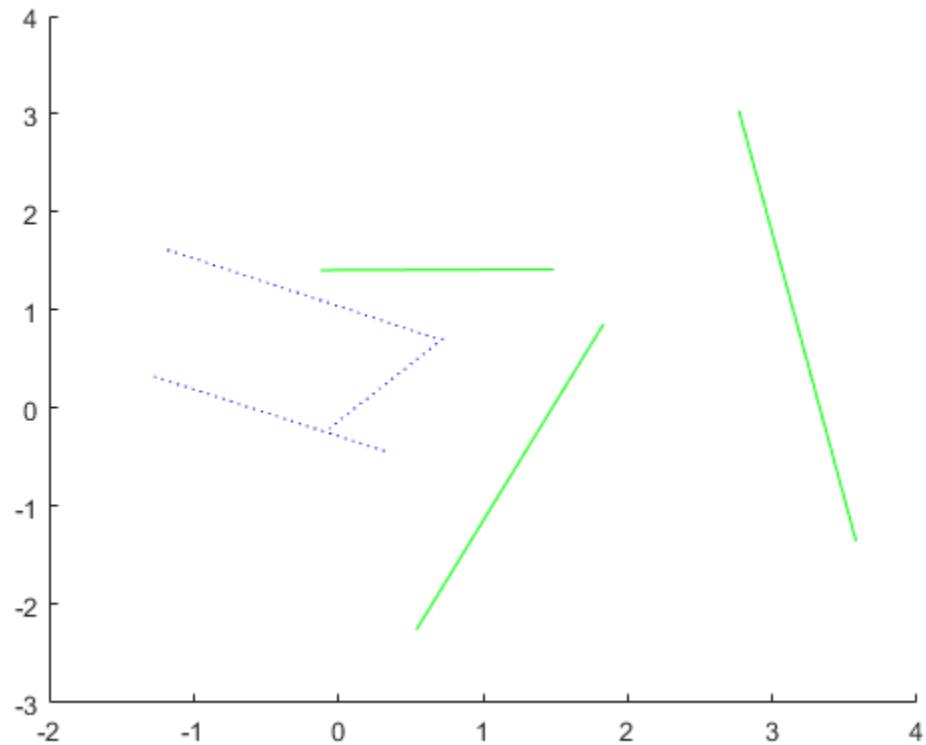
```
greenInk = Whiteboard;
blueInk = Whiteboard;
```

更改颜色并设置蓝色线型。

```
greenInk.Color = "green";
blueInk.Color = "blue";
blueInk.Style = "dot";
```

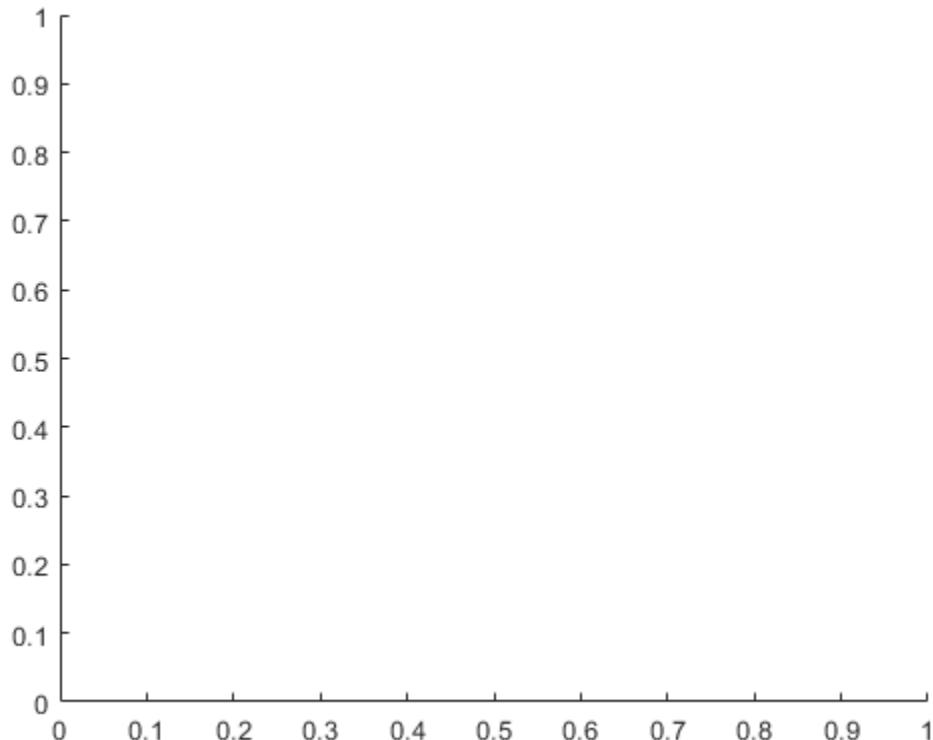
绘制几个线条。

```
for i=1:3
    greenInk();
    blueInk();
end
```



清空白板

```
release(greenInk);
release(blueInk);
```



## 另请参阅

### 相关示例

- “验证属性值”
- “枚举”
- “Code Generation for Enumerations” (MATLAB Coder)

## 处理调整后的属性

此示例说明如何指定当可调属性值在仿真过程中更改时要执行的操作。

`processTunedPropertiesImpl` 方法对于管理操作以避免重复执行十分有用。在许多情况下，更改多个相互依赖的属性之一会导致执行某个操作。通过 `processTunedPropertiesImpl` 方法，您可以控制执行该操作的时间，以使其不会被不必要的重复执行。

### 控制生成查找表的时间

以下示例的 `processTunedPropertiesImpl` 将导致在 `NumNotes` 或 `MiddleC` 属性更改时重新生成 `pLookupTable`。

```
methods (Access = protected)
    function processTunedPropertiesImpl(obj)
        propChange = isChangedProperty(obj,'NumNotes') || ...
            isChangedProperty(obj,'MiddleC')
        if propChange
            obj.pLookupTable = obj.MiddleC * ...
                (1+log(1:obj.NumNotes)/log(12));
        end
    endend
```

### 包含调整后的属性处理的完整类定义文件

```
classdef TuningFork < matlab.System
    % TuningFork Illustrate the processing of tuned parameters
    %

    properties
        MiddleC = 440
        NumNotes = 12
    end

    properties (Access = private)
        pLookupTable
    end

    methods (Access = protected)
        function resetImpl(obj)
            obj.MiddleC = 440;
            obj.pLookupTable = obj.MiddleC * ...
                (1+log(1:obj.NumNotes)/log(12));
        end

        function hz = stepImpl(obj,noteShift)
            % A noteShift value of 1 corresponds to obj.MiddleC
            hz = obj.pLookupTable(noteShift);
        end

        function processTunedPropertiesImpl(obj)
            propChange = isChangedProperty(obj,'NumNotes') || ...
                isChangedProperty(obj,'MiddleC')
            if propChange
                obj.pLookupTable = obj.MiddleC * ...
                    (1+log(1:obj.NumNotes)/log(12));
            end
        end
    end
```

```
end  
end
```

**另请参阅**

processTunedPropertiesImpl

## 定义复合的 System object

此示例说明如何定义包含其他 System object 的 System object。基于单独的高通和低通滤波器 System object 定义一个带通滤波器 System object。

### 将 System object 存储在属性中

要基于其他 System object 来定义 System object，请将这些其他对象以属性方式存储在类定义文件中。在此示例中，高通滤波器和低通滤波器为它们各自的类定义文件中定义的单独 System object。

```
properties (Access = private)
    % Properties that hold filter System objects
    pLowpass
    pHightpass
end
```

### 带通滤波器复合 System object 的完整类定义文件

```
classdef BandpassFIRFilter < matlab.System
% Implements a bandpass filter using a cascade of eighth-order lowpass
% and eighth-order highpass FIR filters.
```

```
properties (Access = private)
    % Properties that hold filter System objects
    pLowpass
    pHightpass
end

methods (Access = protected)
    function setupImpl(obj)
        % Setup composite object from constituent objects
        obj.pLowpass = LowpassFIRFilter;
        obj.pHighpass = HighpassFIRFilter;
    end

    function yHigh = stepImpl(obj,u)
        yLow = obj.pLowpass(u);
        yHigh = obj.pHighpass(yLow);
    end

    function resetImpl(obj)
        reset(obj.pLowpass);
        reset(obj.pHighpass);
    end
end
end
```

### 带通滤波器的低通 FIR 组件的类定义文件

```
classdef LowpassFIRFilter < matlab.System
% Implements eighth-order lowpass FIR filter with 0.6pi cutoff

properties (Nontunable)
    % Filter coefficients
    Numerator = [0.006,-0.0133,-0.05,0.26,0.6,0.26,-0.05,-0.0133,0.006];
end
```

```
properties (DiscreteState)
```

```

State
end

methods (Access = protected)
function setupImpl(obj)
    obj.State = zeros(length(obj.Numerator)-1,1);
end

function y = stepImpl(obj,u)
[y,obj.State] = filter(obj.Numerator,1,u,obj.State);
end

function resetImpl(obj)
    obj.State = zeros(length(obj.Numerator)-1,1);
end
end

```

### 带通滤波器的高通 FIR 组件的类定义文件

```

classdef HighpassFIRFilter < matlab.System
% Implements eighth-order highpass FIR filter with 0.4pi cutoff

properties (Nontunable)
% Filter coefficients
    Numerator = [0.006,0.0133,-0.05,-0.26,0.6,-0.26,-0.05,0.0133,0.006];
end

properties (DiscreteState)
    State
end

methods (Access = protected)
function setupImpl(obj)
    obj.State = zeros(length(obj.Numerator)-1,1);
end

function y = stepImpl(obj,u)
[y,obj.State] = filter(obj.Numerator,1,u,obj.State);
end

function resetImpl(obj)
    obj.State = zeros(length(obj.Numerator)-1,1);
end
end

```

### 另请参阅

nargin

## 定义有限源对象

此示例说明如何定义一个从文件执行特定步数或特定读取次数的 System object。

### 本节内容

- “使用 FiniteSource 类并指定源的结尾”（第 35-36 页）
- “包含有限源的完整类定义文件”（第 35-36 页）

## 使用 FiniteSource 类并指定源的结尾

- 1 从有限源类创建子类。

```
classdef RunTwice < matlab.System & ...
    matlab.system.mixin.FiniteSource
```

- 2 使用 `isDoneImpl` 方法指定源的结尾。在此示例中，源带有两次迭代。

```
methods (Access = protected)
    function bDone = isDoneImpl(obj)
        bDone = obj.NumSteps==2
    end
```

## 包含有限源的完整类定义文件

```
classdef RunTwice < matlab.System & ...
    matlab.system.mixin.FiniteSource
    % RunTwice System object that runs exactly two times
    %
properties (Access = private)
    NumSteps
end

methods (Access = protected)
    function resetImpl(obj)
        obj.NumSteps = 0;
    end

    function y = stepImpl(obj)
        if ~obj.isDone()
            obj.NumSteps = obj.NumSteps + 1;
            y = obj.NumSteps;
        else
            y = 0;
        end
    end

    function bDone = isDoneImpl(obj)
        bDone = obj.NumSteps==2;
    end
end
end
```

## 另请参阅

`matlab.system.mixin.FiniteSource`

## 详细信息

- “Subclassing Multiple Classes”
- “在方法定义中使用 ~ 作为输入参数” (第 35-50 页)

## 保存和加载 System object

此示例说明如何加载和保存 System object。

### 保存 System object 和子对象

定义 `saveObjectImpl` 方法，以指定当用户保存 System object 时不仅仅应保存公共属性。在此方法内，使用默认的 `saveObjectImpl@matlab.System` 将公共属性保存到结构体 `s`。使用 `saveObject` 方法保存子对象。保存受保护的属性和从属属性，最后，如果该对象已被调用并且未释放，则保存对象状态。

```
methods (Access = protected)
    function s = saveObjectImpl(obj)
        s = saveObjectImpl@matlab.System(obj);
        s.child = matlab.System.saveObject(obj.child);
        s.protectedprop = obj.protectedprop;
        s.pdependentprop = obj.pdependentprop;
        if isLocked(obj)
            s.state = obj.state;
        end
    end
end
```

### 加载 System object 和子对象

定义 `loadObjectImpl` 方法以加载以前保存的 System object。在此方法内，使用 `loadObject` 加载子级 System object、加载受保护的属性和私有属性、加载状态（如果对象被调用且未释放），然后使用基类中的 `loadObjectImpl` 加载公共属性。

```
methods (Access = protected)
    function loadObjectImpl(obj,s,isInUse)
        obj.child = matlab.System.loadObject(s.child);

        obj.protectedprop = s.protectedprop;
        obj.pdependentprop = s.pdependentprop;

        if isInUse
            obj.state = s.state;
        end

        loadObjectImpl@matlab.System(obj,s,isInUse);
    end
end
```

### 包含保存和加载的完整类定义文件

`Counter` 类定义文件使用 `count` 属性来设置对象。此计数器在 `MySaveLoader` 类定义文件中用来计算子对象的数目。

```
classdef Counter < matlab.System
    properties(DiscreteState)
        Count
    end
    methods (Access=protected)
```

```

function setupImpl(obj, ~)
    obj.Count = 0;
end
function y = stepImpl(obj, u)
    if u > 0
        obj.Count = obj.Count + 1;
    end
    y = obj.Count;
end
end
end

classdef MySaveLoader < matlab.System

properties (Access = private)
    child
    pdependentprop = 1
end

properties (Access = protected)
    protectedprop = rand;
end

properties (DiscreteState = true)
    state
end

properties (Dependent)
    dependentprop
end

methods
    function obj = MySaveLoader(varargin)
        obj@matlab.System();
        setProperties(obj,nargin,varargin{:});
    end

    function set.dependentprop(obj, value)
        obj.pdependentprop = min(value, 5);
    end

    function value = get.dependentprop(obj)
        value = obj.pdependentprop;
    end
end

methods (Access = protected)
    function setupImpl(obj)
        obj.state = 42;
        obj.child = Counter;
    end
    function out = stepImpl(obj,in)
        obj.state = in + obj.state + obj.protectedprop + ...
            obj.pdependentprop;
        out = obj.child(obj.state);
    end
end

```

```
% Serialization
methods (Access = protected)
    function s = saveObjectImpl(obj)
        % Call the base class method
        s = saveObjectImpl@matlab.System(obj);

        % Save the child System objects
        s.child = matlab.System.saveObject(obj.child);

        % Save the protected & private properties
        s.protectedprop = obj.protectedprop;
        s.pdependentprop = obj.pdependentprop;

        % Save the state only if object called and not released
        if isLocked(obj)
            s.state = obj.state;
        end
    end

function loadObjectImpl(obj,s,isInUse)
    % Load child System objects
    obj.child = matlab.System.loadObject(s.child);

    % Load protected and private properties
    obj.protectedprop = s.protectedprop;
    obj.pdependentprop = s.pdependentprop;

    % Load the state only if object is in use
    if isInUse
        obj.state = s.state;
    end

    % Call base class method to load public properties
    loadObjectImpl@matlab.System(obj,s,isInUse);
end
end
end
```

### 另请参阅

[loadObjectImpl](#) | [saveObjectImpl](#)

# 定义 System object 信息

此示例说明如何为 System object 定义要显示的信息。

## 定义 System object 信息

您可以定义您自己的 `info` 方法，以显示您的 System object 的特定信息。默认的 `infoImpl` 方法会返回一个空结构体。使用 `info(x,'details')` 调用 `info` 时，此 `infoImpl` 方法将返回详细信息；使用 `info(x)` 调用时，则仅返回计数信息。

```
methods (Access = protected)
    function s = infoImpl(obj,varargin)
        if nargin>1 && strcmp('details',varargin(1))
            s = struct('Name','Counter',...
                'Properties', struct('CurrentCount',obj.Count, ...
                'Threshold',obj.Threshold));
        else
            s = struct('Count',obj.Count);
        end
    end
end
```

## 使用 InfoImpl 完成类定义文件

```
classdef Counter < matlab.System
    % Counter Count values above a threshold

    properties
        Threshold = 1
    end

    properties (DiscreteState)
        Count
    end

    methods (Access = protected)
        function setupImpl(obj)
            obj.Count = 0;
        end

        function resetImpl(obj)
            obj.Count = 0;
        end

        function y = stepImpl(obj,u)
            if (u > obj.Threshold)
                obj.Count = obj.Count + 1;
            end
            y = obj.Count;
        end

        function s = infoImpl(obj,varargin)
            if nargin>1 && strcmp('details',varargin(1))
                s = struct('Name','Counter',...
                    'Properties', struct('CurrentCount', ...
                    'obj.Count','Threshold',obj.Threshold));
            else
```

```
s = struct('Count',obj.Count);
end
end
end
end
```

**另请参阅**

infoImpl

# 处理输入设定更改

此示例说明如何控制 System object 的输入设定。您可以控制输入设定更改后产生的效果。

您还可以限制能否在对象使用过程中更改输入的复/实性、数据类型或大小。在您施加限制后，用户必须调用 `release`，才能更改这些方面。

## 响应输入设定更改

要在输入更改了大小、数据类型或复/实性时修改 System object 算法或属性，请实现 `processInputSpecificationChangeImpl` 方法。指定 System object 的各次调用之间输入设定更改时要执行的操作。

在此示例中，当任一输入为复数时，`processInputSpecificationChangeImpl` 会更改 `isComplex` 属性。

```
properties(Access = private)
    isComplex (1,1) logical = false;
end

methods (Access = protected)
    function processInputSpecificationChangeImpl(obj,input1,input2)
        if(isreal(input1) && isreal(input2))
            obj.isComplex = false;
        else
            obj.isComplex = true;
        end
    end
end
```

## 限制输入设定更改

要指定不能在使用 System object 使用过程中更改输入的复/实性、数据类型和大小，请实现 `isInputComplexityMutableImpl`、`isInputDataTypeMutableImpl`、`isInputSizeMutableImpl` 方法以返回 `false`。如果只想限制 System object 输入的某些方面，您可以只包含其中一个或两个方法。

```
methods (Access = protected)
    function flag = isInputComplexityMutableImpl(~,~)
        flag = false;
    end
    function flag = isInputSizeData-TypeImpl(~,~)
        flag = false;
    end
    function flag = isInputSizeMutableImpl(~,~)
        flag = false;
    end
end
```

## 完整的类定义文件

以下 CounterSystem object 限制输入设定的所有三个方面。

```
classdef Counter < matlab.System
    %Counter Count values above a threshold
```

```
properties
    Threshold = 1
end

properties (DiscreteState)
    Count
end

methods
    function obj = Counter(varargin)
        setProperties(obj,nargin,varargin{:});
    end
end

methods (Access=protected)
    function resetImpl(obj)
        obj.Count = 0;
    end

    function y = stepImpl(obj, u1)
        if (any(u1 >= obj.Threshold))
            obj.Count = obj.Count + 1;
        end
        y = obj.Count;
    end

    function flag = isInputComplexityMutableImpl(~,~)
        flag = false;
    end

    function flag = isInputDataTypeMutableImpl(~,~)
        flag = false;
    end

    function flag = isInputSizeMutableImpl(~,~)
        flag = false;
    end
end
```

### 另请参阅

[isInputSizeMutableImpl](#)

### 相关示例

- “当系统正在运行时您无法更改的内容” (第 35-9 页)

# 调用序列摘要

## 本节内容

- “setup 调用序列” (第 35-45 页)
- “运行对象或 step 调用序列” (第 35-45 页)
- “reset 方法调用序列” (第 35-46 页)
- “release 方法调用序列” (第 35-47 页)

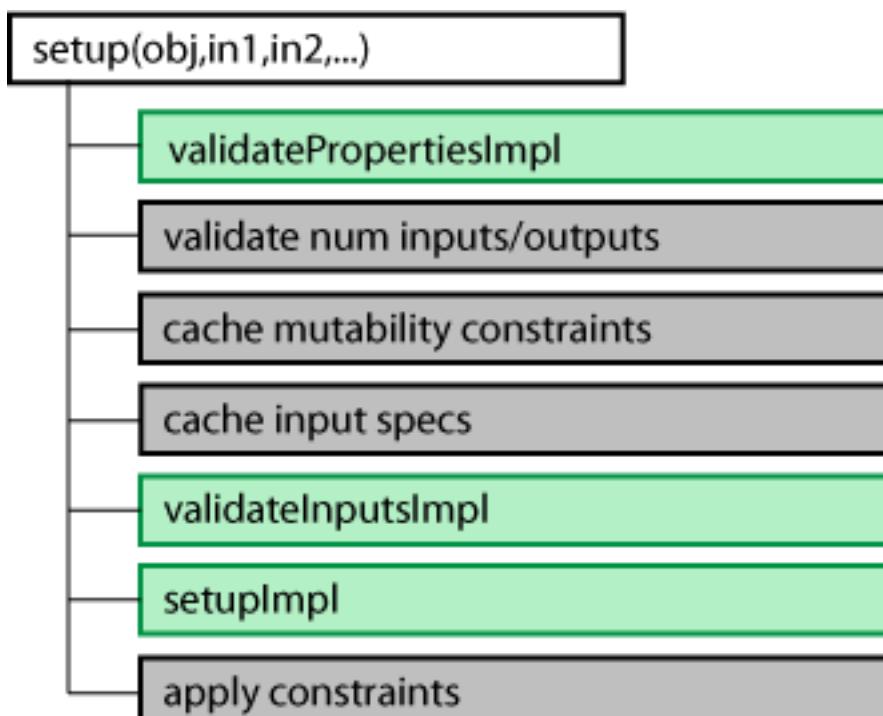
下列各图显示调用 System object 时所执行的操作的摘要视图。各项操作的背景色表示调用的类型。

- 灰色背景 - 内部操作
- 绿色背景 - 作者实现的方法
- 白色背景 - 用户可访问的函数

如果您需要更详细的调用序列, 请参阅 “详细的调用序列” (第 35-48 页)。

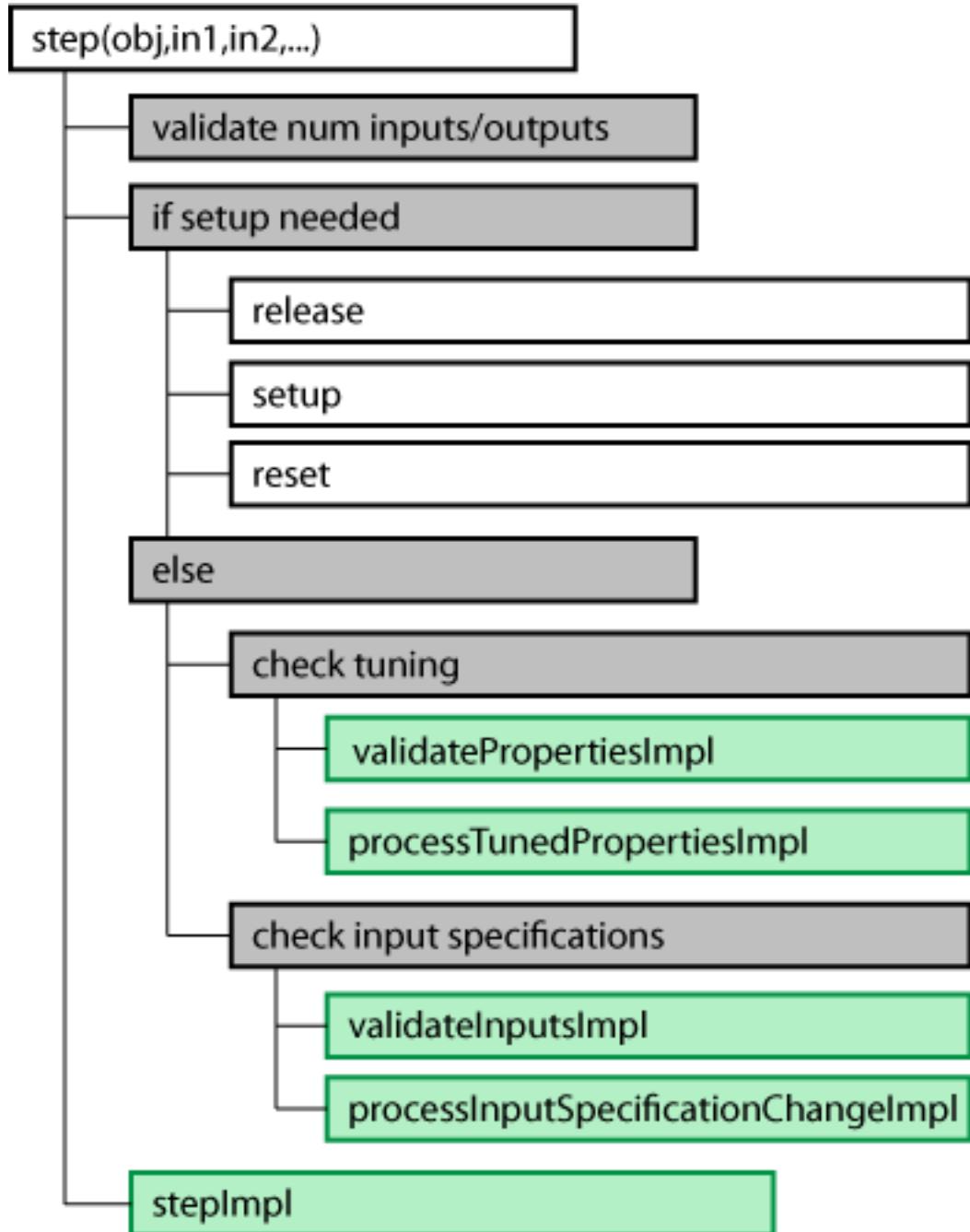
## setup 调用序列

以下层次结构显示了在您调用 **setup** 函数时执行的操作。



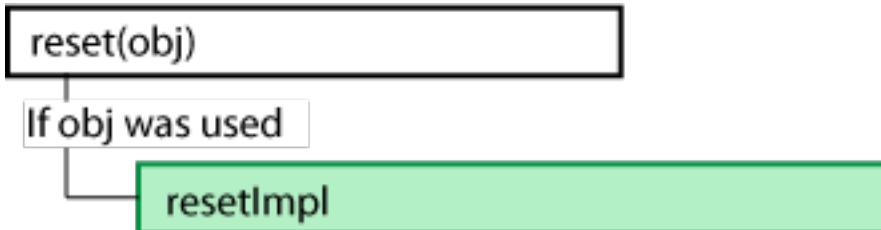
## 运行对象或 step 调用序列

以下层次结构显示了在您调用 **step** 函数时执行的操作。



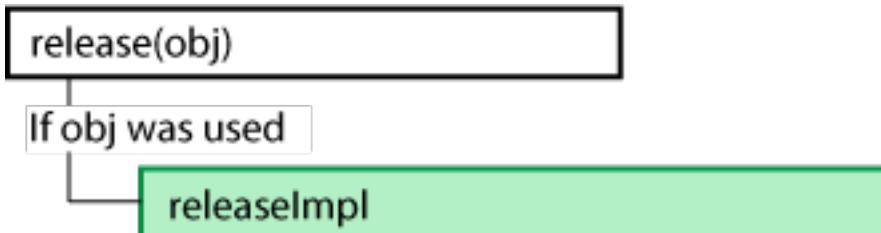
### reset 方法调用序列

以下层次结构显示了在您调用 `reset` 函数时执行的操作。



## release 方法调用序列

以下层次结构显示了在您调用 `release` 函数时执行的操作。



## 另请参阅

[releaseImpl](#) | [resetImpl](#) | [setupImpl](#) | [stepImpl](#)

## 相关示例

- “释放 System object 资源” (第 35-22 页)
- “重置算法状态” (第 35-22 页)
- “构造时设置属性值” (第 35-20 页)
- “定义基本 System object” (第 35-11 页)

## 详细的调用序列

本节内容
“setup 调用序列”（第 35-48 页）
“运行对象或 step 调用序列”（第 35-48 页）
“reset 调用序列”（第 35-49 页）
“release 调用序列”（第 35-49 页）

调用序列图显示了在您运行指定方法时调用内部方法的顺序。如果 System object 不覆盖指定的方法，则使用该方法的默认实现。

如果您要查看各类方法调用的摘要视图，请参阅“调用序列摘要”（第 35-45 页）。

### setup 调用序列

首次运行 System object 时，将调用 `setup` 以执行一次性设置任务。各方法的调用序列如下：

- 1 如果 System object 不在使用中，`release`（第 35-49 页）
- 2 `validatePropertiesImpl`
- 3 `isDiscreteStateSpecificationMutableImpl`
- 4 `isInputDataTypeMutableImpl`
- 5 `isInputComplexityMutableImpl`
- 6 `isInputSizeMutableImpl`
- 7 `isTunablePropertyDataTypeMutableImpl`
- 8 `validateInputsImpl`
- 9 如果 System object 使用非直接馈通方法，请调用 `isInputDirectFeedthroughImpl`
- 10 `setupImpl`

### 运行对象或 step 调用序列

以函数形式调用对象或调用 `step` 在 MATLAB 中运行 System object 时，方法的调用序列如下：

- 1 如果 System object 不在使用中（对象刚刚创建或释放），则依次调用
  - a `release`（第 35-47 页）
  - b `setup`（第 35-48 页）
  - c `reset`（第 35-49 页）

否则，如果对象正在使用中（调用了对象但未调用 `release`），则：

- a 如果可调属性已更改，则依次调用
  - i `validatePropertiesImpl`
  - ii `processTunedPropertiesImpl`
- b 如果输入大小、数据类型或复/实性已更改，则依次调用

- i validateInputsImpl
- ii processInputSpecificationChangeImpl

## reset 调用序列

当调用 **reset** 时，将执行以下操作。

- 1 如果对象正在使用中（对象被调用且未释放），则调用 **resetImpl**

## release 调用序列

当调用 **release** 时，将执行以下操作。

- 1 如果对象正在使用中（对象被调用且未释放），则调用 **releaseImpl**

## 另请参阅

[releaseImpl](#) | [resetImpl](#) | [setupImpl](#) | [stepImpl](#)

## 相关示例

- “释放 System object 资源”（第 35-22 页）
- “重置算法状态”（第 35-22 页）
- “构造时设置属性值”（第 35-20 页）
- “定义基本 System object”（第 35-11 页）

## 定义 System object 的技巧

System object 是一种专用的 MATLAB 对象，针对迭代处理进行了优化。当您需要多次运行某个对象或循环处理数据时，可使用 System object。定义您自己的 System object 时，请按照以下建议帮助您的 System object 更快地运行。

### 常规

- 在 `setupImpl` 方法中定义所有一次性计算，并将结果缓存在私有属性中。对重复的计算使用 `stepImpl` 方法。
- 分别使用 `true` 或 `false` 而不是 `1` 或 `0` 来指定布尔值。
- 如果某个方法中的变量不需要在两次调用之间保留其值，请对该方法中的这些变量使用局部作用域。

### 输入和输出

- 有些方法使用 `stepImpl` 算法输入作为输入，例如 `setupImpl`、`updateImpl`、`validateInputsImpl`、`isInputDirectFeedThroughImpl` 和 `processInputSizeChangeImpl`。这些输入必须与 `stepImpl` 的输入顺序匹配，但不需要与输入数目匹配。如果您的实现不需要为 System object 提供任何输入，则可以将它们全部关闭。
- 对于 `getNumInputsImpl` 和 `getNumOutputsImpl` 方法，如果您从某个对象属性设置返回参数，则该对象属性必须具有 `Nontunable` 特性。

### 在方法定义中使用 ~ 作为输入参数

所有方法（静态方法除外）都期望将 System object 句柄作为第一个输入参数。您可以对 System object 句柄使用任何名称。由 MATLAB 编辑器菜单插入的代码使用 `obj`。

在许多示例中，使用 `~` 来指示函数中未使用对象句柄，而不是传入对象句柄。使用 `~` 而不是对象句柄可防止出现有关未使用的变量的警告。

### 属性

- 对于不会更改的属性，请将它们定义为 `Nontunable` 属性。`Tunable` 属性比 `Nontunable` 属性的访问时间慢。
- 只要有可能，尽量对属性使用 `protected` 或 `private` 特性，而不要使用 `public` 特性。某些 `public` 属性比 `protected` 和 `private` 属性的访问时间慢。
- 如果在 `stepImpl` 方法中多次访问了属性，请将这些属性以局部变量的方式缓存在该方法中。多次属性访问的典型示例为循环。使用缓存的局部变量的迭代计算的运行速度比必须访问对象属性的计算快。当此方法的计算完成后，您可以将本地缓存的结果重新保存到该 System object 的属性中。将经常使用的可调属性复制到私有属性中。该最佳做法也适用于 `updateImpl` 和 `outputImpl` 方法。

例如，在下面的代码中，`k` 在每次循环迭代中被多次访问，但只有一次保存到对象属性中。

```
function y = stepImpl(obj,x)
    k = obj.MyProp;
    for p=1:100
        y = k * x;
        k = k + 0.1;
    end
```

```

obj.MyProp = k;
end

```

- 属性的默认值在对象的所有实例之间共享。一个类的两个实例可以访问同一个默认值（如果该属性尚未被任何一个实例覆盖）。

## 文本比较

不要在 `stepImpl` 方法中使用字符向量比较或基于字符向量的 `switch` 语句，而应在 `setupImpl` 中创建一个方法句柄。此句柄指向同一类定义文件中的方法。在 `stepImpl` 的循环中使用该句柄。

此示例说明如何在循环中使用方法句柄和缓存的局部变量，以实现一个高效的对象。在 `setupImpl` 中，根据字符向量比较选择 `myMethod1` 或 `myMethod2`，然后将方法句柄赋给 `pMethodHandle` 属性。由于 `stepImpl` 中存在一个循环，因此请将 `pMethodHandle` 属性赋给本地方法句柄 `myFun`，然后在该循环中使用 `myFun`。

```

classdef MyClass < matlab.System
function setupImpl(obj)
    if strcmp(obj.Method, 'Method1')
        obj.pMethodHandle = @myMethod1;
    else
        obj.pMethodHandle = @myMethod2;
    end
end
function y = stepImpl(obj,x)
    myFun = obj.pMethodHandle;
    for p=1:1000
        y = myFun(obj,x)
    end
end
function y = myMethod1(x)
    y = x+1;
end
function y = myMethod2(x)
    y = x-1;
end
end

```

## Simulink

对于要包含在 Simulink 中的 System object，请添加 `StrictDefaults` 属性。该属性将所有 `MutableImpl` 方法设置为默认返回 `false`。

## 代码生成

有关 System object 和代码生成的信息，请参阅“MATLAB 代码生成中的 System object”(MATLAB Coder)。

## 使用 MATLAB 编辑器插入 System object 代码

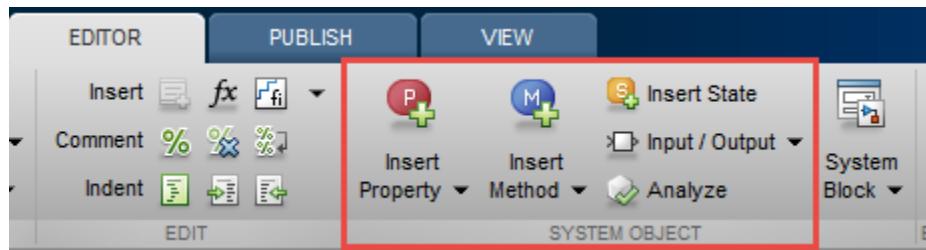
### 本节内容

- “通过代码插入定义 System object”（第 35-52 页）
- “创建温度枚举属性”（第 35-54 页）
- “为冻结点创建自定义属性”（第 35-55 页）
- “添加方法来验证输入”（第 35-56 页）

### 通过代码插入定义 System object

您可以在 MATLAB 编辑器中使用代码插入选项来定义 System object。当您选择这些选项时，MATLAB 编辑器可为 System object 添加预定义的属性、方法、状态、输入或输出。使用这些工具可更快地创建并修改 System object，以及通过减少键入错误来提高准确性。

要访问 System object 编辑选项，请创建一个新的或打开一个现有的 System object。

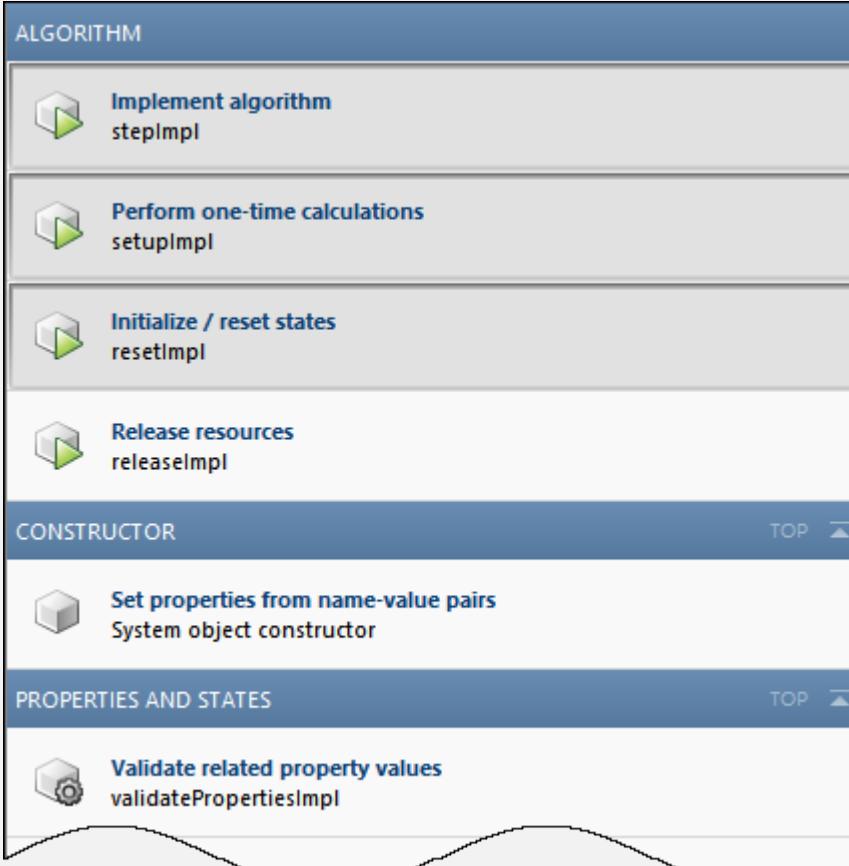


要为 System object 添加预定义的代码，请从相应的菜单中选择该代码。例如，当您点击**插入属性 > 数值**时，MATLAB 编辑器将添加以下代码：

```
properties(Nontunable)
    Property
end
```

MATLAB 编辑器会插入具有默认名称 **Property** 的新属性，您可以对该名称进行重命名。如果您的现有属性组带有 **Nontunable** 特性，则 MATLAB 编辑器会将新属性插入到该组中。如果您没有属性组，MATLAB 编辑器将创建一个具有正确特性的属性组。

## 插入选项

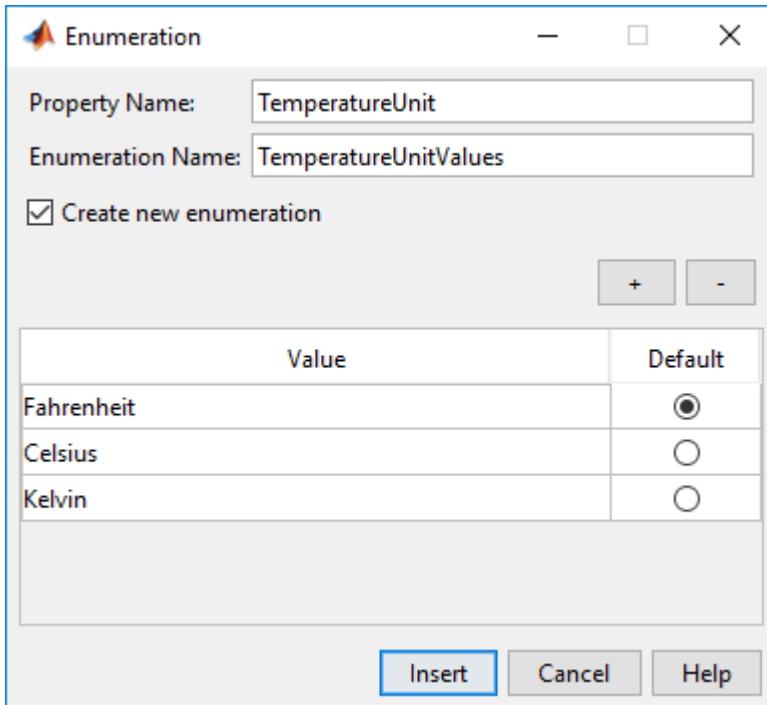
属性	System object 的属性：数值、逻辑、枚举、正整数、可调数值、私有、受保护和自定义。当您选择“枚举”或“自定义”属性时，将打开一个单独的对话框，以指导您创建这些属性。
方法	<p>System object 定义中的常用方法。MATLAB 编辑器只会创建方法结构体。您需要指定该方法的操作。</p> <p><b>插入方法</b>菜单按类别来整理方法，例如<b>算法</b>、<b>输入和输出</b>以及<b>属性和状态</b>。当您从该菜单中选择一种方法时，MATLAB 编辑器会将方法模板插入到 System object 代码中。在此示例中，选择<b>插入方法</b> &gt; <b>释放资源</b>将插入以下代码：</p> <pre>function releaseImpl(obj)     % Release resources, such as file handles end</pre> <p>如果<b>插入方法</b>菜单中的某个方法已存在于 System object 代码中，则该方法在<b>插入方法</b>菜单中灰显：</p> 
状态	包含 DiscreteState 特性的属性。

输入/输出	<p>输入、输出以及相关方法，例如验证输入和不允许更改输入大小。</p> <p>当您选择某个输入或输出时，MATLAB 编辑器将在 <code>stepImpl</code> 方法中插入指定代码。在此示例中，选择插入 &gt; 输入后，MATLAB 编辑器将插入所需的输入变量 <code>u2</code>。MATLAB 编辑器会确定变量名称，但您可以在插入变量后对其进行更改。</p> <pre><code>function y = stepImpl(obj,u,u2)     % Implement algorithm. Calculate y as a function of     % input u and discrete states.     y = u; end</code></pre>
-------	--

## 创建温度枚举属性

- 1 打开一个新的或现有的 System object。
- 2 在 MATLAB 编辑器中，选择插入属性 > 枚举。
- 3 在枚举对话框中，输入：
  - a 属性名称为 `TemperatureUnit`。
  - b 枚举名称为 `TemperatureUnitValues`。
- 4 选中创建新枚举复选框。
- 5 使用 - (减号) 按钮删除现有枚举值。
- 6 使用 + (加号) 按钮和以下值添加三个枚举值：
  - `Fahrenheit`
  - `Celsius`
  - `Kelvin`
- 7 点击默认选择 `Fahrenheit` 作为默认值。

该对话框现在看上去如下所示：



- 8 要创建此枚举和关联的类, 请点击**插入**。
- 9 在 MATLAB 编辑器中, 将创建具有该枚举定义的附加类文件。将该枚举类定义文件另存为 TemperatureUnitValues.m。

```
classdef TemperatureUnitValues < int32
    enumeration
        Fahrenheit (0)
        Celsius (1)
        Kelvin (2)
    end
end
```

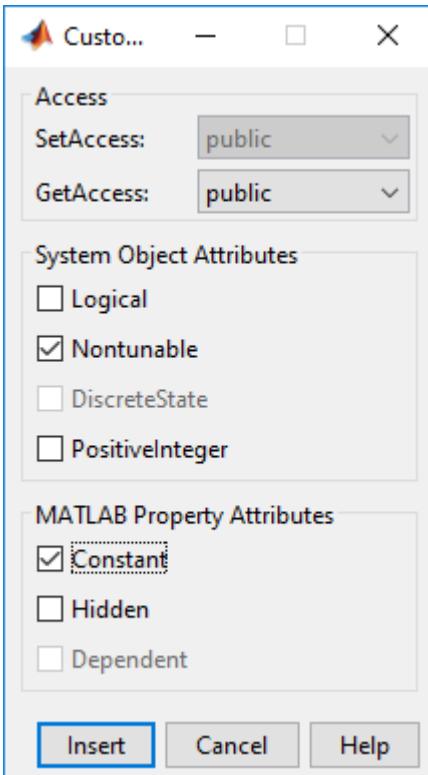
在 System object 类定义中, 添加了以下代码:

```
properties(Nontunable)
    TemperatureUnit (1, 1) TemperatureUnitValues = TemperatureUnitValues.Fahrenheit
end
```

有关枚举的详细信息, 请参阅“将属性值限制为有限列表”(第 35-28 页)。

## 为冻结点创建自定义属性

- 1 打开一个新的或现有的 System object。
- 2 在 MATLAB 编辑器中, 选择**插入属性** > **自定义属性**。
- 3 在“自定义属性”对话框的 **System object 属性**下, 选择 **Nontunable**。在 **MATLAB 属性特性**下, 选择 **Constant**。将 **GetAccess** 保留设置为“public”。**SetAccess** 将被灰显, 因为无法使用 System object 方法来设置类型为常量的属性。



4 点击插入，并将以下代码插入到 System object 定义中：

```
properties(Nontunable, Constant)
    Property
end
```

5 将 Property 替换为您的属性。

```
properties(Nontunable, Constant)
    FreezingPointFahrenheit = 32;
end
```

### 添加方法来验证输入

- 1 打开一个新的或现有的 System object。
- 2 在 MATLAB 编辑器中，选择插入方法 > 验证输入。

MATLAB 编辑器会将以下代码插入到 System object 中：

```
function validateInputsImpl(obj,u)
    % Validate inputs to the step method at initialization
end
```

### 另请参阅

### 相关示例

- “分析 System object 代码”（第 35-57 页）

# 分析 System object 代码

## 本节内容

“查看并导航 System object 代码” (第 35-57 页)

“示例：使用分析器转至 StepImpl 方法” (第 35-57 页)

## 查看并导航 System object 代码

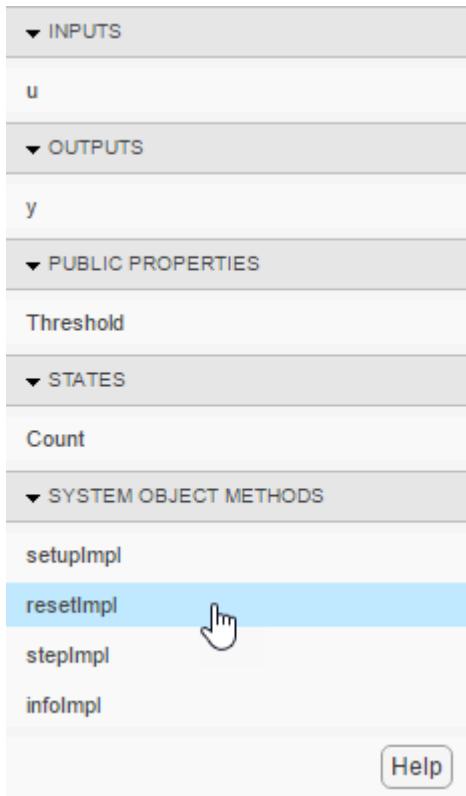
使用分析器查看并导航 System object 代码。

分析器会显示 System object 代码中的所有元素。

- 通过点击元素的名称导航到特定的输入、输出、属性、状态或方法。
- 展开或折叠具有箭头按钮的元素部分。
- 使用 + (public)、# (protected) 和 – (private) 符号标识属性和自定义方法的访问级别。

## 示例：使用分析器转至 StepImpl 方法

- 1 打开一个现有的 System object。
- 2 选择分析。
- 3 点击 resetImpl。



MATLAB 编辑器窗口中的游标将跳转到 resetImpl 方法。

```
10      end
11
12      methods (Access = protected)
13          function setupImpl(obj)
14              obj.Count = 0;
15          end
16
17          function resetImpl(obj)
18              obj.Count = 0;
19          end
20
21          function y = stepImpl(obj,u)
22              if (u > obj.Threshold)
23                  obj.Count = obj.Count + 1;
24              end

```

### 另请参阅

### 相关示例

- “使用 MATLAB 编辑器插入 System object 代码” (第 35-52 页)

# 在 System object 中使用全局变量

全局变量为您可以在其他 MATLAB 函数或 Simulink 模块中访问的变量。

## MATLAB 中的 System object 全局变量

对于仅在 MATLAB 中使用的 System object，在 System object 类定义文件中定义全局变量的方式与在其他 MATLAB 代码中定义全局变量的方式相同（请参阅“全局变量”（第 20-10 页））。

## Simulink 中的 System object 全局变量

对于 Simulink 的 MATLAB System 模块中使用的 System object，您也可以像在 MATLAB 中一样定义全局变量。但是，要在 Simulink 中使用全局变量，您必须在 `stepImpl`、`updateImpl` 或 `outputImpl` 方法中声明全局变量（如果您已分别在 `stepImpl`、`updateImpl` 或 `outputImpl` 调用的方法中对全局变量进行了声明）。

对 MATLAB System 模块设置和使用全局变量的方式与对 MATLAB Function 模块的方式相同（请参阅“数据存储”（Simulink）和“全局共享数据”（Simulink）。与 MATLAB Function 模块一样，您也必须使用与 Data Store Memory 模块匹配的变量名称，才能在 Simulink 中使用全局变量。

例如，以下类定义文件定义了一个 System object，该对象在每个时间步将矩阵的第一行增加 1。如果类文件采用 P 代码编码，则您必须包含 `getGlobalNamesImpl`。

```
classdef GlobalSysObjMatrix < matlab.System
methods (Access = protected)
    function y = stepImpl(obj)
        global B;
        B(1,:) = B(1,:)+1;
        y = B;
    end

    % Include getGlobalNamesImpl only if the class file is P-coded.
    function globalNames = getGlobalNamesImpl(~)
        globalNames = {"B"};
    end
end
end
```

此模型将 `GlobalSysObjMatrix` 对象包含在 MATLAB System 模块和关联的 Data Store Memory 模块中。

