MWORKS.Syslab 函数库

开发规范

苏州同元软控信息技术有限公司

编制	张和华	**生效日期**	2022-10-11
审核		**批准**	

文件变更摘要

日期	版本	变更说明	修订	审核	批准
2022-10-11	v0.1	初始建立			

目录

1. 概述 1

1.1 目的 1
1.2 参考文献 1
1.3 前期准备 1

2. 什么是 Julia 包？ 2

2.1 Julia 包 2
2.2 快速入门 2

3. 函数库开发规范 4

3.1 新建函数库 4
3.2 库的目录结构 7
3.3 库的外部接口 9
3.3.1 导出列表 9
3.3.2 函数定义 9
3.4 设置库的依赖 12
3.4.1 库的项目文件 12
3.4.2 添加库的依赖 14
3.4.3 移除库的依赖 15
3.5 库的单元测试 17
3.5.1 单元测试 17
3.5.2 常用工具 18
3.6 库的测例 22
3.7 函数库编码规范 23
3.8 其它注意事项 23

1. 概述

1.1 目的

本文目的在于指导用户快速开发一个 Julia 函数库。

1.2 参考文献

(1). Julia 帮助文档

- Julia Documentation · The Julia Language
- 主页 · Julia 中文文档 (juliacn.com)
- Julia 教程 | 菜鸟教程 (runoob.com)
- 性能建议 · Julia 中文文档 (julialang.org)
- What scientists must know about hardware to write fast code | BioJulia：写高性能代码时必须了解的一些硬件知识

(2). 包：

- Pkg · Julia 中文文档 (juliacn.com)
- 模块 · Julia 中文文档 (juliacn.com)
- JuliaTesting/ReferenceTests.jl: Utility package for comparing data against reference files (github.com)
- Home · Revise.jl (timholy.github.io)
- simonster/Reexport.jl: Julia macro for re-exporting one module from another (github.com)

(3). 编码规范：

- invenia/BlueStyle: A Julia style guide that lives in a blue world (github.com)

(4). Syslab 使用手册

1.3 前期准备

- 安装 MWORKS.Syslab 2022b 及以上版本，软件下载地址：https://www.tongyuan.cc/download

2. 什么是 Julia 包？

2.1 Julia 包

包（Package）：一个提供可重用功能的项目，其它 Julia 项目可以同 import X 或 using X 使用它。一个包应该包含一个具有 uuid 条目的项目文件。此 UUID 用于在依赖它的项目中标识该包。

我们习惯将 Julia 包又称为 Julia 函数库。

2.2 快速入门

A. 掌握 Julia 常规使用

提供交互式命令行窗口（Read-Eval-Print-Loop，缩写 REPL），用于在窗口输入命令并查看结果。具体请参见 Syslab 使用手册的【Syslab > Syslab 快速入门 > 命令行窗口】。

- REPL 基本命令

操作	命令
上一次运算的结果	ans
中断命令执行	Ctrl+C
清屏	Ctrl+L
运行程序文件	include("filename.jl")
查找 func 相关的帮助	?func
查找 func 的所有定义	apropos("func")
命令行模式	;
包管理模式	]
帮助模式	?
查找特殊符号输入方式	?☆ # "☆" can be typed by \bigwhitestar<tab>
退出特殊模式返回到REPL	在空行上按回格键[Backspace]
退出REPL	exit() 或 Ctrl+D

图 2-1 Julia 命令行窗口的基本命令

- 包管理模式

在 REPL 中输入]，然后按回车，进入包管理模式，常用命令如下：

功能	命令
添加包	add PackageName
删除包	rm PackageName
更新包	update PackageName
使用开发版本	dev PackageName 或 dev GitRepoUrl

图 2-2 包管理模式的常用命令

例如，进入包管理模式，安装 Example 包，命令如下：

```
(@v1.7) pkg> add Example
```

- 使用包：

通过 import 或 using 来导入一个包。

```
# 方法 1：使用 import
import Example
Example.domath(2) # 返回 7

# 方法 2：使用 using
using Example
domath(2) # 返回 7
```

B. 一个简单的 Julia 包

Example.jl 是源上最简单的包，里面包含项目文件（Project.toml）、源码文件（src/Example.jl）、单元测试（test/runtests.jl）、帮助文档（docs）、许可说明（LICENSE.md）、包说明（README.md）、其它配置文件等。

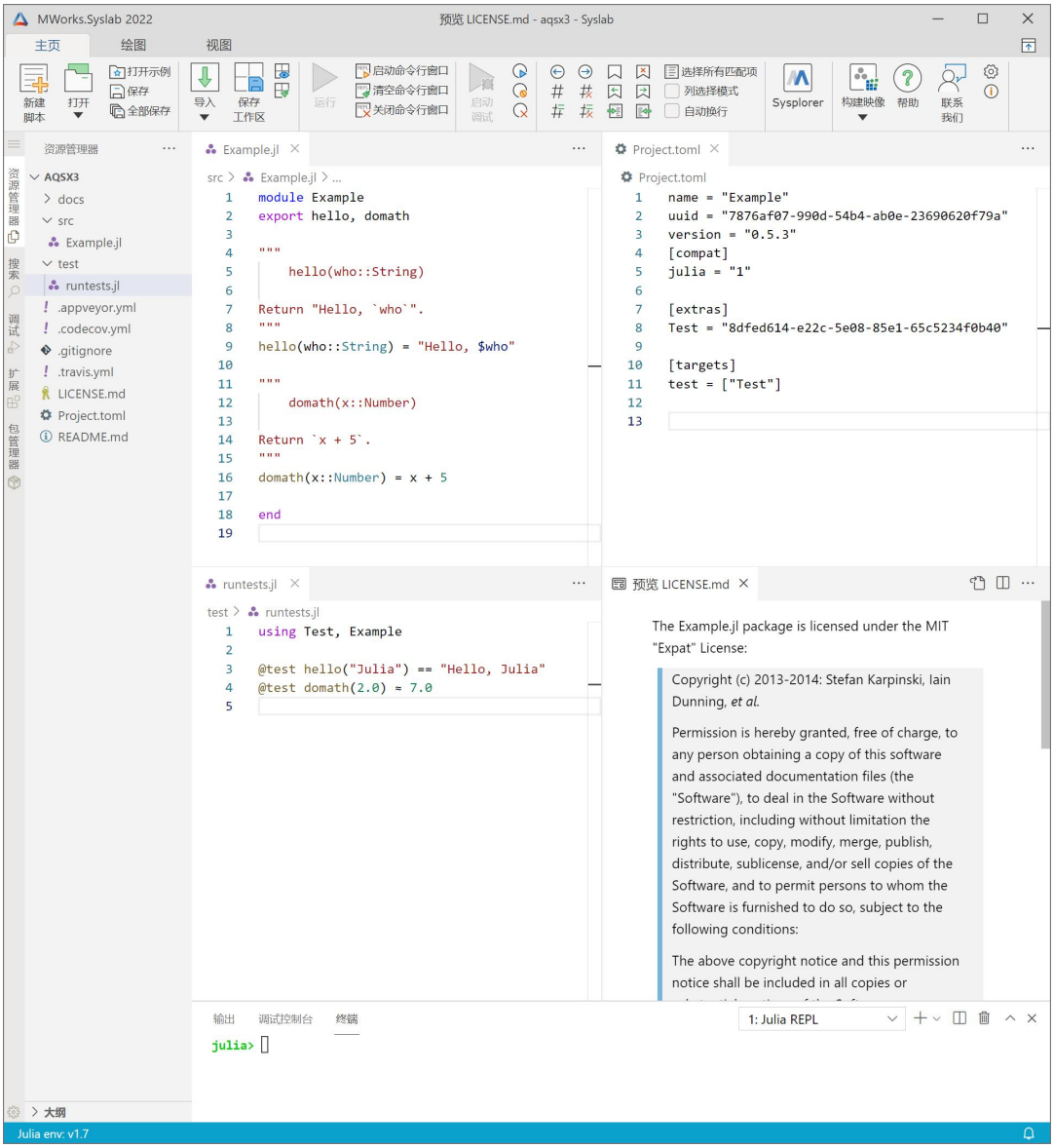

图 2-3 Example 包的内容

C. 进阶学习

如果用户想了解更多深层次的 Julia 编程知识，建议多阅读 Julia 社区提供的包源码，如 Pkg、DataFrames 等。

3. 函数库开发规范

3.1 新建函数库

打开 Syslab，将左侧面板切换到【包管理器】，点击开发库面板上的【新建包】按钮，打开新建包的配置界面，如下图所示。

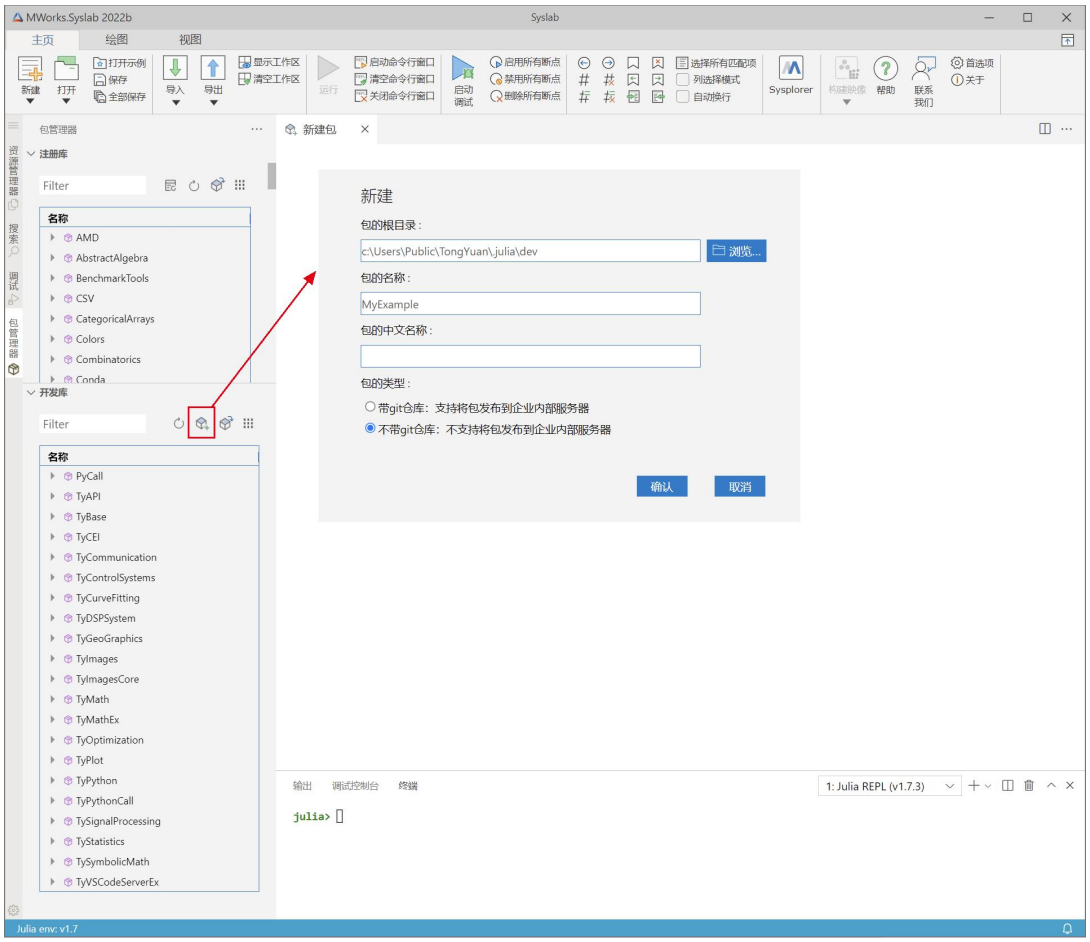

图 3-1 新建包的配置界面

新建包配置完成后，点击【确认】按钮，将自动生成并安装包，此时在左侧开发库面板中可以看到新建的“MyExample”包。选中此包并点击右键菜单【在新窗口中打开】，将打开此包的源码文件夹。

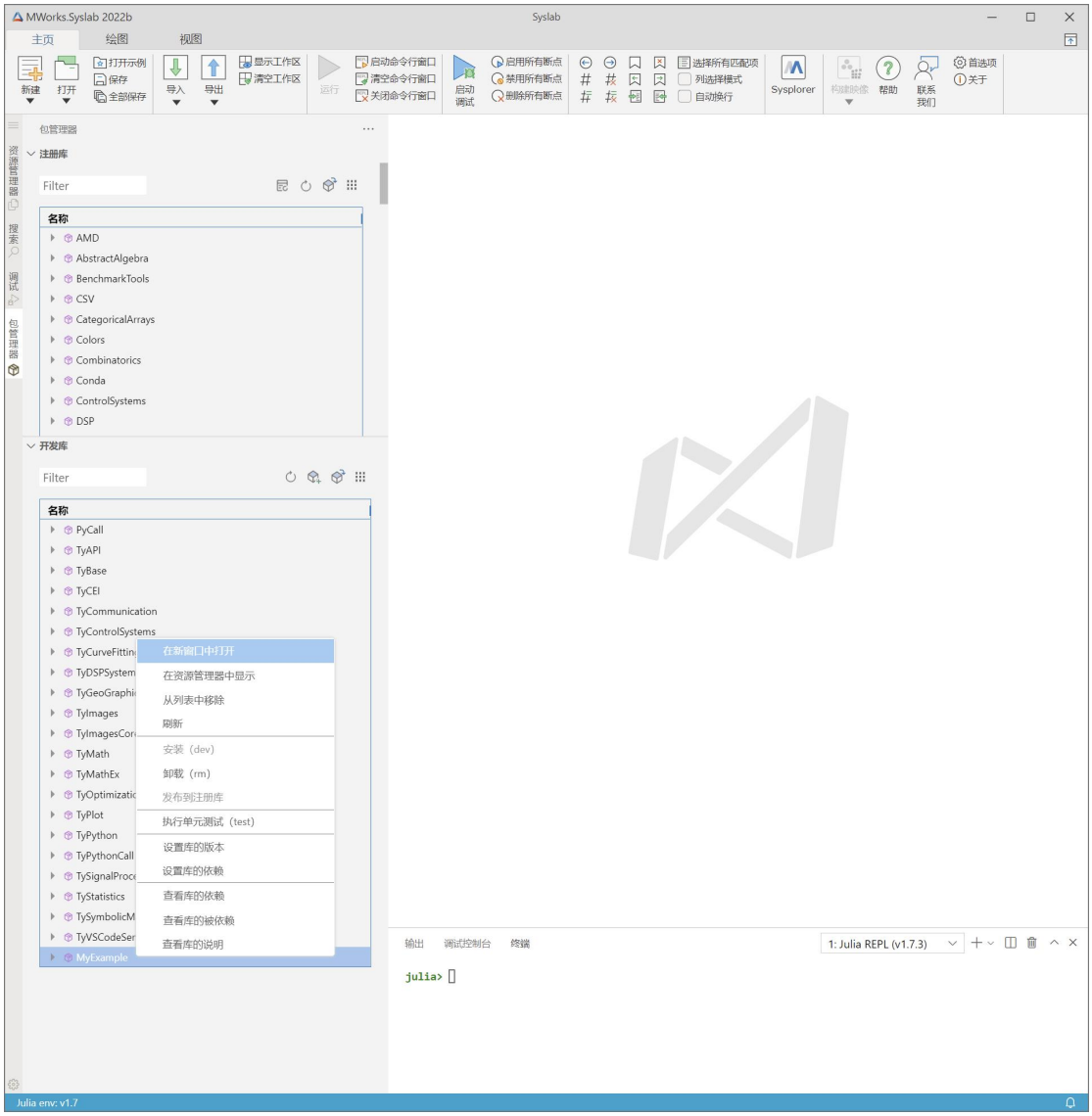
MWorks.Syslab 2022b
Syslab
主页
绘图
视图
新建
打开
打开示例
保存
全部保存
导入
导出
显示工作区
清空工作区
运行
启动命令行窗口
清空命令行窗口
关闭命令行窗口
启动调试
启用所有断点
禁用所有断点
删除所有断点
选择所有匹配项
列选择模式
自动换行
Sysplorer
帮助
联系我们
首选项
关于
包管理器
注册库
Filter
名称
AMD
AbstractAlgebra
BenchmarkTools
CSV
CategoricalArrays
Colors
Combinatorics
Conda
ControlSystems
DSP
开发库
Filter
名称
PyCall
TyAPI
TyBase
TyCEI
TyCommunication
TyControlSystems
在新窗口中打开
在资源管理器中显示
从列表中移除
刷新
安装（dev）
卸载（rm）
发布到注册库
执行单元测试（test）
设置库的版本
设置库的依赖
查看库的依赖
查看库的被依赖
查看库的说明
MyExample
输出
调试控制台
终端
1: Julia REPL (v1.7.3)
julia>
Julia env: v1.7

图 3-2 在新窗口中打开

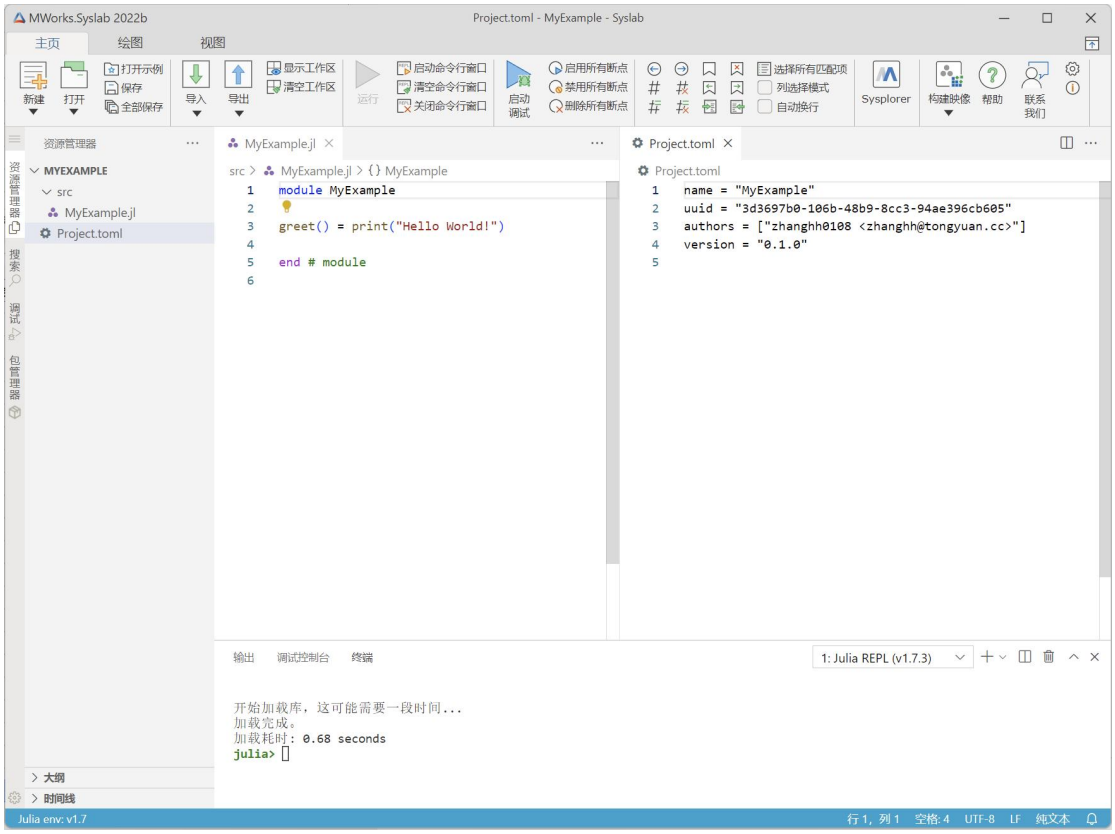

图 3-3 打开 MyExample 包的源码

3.2 库的目录结构

一个标准的包（如 Revise），其目录结构如下图所示：

- docs：帮助文档文件夹
- examples：可选，测例文件夹
- images：可选，资源文件夹
- **src：库源码文件夹**
- **test：单元测试文件夹**
- **Project.toml：项目文件**
- LICENSE.md：许可证文件
- README.md：库说明文件
- …：用户还可以添加其它文件夹

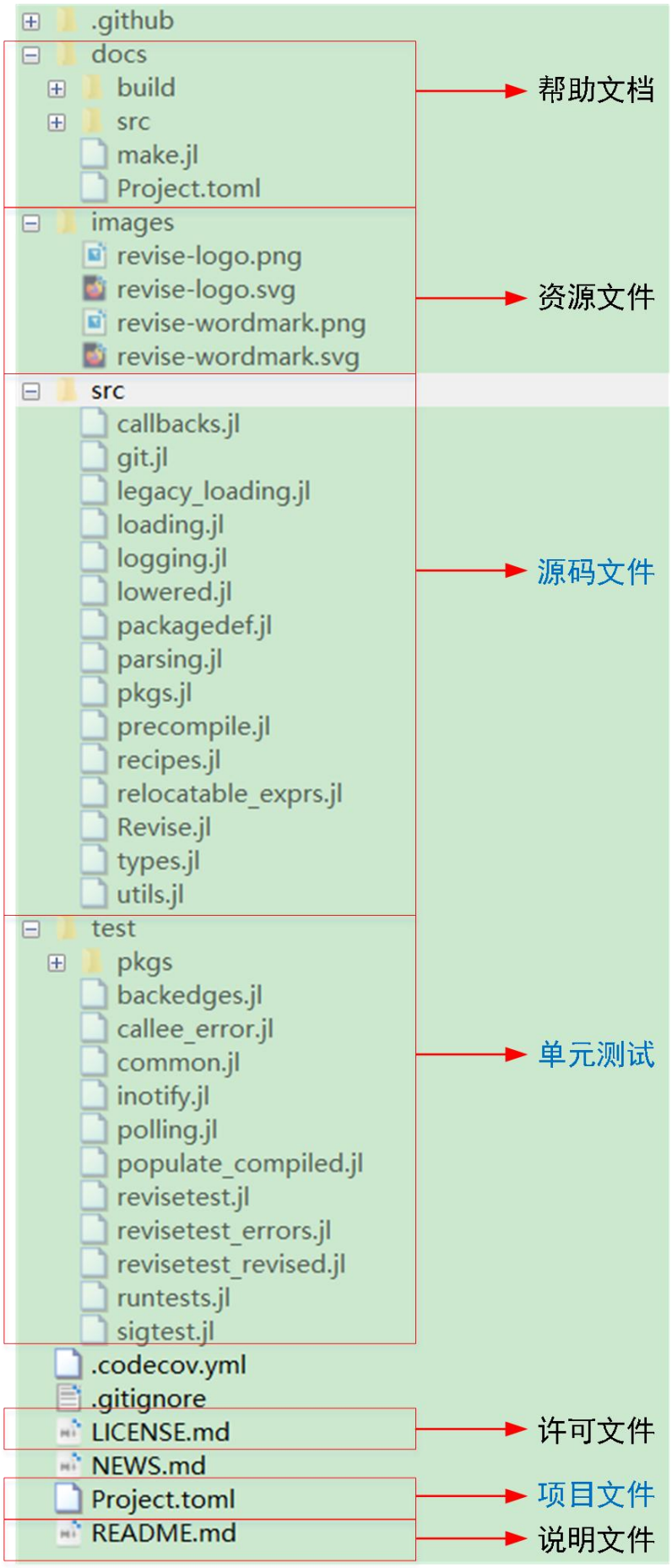
.github
docs
build
src
make.jl
Project.toml
帮助文档
images
revise-logo.png
revise-logo.svg
revise-wordmark.png
revise-wordmark.svg
资源文件
src
callbacks.jl
git.jl
legacy_loading.jl
loading.jl
logging.jl
lowered.jl
packagedef.jl
parsing.jl
pkgs.jl
precompile.jl
recipes.jl
relocatable_exprs.jl
Revise.jl
types.jl
utils.jl
源码文件
test
pkgs
backedges.jl
callee_error.jl
common.jl
inotify.jl
polling.jl
populate_compiled.jl
revisetest.jl
revisetest_errors.jl
revisetest_revised.jl
runtests.jl
sigtest.jl
单元测试
.codecov.yml
.gitignore
LICENSE.md
许可文件
NEWS.md
Project.toml
项目文件
README.md
说明文件

图 3-4 （Revise）函数库的目录结构

3.3 库的外部接口

3.3.1 导出列表

函数、类型、全局变量和常量等，可以通过 export 添加到模块的导出列表。通常，导出列表位于或靠近模块定义的顶部，以便读者可以轻松找到它们。

首先，可以看下典型 Julia 包的做法，如下图所示：

```
if isdefined(Base, :Experimental) && isdefined(Base.Experimental, Symbol("@optlevel"))
    @eval Base.Experimental.@optlevel 1
end

using FileWatching, REPL, Distributed, UUIDs
import LibGit2
using Base: PkgId
using Base.Meta: isexpr
using Core: CodeInfo

export revise, includet, entr, MethodSummary
```

图 3-5 Revise 库的导出列表

例如，MyExample 包的导出列表，定义如下：

```
module MyExample

export greet

greet() = print("Hello World!")

…

end # module
```

3.3.2 函数定义

Julia 包中最常用的就是函数，函数定义由两部分组成：一是函数注释，包括函数原型和函数功能说明；二是函数的算法实现。

首先，可以看下典型 Julia 包的做法，如下图所示：

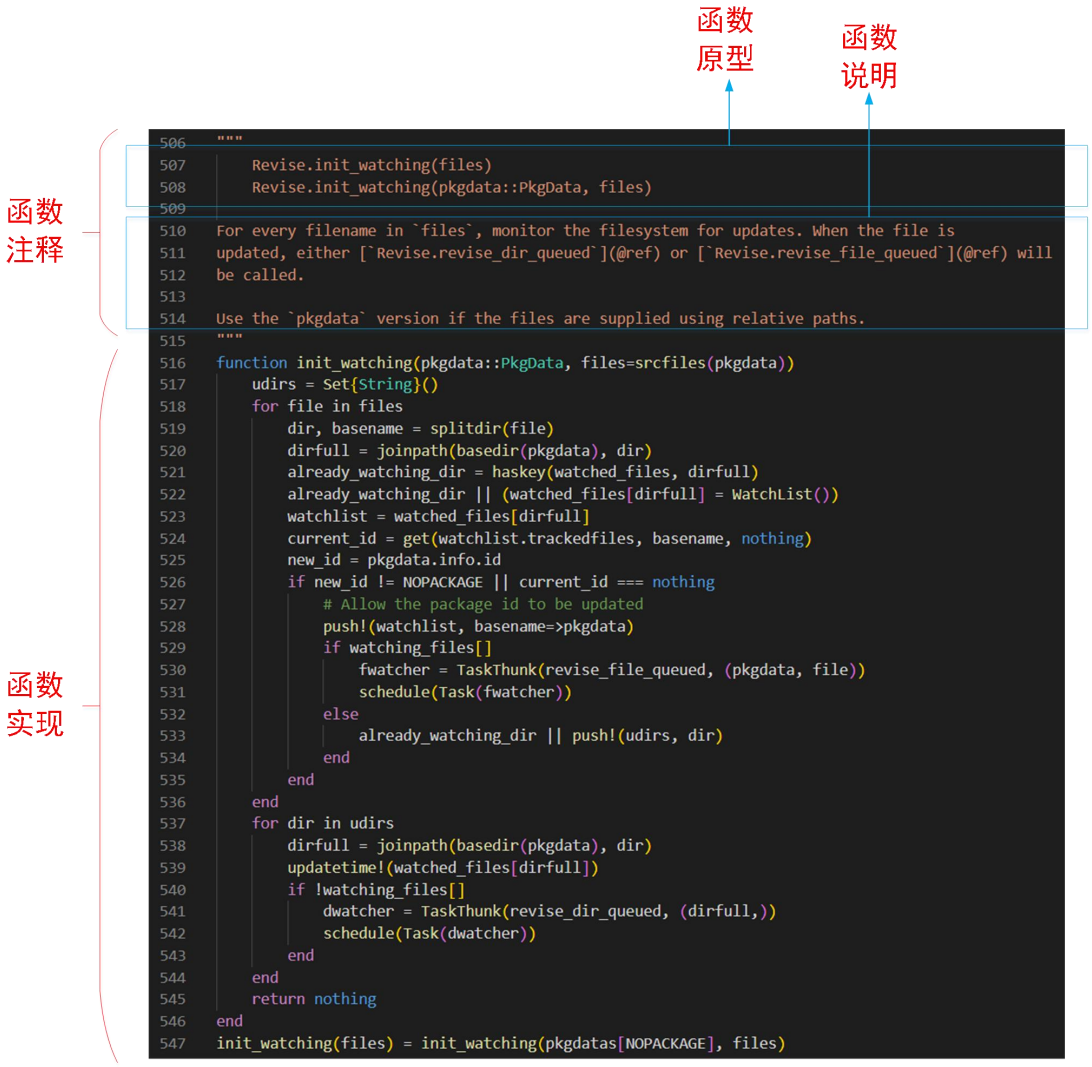

图 3-6 Revise 库的函数定义

例如，为 MyExample 包添加“domath”和“pythagoras”函数，如下所示：

```
module MyExample

export greet, domath, pythagoras

greet() = print("Hello World!")

"""
    domath(x::Number)

Return `x + 5`.
"""
domath(x::Number) = x + 5

include("math.jl")
```

```
end # module
```

其中，MyExample/src/math.jl 中的函数定义如下：

```
"""
    pythagoras(a,b)

勾股定理，英文名为 Pythagoras 也称为毕达哥拉斯定理。

在平面上的一个直角三角形中，两个直角边边长的平方加起来等于斜边长的平方。
如果设直角三角形的两条直角边长度分别是`a`和`b`，斜边长度是`c`，那么数学公式为：


``{\\rm{c = }}\\sqrt {{a^2} + {b^2}}``

返回斜边长`c`
"""
function pythagoras(a, b)
    c = sqrt(a^2 + b^2)
end
```

函数帮助主要有两种形式：一是函数简要说明，从函数定义中自动提取；二是完整详细的帮助手册，需要用户手工编写并集成到 Syslab 帮助手册。其中，查看函数简要说明也有两种方法：一是通过开发库面板提供的右键菜单；二是通过 REPL 中输入“?函数名”来查看。

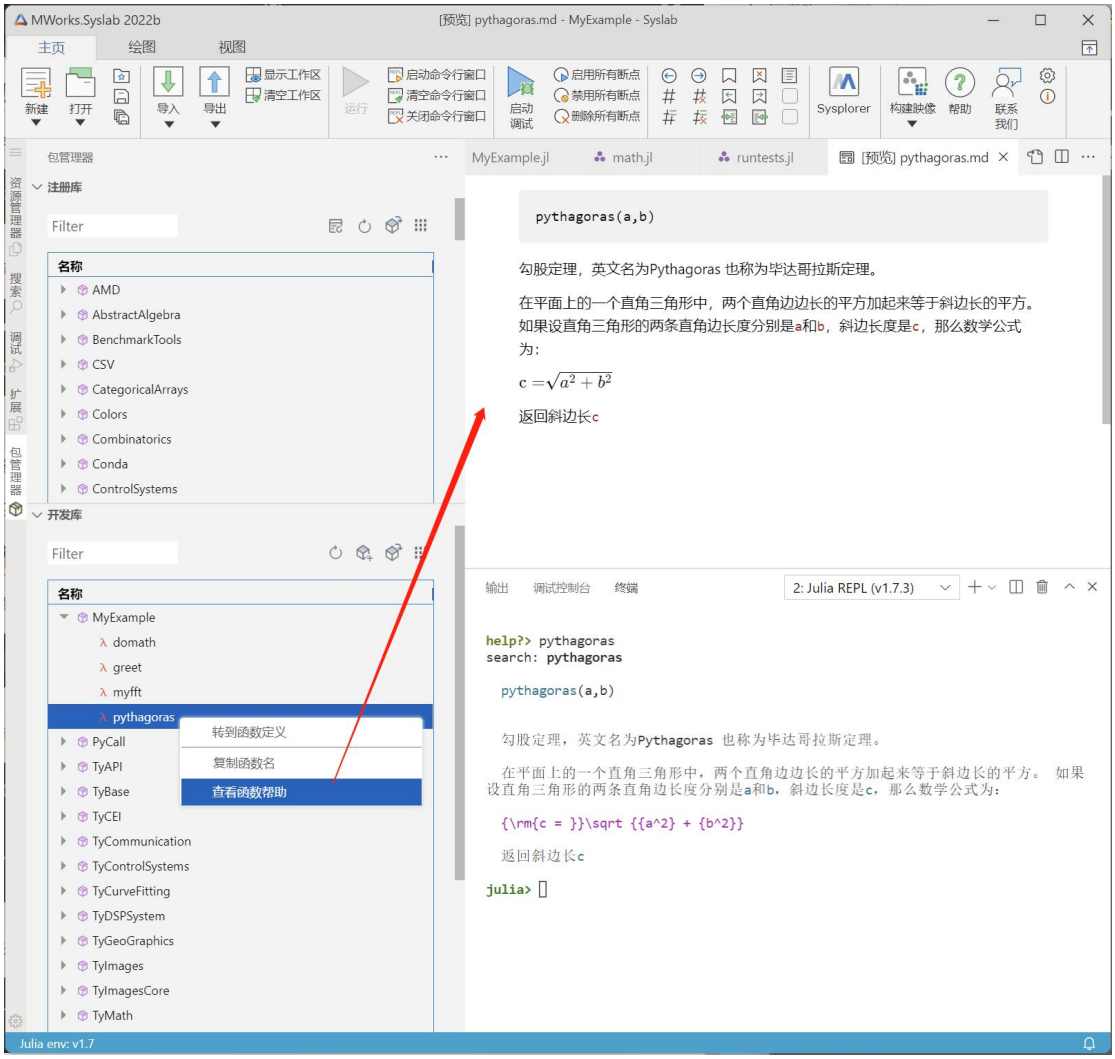

图 3-7 查看函数简要说明

3.4 设置库的依赖

3.4.1 库的项目文件

一个库往往需要依赖其它函数库，因此该库的项目文件 Project.toml 记录了其所有依赖信息，如下图所示：

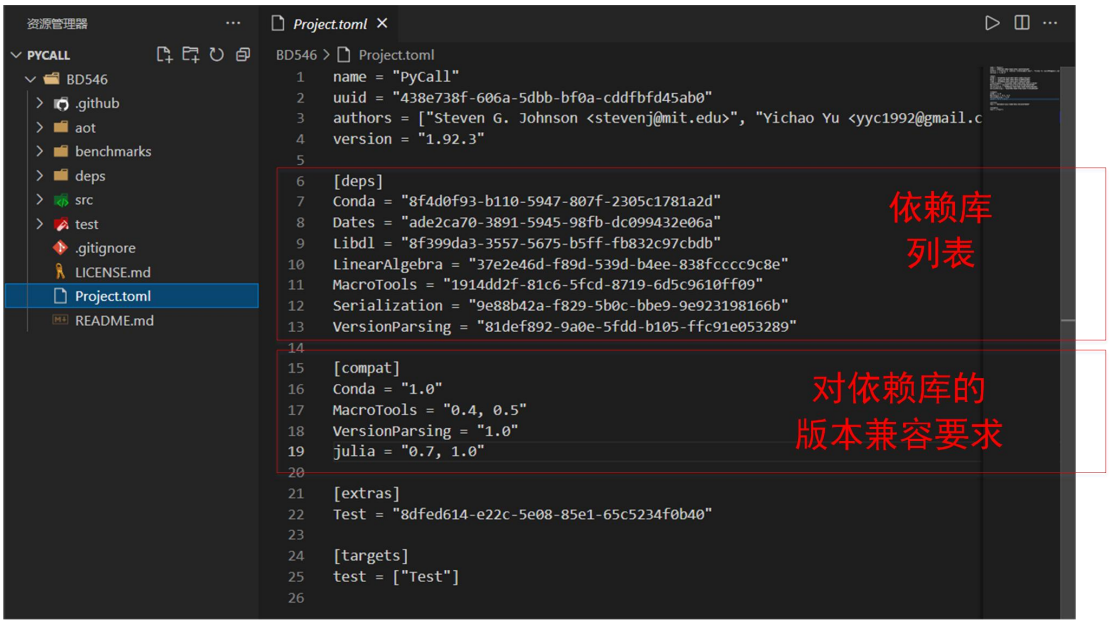

图 3-8 依赖库及其版本兼容要求

包的项目文件 project.toml，内容解释如下：

- name：包的名称；
- uuid：包的唯一标识；
- authors：包的作者，书写规则为“[NAME <EMAIL>, NAME <EMAIL>]”；
- version：包的版本。必须遵守 SemVer 语义化版本，简而言之：
  - 破坏性更新：主版本 major release
  - 新特性：小版本 minor release
  - bug 修复：补丁版本 patch release
- [deps]：该库依赖的其它函数库，书写规则为“name = uuid”；
- [compat]：该库对依赖库的版本兼容要求。关于 compat 字段的规则，请参考 http://pkgdocs.julialang.org/v1/compatibility/。典型的写法是：

```
[compat]
# 指 D 的所有 0.1.* 和 0.2.* 的版本都兼容
D = "0.1, 0.2"
```

- [extras]：单元测试规定的依赖库，与“[targets]”一起使用；
- [targets]：单元测试规定的依赖库。

  注，extras 和 targets 请参考 https://pkgdocs.julialang.org/v1/creating-packages/#Test-specific-dependencies-1。

3.4.2 添加库的依赖

目前提供两种方式来设置库的依赖。

A. 方法 1：

按照上一节介绍，手工修改项目文件 project.toml，参见图 3-8。

B. 方法 2：

Syslab 提供了“包管理器”，通过界面操作即可完成库的依赖设置。

首先，点击 Syslab 左侧边栏的【包管理器】，在开发库中选中“MyExample”，点击其右键菜单的【设置库的依赖】。

注，如果开发库面板显示为空，可以点击如下图的【刷新面板】按钮。

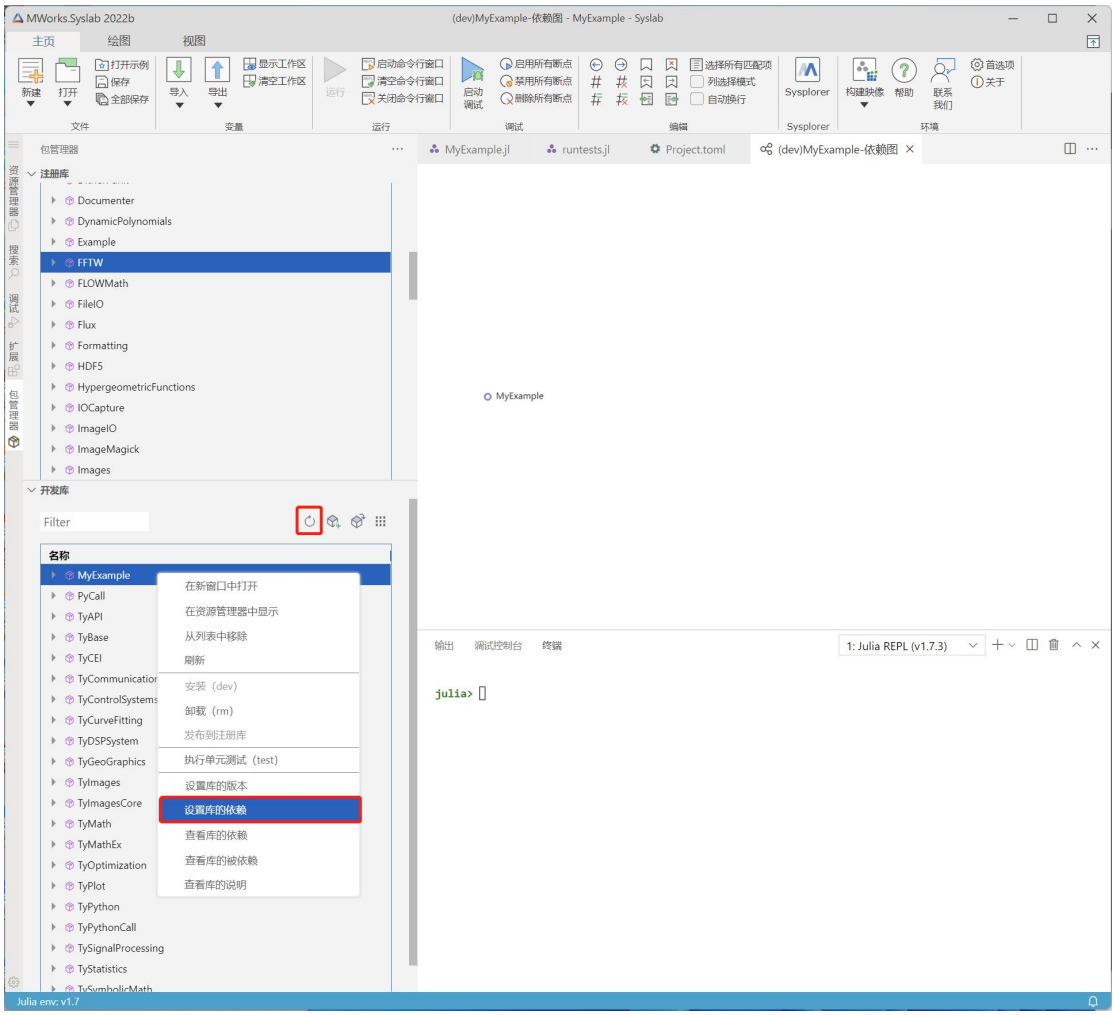

图 3-9 设置库的依赖

接着，将自动弹出选择对话框，此时可以选择需要依赖的包，如 FFTW。

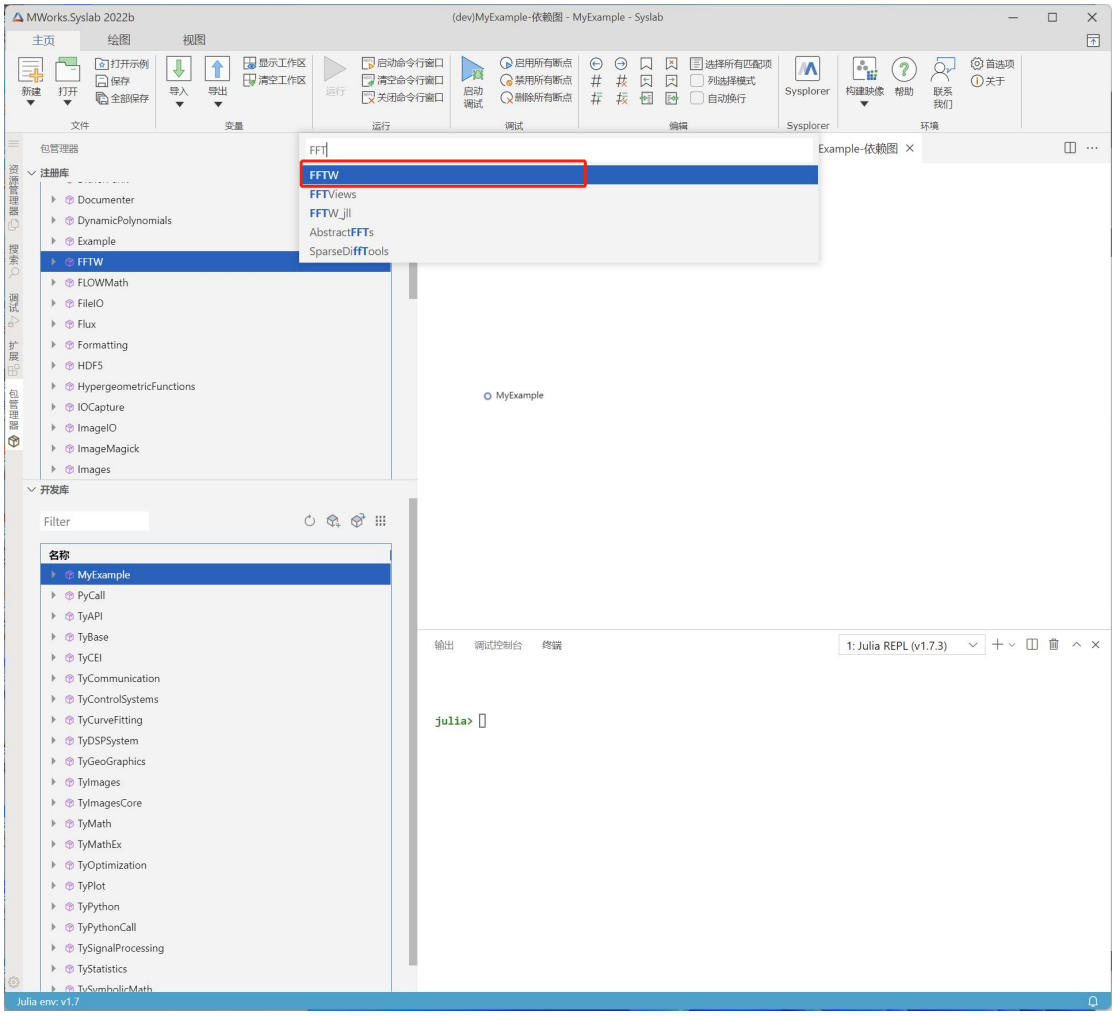

图 3-10 选择依赖的包

上述操作完成后，系统将弹出消息框“添加包的依赖成功!”。

3.4.3 移除库的依赖

目前提供两种方式来移除库的依赖。

A. 方法 1：

按照上一节介绍，手工修改项目文件 project.toml，参见图 3-8。

B. 方法 2：

Syslab 提供了“包管理器”，通过界面操作即可完成库的依赖设置。

首先，点击 Syslab 左侧边栏的【包管理器】，在开发库中选中“MyExample”，点击其右键菜单的【查看库的依赖】。

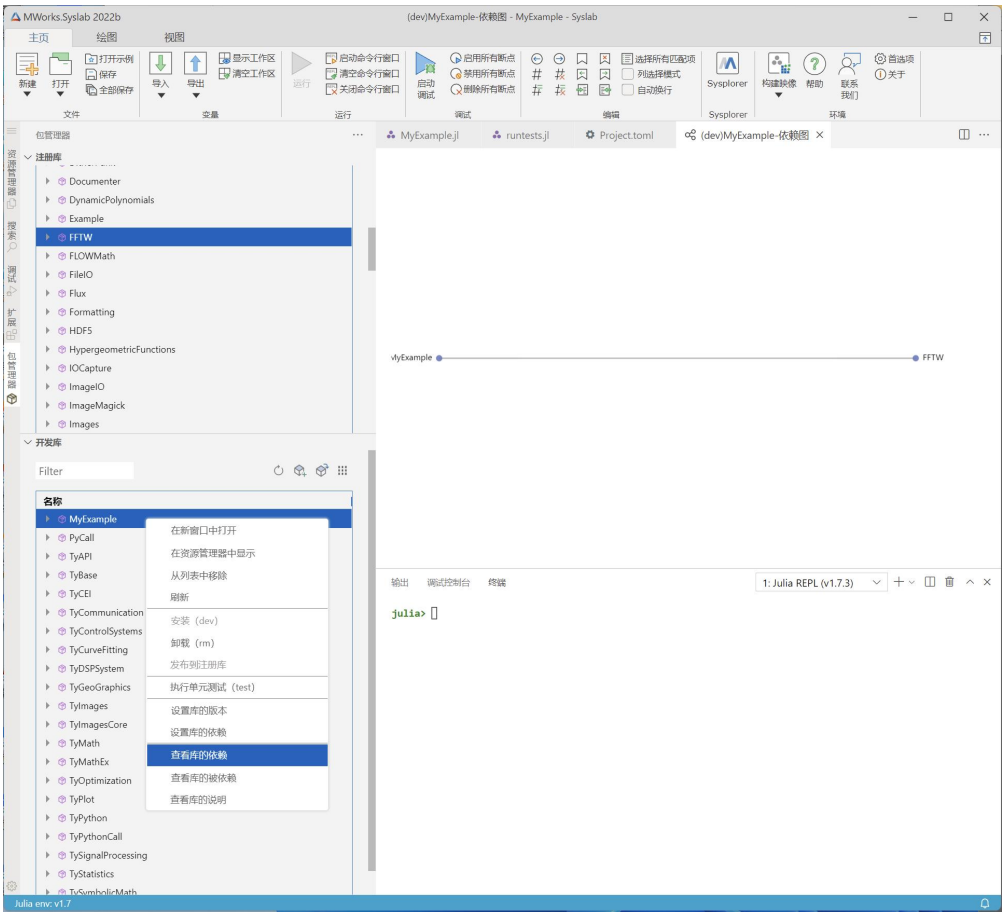

图 3-11 查看库的依赖

接着，在【MyExample-依赖图】视图中选择 FFTW 节点，点击其右键菜单【移除依赖】。

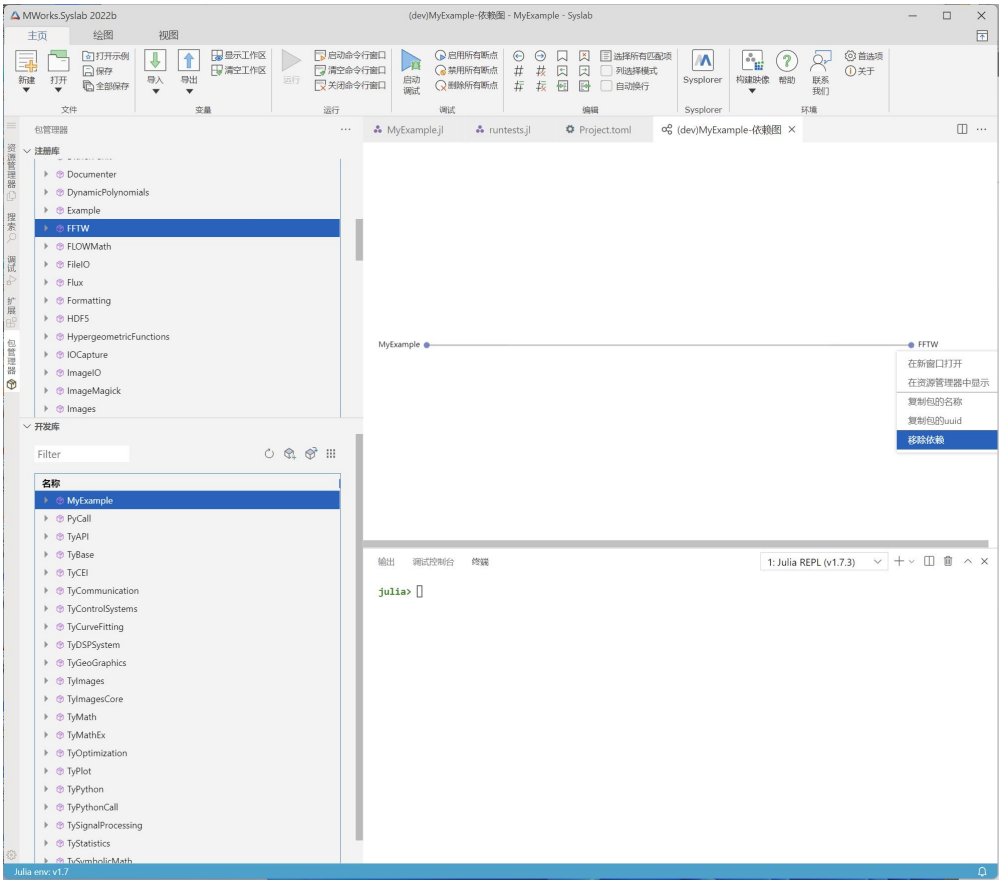

图 3-12 移除依赖

上述操作完成后，系统将弹出消息框“移除包的依赖成功!”。

3.5 库的单元测试

3.5.1 单元测试

对于绝大多数代码开发来说， 测试是保障代码质量和可靠性的最有力的工具。在这里我们介绍测试的基本内容和手段，以单元测试为主。

Julia 的测试代码存放在“<包根目录>/test”文件夹内并通过“pkg> test”调用 “test/runtests.jl”文件。该文件可以理解为 Julia 单元测试的 main 文件。

(1). **@test**：检查表达式的结果是否为 true。如果为 true 则测试通过。

```
using Test
@test 1 + 1 == 2
@test ones(2, 2) == [1 1; 1 1]
```

```
julia> using Test

julia> @test 1 + 1 == 2
Test Passed
  Expression: 1 + 1 == 2
   Evaluated: 2 == 2

julia> @test ones(2, 2) == [1 1; 1 1]
Test Passed
  Expression: ones(2, 2) == [1 1; 1 1]
   Evaluated: [1.0 1.0; 1.0 1.0] == [1 1; 1 1]
```

图 3-13 @test 的运行结果

(2). **@testset**：用于将各个测试组织在一起，例如：

```
@testset "math" begin
    @test 1 + 1 == 2
    @test 1 - 1 == 0
    @test 1 / 0 == 1
end
```

其中，不通过的测试会被标记为 "Fail"， 此时要么是测试代码没写对， 要么是对应的功能存在 bug。

```
math: Test Failed at REPL[9]:4
  Expression: 1 / 0 == 1
   Evaluated: Inf == 1
Stacktrace:
 [1] macro expansion
   @ C:\Program Files\MWorks.Syslab 2022\Tools\julia-1.7.3\share\julia\stdlib\v1.7\Test\src\Test.jl:445 [inlined]
 [2] macro expansion
   @ REPL[9]:4 [inlined]
 [3] macro expansion
   @ C:\Program Files\MWorks.Syslab 2022\Tools\julia-1.7.3\share\julia\stdlib\v1.7\Test\src\Test.jl:1283 [inlined]
 [4] top-level scope
   @ REPL[9]:2
Test Summary: | Pass  Fail  Total
math          |    2     1      3
ERROR: Some tests did not pass: 2 passed, 1 failed, 0 errored, 0 broken.
```

图 3-14 @testset 的运行结果

3.5.2 常用工具

A. Julia 下最常用的测试工具

- Test：为 Julia 标准库；
- ReferenceTests：将结果与参考文件绑定做交叉对比测试，常用于图片的测试；
- Suppressor：用于抑制或捕获 stdout/stderr 的输出结果，经常与 Test 联合使用；
- Random.seed!：用于重置随机数种子，对于某些不定的情况会有用。

B. 常用测试命令：

(1). **@test_throws:** 用于测试函数的报错行为

有些时候我们会希望测试一个函数按照预期的方式报错，这件事则需要借助 @test_throws 进行。例如：

```
julia> rand(1, 1) * rand(3, 1)
ERROR: DimensionMismatch: A has dimensions (1,1) but B has dimensions
(3,1)
```

可以通过

```
msg = "A has dimensions (1,1) but B has dimensions (3,1)"
@test_throws DimensionMismatch(msg) rand(1, 1) * rand(3, 1)
```

```
julia> rand(1, 1) * rand(3, 1)
ERROR: DimensionMismatch("A has dimensions (1,1) but B has dimensions (3,1)")
Stacktrace:
 [1] gemm_wrapper!(C::Matrix{Float64}, tA::Char, tB::Char, A::Matrix{Float64}, B::Matrix{Float64
}, _add::LinearAlgebra.MulAddMul{true, true, Bool, Bool})
   @ LinearAlgebra C:\Program Files\MWorks.Syslab 2022\Tools\julia-1.7.3\share\julia\stdlib\v1.7
\LinearAlgebra\src\matmul.jl:643
 [2] mul!
   @ C:\Program Files\MWorks.Syslab 2022\Tools\julia-1.7.3\share\julia\stdlib\v1.7\LinearAlgebra
\src\matmul.jl:169 [inlined]
 [3] mul!
   @ C:\Program Files\MWorks.Syslab 2022\Tools\julia-1.7.3\share\julia\stdlib\v1.7\LinearAlgebra
\src\matmul.jl:275 [inlined]
 [4] *(A::Matrix{Float64}, B::Matrix{Float64})
   @ LinearAlgebra C:\Program Files\MWorks.Syslab 2022\Tools\julia-1.7.3\share\julia\stdlib\v1.7
\LinearAlgebra\src\matmul.jl:160
 [5] top-level scope
   @ REPL[5]:1

julia> @test_throws DimensionMismatch(msg) rand(1, 1) * rand(3, 1)
Test Passed
  Expression: rand(1, 1) * rand(3, 1)
      Thrown: DimensionMismatch
```

图 3-15 @test_throws 测试结果

(2). **@test_warn**: 测试 warn 日志信息

@test_warn 可以理解为 @test_logs 针对 warn 的特殊版本。

```
function f()
    @warn "don't do this"
    return 40
end
```

```
julia> @test 40 == @test_warn "don't do this" f()
Test Passed
```

(3). **@capture_out**: 捕获 stdout 输出

```
julia> print("hello world")
```

```
hello world
```

这一行为没办法直接用 @test 进行测试： 因为 print 返回的是 nothing。但是可以通过 Suppressor 来进行 IO 的捕捉从而达到测试的目的：

```
julia> using Suppressor
julia> using Test
julia> msg = @capture_out print("hello world")
"hello world"
julia> @test msg == "hello world"
Test Passed
```

(4). **@capture_err**: 捕获 stderr 输出

日志行为属于 stderr 输出，因此也可以通过 Suppressor 进行捕获。

```
julia> using Suppressor
julia> using Test
julia> msg = @capture_err @info "some information"
julia> @test occursin("some information", msg)
```

```
julia> using Suppressor

julia> using Test

julia> msg = @capture_err @info "some information"
"[ Info: some information\n"

julia> @test occursin("some information", msg)
Test Passed
  Expression: occursin("some information", msg)
   Evaluated: occursin("some information", "[ Info: some information\n")
```

图 3-16 @capture_err 捕获 stderr 输出

(5). **@test_broken**: 标记本应该通过的测试

在测试驱动的开发模式 TDD (test driven developement) 下，有一些功能暂时还没开发完成从而导致一些测试无法通过。

```
my_sum(A) = error("待实现")
```

此时通过 @test_broken 宏可以将其标记为已知的损坏测例：

```
julia> @test_broken my_sum([1, 3]) == 4
Test Broken
  Expression: my_sum([1, 3]) == 4
```

这与 @test 最大的差别在于 @test_broken 不会影响单元测试的最终结果。@test_broken 标记的测例在 CI/CD 中依然会显示为绿色（通过） 而非红色（不通过）。

类似的还有 @test_skip，它会直接跳过执行，但留下一个 broken 记录。

(6). **@inferred**: 测试类型不稳定

我们已经知道下面这种类型不稳定的代码在 Julia 下会得到很糟糕的速度。

```
rand_v1() = rand() > 0.5 ? 1 : 0.0 # bad：类型不稳定
rand_v2() = rand() > 0.5 ? 1.0 : 0.0 # good
```

我们可以使用 @inferred 来捕捉类型不稳定：

```
julia> rand_v1() = rand() > 0.5 ? 1 : 0.0 # bad
rand_v1 (generic function with 1 method)

julia> rand_v2() = rand() > 0.5 ? 1.0 : 0.0 # good
rand_v2 (generic function with 1 method)

julia> @inferred rand_v1()
ERROR: return type Float64 does not match inferred return type Union{Float64, Int64}
Stacktrace:
 [1] error(s::String)
   @ Base .\error.jl:33
 [2] top-level scope
   @ REPL[20]:1

julia> @inferred rand_v2()
0.0
```

图 3-17 @inferred 测试类型是否稳定

另， 在日常开发过程中，可以通过 @code_warntype 来检查类型不稳定，

```
julia> @code_warntype rand_v1()
  from rand_v1() in Main at REPL[18]:1
Arguments
  #self#::Core.Const(rand_v1)
Body::Union{Float64, Int64}
1 — %1 = Main.rand()::Float64
│   %2 = (%1 > 0.5)::Bool
└──      goto #3 if not %2
2 —      return 1
3 —      return 0.0


julia> @code_warntype rand_v2()
MethodInstance for rand_v2()
  from rand_v2() in Main at REPL[19]:1
Arguments
  #self#::Core.Const(rand_v2)
Body::Float64
1 — %1 = Main.rand()::Float64
│   %2 = (%1 > 0.5)::Bool
└──      goto #3 if not %2
2 —      return 1.0
3 —      return 0.0
```

图 3-18 检查类型是否稳定

(7). Random.seed!: 固定随机数种子

如果一个函数内部有 rand 等随机行为， 则每次的结果可能会不一样。 这时可以选择两种方案：

- 测试基本属性（而非具体数值）：例如尺寸、值域范围、数值分布等；
- 使用随机数种子进行固定。

```
using Random

Random.seed!(0)
@test sum(rand(4, 4)) == 8.444269386259387
```

3.6 库的测例

单元测试主要是用于自动化回归测试的，并不适合用户学习、使用和演示。为了解决这个问题，可以在包根目录下新增一个 examples 文件夹，用于存放一些使用示例。例如，TiffImages 库就是这么做的，里面的示例都是可以运行体验的。当然，Julia 函数库不带 examples 也是允许的。

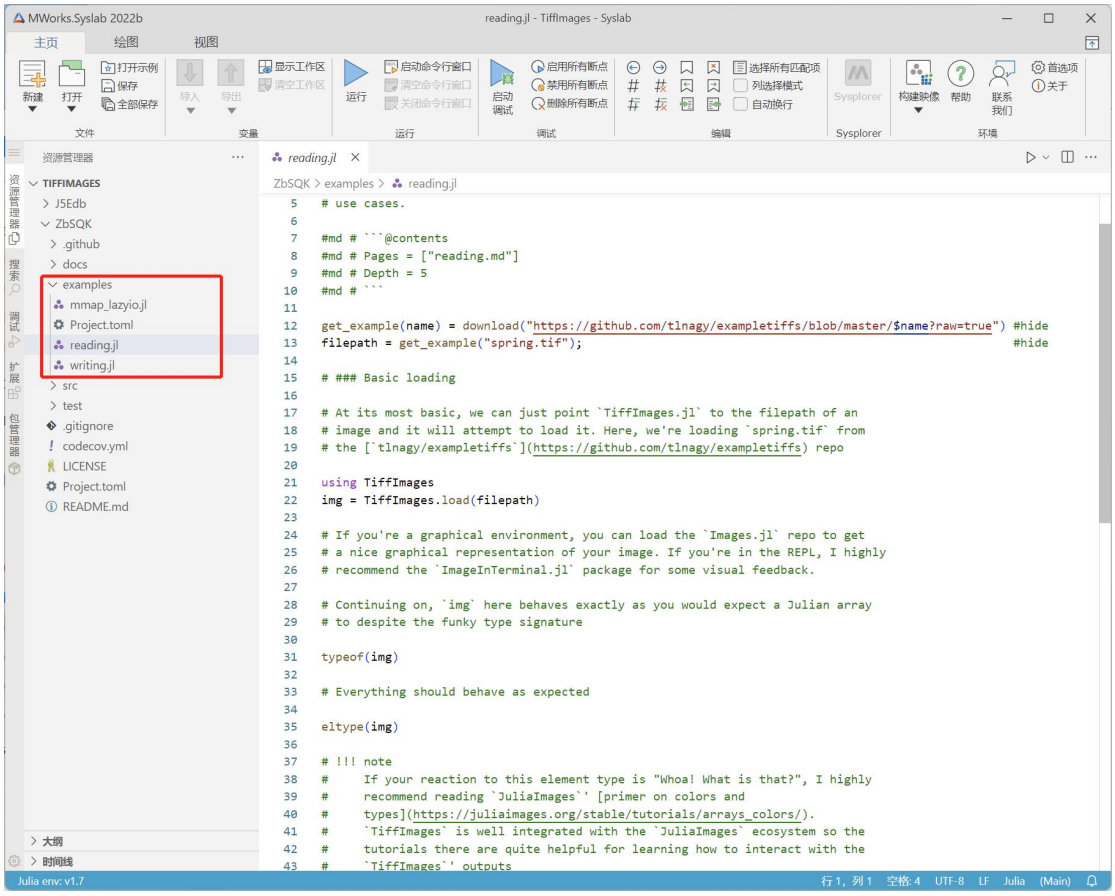

图 3-19 TiffImages 库的示例集

3.7 函数库编码规范

Julia 编码规范，请参考《Julia 编码规范：Blue-aStyleGuideforJulia.pdf》。

3.8 其它注意事项

A. 不推荐在函数库中提交 Manifest.toml 文件

Manifest.toml 文件的目的是为了将整个项目的依赖完全锁定在一个具体版本， 从而可以通过 pkg> instantiate 来得到一个指定的运行版本。对于函数库的开发来说，当你在 Project.toml 中记录了足够完整的 compat 字段后，就不太会再需要提供 Manifest.toml 文件了。

B. 禁止使用不带限制的 @reexport 命令

Reexport.jl 提供了一个非常方便的宏命令 @reexport 用于将其他 Julia 包所导出的名字再一次导出。但是，在使用时需要注意：

- 禁止使用不带限制的 @reexport using SomePkg
- 推荐使用限定符号的 @reexport using SomePkg: sym1