node repl

目录

[概念 1](#_Toc498936952)

[repl模块解读 1](#_Toc498936953)

[默认求值函数 1](#_Toc498936954)

[1.JavaScript表达式 1](#_Toc498936955)

[2.默认求值函数作用域问题 2](#_Toc498936956)

[3.提供Node.js内置模块的访问方式 2](#_Toc498936957)

[所有repl实例支持命令（command） 2](#_Toc498936958)

[扩展：Object.defineProperty(obj, prop, descriptor)使用 3](#_Toc498936959)

概念

REPL（Read-Eval-Print Loop）为运行JavaScript脚本与查看运行结果提供了一种交互方式，可以用于调试、测试。

“读入*-*求值*-*输出”循环 的名字来自于以下几个用来实现这种机制的内置函数：

* 读入函数，接收一个来自于用户的表达式，将其解析成数据结构并存入内存。例如，用户可能会输入一个s-表达式　(+ 1 2 3)，这句话会被解析成一个包含四个元素的链表。
* 求值函数，负责处理内部的数据结构并对其求值。在Lisp中，求一个以函数名开头的ｓ－表达式。意味着对接下来的参数调用那个函数。所以函数"+"被在参数1 2 3上调用，产生结果6。
* 输出函数，接受求值结果，并呈现将其给用户。尽管当前的结果“6”并不具有复杂的格式，但如果是一个较为复杂的表达式，那么它将会被精心处理，以便于更方便地被理解。

个人理解：node很好的支持了repl环境，repl就是在linux命令行中输入node，就进入了一个repl的环境，在这个环境中，我们可以直接践行node的语句，检查输出结果。而不需要单独写一个文件。

repl模块解读

repl模块由repl.REPLServer类导出；

默认求值函数

1.JavaScript表达式

所有的repl.REPLServer实例都支持使用求值函数来计算JavaScript表达式；当repl.REPLServer实例被创建的时候，可以覆盖（overridden）求值函数。

|  |
| --- |
| repl.start({eval: myEval}); |

1. 默认的求值函数支持直接对JavaScript表达式的计算

|  |
| --- |
| > 1 + 1  2  > const m = 2  undefined  > m + 1 |

除非变量在blocks和function中声明，否则不管是隐式声明还是用const, let, var声明，都是声明在全局作用域内

默认的求值程序，默认会将最近一个表达式计算出来的值赋给那个特殊的下划线(\_)变量。显式地将下划线(\_)设置为某个值将会禁用该行为。

|  |
| --- |
| > [ 'a', 'b', 'c' ]  [ 'a', 'b', 'c' ]  > \_.length  3  > \_ += 1  Expression assignment to \_ now disabled.  4  > 1 + 1  2  > \_ //失效，不是输出2了  4 |

2.默认求值函数作用域问题

1. 默认的求值函数提供对任何全局变量的访问。

可以通过将变量分配给context对象（因为context对象与每个REPLServer相关联）将变量暴露给REPL，context对象的属性表现为REPL的本地变量

文件：repl\_test.js

|  |
| --- |
| const repl = require('repl');  const msg = 'message';  repl.start('> ').context.m = msg; |

命令行

|  |
| --- |
| $ node repl\_test.js  > m  'message' |

context的属性并不是只读的，可以对其进行修改；通过Object.defineProperty(),只读形式，可是不会报错，只是值不会被改变（Object.defineProperty()的使用查看扩展）

|  |
| --- |
| const repl = require('repl');  const msg = 'message';  const r = repl.start('> ');  Object.defineProperty(r.context, 'm', {  configurable: false,  enumerable: true,  value: msg  }); |

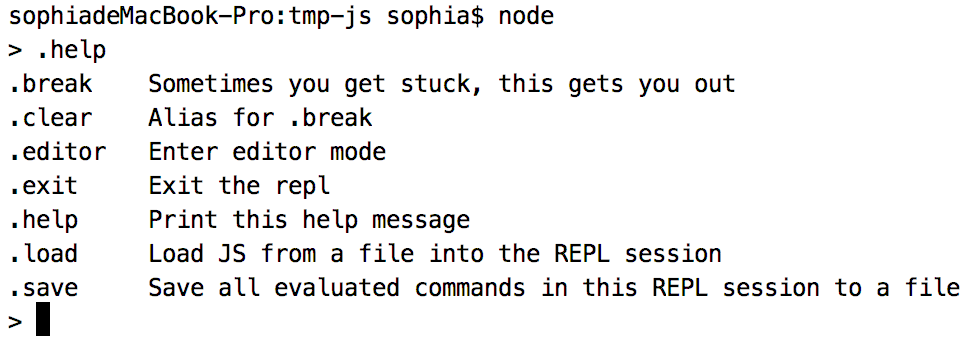
3.提供Node.js内置模块的访问方式

默认的求值函数在使用时会自动将Node.js核心模块加载到REPL环境中，在默认函数中可以直接使用

所有repl实例支持命令（command）

重点：知道用.editor实现多行编辑，利用<ctrl>-C 组合键取消，<ctrl>-D 组合键完成

|  |
| --- |
| > .editor  // Entering editor mode (^D to finish, ^C to cancel)  function welcome(name) {  return `Hello ${name}!`;  }  welcome('Node.js User');  // ^D  'Hello Node.js User!'  > |



|  |
| --- |
| * .help - 显示专用命令列表。 * .break -或者按下<ctrl>-C 组合键，将会中断后续的输入或停止表达式的处理。 * .clear - .break的别名 * .exit - 或者按下<ctrl>-D 组合键关闭输入/输出流，导致REPL退出。 * .save - 将当前的REPL会话保持到文件中: > .save ./file/to/save.js * .load - 将文件加载到当前的REPL会话中: >.load ./file/to/load.js * .editor - 进入编辑模式,使用.exit并不能退出编辑模式,需用<ctrl>-D组合键关闭输入/输出流，表示finish。如果用<ctrl>-C 组合键取消，书写的内容不生效。(^D to finish, ^C to cancel) |

扩展：Object.defineProperty(obj, prop, descriptor)使用

* obj:目标对象
* prop:需要定义的属性或方法的名字。
* descriptor:目标属性所拥有的特性。

可供定义的特性列表：

|  |
| --- |
| value:属性的值（不能同时与set和get使用）  get: 一旦目标属性被访问就会调回此方法，并将此方法的运算结果返回用户。  set:一旦目标属性被赋值，就会调回此方法。  configurable:如果为false，则任何尝试删除目标属性或修改属性以下特性（writable, configurable, enumerable）的行为将被无效化。  writable:如果为false，属性的值就不能被重写。  enumerable:是否能在for…in循环中遍历出来或在Object.keys(obj)中列举出来。 |