 [回雪拓展技巧文档](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#回雪拓展技巧文档)

 [**1 多分支**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#1-多分支)

* [**1.1 套娃**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#11-套娃)
* [**1.2 并列**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#12-并列)
* [**1.3 指向**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#13-指向)
* [**1.4 随取**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#14-随取)

 [**2 可变的变量名**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#2-可变的变量名)

* [**2.1 以QQ、群号作为变量名**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#21-以qq群号作为变量名)
* [**2.2 以日期作为变量名**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#22-以日期作为变量名)
* [**2.3 以时间戳作为变量名**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#23-以时间戳作为变量名)
* [**2.4 指针**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#24-指针)

 [**3 数组/对象**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#3-数组对象)

* [**3.1 数组**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#31-数组)
  + [**3.1.1 数组的total属性**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#311-数组的total属性)
  + [**3.1.2 多维数组**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#312-多维数组)
    - [**矩阵**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#矩阵)
* [**3.2 对象**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#32-对象)
  + [**3.2.1 属性枚举**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#321-属性枚举)
  + [**3.2.2 类**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#322-类)

 [**4 递归**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#4-递归)

* [**4.1递归是什么**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#41递归是什么)
* [**4.2 循环**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#42-循环)
  + [**循环嵌套**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#循环嵌套)

 [**5 排序**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#5-排序)

* [**冒泡排序**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#冒泡排序)
* [**选择排序**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#选择排序)
* [**插入排序**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#插入排序)

 [**6 API**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#6-api)

* [**头文件**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#头文件)

 [**7 链表**](file:///C:\Users\ylzy2\AppData\Local\Temp\Rar$EXa27340.17860\%E5%9B%9E%E9%9B%AA%E6%8B%93%E5%B1%95%E6%8A%80%E5%B7%A7%E6%96%87%E6%A1%A3.html#7-链表)

**回雪拓展技巧文档**

联的冮山

  本文档的内容均为回雪技巧，但是会依靠JS等进行类比对照以助于理解。

  文档内容多数为回雪进阶技巧，是本人对于回雪编写的经验总结。文档并不适合插件初学者阅览，阅览前，你应当首先确认自己对于回雪基本编写格式以及代码足够熟悉，并且能够编写一些简单的回雪插件，例如，好感度插件。

  文档的编写动机主要源于赵赵的跑路。除供各位编写者参考外，文档对本人的意义大约相当于回雪毕业作。

  感谢赵赵提供的舞台，感谢赵赵、蛇与魔骨在我早期学习时提供的帮助，感谢MS-DICE的各位群友的交流反馈，感谢没病装病为文档的版面、校对及导出提供的帮助。

  Ps.实际上编写动机没那么煽情。这篇文档是脑创建的，他留了一个“教程V2.0”标题后就没再动过。作为带新癖重症患者，这是我所不能忍受的事。

**1 多分支**

  多分支是一个很常见的情况，在判断检定的成功等级时，在判定好感度等级时，在多重条件检测时都是多分支。多分支的处理方式并不固定，在此介绍四种。

**1.1 套娃**

  套娃是回雪处理分支的经典结构，分支数量理论可以达到无限。缺陷是如果没有合理的排版，在编写上会有较大困难，debug或需要改动时更是一场灾难。

  以判断检定的成功等级为例：

【比较 1,【变量 结果】,大成功,【比较 【变量 结果】,【比较 【变量 目标值】,50,100,96】,大失败,【比较 【计算 【变量 目标值】/5】,【变量 结果】,极难成功,【比较 【计算 【变量 目标值】/2】,【变量 结果】,困难成功,【比较 【变量 目标值】,【变量 结果】,成功,失败】】】】】

  这就是一段套娃。显而易见，它的可读性很差，但它也是技术难度最低、最容易学会且应用最普遍的办法。把它转换为常见编程语言的话，可以理解为if(){}elseif(){}else{}的嵌套。

  套娃的实际自由度很高，它的结构实际上更接近于二叉树，而非普通的if(){}elseif(){}else{}嵌套。

**1.2 并列**

  并列结构一般用于执行条件的判断，或者执行结果的输出。并列结构主要运用到【返回】。编写难度不大，但需要理清条件判断的排序。

  以判断检定的成功等级为例：

【比较 1,【变量 结果】,【返回 大成功】,【块】】

【比较 【变量 结果】,【比较 【变量 目标值】,50,100,96】,【返回 大失败】,【块】】

【比较 【计算 【变量 目标值】/5】,【变量 结果】,【返回 极难成功】,【块】】

【比较 【计算 【变量 目标值】/2】,【变量 结果】,【返回 困难成功】,【块】】

【比较 【变量 目标值】,【变量 结果】,成功,失败】

  【返回】的作用是阻止后续代码的运行，只运行本分支。这有些类似switch，【返回】即为break，并列结构之后的代码则相当于default中的代码。

**1.3 指向**

  指向运用了【常量】或者【读项目】，每个常量或项目装载着一个分支，而【赋值变量】【执行变量】就是分支点——选择哪个常量或项目，一般由一个变量或者【正则匹配到的】决定。例如：

[[define]]

name="攻击脚本"

[[define]]

name="防御脚本"

[[define]]

name="逃跑脚本"

[[auto]]

keywordRegexp="^战斗：(攻击|防御|逃跑)$"

program="【赋值变量 fight,【常量 【正则匹配到的】脚本】】【执行变量 fight】"

**1.4 随取**

  是的，【随取】。【随取】是随机选择分支的方法，不过【随取】不能直接使用，而必须配合【常量】与【赋值变量】【执行变量】使用。

【赋值变量 下一步,【随取 【常量 攻击】,【常量 防御】,【常量 逃跑】】】

【执行变量 下一步】

**2 可变的变量名**

  这得益于回雪高度自由的文本拼接。如果想要在js里进行所谓变量名的拼接，大概对象的属性名才能做到这点。

  可变的变量名是指，变量（常量/项目）名的一部分可以是其他回雪代码。

[[define]]

name="石头对剪刀"

content="我出剪刀！\n呜……输了……"

[[define]]

name="布对布"

content="……"

……

[[auto]]

keywordRegexp="^石头剪刀布，我出(石头|剪刀|布)^"

program="【常量 【正则匹配到的】对【随取 石头,剪刀,布】】"

  这样做的好处之一是可以精简代码，不过它还有其他用处。

**2.1 以QQ、群号作为变量名**

  虽然回雪有【设置变量针对个人】，但是它只能设置变量针对指令触发者，针对他人需要用到JS的Lib.setTag()。不过，有一个代替方法，就是直接将所针对的QQ号置入变量名中。

//使用【设置变量针对个人】

[[auto]]

keywordFull="增加好感度"

program="【设置固态变量 好感度】【设置变量针对个人 好感度】【赋值变量 好感度,【计算 【变量 好感度】+5】】"

//使用QQ号作为变量名

[[auto]]

keywordFull="增加好感度"

program="【设置固态变量 好感度\_【发送者QQ】】【赋值变量 好感度,【计算 【变量 好感度\_【发送者QQ】】+5】】"

  【设置变量针对群】同理。

  使用QQ/群号作为变量名，除了可以设置变量针对其他人/群外，还有一个优势：这样做其实能够杜绝串数据的风险。

**2.2 以日期作为变量名**

  以日期作为变量名，可以完成数据的每日刷新。以日期作为变量名的方法可以应用于各种需要每日刷新且仅刷新一次的变量，如每日运势、签到等功能。

  需要注意的是，一定要记得给这类变量设置有效期，避免长时间使用后出现大量过期未清理的数据堆积的现象。

**2.3 以时间戳作为变量名**

  以时间戳为变量名的变量通常用来记录“事件”。“事件”是个笼统的名词，因为你可能在监视器、好感度变化记录、操作日志等各处用到以时间戳为变量名的变量。

  【10位时间戳】是计算从1970年1月1日00:00至今经过的秒数，可以记录所有间隔至少一秒的事件。如果你实在担心出现两个事件间隔较短造成的数据覆盖，可以用JS的new Date().getTime()。你会得到精确到毫秒的13位时间戳。

  不过，以时间戳为变量名的变量在使用时有一个难点：如何查看它们？

  我如何知道哪个时间戳记录了事件，哪个没有？或者，我如何知道哪些时间戳记录了事件？

  以QQ/群号为名的变量可以用【发送者QQ】或【当前群号】，亦或者由触发者给出号码；而以日期为名的变量每天刷新一次，一般没有查看昨天、前天的变量的需求（即使有，昨天、前天的日期也很容易得到）。

  看来，我需要将记录了事件的时间戳记录下来，以便我进行查看。

**2.4 指针**

  你可能不知道什么是指针，甚至去搜了一遍也没能搞明白。不过没关系，我们打个比方。

  一个程序会占用一块内存。我们将这块内存比作一个街道，变量住在这些街道里。变量的地址，就是它们的门牌号，而指针变量所储存的就是这些门牌号。

——这个比喻诞生于弥与上学的插件学习过程中

  总而言之，指针的作用是更加有效、便捷地组织一组不规律的变量。这个过程并不一定会用到指针变量，尤其是在没有指针概念的编程语言中。指针最重要的地方在于它所代表的思想。打个比方，你或许曾在一些游戏中看到过物品ID、事件ID的概念，这其实就是指针思想的体现：人为地以ID作为物品、事件的地址，使数据能够被更好地组织。

  回雪同样没有指针及内存地址的概念，但是，正如前文所说，我们可以人为地赋以代号，作为变量的地址。在3.3中，变量的地址就是时间戳，我们用一组更容易组织的变量将这些地址保存起来。那么，什么样的变量更容易组织？

**3 数组/对象**

  回雪没有给出“数组”或“对象”这两种数据形式，但它们确实是两种常用且实用的数据形式，因此，我们需要自己“创造”它们。

  js中的数组/对象的引用格式是“数组/对象名[属性]”，但回雪可以用一个更为简便的方式：用下划线（如果你愿意，也可以用其他字符）作为操作符，即“数组/对象名\_属性”。

本节内容以第2节“可变的变量名”为基本原理。

**3.1 数组**

  以“数组名\_序号”的形式命名的一系列变量可以近似组成一个数组。

JS:

number[0]:18335799466

number[1]:14396228577

number[2]:19365839604

回雪：

号码\_1:"18335799466"

号码\_2:"14396228577"

号码\_3:"19365839604"

  我们可以使用第2节的原理，将序号用回雪代码指定。

[[auto]]

keywordRegexp="^查看第(\\d+)个号码$"

program="【变量 号码\_【正则匹配到的】】"

**3.1.1 数组的total属性**

  数组除了一系列数据外，还包括了一个特殊但关键的属性：数组的长度，即数据的数量。

JS:

name:["张三","李四","王五","赵六"]

name.length=4

回雪：

姓名\_1:"张三"

姓名\_2:"李四"

姓名\_3:"王五"

姓名\_4:"赵六"

姓名\_Total:"4"

  数组长度是一个不可或缺的变量。绝大多数遍历操作都需要用到这个变量。

**3.1.2 多维数组**

  多维数组的原理并不是构建什么多维空间，而是将数组作为数组元素，形成的数组嵌套。

  相比之下，回雪数组的所谓多维更加自由，不过是多几个下划线罢了。

JS:

arr:[[1,2,3],[4,5,6],[7,8,9]]

arr[1][2]:6

回雪：

数组\_1\_1:1

数组\_2\_2:5

数组\_3\_1:7

数组\_3\_2\_2:13

**矩阵**

  矩阵是多维数组的一种，它的特点是所有同级数组的长度相同。即对任意m，“数组\_m\_Total”完全相等；对任意m,n,“数组\_m\_n\_Total”完全相等……

设一个二维数组arr,它是一个3\*3的矩阵。则有：

arr\_1\_1

arr\_1\_2

arr\_1\_3

……

arr\_3\_3

以及四个Total变量

arr\_Total

arr\_1\_Total

arr\_2\_Total

arr\_3\_Total

但是，后三个Total完全相等，所以实际只需要两个Total变量

arr\_Total

arr\_\_Total

  如果一个多维数组是矩阵的话，那么我们只需要数量等于维度数的变量即可储存所有的Total，这可以精简变量数，代价是数组长度变化的自由度降低了。

**3.2 对象**

  回雪对象同样难度不大：

JS:

person:{"name":"老八","age":38,"height":180}

person.name:"老八"

回雪：

人物\_姓名:"老八"

人物\_年龄:"38"

人物\_身高:"180"

**3.2.1 属性枚举**

  属性枚举是指，输出一个对象中的所有属性。如在上一个例子中的“name”、“age”和“height”。遗憾的是，回雪并不能直接枚举一个对象的所有属性，它们必须被另行储存：用一个数组存储该对象的所有属性名。

JS:

Object.keys(person):["name","age","height"]

回雪：

人物~1:"姓名"

人物~2:"年龄"

人物~3:"身高"

**3.2.2 类**

  类，可以看做是一个对象模板。例如，“车”是一个普遍的概念，是类，但车与车不一样，是对象。

JS:

class Person{

   constructor(n,a,h){

       this.name=n;

       this.age=a;

       this.height=h;

  }

}

var customer=new Person(name,age,height)

回雪：

[[define]]

name="生成人物对象"

content="【赋值变量 【变量 类名】\_姓名,【变量 name】】

【赋值变量 【变量 类名】\_年龄,【变量 age】】

【赋值变量 【变量 类名】\_身高,【变量 height】】"

[[auto]]

keywordRegexp="^一个身高为(\\d+),(\\d+)岁的名叫(.+)的人。$"

program="【赋值变量 类名,顾客】

【赋值变量 name,【正则匹配到的 2】】

【赋值变量 age,【正则匹配到的 1】】

【赋值变量 height,【正则匹配到的 0】】

【赋值变量 class,【常量 生成人物对象】】

【执行变量 class】"

  类是快速构造同类对象的途径，学会回雪类的运用更有利于插件编写。

  不过，都学到类了，为什么不去写JS和Lua呢？

**4 递归**

  我们在第3节学会了回雪数组。但是对于回雪数组的很多操作，例如查找数组中的元素，都必须对数组进行遍历。如果是JS，我们可以直截了当地使用for、while等循环结构。但是回雪并没有【循环】这种代码，我们似乎需要自行寻找解决方法。

**4.1递归是什么**

  递归的核心是：自我调用。这是一个在回雪中常用的算法思想。

设a[1]=1,a[2]=1,a[n]+a[n+1]=a[n+2]，请写出一个算法，可以在给出任意大于2的正整数x时求出a[x].

function plus(n){

if(n<3)return 1;

return plus(n-1)+plus(n-2);

}

  递归在程序中比较消耗内存。赵骰也对递归做出一定限制，在递归层数过多时会强制停止并报错。

无法执行！你这行写错了：

line 0: 【执行变量  goto】

  但是，即便如此，递归仍然是回雪常用的算法之一。

**4.2 循环**

  常用的原因是，递归是目前唯一已知的使用回雪实现循环的方法。

  将要重复执行的代码装入一个常量里，附加一个【判断】【比较】或者【判空】，作为递归结束的开关，就是回雪循环的基本原理。

[[define]]

name="数数"

content="【赋值变量 number,【计算 【变量 number】+1】】【比较 【变量 number】,11,【变量 result】,【赋值变量 result,【变量 result】【换行】【变量 number】】【执行变量 goto】】"

[[auto]]

keywordFull="数到十"

program="【赋值变量 goto,【常量 数数】】【执行变量 goto】"

  随着编写的插件逐渐变得复杂，循环的运用也可能越来越多，甚至会出现循环嵌套、循环与数组的结合运用等。你应当学会变通运用。

  这时，赵骰对于递归的限制就有了另一种意义：可以中断死循环。但是，这种限制也导致赵骰插件不接受过多次数的循环。

**循环嵌套**

  所谓循环嵌套其实就是在循环中执行另一个循环，多层循环。

在一个二维地图中寻找发电厂的坐标。

JS:

var x;

var y;

var find=false;

for(x=0;x<xMax;x++){

   for(y=0;y<yMax;y++){

       if(map[x][y]==target){

           find=true;

           break;

      }

  }

   if(find)break;

}

if(x==xMax){"查找无结果。"}

else{target+"的坐标为("+x+","+y")"}

回雪：

[[define]]

name="查找y"

content="【判断 map\_【变量 x】\_【变量 y】,【正则匹配的 0】,【比较 y,yMax,【块】,【赋值变量 y,【计算 【变量 y】+1】】【执行变量 fy】】,true】"

[[define]]

name="查找x"

content="【判断 【执行变量 fy】,true,【比较 x,xMax,【返回 查找无结果】,【赋值变量 x,【计算 【变量 x】+1】】【赋值变量 y,0】【执行变量 fx】】,【块】】"

[[auto]]

keywordRegexp="^查找(.+)$"

program="【赋值变量 x,0】【赋值变量 y,0】【赋值变量 fx,【常量 查找x】】【赋值变量 fy,【查找 y】】【执行变量 fx】【正则匹配到的 0】的坐标为(【变量 x】,【变量 y】)"

  比较需要注意的地方在于内层循环的复位，即数据在新一次循环前需要重新设置。

**5 排序**

  排序单独列出来一节，是因为这确实是一个值得探讨的话题。如果你打开浏览器进行搜索，便能找到各种各样的排序算法。

  排行榜一类的功能会需要排序，根据实际情况的不同，你也应当有所变通地选用不同算法。

**冒泡排序**

  冒泡排序算法会不断地从一个方向或者往返遍历数据组，对遍历到的两个相邻数据进行比较，判断是否需要交换。然后，不论是否交换，向后遍历一位，继续判断，直到一轮遍历没有进行交换为止。

单向遍历：

[52]314→

2[53]14→

23[51]4→

231[54]→

[23]145→

2[31]45→

21[34]5→

213[45]→

[21]345→

1[23]45→

12[34]5→

123[45]→

[12]345→

1[23]45→

12[34]5→

123[45]

往返遍历：

[52]314→

2[53]14→

23[51]4→

231[54]→

23[14]5→

2[31]45→

[21]345→

1[23]45→

12[34]5→

123[45]

  我不打算给出代码。排序算法的编写是对数组和循环的运用进行练习的大好机会。

**选择排序**

  选择排序是对未排序的数组反复遍历，将每次遍历得到的最大值或最小值插入有序数组。

<[3]>1524→

[3]<1>524→

31<[5]>24→

31[5]<2>4→

31[5]2<4>→

<[3]>124|5→

[3]<1>24|5→

[3]1<2>4|5→

312<[4]>|5→

<[3]>12|45→

[3]<1>2|45→

[3]1<2>|45→

<[1]>2|345→

1<[2]>|345→

1|2345

**插入排序**

  插入排序是一个适用于对一组最大值已知的非负整数进行排序的算法。

已知10个最大值为9的非负整数，新建一个长度为[9+1]的数组。

1038539240 [0,0,0,0,0,0,0,0,0,0]

038539240 [0,1,0,0,0,0,0,0,0,0]

38539240   [1,1,0,0,0,0,0,0,0,0]

8539240   [1,1,0,1,0,0,0,0,0,0]

539240     [1,1,0,1,0,0,0,0,1,0]

39240     [1,1,0,1,0,1,0,0,1,0]

9240       [1,1,0,2,0,1,0,0,1,0]

240       [1,1,0,2,0,1,0,0,1,1]

40         [1,1,1,2,0,1,0,0,1,1]

0         [1,1,1,2,1,1,0,0,1,1]

[2,1,1,2,1,1,0,0,1,1]

[1,1,2,1,1,0,0,1,1]  00

[1,2,1,1,0,0,1,1]    001

[2,1,1,0,0,1,1]      0012

[1,1,0,0,1,1]        001233

[1,0,0,1,1]          0012334

[0,0,1,1]            00123345

[0,1,1]              00123345

[1,1]                00123345

[1]                  001233458

0012334589

**6 API**

  我并不保证本节内容所讲的“API”能够严谨地符合学术概念。

  如果一个功能，不需要知晓内部的详细原理即可使用，那么便可以将它封装起来，只留出必要的配置与调用口，这就形成了API（应用程序编程接口）。比如，赵骰JS的Lib库就是一系列由赵赵编写的API。你并不需要知晓Lib.sendPrivateMessage()是怎么工作的，但你仍然可以便捷地使用它。

现在，我们来编写一个检定功能。

[[define]]

name="规则书检定"

content="【赋值变量 结果,【随机数 1-100】】【比较 1,【变量 结果】,大成功,【比较 【变量 结果】,【比较 【变量 目标值】,50,100,96】,大失败,【比较 【计算 【变量 目标值】/5】,【变量 结果】,极难成功,【比较 【计算 【变量 目标值】/2】,【变量 结果】,困难成功,【比较 【变量 目标值】,【变量 结果】,成功,失败】】】】】"

编写完成。这个功能的使用方法很简单，只需要一个“目标值”的变量，然后将这个常量【赋值变量】【执行变量】即可。

将这个常量给别人用，对方也不需要研究解读内部的具体代码，这就是一个API.

JS:

function rc(target){

   let roll=Math.ceil(Math.random()\*100);

   if(roll==1){return "大成功";}

   else{

       let BigFail;

       if(target<50){BigFail=96;}

       else{BigFail=100;}

       if(roll<BigFail){

           if(roll>target/5){

               if(roll>target/2){

                   if(roll>target){return "失败";}

                   else{return "普通成功";}

              }

               else{return "困难成功";}

          }

           else{return "极难成功";}

      }

       else{return "大失败";}

  }

}

**头文件**

  头文件是我从我的C++课中借来的概念。

  简单地讲，头文件是由一系列功能相近、相关联或属于同一类别的API组成，供编写代码者调用的文件。例如，包含了π、e、开平方、指数等数学操作的头文件，用于为机械电路编程的头文件等。

  如果你发现你有一些功能会被多个插件用到，那么这些功能便可以包装起来，单独装入一个文件，这就是一个头文件。

  如果将回雪各种常用的功能包装起来，制成一个便于传播的头文件，供各位编写者引用，想来将是一件莫大的功德吧。

头文件

[[define]]

name="规则书检定"

content="……"

插件1

"……【赋值变量 目标值,【变量 好感度】】【赋值变量 好感度检定,【常量 规则书检定】】【执行变量 好感度检定】……"

插件2

"……【赋值变量 目标值,【计算 【计算 【变量 攻击力】-【变量 防御力】】\*10】】【赋值变量 战斗检定,【常量 规则书检定】】【执行变量 战斗检定】……"

**7 链表**

  程序有一种数据结构叫做“线性表”。“线性表”分为两种，一种叫做“顺序表”，又叫“数组”，另一种称为“链表”。

  你不需要对以上这句话中的生词有多么深刻的理解，望文生义即可。（毕竟我也没有多专业）

  链表与顺序表的本质区别在于内存地址的分布上。顺序表，即数组，所有元素在内存上的分布是相邻的，且顺序排列；链表各元素的内存分布则是毫无规律（或许不是“规律”而是“限制”）的，任意两个元素的内存地址间毫无关联。

  因此，链表的每个元素都由数据与指针构成。以单向链表为例，每个元素分为自己储存的数据与指向下一元素的指针两个部分；而双向链表则额外多出一个指向上一元素的指针。

  链表与数组各有优劣。数组的优势在于能够更快速地调用、修改中间元素，特别是尾部元素。而链表则必须先进行遍历查找，然后在进行调用、修改。

设有一个数组与一个链表，它们的长度均为8，现在要调用它们各自第6个元素。

数组：

查看元素6

链表：

查看元素1

查看元素2

查看元素3

查看元素4

查看元素5

查看元素6

链表的优势在于能够更快地删减元素，特别是头部元素。而数组为了保证数据内存分布的连续性，必须对其他数据进行操作。

删除数组和链表各自的第2个元素。

数组：

删除元素2

元素3移至元素2

元素4移至元素3

元素5移至元素4

元素6移至元素5

元素7移至元素6

元素8移至元素7

链表：

查看元素1

元素1指针指向元素3

删除元素2

在数组与链表各自第1、2个元素间添加新元素。

数组：

元素8移至元素9

元素7移至元素8

元素6移至元素7

元素5移至元素6

元素4移至元素5

元素3移至元素4

元素2移至元素3

添加新元素至元素2

链表：

查看元素1

新元素指针指向元素2

元素1指针指向新元素

  另外，链表一般不关注长度这个数据，如果需要，应当另外设立一个数据来存储链表的长度。