I'm not a bot

Ollama 提供了多种命令行工具（CLI）供用户与本地运行的模型进行交互。 我们可以用 ollama --help 查看包含有哪些命令： Large language model runner Usage: ollama [flags] ollama [command] Available Commands: serve Start ollama create Create a model from a Modelfile show Show information for a model run Run a model stop Stop a running model pull Pull a model from a registry push Push a model to a registry list List models ps List running models cp Copy a model rm Remove a model help Help about any command Flags: -h, --help help for ollama -v, --version Show version information 1、使用方法 ollama [flags]：使用标志（flags）运行 ollama。 ollama [command]：运行 ollama 的某个具体命令。 2、可用命令 serve：启动 ollama 服务。 create：根据一个 Modelfile 创建一个模型。 show：显示某个模型的详细信息。 run：运行一个模型。 stop：停止一个正在运行的模型。 pull：从一个模型仓库（registry）拉取一个模型。 push：将一个模型推送到一个模型仓库。 list：列出所有模型。 ps：列出所有正在运行的模型。 cp：复制一个模型。 rm：删除一个模型。 help：获取关于任何命令的帮助信息。 3、标志（Flags） -h, --help：显示 ollama 的帮助信息。 -v, --version：显示版本信息。 1. 模型管理 拉取模型 从模型库中下载模型： ollama pull 例如： ollama pull llama2 运行模型 运行已下载的模型： ollama run 例如： ollama run llama2 列出本地模型 查看已下载的模型列表： ollama list 删除模型 删除本地模型： ollama rm 例如： ollama rm llama2 2. 自定义模型 创建自定义模型 基于现有模型创建自定义模型： ollama create -f 例如： ollama create my-llama2 -f ./Modelfile 复制模型 复制一个已存在的模型： ollama cp 例如： ollama cp llama2 my-llama2-copy 推送自定义模型 将自定义模型推送到模型库： ollama push 例如： ollama push my-llama2 3. 服务管理 启动 Ollama 服务 启动 Ollama 服务以在后台运行： ollama serve 停止 Ollama 服务 停止正在运行的 Ollama 服务： ollama stop 重启 Ollama 服务 重启 Ollama 服务： ollama restart 4. 其他常用命令 查看帮助 查看所有可用命令： ollama --help 查看版本信息 查看当前安装的 Ollama 版本： ollama version 更新 Ollama 更新 Ollama 到最新版本： ollama update 查看日志 查看 Ollama 的日志信息： ollama logs 清理缓存 清理 Ollama 的缓存： ollama clean 5. 模型信息 查看模型详细信息 查看指定模型的详细信息： ollama show 例如： ollama show llama2 查看模型依赖 查看模型的依赖关系： ollama deps 例如： ollama deps llama2 查看模型配置 查看模型的配置文件： ollama config 例如： ollama config llama2 6. 导入与导出 导出模型 将模型导出为文件： ollama export 例如： ollama export llama2 llama2.tar 导入模型 从文件导入模型： ollama import 例如： ollama import llama2.tar 7. 系统信息 查看系统信息 查看 Ollama 的系统信息： ollama system 查看资源使用情况 查看模型的资源使用情况： ollama resources 例如： ollama resources llama2 8. 模型性能 查看模型性能 查看模型的性能指标： ollama perf 例如： ollama perf llama2 9. 模型历史 查看模型历史记录 查看模型的历史记录： ollama history 例如： ollama history llama2 10. 模型状态 检查模型状态 检查指定模型的状态： ollama status 例如： ollama status llama2 Python3 实例 在 Python 中，我们可以使用内置的 sort() 方法或 sorted() 函数来对列表中的元素进行排序。sort() 方法会直接修改原列表，而 sorted() 函数会返回一个新的排序后的列表，原列表保持不变。 # 定义一个列表 numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5] # 使用 sort() 方法对列表进行排序（原地排序） numbers.sort() print("使用 sort() 方法排序后的列表:", numbers) # 使用 sorted() 函数对列表进行排序（返回新列表） sorted_numbers = sorted(numbers) print("使用 sorted() 函数排序后的列表:", sorted_numbers) 代码解析： numbers.sort() : sort() 方法会对列表 numbers 进行原地排序，即直接修改原列表。 sorted(numbers) : sorted() 函数会返回一个新的排序后的列表，原列表 numbers 保持不变。 输出结果： 使用 sort() 方法排序后的列表: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9] 使用 sorted() 函数排序后的列表: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9] Python3 实例 Python3 实例 在 Python 中，列表切片是一种非常强大的功能，它允许你从一个列表中提取一部分元素，形成一个新的子列表。切片操作使用方括号 [] 和冒号：来指定起始索引、结束索引和步长。 假设我们有一个列表 my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]，我们想要提取其中的一部分元素，可以使用切片操作。 my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] # 提取索引 2 到 5 的元素（不包括索引 5） sub_list = my_list[2:5] print(sub_list) 代码解析： my_list[2:5] : 这是一个切片操作，2 是起始索引，5 是结束索引。切片操作会提取从索引 2 开始到索引 5 之前的元素（即不包括索引 5 的元素）。 sub_list : 这是切片操作后得到的新列表。 输出结果： [2, 3, 4] Python3 实例 序列是 Python 中最基本的数据结构。序列中的每个值都有对应的位置值，称之为索引，第一个索引是 0，第二个索引是 1，依此类推。 Python 有 6 个序列的内置类型，但最常见的是列表和元组。 列表都可以进行的操作包括索引，切片，加，乘，检查成员。 此外，Python 已经内置确定序列的长度以及确定最大和最小的元素的方法。 列表是最常用的 Python 数据类型，它可以作为一个方括号内的逗号分隔值出现。 列表的数据项不需要具有相同的类型 创建一个列表，只要把逗号分隔的不同的数据项使用方括号括起来即可。 如下所示： list1 = ['Google', 'Runoob', 1997, 2000] list2 = [1, 2, 3, 4, 5] list3 = ["a", "b", "c", "d"] list4 = ['red', 'green', 'blue', 'yellow', 'white', 'black'] 访问列表中的值 与字符串的索引一样，列表索引从 0 开始，第二个索引是 1，依此类推。 通过索引列表可以进行截取、组合等操作。 #!/usr/bin/python3 list = ['red', 'green', 'blue', 'yellow', 'white', 'black'] print(list[0]) print(list[1]) print(list[2]) 以上实例输出结果： red green blue 索引也可以从尾部开始，最后一个元素的索引为 -1，往前一位为 -2，以此类推。 #!/usr/bin/python3 list = ['red', 'green', 'blue', 'yellow', 'white', 'black'] print(list[-1]) print(list[-2]) print(list[-3]) 以上实例输出结果： black white yellow 使用下标索引来访问列表中的值，同样你也可以使用方括号 [] 的形式截取字符，如下所示： #!/usr/bin/python3 nums = [10, 20, 30, 40, 50, 60, 70, 80, 90] print(nums[0:4]) 以上实例输出结果： [10, 20, 30, 40] 使用负数索引值截取： #!/usr/bin/python3 list = ['Google', 'Runoob', "Zhihu", "Taobao", "Wiki"] # 读取第二位 print ("list[1]: ", list[1]) # 从第二位开始（包含）截取到倒数第二位（不包含） print ("list[1:-2]: ", list[1:-2]) 以上实例输出结果： list[1]: Runoob list[1:-2]: ['Runoob', 'Zhihu'] 更新列表 你可以对列表的数据项进行修改或更新，你也可以使用 append() 方法来添加列表项，如下所示： #!/usr/bin/python3 list = ['Google', 'Runoob', 1997, 2000] print ("第三个元素为 : ", list[2]) list[2] = 2001 print ("更新后的第三个元素为 : ", list[2]) list1 = ['Google', 'Runoob', 'Taobao'] list1.append('Baidu') print ("更新后的列表 : ", list1) 注意：我们会在接下来的章节讨论 append() 方法的使用。 以上实例输出结果： 第三个元素为 : 1997 更新后的第三个元素为 : 2001 更新后的列表 : ['Google', 'Runoob', 'Taobao', 'Baidu'] 删除列表元素 可以使用 del 语句来删除列表中的元素，如下实例： #!/usr/bin/python3 list = ['Google', 'Runoob', 1997, 2000] print ("原始列表 : ", list) del list[2] print ("删除第三个元素 : ", list) 以上实例输出结果： 原始列表 : ['Google', 'Runoob', 1997, 2000] 删除第三个元素 : ['Google', 'Runoob', 2000] 注意：我们会在接下来的章节讨论 remove() 方法的使用 Python列表脚本操作符 列表对 + 和 * 的操作符与字符串相似。+ 号用于组合列表，* 号用于重复列表。 如下所示： Python 表达式结果 描述 len([1, 2, 3])3长度 [1, 2, 3] + [4, 5, 6][1, 2, 3, 4, 5, 6]组合 ['Hi!'] * 4['Hi!', 'Hi!', 'Hi!', 'Hi!']重复 3 in [1, 2, 3]True元素是否存在于列表中 for x in [1, 2, 3]: print(x, end=" ")1 2 3迭代 Python 列表截取与拼接 Python的列表截取与字符串操作类似，如下所示： L=['Google', 'Runoob', 'Taobao'] 操作： Python 表达式结果 描述 L[2]'Taobao'读取第三个元素 L[-2]'Runoob'从右侧开始读取倒数第二个元素: count from the right L[1:]['Runoob', 'Taobao']输出从第二个元素开始后的所有元素 >>> L=['Google', 'Runoob', 'Taobao'] >>> L[2] 'Taobao' >>> L[-2] 'Runoob' >>> L[1:] ['Runoob', 'Taobao'] >>> 列表还支持拼接操作： >>> squares = [1, 4, 9, 16, 25] >>> squares += [36, 49, 64, 81, 100] >>> squares [1, 4, 9, 16, 25, 36, 49, 64, 81, 100] >>> 嵌套列表 使用嵌套列表即在列表里创建其它列表，例如： >>> a = ['a', 'b', 'c'] >>> n = [1, 2, 3] >>> x = [a, n] >>> x [['a', 'b', 'c'], [1, 2, 3]] >>> x[0] ['a', 'b', 'c'] >>> x[0][1] 'b' 列表比较 列表比较需要引入 operator 模块的 eq 方法（详见：Python operator 模块）： # 导入 operator 模块 import operator a = [1, 2] b = [2, 3] c = [2, 3] print("operator.eq(a,b): ", operator.eq(a,b)) print("operator.eq(c,b): ", operator.eq(c,b)) 以上代码输出结果为： operator.eq(a,b): False operator.eq(c,b): True Python列表函数&方法 Python包含以下函数: Python包含以下方法: C++ 标准库提供了丰富的功能，其中 是一个非常重要的容器类，用于存储元素集合，支持双向迭代器。 是 C++ 标准模板库（STL）中的一个序列容器，它允许在容器的任意位置快速插入和删除元素。与数组或向量（ ）不同， 不需要在创建时指定大小，并且可以在任何位置添加或删除元素，而不需要重新分配内存。 语法 以下是 容器的一些基本操作： 包含头文件：#include 声明列表：std::list mylist;，其中 T 是存储在列表中的元素类型。 插入元素：mylist.push_back(value); 删除元素：mylist.pop_back(); 或 mylist.erase(iterator); 访问元素：mylist.front(); 和 mylist.back(); 遍历列表：使用迭代器 for (auto it = mylist.begin(); it != mylist.end(); ++it) 特点 双向迭代： 提供了双向迭代器，可以向前和向后遍历元素。 动态大小：与数组不同， 的大小可以动态变化，不需要预先分配固定大小的内存。 快速插入和删除：可以在列表的任何位置快速插入或删除元素，而不需要像向量那样移动大量元素。 声明与初始化 的声明和初始化与其他容器类似： #include #include int main() { std::list lst1; // 空的list std::list lst2(5); // 包含5个默认初始化元素的list std::list lst3(5, 10); // 包含5个元素，每个元素为10 std::list lst4 = {1, 2, 3, 4}; // 使用初始化列表 return 0; } 实例 下面是一个使用 的简单示例，包括创建列表、添加元素、遍历列表和输出结果。 #include #include int main() { // 创建一个整数类型的列表 std::list numbers; // 向列表中添加元素 numbers.push_back(10); numbers.push_back(20); numbers.push_back(30); // 访问并打印列表的第一个元素 std::cout li[-1] 'example' >>> li[-3] 'mpilgrim' >>> li ['a', 'b', 'mpilgrim', 'z', 'example'] >>> li[1:3] ['b', 'mpilgrim'] >>> li[1:-1] ['b', 'mpilgrim', 'z'] >>> li[0:3] ['a', 'b', 'mpilgrim']