http考察 请求首部字段host是不是必须的？为什么？

Host 首部字段在 HTTP/1.1 规范内是唯一一个必须被包含在请求内的首部字段。  
首部字段 Host 和以单台服务器分配多个域名的虚拟主机的工作机制有很密切的关联，这是首部字段 Host 必须存在的意义。

http头中可以没有host字段吗？在http 1.1中不能缺失host字段,如果缺失, 服务器返回400 bad request，http1.1中不能缺失host字段，但host字段可以是空值。在http 1.0中可以缺失host字段。

4XX 客户端错误？

· 400 bad request，请求报文存在语法错误  
· 401 unauthorized，表示发送的请求需要有通过 HTTP 认证的认证信息  
· 403 forbidden，表示对请求资源的访问被服务器拒绝  
· 404 not found，表示在服务器上没有找到请求的资源

301跳转和302跳转具体的区别在哪里？

定义如下：  
301 Moved Permanently 被请求的资源已永久移动到新位置，并且将来任何对此资源的引用都应该使用本响应返回的若干个URI之一。如果可能，拥有链接编辑功能的客户端应当自动把请求的地址修改为从服务器反馈回来的地址。除非额外指定，否则这个响应也是可缓存的。  
302 Found 请求的资源现在临时从不同的URI响应请求。由于这样的重定向是临时的，客户端应当继续向原有地址发送以后的请求。只有在Cache-Control或Expires中进行了指定的情况下，这个响应才是可缓存的。  
字面上的区别就是301是永久重定向，而302是临时重定向。

题目内容是这样的：烧一根不均匀的绳，从头烧到尾总共需要1个小时。现在有若干条材质相同的绳子，问如何用烧绳的方法来计时一个小时十五分钟呢?

需要三根绳子：  
第一根和第二根同时点,第一根点一头,第二根点两头  
在第二根燃完后（30分钟）,点燃第一根的另一头  
前两根燃完后（15分钟）,点燃第三根的两头  
全部燃完后（30分钟）  
一共就是1小时15分钟.

STL 是“Standard Template Library”的缩写，中文译为“标准模板库”。STL 是 C++ 标准库的一部分，不用单独安装。

C++ 对模板（Template）支持得很好，STL 就是借助模板把常用的数据结构及其算法都实现了一遍，并且做到了数据结构和算法的分离。例如，vector 的底层为顺序表（数组），list 的底层为双向链表，deque 的底层为循环队列，set 的底层为红黑树，hash\_set 的底层为哈希表。  
https://www.cnblogs.com/skyfsm/p/6934246.html

两个有序数组合并成一个：

public class MergeArrayAndOrder {  
    public static void main(String[] args) {  
        int[] a = new int[]{1,3,5,7,9,16,17};  
        int[] b = new int[]{0,2,4,6,8,10,11,12,13,14,18,19};  
        int[] c = MergeArrayAndOrder.mergeOrder(a,b);  
        for(int i : c){  
            System.out.print(i);  
        }  
    }  
    public static int[] mergeOrder(int[] arr1,int[]arr2){  
        int[] result = new int[arr1.length + arr2.length];  
        int m = 0;  
        int n = 0;  
        for (int i=0;i<result.length;i++){  
            if(m == arr1.length){  
                result[i] = arr2[n++];  
            }else if(n == arr2.length){  
                result[i] = arr1[m++];  
            }else {  
                if(arr1[m]<arr2[n]){  
                    result[i] = arr1[m++];  
                }else{  
                    result[i] = arr2[n++];  
                }  
            }  
        }  
        return result;  
    }  
}

快速排序   C++语言

#include <iostream>  
using namespace std;  
void Qsort(int a[], int low, int high)  
{  
    if(low >= high)  
    {  
        return;  
    }  
    int first = low;  
    int last = high;  
    int key = a[first];/\*用字表的第一个记录作为枢轴\*/  
  
    while(first < last)  
    {  
        while(first < last && a[last] >= key)  
        {  
            --last;  
        }  
        a[first] = a[last];/\*将比第一个小的移到低端\*/  
        while(first < last && a[first] <= key)  
        {  
            ++first;  
        }     
        a[last] = a[first];      
/\*将比第一个大的移到高端\*/  
    }  
    a[first] = key;/\*枢轴记录到位\*/  
    Qsort(a, low, first-1);  
    Qsort(a, first+1, high);  
}  
int main()  
{  
    int a[] = {57, 68, 59, 52, 72, 28, 96, 33, 24};  
    Qsort(a, 0, sizeof(a) / sizeof(a[0]) - 1);/\*这里原文第三个参数要减1否则内存越界\*/  
  
    for(int i = 0; i < sizeof(a) / sizeof(a[0]); i++)  
    {  
        cout << a[i] << "";  
    }  
  
    return 0;  
}

快速排序  Python 语言   分而治之+递归

# -\*- coding: UTF-8 -\*-  
#quick sort  
def quickSort(array):  
    if len(array) < 2:  # 基线条件（停止递归的条件）  
        return array  
    else:    
        baseValue = array[0]  # 选择基准值  
        less, equal, greater = [], [baseValue], []  
        for m in array[1:]:  
            if m < baseValue:  
                less.append(m)    # 由所有小于基准值的元素组成的子数组  
            elif m > baseValue:  
                greater.append(m)  # 由所有大于基准值的元素组成的子数组  
            else:  
                # 包括基准在内的同时和基准相等的元素，  
                equal.append(m)  
        return quickSort(less) + equal + quickSort(greater)  
# 示例：  
array = [2,3,5,7,1,4,6,15,5,2,7,9,10,15,9,17,12]  
print(quickSort(array))  
# 输出为[1, 2, 2, 3, 4, 5, 5, 6, 7, 7, 9, 9, 10, 12, 15, 15, 17]

关于在vim中的查找和替换

##### **1，查找**

在normal模式下按下/即可进入查找模式，输入要查找的字符串并按下回车。 Vim会跳转到第一个匹配。按下n查找下一个，按下N查找上一个。

##### **2，大小写敏感查找**

在查找模式中加入\c表示大小写不敏感查找，\C表示大小写敏感查找。例如：  
/foo\c  
将会查找所有的"foo","FOO","Foo"等字符串。

##### **3，查找与替换**

:s（substitute）命令用来查找和替换字符串。语法如下：  
:{作用范围}s/{目标}/{替换}/{替换标志}  
例如:  
%s/foo/bar/g会在全局范围(%)查找foo并替换为bar，所有出现都会被替换（g）

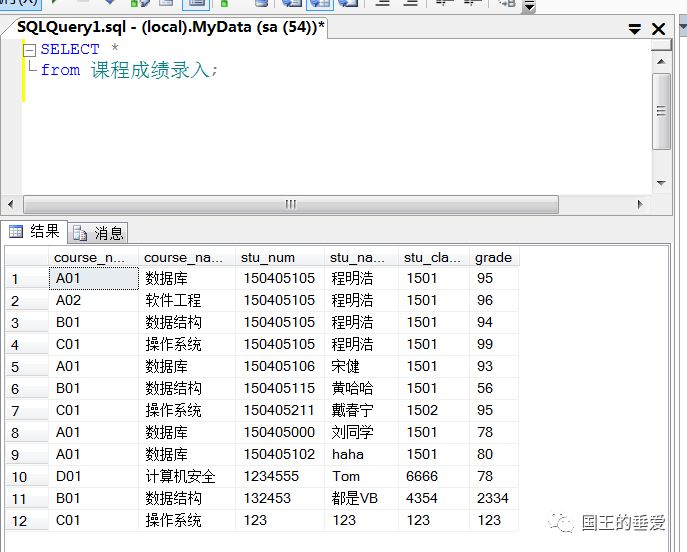
##### **4，作用范围**

作用范围分为当前行、全文、选区等等。  
当前行：  
:s/foo/bar/g  
全文：  
:%s/foo/bar/g  
选区，在Visual模式下选择区域后输入:，Vim即可自动补全为 :'<,'>。  
:'<,'>s/foo/bar/g  
2-11行：  
:5,12s/foo/bar/g  
当前行.与接下来两行+2：  
:.,+2s/foo/bar/g

##### **5，替换标志**

上文中命令结尾的g即是替换标志之一，表示全局global替换（即替换目标的所有出现）。 还有很多其他有用的替换标志：  
【1】空替换标志表示只替换从光标位置开始，目标的第一次出现：  
:%s/foo/bar  
【2】i表示大小写不敏感查找，I表示大小写敏感：  
:%s/foo/bar/i   # 等效于模式中的\c（不敏感）或\C（敏感）:%s/foo\c/bar  
【3】c表示需要确认，例如全局查找"foo"替换为"bar"并且需要确认：  
:%s/foo/bar/gc  
回车后Vim会将光标移动到每一次"foo"出现的位置，并提示  
replace with bar (y/n/a/q/l/^E/^Y)?  
按下y表示替换，n表示不替换，a表示替换所有，q表示退出查找模式， l表示替换当前位置并退出。

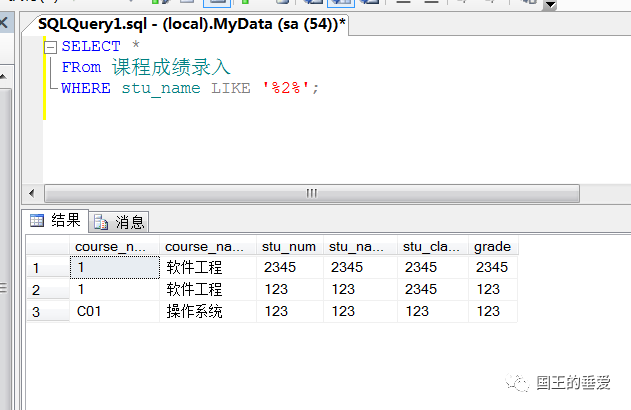
数据库语句





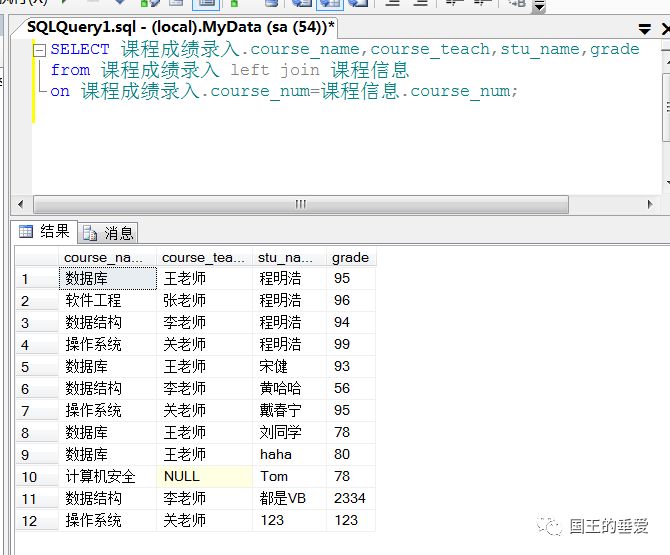


 （ SqlServer 和 Mysql 的写法 ）









**可以使用的不同的 SQL JOIN 类型：**

**·**INNER JOIN：如果表中有至少一个匹配，则返回行

**·**LEFT JOIN：即使右表中没有匹配，也从左表返回所有的行

**·**RIGHT JOIN：即使左表中没有匹配，也从右表返回所有的行

**·**FULL JOIN：只要其中一个表中存在匹配，则返回行