139.单词拆分

题目链接：https://leetcode-cn.com/problems/word-break/

给定一个非空字符串 s 和一个包含非空单词的列表 wordDict，判定 s 是否可以被空格拆分为一个或多个在字典中出现的单词。

说明：

拆分时可以重复使用字典中的单词。

你可以假设字典中没有重复的单词。

示例 1：

输入: s = "leetcode", wordDict = ["leet", "code"]

输出: true

解释: 返回 true 因为 "leetcode" 可以被拆分成 "leet code"。

示例 2：

输入: s = "applepenapple", wordDict = ["apple", "pen"]

输出: true

解释: 返回 true 因为 "applepenapple" 可以被拆分成 "apple pen apple"。

注意你可以重复使用字典中的单词。

> 注意拆分的反义词就是组合
> **单词就是物品，字符串s就是背包，单词能否组成字符串s意思就是能不能装满背包**
> **由于拆分时可以重复使用字典中的单词，说明就是一个完全背包。。**

动规五部曲分析如下：

> **1、确定dp数组以及下标的含义**
> **dp[i]:字符串长度为i的话,dp[i]为true,表示可以拆分为一个或多个在字典中出现的单词**

2、确定递推公式
如果确定dp[j]是true,且字符串s中[j,i]的子字符串在字典中,那么dp[i]一定是true(j<i)
所以递推公式是dp[i]=dp[i]&&dp[j]&&(set.find(s.subString(j,i-j+1))!=set.end())

3、dp数组如何初始化:
dp[0]=true,dp[其余]=false

4、遍历顺序:
对于01背包，二维dp数组遍历可以是先物品后背包或者先背包后物品都可,一维dp数组只能是先物品后背包。
对于完全背包，二维dp数组遍历可以是先物品后背包或者先背包或物品，但是如果求的是组合数，则遍历顺序为前者；如果求的是排列数，那么遍历顺序是后者。

```
class Solution {
public:
    bool wordBreak(string s, vector<string>& wordDict) {
 //初始化单词集合
 unordered_set<string> wordSet(wordDict.begin(),wordDict.end());
 vector<bool> dp(s.size()+1,false);
 dp[0]=0;
 //先遍历背包后物品
 for(int i=1;i<=s.size();i++){//遍历背包
  for(int j = 0;j < i;j++){//遍历物品
  string word=s.substr(j,i-j);//substr(其实位置，截取长度)
  if(wordSet.find(word)!=wordSet.end()){
  dp[i]=dp[j];
  }
  }
  }
  return dp[s.size()];
     }
}
```