数独的解法需 遵循如下规则：
数字 1-9 在每一行只能出现一次。
数字 1-9 在每一列只能出现一次。
数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。（请参考示例图）
数独部分空格内已填入了数字，空白格用 '.' 表示。

分析：

1. 对数独，N皇后问题，求解一定采用回溯法。
2. 回溯时，每次都从当前节点<x,y>开始，判断是否完成回溯；
3. 状态遍历从[x..row-1][y..col-1],如果不需要填写数字直接递归下一列；
4. 否则从[0~9]中选取合适数字进行回溯，做出选择后，回溯下一列，如果下一列回溯成功，找到一个接解，直接返回；
5. 否则撤销选择，穷举下一个数字。

```
class Solution {
public:
  void solveSudoku(std::vector<std::vector<char>>& board) {
    int row = board.size(), col = 0;
    if (0 == row) {
      return;
    } else {
      col = board[0].size();
    }

    if (row != col) {
      return;
    }

    backtrace(board, row, col, 0, 0);
  }

private:
  bool backtrace(std::vector<std::vector<char>>& board,
                 int                             row,
                 int                             col,
                 int                             x,
                 int                             y) {
    if (y == col) {
      // 计算到最后一列，计算下一行
      return backtrace(board, row, col, x + 1, 0);
    }

    // 计算到最后一行，得到一个结果
    if (x == row) {
      return true;
    }

    // 对每个位置进行穷举
```

```
    for (int i = x; i < row; i++) {
      for (int j = y; j < col; j++) {
        if ('.' != board[i][j]) {
          // 如果有预设数字，遍历下一列
          return backtrace(board, row, col, i, j + 1);
        }

        // 非预设数字
        for (char ch = '1'; ch <= '9'; ch++) {
          // 当前选中解无效，继续
          if (!isValid(board, row, col, i, j, ch)) {
            continue;
          }

          // 做选择
          board[i][j] = ch;
          if (backtrace(board, row, col, i, j + 1)) {  // 找到可行解，返回
            return true;
          }

          // 撤销选择
          board[i][j] = '.';
        }

        return false;  // 穷举完所有选择，则直接放回
      }
    }

    return false;  // 没有可行解
  }

  // 判断所填数字是否合法
  bool isValid(std::vector<std::vector<char>>& board,
               int                             row,
               int                             col,
               int                             x,
               int                             y,
               char                            ch) {
    for (int i = 0; i < row; i++) {
      // 同一行是否有相同字符
      if (ch == board[x][i]) {
        return false;
      }
      // 同一列是否有相同字符
      if (ch == board[i][y]) {
        return false;
      }

      // 临近3*3的方格是否有相同字符
      if (ch == board[(x / 3) * 3 + i / 3][(y / 3) * 3 + i % 3]) {
        return false;
      }
    }
    return true;
```

```
    }
};
```