Python

类设计

- `__init__(self, board_str)`: 接受一个Sudoku字符串输入，并将其解析为9x9的矩阵。
- `_parse_board(self, board_str)`: 内部方法，将字符串转换为二维数组表示的Sudoku棋盘。
- `get_candidates(self, row, col)`: 计算给定单元格的候选值，通过排除规则得到可选的数字集合。
- `solve(self)`: 进行简单的推理，输出每个空白单元格的候选值。
- `clone(self)`: 通过 `deepcopy` 实现Sudoku对象的克隆。
- `to_json(self)`: 将当前棋盘序列化为JSON格式。
- `__eq__(self, other)`: 比较两个Sudoku对象是否相等。
- `__str__(self)`: 返回棋盘的可读字符串表示。

其他OO技术方法

- **克隆**：使用 `deepcopy` 来复制当前对象。
- **序列化**：提供了 `to_json` 方法来将对象的棋盘转换为JSON格式。
- **比较**：通过 `__eq__` 方法来比较两个Sudoku对象是否相等。

测试代码

- 测试了Sudoku对象的创建、候选值推理、克隆对象、序列化以及比较功能。

1. `solve_sudoku(self)`: 公开方法，启动回溯算法求解数独。
2. `_solve(self)`: 私有递归回溯函数，遍历数独棋盘，尝试为每一个空格填入可能的候选值。如果某个候选值能使整个数独最终解出，则填入该值并继续；如果不能，则回溯，尝试下一个候选值。

回溯算法的基本步骤：

- **找到空白单元格**：对于每一个空单元格（值为0），获取其候选值。
- **尝试填入候选值**：依次填入候选值，并继续递归求解后续单元格。
- **回溯**：如果某个候选值导致后续单元格无解，则重置该单元格为0，回溯到上一步，尝试下一个候选值。

边界情况：

- 当棋盘已经全部填满时，回溯终止并返回True，表示数独已解出。
- 如果在某一步无法填入有效的候选值，则返回False，触发回溯。

测试结果

- **初始化棋盘**：从输入的字符串生成数独棋盘。
- **回溯求解**：调用 `solve_sudoku()`，如果数独可以解出，打印最终的解；如果不能解出，打印相应提示。

C++

1. `Sudoku(const string& board_str)`:

    - 构造函数，从输入的字符串初始化数独棋盘。字符串必须长度为81，表示9x9的棋盘，其中`'0'`表示空格。
    - 初始化一个9x9的二维数组，将字符串中的字符逐个转换为整数并存入棋盘中。
2. `solveSudoku()`：
    - 启动回溯求解数独，调用私有的`solve()`函数，返回是否成功求解的布尔值。
3. `solve()`：
    - 核心的回溯算法函数。递归遍历棋盘中的每一个空格，尝试为其填入1到9的数字。
    - 如果某个数字符合数独规则（行、列和宫格内没有冲突），则填入该数字并继续递归求解下一个空格。
    - 如果所有空格都成功填满，返回`true`。否则，回溯到上一步，重置该格为0，并尝试下一个数字。
4. `isValid(int row, int col, int num)`：
    - 判断在`(row, col)`位置是否可以放置数字`num`。
    - 通过检查该行、该列以及该3x3小宫内是否已有相同的数字来决定。
5. `printBoard()`：
    - 打印当前的数独棋盘。空格显示为`.`，非空格显示对应的数字。
6. `main()`：
    - 主函数中定义了一个数独字符串。程序通过调用`solveSudoku()`进行求解，并根据结果输出求解后的数独棋盘或显示无法求解的提示信息。

测试成功：

初始化棋盘:

```
. 1 7 9 . 3 6 . .
. . . . 8 . . . .
9 . . . . . 5 . 7
. 7 2 . 1 . 4 3 .
. . . 4 . 2 . 7 .
. 6 4 3 7 . 2 5 .
7 . 1 . . . . 6 5
. . . . 3 . . . .
. . 5 6 . 1 7 2 .
```

推理成功:

```
4 1 7 9 5 3 6 8 2
2 5 6 1 8 7 9 4 3
9 8 3 2 4 6 5 1 7
8 7 2 5 1 9 4 3 6
5 3 9 4 6 2 8 7 1
1 6 4 3 7 8 2 5 9
7 9 1 8 2 4 3 6 5
6 2 8 7 3 5 1 9 4
3 4 5 6 9 1 7 2 8
```

测试失败:

初始化棋盘:

```
2 1 7 9 4 3 6 . .
. . . . 8 . . . .
9 . . . . . 5 . 7
. 7 2 . 1 . 4 3 .
. . . 4 . 2 . 7 .
. 6 4 3 7 . 2 5 .
7 . 1 . . . . 6 5
. . . . 3 . . . .
. . 5 6 . 1 7 2 .
```

推理失败.