# 动态类型语言Sudoku文档

## 目录



---

## 介绍

Sudoku由动态类型语言*python*实现，两个主要类组成：`Grid` 和 `Sudoku`。该代码实现了数独棋盘的创建、操作、推理候选值、克隆、序列化及反序列化等功能。数独棋盘可以通过字符串输入进行初始化，并且可以从文件中序列化和反序列化数独对象。该文档将详细说明代码结构和功能。

---

## 类说明

### Grid 类

`Grid` 是一个基础类，表示一个 9x9 的数独棋盘。该类的主要功能是提供获取行、列及小方块内元素的能力。

#### 属性

- `BOX_SIZE`: 小方块的尺寸（3x3）。
- `GRID_SIZE`: 整个棋盘的尺寸（9x9）。
- `grid`: 存储棋盘内容的二维列表。

#### 方法

1. `__init__(grid=None)`:
   初始化 `Grid` 对象，可以选择性地传入一个二维数组 `grid` 作为棋盘。如果未传入，棋盘默认为 9x9 的空数组（即全 0）。

2. `getRow(row)`:
   返回指定行的所有元素。

3. `getColumn(col)`:
   返回指定列的所有元素。

4. `getBox(row, col)`:
   返回指定单元格 (row, col) 所在小方块（3x3）的所有元素。

### Sudoku 类

`Sudoku` 类继承自 `Grid` 类，是整个数独求解器的核心。它增加了棋盘的解析、推理及序列化等高级功能。

#### 属性

继承自 `Grid` 的属性，包括 `BOX_SIZE`、`GRID_SIZE` 和 `grid`。

#### 方法

1. `__init__(board_str=None, grid=None)`:
   使用字符串 `board_str` 或已有二维数组 `grid` 初始化棋盘。`board_str` 应该是一个长度为 81 的字符串，用于表示数独棋盘的所有单元格，0 代表空格。

2. `_parse(board_str)`:
   将字符串 `board_str` 转换为 9x9 的二维数组。

3. `getInference()`:
   推理并返回棋盘中每个空单元格（值为 0）的候选值，候选值由行、列及小方块中未使用的数字组成。

4. `_get_possible_values(row, col)`:
   返回指定单元格 (row, col) 的候选值（排除行、列及小方块中已存在的数字）。

5. `__str__()`:
   返回数独棋盘的字符串表示形式。

6. `__eq__(other)`:
   比较两个 `Sudoku` 对象的棋盘是否相同。

7. `clone()`:
   克隆当前 `Sudoku` 对象并返回一个新的副本。

8. `serialize(filename)`:
   将当前 `Sudoku` 对象序列化到指定的文件 `filename` 中。

9. `deserialize(filename)`:
   从指定的文件 `filename` 中反序列化并返回一个 `Sudoku` 对象。

---

## 代码解释

### Grid 类方法

1. `__init__(grid=None)`:
   该方法用于初始化 `Grid` 对象。如果传入了 `grid` 参数，则将其深拷贝并赋值给 `self.grid`；否则，初始化一个 9x9 的空棋盘。

```
def __init__(self, grid=None):
    self.grid = deepcopy(grid) if grid else [[0] * self.GRID_SIZE for _ in
range(self.GRID_SIZE)]
```

2. `getRow(row)`:
   该方法返回指定行的所有元素。

```
def getRow(self, row):
    return self.grid[row]
```

3. `getColumn(col)`:
   该方法返回指定列的所有元素。

```
def getColumn(self, col):
    return [self.grid[row][col] for row in range(self.GRID_SIZE)]
```

4. `getBox(row, col)`:
   该方法返回指定单元格所在小方块（3x3）的所有元素。通过计算单元格的起始行列位置 `start_row` 和 `start_col` 来获取小方块的内容。

```
def getBox(self, row, col):
    start_row, start_col = (row // self.BOX_SIZE) * self.BOX_SIZE, (col //
self.BOX_SIZE) * self.BOX_SIZE
    return [self.grid[r][c] for r in range(start_row, start_row +
self.BOX_SIZE)
                            for c in range(start_col, start_col +
self.BOX_SIZE)]
```

## Sudoku 类方法

1. `__init__(board_str=None, grid=None)`:
   该方法用于通过字符串或已有棋盘二维数组来初始化 `Sudoku` 对象。

```
def __init__(self, board_str=None, grid=None):
    if board_str:
        super().__init__(self._parse(board_str))
    else:
        super().__init__(grid)
```

2. `_parse(board_str)`:
   将字符串 `board_str` 转换为 9x9 的二维数组形式。

```
def _parse(self, board_str):
    board = [[int(board_str[i * self.GRID_SIZE + j]) for j in
range(self.GRID_SIZE)]
                for i in range(self.GRID_SIZE)]
    return board
```

3. `getInference()`:
   遍历整个棋盘，找到所有空单元格（值为 0）的候选值。使用 `_get_possible_values` 方法来获取候选值。

```
def getInference(self):
    candidates = [[[] for _ in range(self.GRID_SIZE)] for _ in
range(self.GRID_SIZE)]
    for row in range(self.GRID_SIZE):
        for col in range(self.GRID_SIZE):
            if self.grid[row][col] == 0:
                candidates[row][col] = self._get_possible_values(row, col)
    return candidates
```

4. `_get_possible_values(row, col)`:
   获取指定单元格 (row, col) 的候选值。候选值排除行、列及小方块中已存在的数字。

```
def _get_possible_values(self, row, col):
    possible_values = set(range(1, 10))
    used_values = set(self.getRow(row) + self.getColumn(col) +
self.getBox(row, col))
    return list(possible_values - used_values)
```

---

## 文件序列化和反序列化

`Sudoku` 类提供了序列化和反序列化方法，能够将 `Sudoku` 对象保存到文件中，并且可以从文件中恢复对象。

```
def serialize(self, filename):
    with open(filename, 'wb') as f:
        pickle.dump(self, f)

@staticmethod
def deserialize(filename):
    with open(filename, 'rb') as f:
        return pickle.load(f)
```

以上示例展示了如何将数独对象序列化到 `sudoku.pkl` 文件，并且从该文件中反序列化回原对象。

---

## 示例与测试

```
# 测试代码
if __name__ == "__main__":
    input =
"017903600000080000900000507072010430000402070064370250701000065000030000005601720
"

    # 创建 Sudoku 对象
    sudoku = Sudoku(input)

    # 打印棋盘
    print("原始棋盘:")
    print(sudoku)

    # 推理并打印候选值
    print("\n候选值:")
    candidates = sudoku.getInference()
    for row in candidates:
        print(row)

    # 序列化和反序列化
    sudoku.serialize('sudoku.pkl')
    deserialized_sudoku = Sudoku.deserialize('sudoku.pkl')

    # 比较序列化前后的对象是否相等
    print("\n序列化和反序列化是否相等:", sudoku == deserialized_sudoku)

    # 克隆对象并比较
    cloned_sudoku = sudoku.clone()
    print("克隆后的数独对象是否相等:", sudoku == cloned_sudoku)
```