

在您提供的 `main.nice.js` 文件中，模块47和模块48是构建“无限城市”核心结构的关键组成部分。它们分别负责不同的功能，但通过共享数据结构和对象紧密协作，共同实现城市的场景管理和动态更新。以下是对它们关系的详细分析，以及在代码中是如何关联起来的说明。

---

## 模块47：Application

### 功能概述

模块47定义了一个名为 `Application` 的类，继承自 `THREE.Scene`，它是整个城市场景的管理核心。它负责：

- 创建和管理由多个区块（chunks）组成的场景。
- 每个区块是一个独立的单元，包含特定的元素（如道路、建筑等）。

### 关键方法

- `_initChunks`：初始化一个由多个区块组成的网格，并将这些区块添加到场景中。
- `_createChunkAt`：在指定位置（x, y）创建一个新的区块对象，返回一个 `THREE.Object3D` 实例作为区块的容器。
- `forEachChunk`：提供一个方法来遍历所有区块并对它们执行特定操作。

### 代码片段



A screenshot of a code editor window showing the beginning of a JavaScript file. The tab bar at the top shows 'javascript'. The main editor area contains the following code:

```
var Application = function() {
```

The code is written in a monospaced font, and the editor has a dark theme with light-colored syntax highlighting.

```

THREE.Scene.call(this);
this._pickables = [];
this.chunks = [];
this._initChunks();
};

Application.inherit(THREE.Scene, {
    _initChunks: function() {
        for (var j = 0; j < properties.CHUNK_COUNT; j++) {
            for (var i = 0; i < properties.CHUNK_COUNT; i++) {
                if (void 0 === this.chunks[i]) {
                    this.chunks[i] = [];
                }
                var id = this._createChunkAt(i, j);
                this.chunks[i][j] = id;
                this.add(id);
            }
        }
    },
    _createChunkAt: function(x, time) {
        var settings = new THREE.Object3D();
        var pregeom = new THREE.PlaneGeometry(properties.CHUNK_SIZE, properties.CHUNK_SIZE);
        var wheelAxisMat = new THREE.MeshBasicMaterial();
        var data = new THREE.Mesh(pregeom, wheelAxisMat);
        // 设置位置和属性
        settings.position.x = left + x * properties.CHUNK_SIZE;
        settings.position.z = i + time * properties.CHUNK_SIZE;
        settings.add(data);
        return settings;
    }
});

```

- `this.chunks` 是一个二维数组，存储了所有区块的 `THREE.Object3D` 对象。
  - 每个区块通过 `_createChunkAt` 创建，并被添加到场景中。
- 

## 模块48: update

### 功能概述

模块48定义了一个名为 `update` 的构造函数（尽管在代码中它更像是一个类），负责生成和管理区块的具体数据，包括道路、车辆、云等元素。它还处理区块之间的关系以及动态元素的更新。

## 关键方法

- `_generate`：生成整个城市的区块数据，填充 `this.chunks` 数组。
- `_getRandomChunk`：为指定位置生成一个随机的区块，包含道路、车辆等内容，并返回一个 `THREE.Object3D` 对象。
- `update`：更新区块中的动态元素（如车辆和云的位置）。
- `_forEachNeighboringChunk`：遍历指定区块的相邻区块并执行操作。

## 代码片段

javascript

X ≡ ▶ ⌂

```
var update = function(data, options, prop, index, vertices) {  
    // do something with data // THIS IS THE LINE
```

```

this.keys = data, // 地图区块模型
this.lanes = []; // 道路数据
this.intersections = prop; // 交叉口数据
this.carObjects = index; // 车辆模型
this.mobs = []; // 动态元素(车辆、云等)
this.chunks = []; // 区块数据
this.cloudObjects = vertices; // 云模型
this._generate();
};

update.inherit(Object, {
    _generate: function() {
        for (var i = 0; i < options.TABLE_SIZE; i++) {
            for (var x = 0; x < options.TABLE_SIZE; x++) {
                if (void 0 === this.chunks[x]) {
                    this.chunks[x] = [];
                }
                var n = this._getRandomChunk(x, i);
                n.tableX = x;
                n.tableY = i;
                this.chunks[x][i] = { node: n };
            }
        }
    },
    _getRandomChunk: function(x, y) {
        var self = new THREE.Object3D();
        self.name = "chunk";
        var block = this._getRandomBlockAt(x, y); // 获取随机区块内容
        self.add(block);
        // 添加道路、车辆、云等元素
        return self;
    },
    update: function(target) {
        this.mobs.forEach(function(e) {
            e.update(target);
        });
    }
});

```

- `this.chunks` 是一个二维数组，存储了每个区块的节点（`node`），这些节点是 `THREE.Object3D` 实例，包含了区块的视觉内容。

## 模块47和模块48之间的关系

模块47和模块48通过共享区块对象和数据结构紧密关联，分工明确：

- **模块47 ( Application )**：负责场景的整体管理和区块的创建，专注于场景层面的结构。
- **模块48 ( update )**：负责生成区块的具体内容（如道路、车辆等）并更新动态元素，专注于数据和逻辑层面。

## 具体关联方式

### <sup>1</sup> 场景管理与数据生成

- 模块47 创建并管理场景中的区块对象 (`THREE.Object3D`)，通过 `this.chunks` 存储这些对象。

- 模块48 生成区块的详细数据和内容（通过 `_getRandomChunk`），并将这些内容与模块47 中的区块对象关联起来。

## 2 区块创建与内容填充

- 在模块47的 `_initChunks` 中，通过 `_createChunkAt` 创建空的区块容器（`THREE.Object3D`）。
- 在模块48的 `_generate` 中，通过 `_getRandomChunk` 生成包含道路、车辆等的区块内容。这些内容最终被添加到模块47创建的区块容器中。

## 3 数据共享

- 模块48 的 `this.chunks` 数组存储了每个区块的节点（`node`），这些节点是 `THREE.Object3D` 实例，包含了所有视觉元素。
- 模块47 的 `this.chunks` 数组引用了相同的区块对象，通过二维索引（`tableX`, `tableY`）与模块48的数据对应，确保场景和数据的一致性。

## 4 动态更新

- 模块48 的 `update` 方法负责更新动态元素（如车辆和云的移动），这些元素是区块的一部分。
- 由于这些动态元素被添加到了模块47的区块对象中，它们的更新会直接影响场景的整体表现。

### 代码中的具体关联点

#### · 区块创建：

- 模块47 的 `_createChunkAt` 创建了一个空的 `THREE.Object3D` 实例作为区块容器。
- 模块48 的 `_initChunks` 和 `_generate` 生成并关联区块内容（如道路、车辆）。

模块48 的 `_getRandomChunk` 生成具体的区块内容（如道路、车辆），并通过 `self.add()` 将这些内容添加到区块容器中。

- **数据引用：**

- 模块48 的 `this.chunks[x][i] = { node: n }` 中的 `node` 是区块的 `THREE.Object3D` 实例。
- 模块47 的 `this.chunks[i][j]` 直接存储这些对象，确保两者共享相同的区块实例。

- **车辆管理：**

- 模块48 的 `this.mobs` 数组存储了所有车辆和云的实例，这些实例被添加到区块的 `node` 中。
  - 模块47 的场景通过引用这些区块对象，在渲染时显示这些动态元素。
- 

## 总结

模块47和模块48通过以下方式在代码中关联起来：

- **对象共享：** 模块47 创建的区块容器（`THREE.Object3D`）被模块48填充内容，形成完整的区块。
- **数据一致性：** 两者的 `chunks` 数组通过二维索引和对象引用保持同步。
- **功能协作：** 模块47 管理场景结构，模块48 提供数据和动态更新，共同实现无限城市的运行。

这种分工使得代码结构清晰，模块47 专注于场景管理，模块48 专注于内容生成和动态逻辑，便于维护和扩展。