通用模型

> **支持格式**
>
> 支持的格式有 `sbm、gltf (glb)、fbx、sbmx`

loadModel

加载 `model` 模型。

样例：

定义：

```ts
interface ModelInfo extends BaseObject3DInfo, ObjectEvents<Model> {
  url: string;
```

function loadModel(modelInfo: ModelInfo): Promise<Model>;

用法：

```js
ssp
  .loadModel(
    // modelInfo
    {
      id: 'xx',
      name: 'xx',
      url: 'xx/x.fbx',
      level: {
        max: 1000,
        min: null,
      },
      position: { x: 0, y: 0, z: 0 },
      rotation: { x: 0, y: 0, z: 0 },
      scale: { x: 2, y: 2, z: 2 },
      onClick(e) {
        /**
         * 对象的独立事件触发后，默认不传播（类似 DOM 的事件冒泡）到全局事件，
         * 调用 eventPropagation 方法通知事件继续传播到全局。
         *
         * warn:
         *  在 **非箭头函数** 中参数 e 与 this 的指向都是当前模型对象，
         *  在 *箭头函数** 参数 e 依然是模型对象，但 this 指向会发生改变。
         */
        this.eventPropagation();

        console.log('模型自身的点击事件触发', this);
      },
      onDblClick: (e) => {
        /**
         * 这里模拟在 **箭头函数** 中
         */
        e.eventPropagation();

        console.log('模型自身的双击事件触发', e);
      },
      userData: {},
    }
```

```
.catch((err) => console.error(err));
```

参数：

modelInfo

- **描述:** 实例 `Model` 对象所需信息
- **类型:** ModelInfo
- **必填:** ✓

ModelInfo

属性	描述	类型	必填	默认值
id	唯一ID	string | number	✓	
name	名称	string	✗	
url	资源路径	string	✓	
level	显示层级范围	Level	✗	{ max: null, min: null }
visible	是否可见	boolean	✗	true
position	位置坐标	Position	✗	{ x: 0, y: 0, z: 0 }
rotation	旋转弧度	Rotation	✗	{ x: 0, y: 0, z: 0 }
scale	缩放比例	Scale	✗	{ x: 1, y: 1, z: 1 }
userData	用户数据	any	✗	{}
onClick	左键单击事件	(object: Model) => void	✗	null
onDblClick	左键双击事件	(object: Model) => void	✗	null

onRightClick	右键单击事件	(object: Model) => void	✗	null
onLoad	加载完成事件	(object: Model) => void	✗	null

cloneModel

克隆 Model 模型

定义:

```ts
interface CloneModelInfo extends Omit<ModelInfo, 'url'> {}

function cloneModel(
  model: Model,
  modelInfo: CloneModelInfo,
  parent?: BaseObject3D | null
): Promise<Model>;
```

用法:

```js
const clonedModel = await ssp.cloneModel(model, {
  id: 'clone_model',
  position: {
    x: 100,
    y: 0,
    z: 0,
  },
});
```

参数:

model

- **描述:** Model 对象
- **类型:** `Model`
- **必填:** ✓

modelInfo

同 ModelInfo, 但不需要字段 `url` 。

parent

- **描述:** 将 `Model` 克隆到的 `parent` 下
- **类型:** `Model`
- **必填:** ✗

getModelById 👎

通过 `id` 查找

定义:

```ts
function getModelById(id: ModelInfo['id']): Model | null;
```

用法:

```js
const model = ssp.getModelById('xxx');
```

> **弃用警告**
>
> 请使用 `getObjectById` 替代

getModelByName

通过 `name` 查找

定义:

```ts
function getModelByName(name: string): Model[];
```

用法:

```js
const modelList = ssp.getModelByName('xxx');
```

> **弃用警告**
>
> 请使用 `getObjectByName` 替代

getAllModel

获取所有 `Model` 对象

定义:

```ts
function getAllModel(): Model[];
```

用法:

getModelByUserDataProperty 👎

通过 `userData` 属性查找

定义：

```ts
function getModelByUserDataProperty(
  propNameOrFindFunc: string | UserDataPropertyFindFunc,
  value?: any
): Model[];
```

用法：

```js
const modelList = ssp.getModelByUserDataProperty('propKey', 'propVal')
// or
const modelList = ssp.getModelByUserDataProperty(item => item['itemPropKey']
```

参数：

propNameOrFindFunc

- **描述:** `userData` 内属性名 或 `find` 函数
- **类型:** string | function
- **必填:** ✔

propValue

- **描述:** `userData` 内属性值。
- **类型:** any
- **必填:** ✗

```js
model.userData = {
  people: {
    name: 'xiaoming',
    age: 18,
  },
};
const modelList = ssp.getModelByUserDataProperty(
  (userData) => userData?.people?.name === 'xiaoming'
);
```

> **弃用警告**
>
> 请使用 `getObjectByUserDataProperty` 替代

removeModelById

通过 `id` 移除

定义：

```ts
function removeModelById(id: ModelInfo['id']): boolean;
```

用法：

```js
ssp.removeModelById('xxx');
```

> **弃用警告**
>
> 请使用 `removeObjectById` 替代

loadModelToGroup

加载 `model` 到一个组内。

定义：

```ts
function loadModelToGroup(
  groupInfo: GroupInfo,
  modelInfoList: ModelInfo[]
): Promise<Group>;
```

用法：

```js
ssp
  .loadModelToGroup(
    // groupInfo
    {
      id: 'firstModelGroup',
      name: 'name_firstModelGroup',
      // ...
    },
    // modelInfoList
    [modelInfo1, modelInfo2, modelInfo3]
  )
  .then((group) => console.log(group));
```

参数

groupInfo

- **描述:** 实例组对象所需信息
- **类型:** GroupInfo
- **必填:** ✓

modelInfoList

- **描述:** `modelInfo` 集合
- **类型:** modelinfo[]
- **必填:** ✓

createGroupForModel 👎

为 `model` 提前创建一个空组。

> **使用场景**
>
> 与 `loadModelToGroup` 不同，有些时候可能你还没有具体的 `modelInfo` 数据，但你想提前创建一个批量管理的空组，当有数据时再使用 addModelForGroup 插入。

定义：

```ts
function createGroupForModel(groupInfo: GroupInfo): Group;
```

用法：

```js
ssp.createGroupForModel({
  id: 'firstModelGroup',
  name: 'name_firstModelGroup',
  // ...
});
```

参数

groupInfo

- **描述:** 实例组对象所需信息
- **类型:** GroupInfo

> **弃用警告**
>
> 请使用 `createGroup` 替代

addModelForGroup

向一个已经存在的组内添加 `model` 对象。

定义:

```ts
function addModelForGroup(
  groupId: GroupInfo['id'],
  modelInfoList: ModelInfo[]
): Promise<Group | null>;
```

用法:

```js
ssp
  .addModelForGroup(
    // groupId
    'firstModelGroup',
    // modelInfoList
    [modelInfo4, modelInfo5],
    // onProgress
    (progress) => console.log('进度信息: ', progress)
  )
  .then((group) => console.log(group));
```

参数

groupId

- **描述:** 组 `id`

modelInfoList

- 描述: `modelInfo` 集合
- 类型: modelinfo[]
- 必填: ✓

getModelGroupById

通过 `id` 查找 `model` 组

定义:

```ts
function getModelGroupById(id: GroupInfo['id']): Group | null;
```

用法:

```js
const group = ssp.getModelGroupById('firstModelGroup');
```

> **弃用警告**
>
> 请使用 `getObjectById` 替代

getModelGroupByName

通过 `name` 查找 `model` 组

定义:

用法:

```js
const groupList = ssp.getModelGroupByName('name_firstModelGroup');
```

> **弃用警告**
>
> 请使用 `getObjectByName` 替代

getAllModelGroup

获取所有 `Model` 对象组

定义:

```ts
function getAllModelGroup(): Group[];
```

用法:

```js
const allModelGroupList = ssp.getAllModelGroup();
```

> **弃用警告**
>
> 请使用 `getAllGroup` 替代

removeModelGroupById

通过 `id` 移除 `model` 组

定义：

```ts
function removeModelGroupById(id: GroupInfo['id']): boolean;
```

用法：

```js
const isRemoveSuccess = ssp.removeModelGroupById('firstModelGroup');
```

> **弃用警告**
>
> 请使用 `removeObjectById` 替代

clearModel

清除当前场景内所有 `model` 对象。

定义：

```ts
function clearModel(): void;
```

用法：

```js
ssp.clearModel();
```

showAllModel

定义：

```ts
function showAllModel(): void;
```

用法：

```js
ssp.showAllModel();
```

hideAllModel

隐藏当前场景内所有 `model` 对象。

定义：

```ts
function hideAllModel(): void;
```

用法：

```js
ssp.hideAllModel();
```

playModelAnimation

播放模型动画。

定义：

```
    animation: AnimationClip,
    index: number,
    animations: AnimationClip[]
  ): boolean;
}

function playModelAnimation(
  model: Model,
  animation: number | AnimationClip | ModelAnimationFindFunc
): AnimationAction | undefined;
```

用法：

```js
const model = ssp.getObjectById('xxxx');

// number
const action = ssp.playModelAnimation(model, 0);
// or AnimationClip
ssp.playModelAnimation(model, model.animations[0]);
// or ModelAnimationFindFunc
ssp.playModelAnimation(model, (itemAnimation) => itemAnimation.name === 'run'
```

> **提示**
>
> 返回的 `action` 对象请参考 three.js AnimationAction

参数：

model

- **描述:** 模型对象
- **类型:** Model
- **必填:** ✓

animation

- **必填:** ✓

stopModelAnimation

停止模型动画。

定义:

```ts
function stopModelAnimation(
  model: Model,
  animation: number | AnimationClip | ModelAnimationFindFunc
): void;
```

用法:

```js
const model = ssp.getObjectById('xxxx');

// number
ssp.stopModelAnimation(model, 0);
// or AnimationClip
ssp.stopModelAnimation(model, model.animations[0]);
// or ModelAnimationFindFunc
ssp.stopModelAnimation(model, (itemAnimation) => itemAnimation.name === 'run'
```

参数:

model

- **描述:** 模型对象
- **类型:** Model
- **必填:** ✓

animation

- **描述:** 动画信息
- **类型:** number | AnimationClip | ModelAnimationFindFunc
- **必填:** ✓

setModelDracoDecoderPath

设置模型的 DRACO 解压库路径

定义:

```ts
function setModelDracoDecoderPath(path: string): void;
```

用法:

```js
ssp.setModelDracoDecoderPath('/draco/');
```

> **提示**
>
> `draco` 目录在 `node_modules/three/examples/jsm/libs/draco` 中
>
> 然后将 `draco` 目录拷贝至所在项目的静态资源目录中，一般是 `public` 目录

computeModelsBoundsTree

计算所有 model 的 BVH

调用此方法可减少控制器、模型事件的计算时间

需要在模型加载完成后调用此方法

SoonSpace.js 2.x

定义：

```ts
type ModelsBoundsTreeOptions = {
  /**
   * block 为阻塞计算，slice 为每帧分片计算，worker 使用 Worker 计算
   */
  type?: 'block' | 'slice' | 'worker';
  force?: boolean;
  frameSliceCount?: number;
  workerCreator?: () => Worker;
};

function computeModelsBoundsTree(
  options?: ModelsBoundsTreeOptions
): Promise<void>;
```

用法：

```js
await ssp.loadModel({
  id: 'model',
  url: 'xxxx',
});

ssp.computeModelsBoundsTree({
  type: 'slice',
  force: false,
  frameSliceCount: 1000,
});
```

参数：

options

- **描述:** 配置选项
- **类型:** ModelsBoundsTreeOptions
- **必填:** ✗

ModelsBoundsTreeOptions

属性	描述	类型	必填	默认值
type	计算的类型	block | slice	✗	slice
force	是否强制重新计算	boolean	✗	false
frameSliceCount	配合 `slice` 使用，每帧的几何结构计算数量	boolean	✗	500
workerCreator	配合 `worker` 使用，请参考下方示例	() => Worker	✗	

Worker 示例

```js
function workerCreator() {
  const worker = new Worker(
    new URL(
      /**
       * 路径填写 generateBVH.worker.js 文件相对于当前文件的位置
       * 可以填写 node_modules 中的相对路径或将文件拷贝至项目的 src 目录中
       */
      './xx/xx/generateBVH.worker.js',
      import.meta.url
    ),
    { type: 'module', name: 'ssp-bvh-worker' }
  );

  return worker;
}

ssp.computeModelsBoundsTree({
  type: 'worker',
  workerCreator,
});
```

文件位于 `node_modules/soonspacejs/dist/generateBVH.worker.js`

> 使用 `workerCreator` 时需要项目的构建工具支持 `new Worker()` 语法
>
> 在 Webpack5、Vite 中无需特殊处理
>
> 如果你使用的是 Webpack4，你可能需要 worker-loader

setTexture

给 mesh 设置纹理贴图，可设置 图片、Cavans 元素、视频 等。

备注：它会恢复原材质中除了贴图之外的所有配置，以保证效果尽可能与原来材质一样。

定义：

```ts
setTexture ( mesh: Mesh, image: string | HTMLImageElement | HTMLCanvasElement
```

用法：

```js
ssp.setTexture(mesh, 'http://xx.com/xx.png');
```

createFindObjectsInBoxNearPosition

创建用于 查找位置附近的 box 空间范围内的物体 的查找器。

当你来查找棱某个位置处 上、下、左、右、前、后 一定距离范围内的某类模型时，可以使用该方法。

定义：

```
 * @param position - 指定查找的位置
 * @returns 所有包含在指定空间的对象
 */
export type FindObjectsNearPosition = ( position: Vector3 ) => Object3D[]

/**
 * 盒子空间
 */
interface BoxSpace {
  top: number;
  bottom: number;
  left: number;
  right: number;
  front: number;
  back: number;
}


createFindObjectsInBoxNearPosition ( boxSpace: BoxSpace, objects: Object3D[]
```

boxSpace

通过上、下、左、右、前、后来描述查找区域。

- 类型： `BoxSpace`

objects

被查找的空间对象。只会从这些对象中进行查找。

- 类型： `Object3D`

用法：

```
//合建一个查找器，需要指定查找的区域 和 从哪些对象中进行查找
const findObjectsNearPosition = ssp.createFindObjectsInBoxNearPosition(
  {
```

```
    left: 30,
    right: 40,
    front: 50,
    back: 60,
  },
  objArr
);

// 使用查找器查找指定位置附近区域的对象
const nearObjs = findObjectsNearPosition({ x: 100, y: 100, z: 100 });
```

createFindObjectsInSphereNearPosition

创建用于 查找位置附近的球形区域范围内的物体 的查找器

当你来查找某个位置处指定半径范围内的某类模型时，可以使用该方法。

定义：

```ts
/**
 * 位置附近的对象查找器
 * @param position - 指定查找的位置
 * @returns 所有包含在指定空间的对象
 */
export type FindObjectsNearPosition = ( position: Vector3 ) => Object3D[]

/**
 * 创建用于 查找位置附近的球形区域范围内的物体 的查找器
 *
 * @param radius - 描述查找半径
 * @param objects - 所有需要被查找的对象
 * @returns 用于查找的函数
 */
createFindObjectsInSphereNearPosition ( radius: number, objects: Object3D[] )
```

radius

查找区域的半径

- 类型： `number`

objects

被查找的空间对象。只会从这些对象中进行查找。

- 类型： `Object3D`

用法：

```js
//合建一个查找器，需要指定查找的区域 和 从哪些对象中进行查找
const findObjectsNearPosition = ssp.createFindObjectsInSphereNearPosition(
  20,
  objArr
);

// 使用查找器查找指定位置附近区域的对象
const nearObjs = findObjectsNearPosition({ x: 100, y: 100, z: 100 });
```

createFindObjectsNearPath

创建用于 查找路径附近的范围内的物体 的查找器

当你来查找路径周围指定半径范围内的某类模型时，可以使用该方法。

定义：

```ts
/**
 * 查找附近的对象
 * @param objs - 所有需要被查找的对象
```

```
/**
 *  创建用于 查找路径附近的范围内的物体 的查找器
 *
 * @param points - 描述路径的点
 * @param radius - 描述查找半径
 * @returns 用于查找的函数
 */
createFindObjectsNearPath ( points: Vector3[], radius: number ): FindNearbyOb
```

points

组成路径的顶点

- 类型： `Vector3[]`

radius

路径附近的查找区域的半径

- 类型： `number`

用法：

```
//合建一个查找器，需要指定查找的区域 和 和被查找的路径
const findNearbyObjects = ssp.createFindObjectsNearPath(
  [
    { x: 0, y: 0, z: 0 },
    { x: 10, y: 0, z: 0 },
    { x: 10, y: 10, z: 0 },
    { x: 10, y: 10, z: 10 },
  ],
  20
);
```

createPathAnimation

创建路径动画

定义：

ts

```ts
/**
 * 路径动画选项
 */
interface PathAnimationOptions {

  /**
   * 移动速度
   */
  speed?: number;

  /**
   * 反向播放
   */
  reverse?: boolean;

  /**
   * 是否需要旋转
   */
  needsRotate?: boolean;

  /**
   * 位置更新回调
   * @remarks
   * 每当目标位置有更新时会触发
   */
  onUpdate?: ( position: Vector3, tween: Tween<Vector3> ) => void;

  /**
   * 动画开始时回调
   */
  onStart?: ( tween: Tween<Vector3> ) => void;
```

```
     * 每段 tween 动画开始时回调
     * @param tween
     */
    onEveryStart?: ( tween: Tween<Vector3> ) => void;

    /**
     * 当到达一个点时回调
     */
    onPoint?: ( index: number, point: Vector3 ) => void;
}


/**
 * 创建路径动画
 * @param target - 被动画的目标对象
 * @param points - 路径的点列表
 * @param options - 选项
 * @returns
 */
createPathAnimation ( target: Object3D, points: Vector3[], options?: PathAnim
```

target

被动画的目标对象

- 类型： `Object3D`

points

路径的点列表

- 类型： `Vector3[]`

用法：

```js
//创建路径动画对象
const pathAnimation = ssp.createPathAnimation(
  model,
```

```
      { x: 10, y: 0, z: 0 },
      { x: 10, y: 10, z: 0 },
      { x: 10, y: 10, z: 10 },
    ],
    {
      speed: 10,
    }
);

// 播放动画
pathAnimation.play();

// 暂停动画
pathAnimation.pause();
```

createTopologyAnimation

创建沿拓扑路径运动的动画

样例：

定义：

ts

```ts
/**
 * 路径动画选项
 */
interface PathAnimationOptions {

  /**
   * 移动速度
   */
  speed?: number;

  /**
   * 反向播放
   */
  reverse?: boolean;

  /**
   * 是否需要旋转
   */
  needsRotate?: boolean;

  /**
   * 位置更新回调
   * @remarks
   * 每当目标位置有更新时会触发
   */
  onUpdate?: ( position: Vector3, tween: Tween<Vector3> ) => void;

  /**
   * 动画开始时回调
   */
  onStart?: ( tween: Tween<Vector3> ) => void;

  /**
   * 每段 tween 动画开始时回调
   * @param tween
   */
  onEveryStart?: ( tween: Tween<Vector3> ) => void;

  /**
```

```
  onPoint?: ( index: number, point: Vector3 ) => void;
}


  /**
   * 创建沿拓扑路径动画的动画
   * @param target - 被动画的目标对象
   * @param topology - 路径点列表
   * @param options - 选项
   * @returns
   */
createTopologyAnimation ( target: Object3D, topology: Topology, options?: Pat
```

target

被动画的目标对象

- 类型： `Object3D`

topology

拓扑路径

- 类型： `Topology`

用法：

```js
//创建沿拓扑路径动画的动画
const pathAnimation = ssp.createTopologyAnimation(model, topology, {
  speed: 10,
});

/**
 * 所有的 options 可以直接修改
 */
pathAnimation.speed = 1;
pathAnimation.reverse = false;
```

```
    .play()
    .then(() => {
      console.log('动画结束');
    })
    .catch(() => {
      console.log('stop方法被调用');
    });

// 暂停动画
pathAnimation.pause();

// 恢复动画
pathAnimation.resume();

// 停止动画
pathAnimation.stop();
```

clearIdb

清空本地 indexedDB 模型缓存数据

定义:

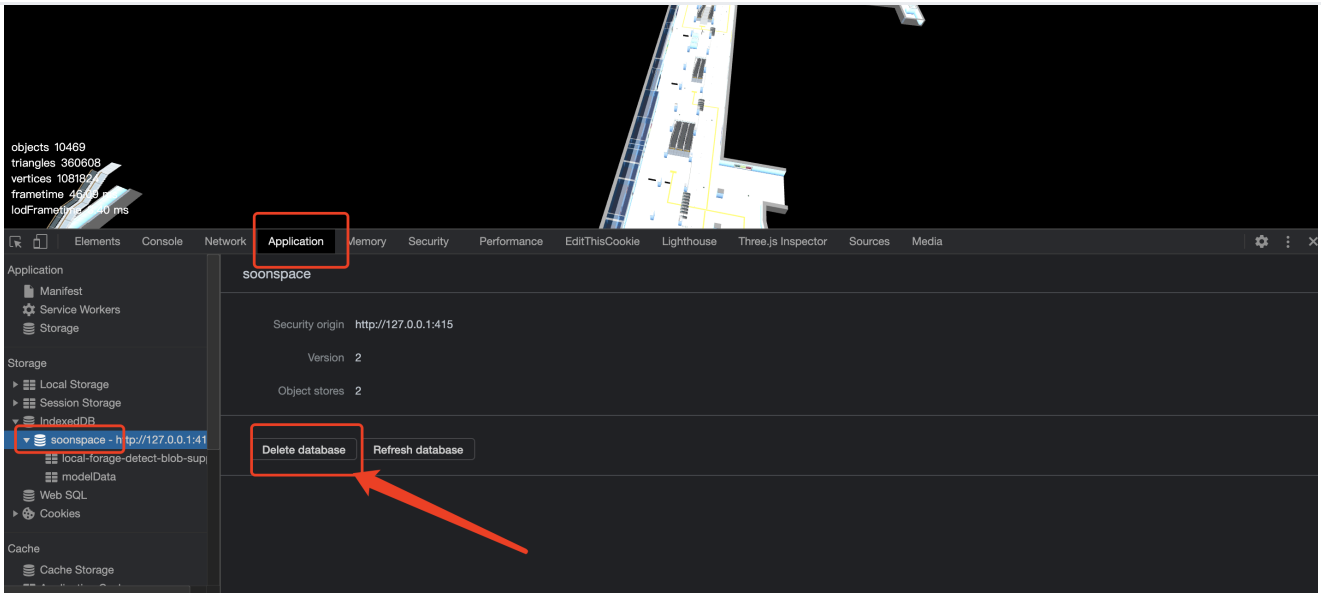
```ts
function clearIdb(): Promise<void>;
```

用法:

```js
ssp.clearIdb().then(() => {
  console.log('本地数据已清空!!!');
});
```

手动清除 indexedDB 缓存

SoonSpace.js 2.x



← Sbm 模型

Poi 对象 →