WTScada 组态软件 Javascript API 手册

ht.List 类

```
ht.List = function (){};
/**
 * 增加元素
 * @param {Object} item 新元素
 * @param {Number} [index] 插入索引
 */
add = function (item, index){};
/**
 * 将一批元素加入到当前集合中
 * @param {Array|ht.List} array 元素数组或集合
 */
addAll = function (array){};
/**
 * 清空集合
 */
clear = function (){};
/**
 * 判断当前集合是否包含参数元素
 * @param {Object} item 是否包含此元素
 */
contains = function (item){};
/**
 * 提供一个回调函数遍历此集合
 * @param {Function} func 遍历函数
 * @param {Object} [scope] 函数域
 * @example list.each(function(item) {
 *   console.log(item);
 * });
 */
each = function (func, scope){};
/**
 * 返回索引位置的的元素
 * @param {Number} index 索引
 * @return {Object} 处于索引位置的元素
 */
get = function (index){};
/**
 * 获取类声明(构造函数)
```

```
 * @return {Function} 类声明(构造函数)
 */
getClass = function (){};

/**
 * 获取类全名
 * @return {String} 类全名
 */
getClassName = function (){};

/**
 * 获取父类声明(构造函数)
 * @return {Function} 父类声明(构造函数)
 */
getSuperClass = function (){};
/**
 * 获得参数元素的索引
 * @param {Object} item 元素
 * @return {Number} 元素的索引
 */
indexOf = function (item){};
/**
 * 判断集合是否为空
 * @return {Boolean} 集合是否为空
 */
isEmpty = function (){};
/**
 * 将参数元素从集合中删除
 * @param {Object} item 要删除的元素
 * @return {Number} 要删除的元素的索引
 */
remove = function (item){};
/**
 * 删除索引位置的元素
 * @param {Number} index 要删除的索引
 * @return {Ojbect} 删除的元素
 */
removeAt = function (index){};
/**
 * 将集合中的元素顺序倒序排序
 */
reverse = function (){};
/**
 * 提供一个回调函数倒序遍历此集合
```

```
 * @param {Function} func 遍历函数
 * @param {Object} [scope] 函数域
 * @example list.reverseEach(function(item) {
 *   console.log(item);
 * });
 */
reverseEach = function (){};
/**
 * 设置索引处的元素
 * @param {Number} index 索引，如果此索引处存在元素则将其替换
 * @param {Object} item 新元素
 */
set = function (index, item){};
/**
 * 获取集合中的元素数
 * @return {Number} 集合中的元素数
 */
size = function (){};
/**
 * 提取集合中的部分元素组成一个新集合并返回
 * @param {Number} start 开始索引(包含)
 * @param {Number} end 结束索引(不包含)
 * @return {ht.List} 新集合
 */
slice = function (start, end){};
/**
 * 根据参数函数将元素排序
 * @param {Function} sortFunc 排序函数
 * @example list.sort(function(item1, item2) {
 *      return item1.age > item2.age;
 * });
 * @return {ht.List} 自身
 */
sort = function (sortFunc){};
/**
 * 以 matchFunc 为过滤函数构建新的元素数组
 * @param {Function} [matchFunc] 过滤函数
 * @param {Object} [scope] 函数域
 * @example var array = list.toArray(function(item) {
 *    if (item.a('visible')) {
 *       return true;
 *    }
 * });
 *
```

```
 * @return {Array} 元素数组
 */
toArray = function (mathFunc, scope){};

/**
 * 以 matchFunc 为过滤函数构建新的元素集合
 * @param {Function} [matchFunc] 过滤函数
 * @param {Object} [scope] 函数域
 * @example var list = list.toList(function(item) {
 *    if (item.a('visible')) {
 *       return true;
 *    }
 * });
 *
 * @return {ht.List} 元素集合
 */
toList = function (mathFunc, scope){};

/**
 * 重写 js 默认的 toString
 * @return {String}
 */
toString = function (){};
```

ht.Notifier 类

```
/**
 * 事件派发器
 * @constructor
 */
/**
 * 增加监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
add = function (listener, scope, ahead){};

/**
 * 是否包含此监听器函数
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
```

```
contains = function (listener, scope){};

/**
 * 派发事件，依次调用所有的监听器函数
 * @param {Object} event 事件对象
 */
fire = function (event){};

/**
 * 删除监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
remove = function (listener, scope){};

/**
 * 数据容器 ht.DataModel 作为承载 Data 数据的模型，
 * 管理着 Data 数据的增删以及变化事件派发，
 * HT 框架所有组件都是通过绑定 DataModel，以不同的形式呈现到用户界面；
 * 同时组件也会监听 DataModel 模型的变化事件， 实时同步更新界面数据信息，
 * 掌握了 DataModel 的操作就掌握了所有组件的模型驱动方式。
 * @constructor
 */
```

ht.DataModel 类

```
/**
 * 获取或设置 attr 属性，仅有一个参数时相当于{@link ht.DataModel#getAttr getAttr}，有两个参数时相当于{@link ht.DataModel#setAttr setAttr}
 * @param {String} name 属性名
 * @param {Object} [value] 属性值
 * @returns {Object}
 */
a = function (name, value){};

/**
 * 增加数据元素
 * @param {ht.Data} data 数据元素
 * @param {Number} [index] 插入索引
 */
add = function (data, index){};

/**
 * 增加数据模型增删变化事件监听器
```

```
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @example dataModel.addDataModelChangeListener(function(event) {
 *      //event 格式：
 *      {
 *       kind: "add"|"remove"|"clear",//事件类型
 *       data: data//事件相关 data
 *      }
 * });
 * @see {@link ht.DataModel#mm mm}
 */
addDataModelChangeListener = function (listener, scope, ahead){};

/**
 * 增加数据模型增删变化事件监听器，{@link ht.DataModel#addDataModelChangeListener
addDataModelChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @example dataModel.mm(function(event) {
 *      //event 格式：
 *      {
 *       kind: "add"|"remove"|"clear",//事件类型
 *       data: data//事件相关 data
 *      }
 * });
 * @see {@link ht.DataModel#addDataModelChangeListener addDataModelChangeListener}
 */
mm = function (listener, scope, ahead){};

/**
 * 增加模型中 Data 元素属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @example dataModel.addDataPropertyChangeListener(function(event) {
 *      //event 格式：
 *      {
 *       property: "name",//发生变化的属性
 *       data: data,//属性发生变化的 data
 *       oldValue: 0,//旧值
 *       newValue: 1//新值
 *      }
```

```
 * });
 * @see {@link ht.DataModel#md md}
 */
addDataPropertyChangeListener = function (listener, scope, ahead){};

/**
 * 增加模型中 Data 元素属性变化事件监听器，{@link
ht.DataModel#addDataPropertyChangeListener addDataPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @example dataModel.md(function(event) {
 *      //event 格式：
 *      {
 *       property: "name",//发生变化的属性
 *       data: data,//属性发生变化的 data
 *       oldValue: 0,//旧值
 *       newValue: 1//新值
 *      }
 * });
 * @see {@link ht.DataModel#addDataPropertyChangeListener addDataPropertyChangeListener}
 */
md = function (listener, scope, ahead){};

/**
 * 增加监听器，监听 Data 在 DataModel 中的层次(用于 TreeView、TreeTableView 等)变化事件
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @example dataModel.addHierarchyChangeListener(function(event) {
 *      //event 格式：
 *      {
 *       data: data,//事件相关 Data
 *       oldIndex: 0,//旧层次
 *       newIndex: 1//新层次
 *      }
 * });
 * @see {@link ht.DataModel#mh mh}
 */
addHierarchyChangeListener = function (listener, scope, ahead){};

/**
 * 增加监听器，监听 Data 在 DataModel 中的层次(用于 TreeView、TreeTableView 等)变化事件，
{@link ht.DataModel#addHierarchyChangeListener addHierarchyChangeListener}的缩写
```

```
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @example dataModel.mh(function(event) {
 *      //event 格式：
 *      {
 *       data: data,//事件相关 Data
 *       oldIndex: 0,//旧层次
 *       newIndex: 1//新层次
 *      }
 * });
 * @see {@link ht.DataModel#addHierarchyChangeListener addHierarchyChangeListener}
 */
mh = function (listener, scope, ahead){};

/**
 * 增加监听器，监听 Data 在 DataModel 中的索引(用于拓扑组件)变化事件
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @example dataModel.addIndexChangeListener(function(event) {
 *      //event 格式：
 *      {
 *       data: data,//事件相关 Data
 *       oldIndex: 0,//旧索引
 *       newIndex: 1//新索引
 *      }
 * });
 */
addIndexChangeListener = function (listener, scope, ahead){};

/**
 * 删除容器中所有 Data 对象，该操作一次性清空，没有逐个 remove 的过程，不会影响 Data 父子关系
 */
clear = function (){};

/**
 * 判断容器是否包含该 data 对象
 * @param {ht.Data} data 要判断的数据元素
 * @return {Boolean} 容器是否包含参数 data
 */
contains = function (data){};
```

```
/**
 * 反序列化数据到数据容器
 * @param {String} json 要被反序列化的 json 字符串
 * @param {ht.Data} rootParent 指定反序列化的数据元素的父元素
 * @param {Boolean} setId 反序列化后的数据元素是否保留原 id
 * @return {ht.List} 被反序列化的数据元素集合
 */
deserialize= function (json, rootParent, setId){};

/**
 * 提供一个回调函数遍历此容器
 * @param {Function} func 遍历函数
 * @param {Object} [scope] 函数域
 * @example dataModel.each(function(data) {
 *   console.log(data);
 * });
 */
each = function (func, scope){};

/**
 * 以 data 为起始深度优先遍历 Data 对象
 * @param {Function} func 遍历函数
 * @param {ht.Data} [data] 遍历起点元素
 * @param {Object} [scope] 函数域
 */
eachByDepthFirst = function (func, data, scope){};

/**
 * 以 data 为起始广度优先遍历 Data 对象
 * @param {Function} func 遍历函数
 * @param {ht.Data} [data] 遍历起点元素
 * @param {Object} [scope] 函数域
 */
eachByBreadthFirst = function (func, data, scope){};

/**
 * 获取 attr 属性
 * @param {String} name 属性名
 * @returns {Object}
 */
getAttr = function (name){};

/**
 * 设置 attr 属性
```

```
 * @param {String} name 属性名
 * @param {Object} value 属性值
 */
setAttr = function (name, value){};

/**
 * 获取 attr 属性对象，该属性默认为空，用于存储用户业务信息
 * @return {Object} attr 属性对象
 */
getAttrObject = function (){};

/**
 * 设置 attr 属性对象，该属性默认为空，用于存储用户业务信息
 * @param {Object} attrObject attr 属性对象
 */
setAttrObject = function (attrObject){};

/**
 * 根据 id 快速查找 Data 对象，模型内部维护着一个 id->data 的映射表，因此查找速度比遍历方式快
 * @param {String|Number} id 要查找的 id
 * @return {ht.Data} 查找到的 Data
 */
getDataById = function (id){};

/**
 * 根据 tag 快速查找，模型内部维护着一个 tag->data 的映射表，因此查找速度比遍历方式快
 * @param {String|Number} tag 要查找的 tag
 * @return {ht.Data} 查找到的 Data
 */
getDataByTag = function (tag){};

/**
 * 获取所有添加到容器的 Data 数据集合
 * @return {ht.List}
 */
getDatas = function (){};

/**
 * 获取历史管理器
 * @return {ht.HistoryManager}
 */
getHistoryManager = function (){};
```

```
/**
 * 获取所有 parent 为空的 Data 对象
 * @return {ht.List}
 */
getRoots = function (){};

/**
 * 获取该容器的选择模型
 * @see {@link ht.DataModel#sm sm}
 * @return {ht.SelectionModel}
 */
getSelectionModel = function (){};

/**
 * 此函数返回一个 map，决定序列化时哪些 attr 属性可被序列化，默认所有 attr 对象里的属性都会被序列化
 * @example function(){
 *     var name, map = {};
 *     for (name in this._attrObject) {
 *         map[name] = 1;
 *     }
 *     return map;
 * }
 * @return {Object} 需要被序列化的 attr 属性 map
 */
getSerializableAttrs = function(){};

/**
 * 获取和 data 同父子层次的兄弟数组，如果 data 父亲为空，则返回 dataModel.getRoots()
 * @param {ht.Data} data 目标 data
 * @return {ht.List}
 */
getSiblings = function(data){};

/**
 * 是否自动调整 data 在容器中索引顺序
 * @return {Boolean}
 */
isAutoAdjustIndex = function(){};

/**
 * 判断容器是否为空
 * @return {Boolean}
 */
```

```
isEmpty = function(){};

/**
 * 移动 data 到同层兄弟数组中的下一个位置
 * @param {ht.Data} data 要移动的数据元素
 */
moveDown = function(data){};

/**
 * 移动当前选中的数据元素到同层兄弟数组中的下一个位置
 * @param {ht.SelectionModel} [sm] 要操作的选中模型，如果为空，使用 dataModel 自身绑定的选中模型
 */
moveSelectionDown = function(sm){};

/**
 * 移动当前选中的数据元素到同层兄弟数组的底部
 * @param {ht.SelectionModel} [sm] 要操作的选中模型，如果为空，使用 dataModel 自身绑定的选中模型
 */
moveSelectionToBottom = function(sm){};

/**
 * 移动当前选中的数据元素到同层兄弟数组的顶部
 * @param {ht.SelectionModel} [sm] 要操作的选中模型，如果为空，使用 dataModel 自身绑定的选中模型
 */
moveSelectionToTop = function(sm){};

/**
 * 移动当前选中的数据元素到同层兄弟数组中的上一个位置
 * @param {ht.SelectionModel} [sm] 要操作的选中模型，如果为空，使用 dataModel 自身绑定的选中模型
 */
moveSelectionUp = function(sm){};

/**
 * 移动数据元素到同层兄弟数组中的指定索引
 * @param {ht.Data} data 要移动的数据元素
 * @param {Number} newIndex 目标索引
 */
moveTo = function(data, newIndex){};

/**
```

```
 * 移动数据元素到同层兄弟数组的底部
 * @param {ht.Data} data 要移动的数据元素
 */
moveToBottom = function(data){};

/**
 * 移动数据元素到同层兄弟数组的顶部
 * @param {ht.Data} data 要移动的数据元素
 */
moveToTop = function(data){};

/**
 * 移动数据元素到同层兄弟数组中的上一个位置
 * @param {ht.Data} data 要移动的数据元素
 */
moveUp = function(data){};

/**
 * 数据元素添加的回调函数，可重载做后续处理
 * @param {ht.Data} data 新添加的数据元素
 */
onAdded = function(data){};

/**
 * 数据元素属性变化回调函数，可重载做后续处理
 * @param {ht.Data} data 发生变化的数据元素
 * @param {Object} e 事件信息
 */
onDataPropertyChanged = function(data, e){};

/**
 * 数据元素删除时回调函数，可重载做后续处理
 * @param {ht.Data} data 被删除的数据元素
 */
onRemoved = function(data){};

/**
 * 删除数据元素，该操作有以下副作用：
 * <ul>
 * <li>其子孙被递归从 DataModel 中删除</li>
 * <li>被断开父子关系 data.setParent(null)</li>
 * <li>Edge 类型通过 edge.setSource(null)和 data.setTarget(null)断开节点关系</li>
 * <li>Node 类型会将其关联的连线从 DataModel 中删除</li>
 * <li>Node 类型通过 data.setHost(null)断开与宿主吸附节点关系</li>
```

```
 * </ul>
 * @param {ht.Data} data 要删除的数据元素
 */
remove = function(data){};

/**
 * 通过 id 删除数据元素
 * @param {Number} id 要删除的数据元素 id
 * @see {@link ht.DataModel#remove remove}
 */
removeDataById = function(id){};

/**
 * 通过 tag 删除数据元素
 * @param {String} tag 要删除的数据元素 tag
 * @see {@link ht.DataModel#remove remove}
 */
removeDataByTag = function(tag){};

/**
 * 删除数据模型增删变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.DataModel#umm umm}
 */
removeDataModelChangeListener = function(listener, scope){};

/**
 * 删除数据模型增删变化事件监听器，{@link ht.DataModel#removeDataModelChangeListener removeDataModelChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.DataModel#removeDataModelChangeListener removeDataModelChangeListener}
 */
umm = function(listener, scope){};

/**
 * 删除模型中 Data 元素属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.DataModel#umd umd}
 */
removeDataPropertyChangeListener = function(listener, scope){};
```

```
/**
 * 删除模型中 Data 元素属性变化事件监听器，{@link
ht.DataModel#removeDataPropertyChangeListener removeDataPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.DataModel#removeDataPropertyChangeListener
removeDataPropertyChangeListener}
 */
umd = function(listener, scope){};

/**
 * 删除监听 Data 在 DataModel 中的层次(用于 TreeView、TreeTableView 等)变化事件的监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.DataModel#umh umh}
 */
removeHierarchyChangeListener = function(listener, scope){};

/**
 * 删除监听 Data 在 DataModel 中的层次(用于 TreeView、TreeTableView 等)变化事件的监听器，
{@link ht.DataModel#removeHierarchyChangeListener removeHierarchyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.DataModel#removeHierarchyChangeListener removeHierarchyChangeListener}
 */
umh = function(listener, scope){};

/**
 * 删除监听 Data 在 DataModel 中的索引(用于拓扑组件)变化事件的监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeIndexChangeListener = function(listener, scope){};

/**
 * 将 data 在拓扑上置底
 * @param {ht.Data} data 要置底的数据元素
 */
sendToBottom = function(data){};

/**
 * 将 data 在拓扑上置顶
 * @param {ht.Data} data 要置顶的数据元素
 */
```

```
sendToTop = function(data){};

/**
 * 将数据模型序列化成 JSON 格式字符串
 * @param {Number} space 缩进空格数
 */
serialize = function(space){};

/**
 * 设置是否自动调整 data 在容器中索引顺序
 * @param {Boolean} autoAdjustIndex 是否自动调整 data 在容器中索引顺序
 */
setAutoAdjustIndex = function(autoAdjustIndex){};

/**
 * 返回当前容器中 Data 对象的总数
 * @return {Number}
 */
size = function(){};

/**
 * 获取该容器的选择模型
 * @see {@link ht.DataModel#getSelectionModel getSelectionModel}
 * @return {ht.SelectionModel}
 */
sm = function(){};

/**
 * 以 matchFunc 为过滤函数构建新的元素集合并返回
 * @param {Function} [matchFunc] 过滤函数
 * @param {Object} [scope] 函数域
 * @return {ht.List} 元素集合
 */
toDatas = function(matchFunc, scope){};

/**
 * 将数据模型序列化成 JSON 格式对象
 * @return {Object} JSON 对象
 */
toJSON = function(){};
```

```
/**
 * 选择模型管理 DataModel 模型中 Data 对象的选择状态，
 * 每个 DataModel 对象都内置一个 SelectionModel 选择模型，
 * 控制这个 SelectionModel 即可控制所有绑定该 DataModel 的视图组件的对象选择状态，
 * 这意味着共享同一 DataModel 的组件默认就具有选中联动功能。<br>
 * 如果希望某些组件不与其他组件选中联动，
 * 可通过调用 view.setSelectionModelShared(false)，
 * 这样该 view 将创建一个专属的 SelectionModel 实例。
 * @constructor
 */
```

SelectionModel

```
/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.SelectionModel#mp mp}
 */
addPropertyChangeListener = function (listener, scope, ahead){};

/**
 * 增加自身属性变化事件监听器，{@link ht.SelectionModel#addPropertyChangeListener addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.SelectionModel#addPropertyChangeListener addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};

/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.SelectionModel#ump ump}
 */
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除自身属性变化事件监听器，{@link ht.SelectionModel#removePropertyChangeListener removePropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
```

```
 * @see {@link ht.SelectionModel#removePropertyChangeListener
removePropertyChangeListener}
 */
ump = function (listener, scope){};


/**
 * 增加监听器，监听选中变化事件
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.SelectionModel#ms ms}
 * @example dataModel.addSelectionChangeListener(function(event) {
 *      //event 格式：
 *      {
 *       kind: "set",//事件类型 set|remove|append|clear
 *       datas: datas,//包含所有选中状态变化的数据元素，之前选中现在取消选中，和之前没选
中现在被选中的数据元素
 *      }
 * });
 */
addSelectionChangeListener = function (listener, scope, ahead){};

/**
 * 增加监听器，监听选中变化事件，{@link ht.SelectionModel#addSelectionChangeListener
addSelectionChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.SelectionModel#addSelectionChangeListener addSelectionChangeListener}
 * @example dataModel.ms(function(event) {
 *      //event 格式：
 *      {
 *       kind: "set",//事件类型 set|remove|append|clear
 *       datas: datas,//包含所有选中状态变化的数据元素，之前选中现在取消选中，和之前没选
中现在被选中的数据元素
 *      }
 * });
 */
ms = function (listener, scope, ahead){};

/**
 * 删除监听选中变化事件的监听器
 * @param {Function} listener 监听器函数
```

```
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.SelectionModel#ums ums}
 */
removeSelectionChangeListener = function (listener, scope){};

/**
 * 删除监听选中变化事件的监听器，{@link ht.SelectionModel#removeSelectionChangeListener removeSelectionChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.SelectionModel#removeSelectionChangeListener removeSelectionChangeListener}
 */
ums = function (listener, scope){};

/**
 * 追加选中一个或多个数据元素，参数可为单个数据元素，也可为 ht.List 或 Array 数组
 * @param {ht.Data|ht.List|Array} datas 数据元素
 * @see {@link ht.SelectionModel#as as}
 */
appendSelection = function (datas){};

/**
 * 追加选中一个或多个数据元素，参数可为单个数据元素，也可为 ht.List 或 Array 数组，{@link ht.SelectionModel#appendSelection appendSelection}的缩写
 * @param {ht.Data|ht.List|Array} datas 数据元素
 * @see {@link ht.SelectionModel#appendSelection appendSelection}
 */
as = function (datas){};

/**
 * 取消所有选中数据元素
 * @see {@link ht.SelectionModel#cs cs}
 */
clearSelection = function (){};

/**
 * 取消所有选中数据元素，{@link ht.SelectionModel#clearSelection clearSelection}的缩写
 * @see {@link ht.SelectionModel#clearSelection clearSelection}
 */
cs = function (){};

/**
 * 判断 data 对象是否被选中
```

```
 * @param {ht.Data} data 要判断的 data 对象
 * @see {@link ht.SelectionModel#co co}
 */
contains = function (data){};

/**
 * 判断 data 对象是否被选中，{@link ht.SelectionModel#contains contains}的缩写
 * @param {ht.Data} data 要判断的 data 对象
 * @see {@link ht.SelectionModel#contains contains}
 */
co = function (data){};

/**
 * 获取{@link ht.DataModel DataModel}，{@link ht.SelectionModel#getDataModel getDataModel}
的缩写
 * @return {ht.DataModel} dataModel
 * @see {@link ht.SelectionModel#getDataModel getDataModel}
 */
dm = function (){};

/**
 * 获取{@link ht.DataModel DataModel}
 * @return {ht.DataModel} dataModel
 * @see {@link ht.SelectionModel#dm dm}
 */
getDataModel = function (){};

/**
 * 提供一个回调函数遍历此选中模型
 * @param {Function} func 遍历函数
 * @param {Object} [scope] 函数域
 * @example selectionModel.each(function(data) {
 *   console.log(data);
 * });
 */
each = function (func, scope){};

/**
 * 返回首个被选中的数据元素，如果没有选中数据元素则返回空
 * @return {ht.Data} 首个被选中的数据元素
 * @see {@link ht.SelectionModel#fd fd}
 */
getFirstData = function (){};
```

```
/**
 * 返回首个被选中的数据元素，如果没有选中数据元素则返回空，此方法是{@link
ht.SelectionModel#getFirstData getFirstData}的缩写
 * @return {ht.Data} 首个被选中的数据元素
 * @see {@link ht.SelectionModel#getFirstData getFirstData}
 */
fd = function (){};

/**
 * 返回最后被选中的数据元素，如果没有选中数据元素则返回空
 * @return {ht.Data} 最后被选中的数据元素
 * @see {@link ht.SelectionModel#ld ld}
 */
getLastData = function (){};

/**
 * 返回最后被选中的数据元素，如果没有选中数据元素则返回空，{@link
ht.SelectionModel#getLastData getLastData}的缩写
 * @return {ht.Data} 最后被选中的数据元素
 * @see {@link ht.SelectionModel#getLastData getLastData}
 */
ld = function (){};

/**
 * 返回选中过滤器函数<br>
 * 有些数据元素不希望被用户选中，可以通过设置此过滤器实现
 * @return {Function} 选中过滤器函数
 * @see {@link ht.SelectionModel#setFilterFunc setFilterFunc}
 */
getFilterFunc = function (){};

/**
 * 设置选中过滤器函数<br>
 * 有些数据元素不希望被用户选中，可以通过设置此过滤器实现
 * @param {Function} func 选中过滤器函数
 * @example //禁止选中 name 为 test 的数据元素
 * selectionModel.setFilterFunc(function(data) {
 *  if (data.getName() === 'test') {
 *   return false;
 *  } else {
 *   return true;
 *  }
 * });
 */
```

```
setFilterFunc = function (func){};

/**
 * 获取所有被选中数据元素集合，注意不可直接对返回的集合进行增删操作，
 * 如果需要增删操作，应使用 toSelection 方法
 * @return {ht.List} 被选中的数据元素集合
 * @see {@link ht.SelectionModel#toSelection toSelection}
 */
getSelection = function (){};

/**
 * 获取所有被选中数据元素集合，注意不可直接对返回的集合进行增删操作，
 * 如果需要增删操作，应使用 toSelection 方法
 * @return {ht.List} 被选中的数据元素集合
 * @see {@link ht.SelectionModel#toSelection toSelection}
 */
toSelection = function (matchFunc, scope){};

/**
 * 以 matchFunc 为过滤函数构建新的元素集合并返回
 * @param {Function} [matchFunc] 过滤函数
 * @param {Object} [scope] 函数域
 * @return {ht.List} 元素集合
 */
setSelection = function (datas){};

/**
 * 设置选中数据元素，参数可为单个数据元素，也可为 ht.List 或 Array 数组，{@link
ht.SelectionModel#setSelection setSelection}的缩写
 * @param {ht.Data|ht.List|Array} datas 数据元素
 */
ss = function (datas){};

/**
 * 获取选中模式
 * @default multiple
 * @return {String} none|single|multiple
 */
getSelectionMode = function (){};

/**
 * 设置选中模式，可选值：
 * <ul>
 * <li>none:不可选中</li>
```

```
 * <li>single:只能选中一个数据元素</li>
 * <li>multiple:可以选中多个数据元素</li>
 * </ul>
 * @default multiple
 * @param {String} mode 选中模式
 */
setSelectionMode = function (mode){};

/**
 * 判断是否有选中的数据元素
 * @return {Boolean}
 */
isEmpty = function(){};

/**
 *
 * 判断数据元素是否可被选中
 * @param {ht.Data} data 要判断的数据元素
 * @return {Boolean}
 */
isSelectable = function(data){};

/**
 * 取消选中数据元素，参数可为单个数据元素，也可为 ht.List 或 Array 数组
 * @param {ht.Data|ht.List|Array} datas 数据元素
 * @see {@link ht.SelectionModel#rs rs}
 */
removeSelection = function(datas){};

/**
 * 取消选中数据元素，参数可为单个数据元素，也可为 ht.List 或 Array 数组，{@link
ht.SelectionModel#removeSelection removeSelection}的缩写
 * @param {ht.Data|ht.List|Array} datas 数据元素
 * @see {@link ht.SelectionModel#removeSelection removeSelection}
 */
rs = function(datas){};

/**
 * 选中 DataModel 中的所有数据元素
 * @see {@link ht.SelectionModel#sa sa}
 */
selectAll = function(){};

/**
```

```
 * 选中 DataModel 中的所有数据元素，{@link ht.SelectionModel#selectAll selectAll}的缩写
 * @see {@link ht.SelectionModel#selectAll selectAll}
 */
sa = function(){};

/**
 * 获取选中模型中数据元素的个数
 * @return {Number}
 */
size = function(){};
```

ht.Data

```
/**
 * 数据元素基类，包含基本属性设置、样式设置、事件派发、父子关系等功能
 * @constructor
 */
ht.Data = function (){};

/**
 * 添加孩子节点，index 为孩子插入索引，为空则插入作为最后的孩子，内部会自动调用 child 的 setParent
 * @param {ht.Data} child 孩子元素
 * @param {Number} [index] 插入索引
 */
addChild = function (child, index){};

/**
 * 增加 icon，icon 参数请参考 beginner guide
 * @param {String} name icon 名
 * @param {Object} icon icon 参数
 * @example
 * data.addStyleIcon("arrow1", {
  *    position: 2,
  *    width: 50,
  *    height: 25,
  *    keepOrien: true,
  *    names: ['arrow']
  *});
 */
addStyleIcon = function (name, icon){};
```

```
/**
 * 删除所有孩子节点，内部会自动调用 setParent
 */
clearChildren = function (){};

/**
 * 获取{@link ht.DataModel DataModel}, {@link ht.Data#getDataModel getDataModel}的缩写
 * @return {ht.DataModel} dataModel
 */
dm = function (dataModel){};

/**
 * 遍历孩子元素
 * @param {Function} func 遍历函数
 * @param {Object} [scope] 函数域
 * @example data.eachChild(function(child) {
 *    console.log(child.getId());
 * });
 *
 */
eachChild = function (func, scope){};

/**
 * 派发属性变化事件
 * @param {String} property 属性名
 * @param {Object} oldValue 旧值
 * @param {Object} newValue 新值
 */
firePropertyChange = function (property, oldValue, newValue){};

/**
 * 派发属性变化事件，同{@link ht.Data#firePropertyChange firePropertyChange}
 * @param {String} property 属性名
 * @param {Object} oldValue 旧值
 * @param {Object} newValue 新值
 */
fp = function (property, oldValue, newValue){};


/**
 * 获取 attr 属性对象，该属性默认为空，用于存储用户业务信息
 * @return {Object} attr 属性对象
 */
```

```
getAttrObject = function (){};

/**
 * 返回指定索引位置的孩子
 * @param {Number} index 索引
 * @return {ht.Data} 索引对应的孩子
 */
getChildAt = function (index){};

/**
 * 获取所有孩子节点
 * @return {ht.List} 孩子元素集合
 */
getChildren = function (){};

/**
 * 获取类声明(构造函数)
 * @return {Function} 类声明(构造函数)
 */
getClass = function (){};

/**
 * 获取类全名，继承 Data 并希望序列化时应该重写此方法返回子类的类名字符串
 * @see {@link ht.Data#getSuperClass}
 * @return {String} 类全名
 */
getClassName = function (){};

/**
 * 获取所属的 DataModel
 * @return {ht.DataModel} DataModel 数据容器
 */
getDataModel = function (){};

/**
 * 获取显示名称，常作为 Column 和 Property 的列头和属性名称显示
 * @return {String} 显示名称
 */
getDisplayName = function (){};

/**
 * 获取小图标名称，常作为 TreeView 和 ListView 等组件上的节点小图标
 * @return {String|Object} 图标名或矢量
 */
```

```
getIcon = function (){};

/**
 * 获取唯一编号
 * @return {Number} 唯一编号
 */
getId = function (){};

/**
 * 获取数据元素在 GraphView 组件中的图层位置
 * @default 0
 * @return {String|Number} 图层名
 */
getLayer = function (){};

/**
 * 获取父元素
 * @return {ht.Data} 父元素
 */
getParent = function (){};

/**
 * 获取数据元素名
 * @return {String} 名称
 */
getName = function (){};

/**
 * 此函数返回一个 map，决定序列化时哪些属性可被序列化，如果有自定义的 get/set 属性并且需要序列化，应该重写此方法
 * @example function(){
 *     return {
 *         name: 1,
 *         displayName: 1,
 *         icon: 1,
 *         toolTip: 1,
 *         parent: 1,
 *         layer: 1,
 *         tag: 1,
 *         adjustChildrenToTop: 1
 *     };
 * }
 * @return {Object} 需要被序列化的属性 map
 */
```

```
getSerializableProperties = function(){};

/**
 * 此函数返回一个 map，决定序列化时哪些 attr 属性可被序列化，默认所有 attr 对象里的属性都会被序列化
 * @example function(){
 *     var name, map = {};
 *     for (name in this._attrObject) {
 *         map[name] = 1;
 *     }
 *     return map;
 * }
 * @return {Object} 需要被序列化的 attr 属性 map
 */
getSerializableAttrs = function(){};

/**
 * 此函数返回一个 map，决定序列化时哪些样式可被序列化，默认所有样式都会被序列化
 * @example function(){
 *     var name, map = {};
 *     for (name in this._styleMap) {
 *         map[name] = 1;
 *     }
 *     return map;
 * }
 * @return {Object} 需要被序列化的样式 map
 */
getSerializableStyles = function(){};

/**
 * 获取或设置 attr 属性，仅有一个参数时相当于{@link ht.Data#getAttr getAttr}，有两个参数时相当于{@link ht.Data#setAttr setAttr}
 * @param {String} name 属性名
 * @param {Object} [value] 属性值
 * @returns {Object}
 */
a = function (name, value){};

/**
 * 获取或设置样式，仅有一个参数时相当于{@link ht.Data#getStyle getStyle}，有两个参数时相当于{@link ht.Data#setStyle setStyle}
 * @param {String} name 样式名
 * @param {Object} [value] 样式值
 * @returns {Object}
```

```
 */
s = function (name, value){};

/**
 * 获取样式属性
 * @param {String} name 样式名
 * @returns {Object}
 */
getStyle = function (name){};
/**
 * 设置样式
 * @param {String} name 样式名
 * @param {Object} value 样式值
 */
setStyle = function (name, value){};

/**
 * 获取 attr 属性
 * @param {String} name 属性名
 * @returns {Object}
 */
getAttr = function (name){};
/**
 * 设置 attr 属性
 * @param {String} name 属性名
 * @param {Object} value 属性值
 */
setAttr = function (name, value){};

/**
 * 获取图元内部样式映射信息
 * @returns {Object} 样式映射表
 */
getStyleMap = function (){};

/**
 * 获取父类声明(构造函数)，继承类时可以用来调用父类构造或函数
 * @example function MyNode() {
 *      this.getSuperClass().call(this);//调用父类构造函数
 * }
 * ht.Default.def(MyNode, ht.Data, {
 *    setName: function(name) {
 *       this.getSuperClass().prototype.setName.call(this, name);//调用父类的 setName 函
数
```

```
 *       this._username = name;
 *    }
 * });
 *
 * @returns {Function} 父类声明(构造函数)
 */
getSuperClass = function (){};

/**
 * 获取标示号，可通过{@link ht.DataModel#getDataByTag getDataByTag}快速查找
 * @returns {String} 标示号
 */
getTag = function (){};

/**
 * 获取文字提示信息
 * @returns {String} 文字提示
 */
getToolTip = function (){};

/**
 * 获取拓扑组件上的 UI 类
 * @returns {Function} UI 类声明(构造函数)
 */
getUIClass = function (){};

/**
 * 判断是否有孩子
 * @returns {Boolean} 是否有孩子
 */
hasChildren = function (){};

/**
 * 强制触发属性变化事件通知界面更新
 */
invalidate = function (){};

/**
 * GraphView 点击图元会自动 sendToTop，该属性决定是否对子图元也进行 sendToTop 操作
 * @return {Boolean} 是否将 children 自动 sendToTop
 */
isAdjustChildrenToTop = function (){};

/**
```

```
 * 判断自身是否为指定 data 的子孙
 * @param {ht.Data} data 要对比的数据元素
 * @return {Boolean} 自身是否为指定 data 的子孙
 */
isDescendantOf = function (data){};

/**
 * 判断是否有孩子，同{@link ht.Data#hasChildren hasChildren}
 * @return {Boolean} 是否有孩子
 */
isEmpty = function (){};

/**
 * 判断自身是否为指定 data 的父亲
 * @param {ht.Data} data 要对比的数据元素
 * @return {Boolean} 自身是否为指定 data 的父亲
 */
isParentOf = function (data){};

/**
 * 判断自身与指定 data 是否有父子或子孙关系
 * @param {ht.Data} data 要对比的数据元素
 * @return {Boolean} 自身与指定 data 是否有父子或子孙关系
 */
isRelatedTo = function (data){};

/**
 * 强制触发属性变化事件通知界面更新，{@link ht.Data#invalidate invalidate}的缩写
 */
iv = function (){};

/**
 * 添加孩子时的回调函数，可重载做后续处理
 * @param {ht.Data} child 新增加的孩子元素
 * @param {Number} index 索引
 */
onChildAdded = function (child, index){};

/**
 * 删除孩子时的回调函数，可重载做后续处理
 * @param {ht.Data} child 被删除的孩子元素
 * @param {Number} index 索引
 */
```

```
onChildRemoved = function (child, index){};

/**
 * 改变父亲元素时的回调函数，可重载做后续处理
 * @param {ht.Data} oldParent 旧的父元素
 * @param {ht.Data} parent 新的父元素
 */
onParentChanged = function (oldParent, parent){};

/**
 * 属性变化回调函数，可重载做后续处理
 * @param {Object} event 属性变化事件
 * @example //event 格式：
 * {
 *   property: 'name',//发生变化的属性
 *   oldValue: 'oldValue',//旧值
 *   newValue: 'newValue',''新值
 *   data: data//发生变化的 data
 * }
 *
 */
onPropertyChanged = function (event){};

/**
 * 样式属性变化时会回调该函数，可重载做后续处理
 * @param {String} name 样式名
 * @param {Object} oldValue 旧的样式值
 * @param {Object} newValue 新的样式值
 */
onStyleChanged = function (name, oldValue, newValue){};

/**
 * 删除指定孩子元素，内部会自动调用孩子元素的 setParent
 * @param {ht.Data} child 要删除的孩子元素
 */
removeChild = function (child){};

/**
 * 删除 icon
 * @param {String} name 要删除的 icon 名
 */
removeStyleIcon = function (name){};

/**
```

```
 * GraphView 点击图元会自动 sendToTop，该属性决定是否对子图元也进行 sendToTop 操作
 * @param {Boolean} adjustToTop 是否将 children 自动 sendToTop
 */
setAdjustChildrenToTop = function (adjustToTop){};

/**
 * 设置 attr 属性对象，该属性默认为空，用于存储用户业务信息
 * @param {Object} attrObject attr 属性对象
 */
setAttrObject = function (attrObject){};

/**
 * 设置显示名称，常作为 Column 和 Property 的列头和属性名称显示
 * @param {String}  displayName 显示名称
 */
setDisplayName = function (displayName){};

/**
 * 设置小图标名称，常作为 TreeView 和 ListView 等组件上的节点小图标
 * @param {String|Object} icon 图标名或矢量
 */
setIcon = function (icon){};

/**
 * 设置唯一编号，如果手工设置，一定要确保在 data 加入到 DataModel 之前设置并且唯一不重复
 * @param {String|Number} id 唯一编号
 */
setId = function (id){};

/**
 * 设置数据元素在 GraphView 组件中的图层位置
 * @param {String|Number} layer 图层名
 */
setLayer = function (layer){};

/**
 * 设置数据元素名称
 * @param {String} name 数据元素名称
 */
setName = function (name){};

/**
 * 设置父元素
 * @param {ht.Data} parent 父元素
```

```
 */
setParent = function (parent){};

/**
 * 设置标示号，可通过{@link ht.DataModel#getDataByTag getDataByTag}快速查找
 * @param {String} tag 标示号
 */
setTag = function (tag){};

/**
 * 设置文字提示信息
 * @param {String} toolTip 文字提示
 */
setToolTip = function (toolTip){};

/**
 * 获取孩子元素总数
 * @return {Number} 孩子元素总数
 */
size = function (){};

/**
 * 以 matchFunc 为过滤函数构建新的孩子元素集合
 * @param {Function} [matchFunc] 过滤函数
 * @param {Object} [scope] 函数域
 * @example var list = data.toChildren(function(child) {
 *    if (child.a('visible')) {
 *       return true;
 *    }
 * });
 *
 * @return {ht.List} 孩子元素集合
 */
toChildren = function (matchFunc, scope){};

/**
 * 返回值作为 TreeView 和 GraphView 等组件上的图元文字标签，默认返回 displayName||name 信息
 * @return {String} 文字标签
 */
toLabel = function (){};

/**
 * 重写 js 默认的 toString
 * @return {String}
```

```
 */
toString = function (){};
```

ht.Node

```
/**
 * 拓扑图元类型
 * @constructor
 * @extends ht.Data
 */
ht.Node = function() {};

/**
 * 获取当前图元代理的连线集合
 * @return {ht.List}
 */
getAgentEdges = function(){};

/**
 * 获取吸附到自身的所有图元
 * @return {ht.List}
 */
getAttaches = function(){};

/**
 * 获取图元四个角的实时坐标(包括旋转后的坐标)
 * @param {Number} xPadding 水平方向 padding
 * @param {Number} yPadding 垂直方向 padding
 * @example //返回值示例：
 * [
 * {x: 0, y: 0},//左上
 * {x: 100, y: 0},//右上
 * {x: 100, y: 100},//右下
 * {x: 0, y: 100}//左下
 * ]
 * @return {Array} 四个角坐标，顺序为左上，右上，右下，左下
 */
getCorners = function(xPadding, yPadding){};
/**
 * 获取图元中心在 3D 坐标系中的 y 坐标
 * @return {Number}
 */
getElevation = function(){};
```

```
/**
 * 获取所有跟图元关联的连线(不包括代理的连线)
 * @return {ht.List}
 */
getEdges = function(){};

/**
 * 获取图元在 2D 拓扑中的高度，或 3D 拓扑中的 z 轴长度
 * @return {Number}
 */
getHeight = function(){};

/**
 * 获取宿主图元，当图元吸附上宿主图元时，宿主移动或旋转时会带动所有吸附者
 * @return {ht.Data}
 */
getHost = function(){};

/**
 * 获取拓扑上展现的图片信息，在 GraphView 拓扑图中图片一般以 position 为中心绘制
 * @return {ht.Data}
 */
getImage = function(){};

/**
 * 获取所有跟节点关联的自环连线
 * @return {ht.List}
 */
getLoopedEdges = function(){};

/**
 * 获取图元中心点坐标
 * @example //返回值示例：
 * {
 *   x: 0,
 *   y: 0
 * }
 *
 * @return {Object}
 */
getPosition = function(){};

/**
 * 获取图元中心点在 3D 拓扑中的三维坐标
```

```
 * @return {Array} 三维坐标数组，格式为[x, y, z]
 */
getPosition3d = function(){};

/**
 * 获取图元的矩形区域(包括旋转)
 * @example //返回值示例：
 * {x: 0, y: 0, width: 100, height: 100}
 * @return {Object} 矩形区域
 */
getRect = function(){};

/**
 * 获取图元的旋转角度，围绕中心点顺时针旋转
 * @return {Number} 旋转角度(弧度制)
 */
getRotation = function(){};

/**
 * 获取图元在 3D 拓扑中的三维旋转角度
 *@return {Array} 三维旋转角度(弧度制)，格式为[x, y, z]，即[getRotationX(), -getRotation(), getRotationZ()]
 */
getRotation3d = function(){};

/**
 * 返回三维旋转模式<br>
 * 图元在 3D 拓扑中旋转时，先沿 x 轴旋转，再沿 y 轴旋转和先沿 y 轴旋转，再沿 x 轴旋转最后的结果是不一样的
 * @return {String} 三维旋转模式，xyz|xzy|yxz|yzx|zxy|zyx
 * @see {@link ht.Node#setRotationMode setRotationMode}
 */
getRotationMode = function(){};

/**
 * 获取图元在 3d 拓扑中沿 x 轴的旋转角度(弧度制)
 * @return {Number}
 */
getRotationX = function(){};

/**
 * 获取图元在 3d 拓扑中沿 y 轴的旋转角度(弧度制)
 * @return {Number}
 */
```

```
getRotationY = function(){};

/**
 * 获取图元在 3d 拓扑中沿 z 轴的旋转角度(弧度制)
 * @return {Number}
 */
getRotationZ = function(){};

/**
 * 获取图元在 2D 拓扑中的尺寸(宽高)
 * @example //返回值示例：
 * {
 *  with: 100,
 *  height: 100
 * }
 * @return {Object}
 */
getSize = function(){};

/**
 * 获取图元在 3D 拓扑中的尺寸(长宽高)
 * @return {Array} 格式为[x, y, z]，即[getWidth(), getTall(), getHeight()]
 */
getSize3d = function(){};

/**
 * 获取代理的起始于该图元的连线
 * @return {ht.List}
 */
getSourceAgentEdges = function(){};

/**
 * 获取跟图元关联的并起始于该图元的连线(不包括代理的连线)
 * @return {ht.List}
 */
getSourceEdges = function(){};

/**
 * 获取图元在 3D 拓扑中的 y 轴长度
 * @return {Number}
 */
getTall = function(){};

/**
```

```
 * 获取图元代理的结束于该图元的连线
 * @return {ht.List}
 */
getTargetAgentEdges = function(){};

/**
 * 获取跟图元关联的并结束于该图元的连线(不包括代理的连线)
 * @return {ht.List}
 */
getTargetEdges = function(){};

/**
 * 获取图元在 2D 拓扑中的宽度，或在 3D 拓扑中 x 轴的长度
 * @return {Number}
 */
getWidth = function(){};

/**
 * 当吸附宿主对象属性发生变化时回调该函数，可重载做后续处理
 * @param {Event} event 事件对象
 */
handleHostPropertyChange = function(event){};

/**
 * 判断当前图元上是否有代理连线
 * @return {Boolean}
 */
hasAgentEdges = function(){};

/**
 * 判断当前图元是否吸附到指定图元对象上
 * @param {ht.Node} node 指定的图元
 * @return {Boolean}
 */
isHostOn = function(node){};

/**
 * 当吸附的宿主对象发生变化时回调该函数，可重载做后续处理
 * @param {ht.Node} oldHost 旧的宿主
 * @param {ht.Node} newHost 新的宿主
 */
onHostChanged = function(oldHost, newHost){};

/**
```

```
 * 获取或设置设置图元中心点坐标，有两个参数时相当于{@link ht.Node#setPosition
setPosition}，没有参数时相当于{@link ht.Node#getPosition getPosition}
 * @param {Number} [x] x 坐标
 * @param {Number} [y] y 坐标
 * @see {@link ht.Node#setPosition setPosition}
 * @see {@link ht.Node#getPosition getPosition}
 * @return {Object} 坐标值，格式为:{x: x, y: y}
 */
p = function(x, y){};

/**
 * 获取或设置图元中心点在 3D 拓扑中的三维坐标，有三个参数时相当于{@link
ht.Node#setPosition3d setPosition3d}，没有参数时相当于{@link ht.Node#getPosition3d
getPosition3d}
 * @param {Number} [x] x 坐标
 * @param {Number} [y] y 坐标
 * @param {Number} [z] z 坐标
 * @see {@link ht.Node#setPosition3d setPosition3d}
 * @see {@link ht.Node#getPosition3d getPosition3d}
 * @return {Array} 三维坐标数组，格式为[x, y, z]
 */
p3 = function(x, y, z){};

/**
 * 获取或设置图元在 3D 拓扑中的三维旋转角度，有三个参数时相当于{@link
ht.Node#setRotation3d setRotation3d}，没有参数时相当于{@link ht.Node#getRotation3d
getRotation3d}
 * @param {Number} [rotationX] 沿 x 轴的旋转角度(弧度制)
 * @param {Number} [rotationY] 沿 y 轴的旋转角度(弧度制)
 * @param {Number} [rotationZ] 沿 z 轴的旋转角度(弧度制)
 * @see {@link ht.Node#setRotation3d setRotation3d}
 * @see {@link ht.Node#getRotation3d getRotation3d}
 *@return {Array} 三维旋转角度(弧度制)，格式为[x, y, z]，即[getRotationX(), -getRotation(),
getRotationZ()]
 */
r3 = function(rotationX, rotationY, rotationZ){};

/**
 * 以指定的坐标为中心旋转图元
 * @param {Number} x 指定 x 坐标
 * @param {Number} y 指定 y 坐标
 * @param {Number} angle 旋转角度(弧度制)
 */
```

```
rotateAt = function(x, y, angle){};

/**
 * 获取或设置图元在 3D 拓扑中的尺寸，有三个参数时相当于{@link ht.Node#setSize3d setSize3d}，没有参数时相当于{@link ht.Node#getSize3d getSize3d}
 * @param {Number} width x 轴方向的长度
 * @param {Number} tall y 轴方向的长度
 * @param {Number} height z 轴方向的长度
 * @see {@link ht.Node#setSize3d setSize3d}
 * @see {@link ht.Node#getSize3d getSize3d}
 * @return {Array} 格式为[x, y, z]，即[getWidth(), getTall(), getHeight()]
 */
s3 = function(width, tall, height){};

/**
 * 设置图元中心在 3D 坐标系中的 y 坐标
 * @param {Number} elevation y 轴方向的坐标值
 */
setElevation = function(elevation){};

/**
 * 设置图元在 2D 拓扑中的高度，或 3D 拓扑中的 z 轴长度
 * @param {Number} height 2D 拓扑中的高度，或 3D 拓扑中的 z 轴长度
 */
setHeight = function(height){};

/**
 * 设置宿主图元，当图元吸附上宿主图元时，宿主移动或旋转时会带动所有吸附者
 * @param {ht.Data} data 宿主图元
 */
setHost = function(data){};

/**
 * 设置拓扑上展现的图片信息，在 GraphView 拓扑图中图片一般以 position 为中心绘制
 * @param {String|Object} image 注册的图片名或 url 或矢量对象
 */
setImage = function(image){};

/**
 * 设置图元中心点坐标
 * @param {Number} x x 坐标
 * @param {Number} y y 坐标
 */
```

```
setPosition = function(x, y){};

/**
 * 设置图元中心点在 3D 拓扑中的三维坐标
 * @param {Number} x x 坐标
 * @param {Number} y y 坐标
 * @param {Number} z z 坐标
 */
setPosition3d = function(x, y, z){};

/**
 * 设置图元矩形区域
 * @param {Number} x x 坐标
 * @param {Number} y y 坐标
 * @param {Number} width 宽度
 * @param {Number} height 高度
 */
setRect = function(x, y, width, height){};

/**
 * 设置图元的旋转角度，围绕中心点顺时针旋转
 * @param {Number} rotation 旋转角度(弧度制)
 */
setRotation = function(rotation){};

/**
 * 设置图元在 3D 拓扑中的三维旋转角度
 * @param {Number} x 沿 x 轴的旋转角度(弧度制)
 * @param {Number} y 沿 y 轴的旋转角度(弧度制)
 * @param {Number} z 沿 z 轴的旋转角度(弧度制)
 */
setRotation3d = function(x, y, z){};

/**
 * 设置三维旋转模式<br>
 * 图元在 3D 拓扑中旋转时，先沿 x 轴旋转，再沿 y 轴旋转和先沿 y 轴旋转，再沿 x 轴旋转最后的结果是不一样的
 * @param {String} rotationMode 旋转模式，可选值如下：
 * <ul>
 * <li>xyz:先进行 x 轴旋转，再进行 y 轴旋转，最后进行 z 轴旋转</li>
 * <li>xzy:先进行 x 轴旋转，再进行 z 轴旋转，最后进行 y 轴旋转</li>
 * <li>yxz:先进行 y 轴旋转，再进行 x 轴旋转，最后进行 z 轴旋转</li>
 * <li>yzx:先进行 y 轴旋转，再进行 z 轴旋转，最后进行 x 轴旋转</li>
 * <li>zxy:先进行 z 轴旋转，再进行 x 轴旋转，最后进行 y 轴旋转</li>
```

```
 * <li>zyx:先进行 z 轴旋转，再进行 y 轴旋转，最后进行 x 轴旋转</li>
 * </ul>
 * @see {@link ht.Node#getRotationMode getRotationMode}
 */
setRotationMode = function(rotationMode){};

/**
 * 设置图元在 3D 拓扑中沿 x 轴的旋转角度(弧度制)
 * @param {Number} rotationX 旋转角度(弧度制)
 */
setRotationX = function(rotationX){};

/**
 * 设置图元在 3D 拓扑中沿 y 轴的旋转角度(弧度制)
 * @param {Number} rotationY 旋转角度(弧度制)
 */
setRotationY = function(rotationY){};

/**
 * 设置图元在 3D 拓扑中沿 z 轴的旋转角度(弧度制)
 * @param {Number} rotationZ 旋转角度(弧度制)
 */
setRotationZ = function(rotationZ){};

/**
 * 设置图元在 2D 拓扑中的尺寸(宽高)
 * @param {Number} width 宽度
 * @param {Number} height 高度
 */
setSize = function(width, height){};

/**
 * 设置图元在 3D 拓扑中的尺寸
 * @param {Number} width x 轴方向的长度
 * @param {Number} tall y 轴方向的长度
 * @param {Number} height z 轴方向的长度
 */
setSize3d = function(width, tall, height){};

/**
 * 设置图元在 3D 拓扑中的 y 轴方向的长度
 * @return {Number} tall y 轴方向的长度
 */
```

```
setTall = function(tall){};

/**
 * 设置图元在 3D 拓扑中的 x 轴方向的长度
 * @return {Number} width x 轴方向的长度
 */
setWidth = function(width){};

/**
 * 在当前坐标的基础上增加 x、y、z 三个方向的平移值，{@link ht.Node#translate3d translate3d}
的缩写
 * @param {Number} tx x 轴方向的平移值
 * @param {Number} ty y 轴方向的平移值
 * @param {Number} tz z 轴方向的平移值
 * @see {@link ht.Node#translate3d translate3d}
 */
t3 = function(tx, ty, tz){};

/**
 * 在当前坐标的基础上增加 x、y 两个方向的平移值
 * @param {Number} tx x 轴方向的平移值
 * @param {Number} ty y 轴方向的平移值
 */
translate = function(tx, ty){};

/**
 * 在当前坐标的基础上增加 x、y、z 三个方向的平移值，{@link ht.Node#t3 t3}的缩写
 * @param {Number} tx x 轴方向的平移值
 * @param {Number} ty y 轴方向的平移值
 * @param {Number} tz z 轴方向的平移值
 * @see {@link ht.Node#t3 t3}
 */
translate3d = function(tx, ty, tz){};

/**
 * 沿向量平移
 * @param {Array} direction 方向向量
 * @param {Number} distance 平移距离
 */
translate3dBy = function(direction, distance){};

/**
 * 沿向量[0, 0, -1]平移
 * @param {Number} distance 平移距离
```

```
 */
translateBack = function(distance){};

/**
 * 沿向量[0, -1, 0]平移
 * @param {Number} distance 平移距离
 */
translateBottom = function(distance){};

/**
 * 沿向量[0, 0, 1]平移
 * @param {Number} distance 平移距离
 */
translateFront = function(distance){};

/**
 * 沿向量[-1, 0, 0]平移
 * @param {Number} distance 平移距离
 */
translateLeft = function(distance){};

/**
 * 沿向量[1, 0, 0]平移
 * @param {Number} distance 平移距离
 */
translateRight = function(distance){};

/**
 * 沿向量[0, 1, 0]平移
 * @param {Number} distance 平移距离
 */
translateTop = function(distance){};
```

ht.Edge

```
/**
 * 拓扑连线，用于连接起始和目标两个 Node 节点，两个节点间可以有多条 Edge 存在，
 * 也允许起始和目标为同一节点。 <br>
 * 连线的 agent 指的是目前拓扑上真正代理连接该连线的节点，
 * 当节点位于关闭的 Group 之内时，Group 将代理内部的节点进行连接。
 * @constructor
```

```
 * @extends ht.Data
 */
ht.Edge = function() {};

/**
 * 获取连线组，起始和目标节点间有多条连线时才有值
 * @return {ht.EdgeGroup} 连线分组
 */
getEdgeGroup = function(){};

/**
 * 获取当前连线在连线组内的索引
 * @return {Number} 索引
 */
getEdgeGroupIndex = function(){};

/**
 * 获取当前连线在连线组内的索引
 * @return {Number} 索引
 */
getEdgeGroupIndex = function(){};

/**
 * 获取当前连线所在的连线组中的连线数
 * @return {Number} 连线数
 */
getEdgeGroupSize = function(){};

/**
 * 获取起始图元
 * @return {ht.Node} 起始图元
 */
getSource = function(){};

/**
 * 获取拓扑上连接的起始图元(代理)
 * @return {ht.Node} 起始图元(代理)
 */
getSourceAgent = function(){};

/**
 * 获取目标图元
 * @return {ht.Node} 目标图元
 */
```

```
getTarget = function(){};

/**
 * 获取拓扑上连接的目标图元(代理)
 * @return {ht.Node} 目标图元(代理)
 */
getTargetAgent = function(){};

/**
 * 获取当前连线是否为所在连线组的代理
 * @return {Boolean}
 */
isEdgeGroupAgent = function(){};

/**
 * 判断当前连线在连线组中是否被隐藏
 * @return {Boolean}
 */
isEdgeGroupHidden = function(){};

/**
 * 是否是自环(起始和目标是同一个图元)
 * @return {Boolean}
 */
isLooped = function(){};

/**
 * 设置起始图元
 * @param {ht.Node} source 起始图元
 */
setSource = function(source){};

/**
 * 设置目标图元
 * @param {ht.Node} target 目标图元
 */
setTarget = function(target){};
```

ht.EdgeGroup

```
/**
 * 连线分组
```

```
 * @constructor
 */
ht.EdgeGroup = function() {};

/**
 * 提供一个回调函数遍历此分组的所有连线
 * @param {Function} func 遍历函数
 * @param {Object} [scope] 函数域
 * @example edgeGroup.each(function(edge) {
 *   console.log(edge);
 * });
 */
each = function(func, scope){};

/**
 * 提供一个回调函数遍历相同起始和目标图元之间其它分组中的连线
 * @param {Function} func 遍历函数
 * @param {Object} [scope] 函数域
 * @example edgeGroup.eachSiblingEdge(function(edge) {
 *   console.log(edge);
 * });
 */
eachSiblingEdge = function(func, scope){};

/**
 * 根据索引获取分组中的连线
 * @param {Number} index 索引
 * @return {ht.Edge}
 */
get = function(index){};

/**
 * 获取分组中所有连线
 * @return {ht.List}
 */
getEdges = function(){};

/**
 * 获取相同起始和目标图元之间的其它分组
 * @return {ht.List}
 */
getSiblings = function(){};

/**
```

```
 * 获取参数连线在分组中的索引
 * @param {ht.Edge} edge 连线
 * @return {Number}
 */
indexOf = function(edge){};

/**
 * 获取分组中的连线数量
 * @return {Number}
 */
size = function(){};
```

ht.Group

```
/**
 * 组图元类型，可以包含 Node 和其它孩子元素，可双击展开或关闭
 * @constructor
 * @extends ht.Node
 */
ht.Group = function() {};

/**
 * 判断 Group 对象是否处于展开状态
 * @return {Boolean}
 */
isExpanded = function(){};

/**
 * 设置 Group 对象的展开关闭状态
 * @param {Boolean} expanded 为 true 展开，false 关闭
 */
setExpanded = function(expanded){};

/**
 * 切换展开关闭状态
 */
toggle = function(){};
```

ht.Shape

```
/**
```

```
 * 多边形元素，由多点组合形成的多边形，如果不填充背景色，可作为折线或曲线
 * @constructor
 * @extends ht.Node
 */
ht.Shape = function() {};

/**
 * 增加点
 * @param {Object} point 坐标点
 * @param {Number} [index] 索引，如果不指定索引则加到最后
 */
addPoint = function(point, index){};

/**
 * 计算 Shape 的长度
 * @param {Number} [resolution] 曲线分段微分数，默认为 12，数字越大计算结果越精确，同时也越耗费性能
 * @return {Number}
 */
getLength = function(resolution){};

/**
 * 获取点集合
 * @return {ht.List}
 */
getPoints = function(){};

/**
 * 获取线段类型集合
 * @return {ht.List}
 */
getSegments = function(){};

/**
 * 获取 3D 拓扑中的线段厚度，小于 0 时可实现类似地板的填充效果
 * @return {Number}
 */
getThickness = function(){};

/**
 * 是否闭合 Shape
 * @return {Boolean}
 */
```

```
isClosePath = function(){};

/**
 * 根据索引删除点
 * @param {Number} index 索引
 */
removePointAt = function(index){};

/**
 * 设置是否闭合 Shape
 * @param {Boolean} v
 */
setClosePath = function(v){};

/**
 * 修改索引指向的点坐标
 * @param {Number} index 索引
 * @param {Object} point 新坐标点
 */
setPoint = function(index, point){};

/**
 * 设置 Shape 的点
 * @param {ht.List} points
 */
setPoints = function(points){};

/**
 * 设置 Shape 的线段类型
 * @param {ht.List} segments
 */
setSegments = function(segments){};

/**
 * 设置 3D 拓扑中的线段厚度，小于 0 时可实现类似地板的填充效果
 * @param {Number} thickness
 */
setThickness = function(thickness){};

/**
 * 构建一个新的 Shape 点集合并返回
 * @return {ht.List}
 */
```

```
toPoints = function(){};

/**
 * 构建一个新的线段类型集合并返回
 * @return {ht.List}
 */
toSegments = function(){};
```

ht.Grid

```
/**
 * 网格类型，可嵌套包含附属节点，以网格方式定位附属节点，一般用于呈现设备面板
 * @constructor
 * @extends ht.Node
 */
ht.Grid = function() {};

/**
 * 获取单元格的矩形范围
 * @param {Number} rowIndex 行索引
 * @param {Number} columnIndex 列索引
 */
getCellRect = function(rowIndex, columnIndex){};
/**
 * 子网类型，分层次管理呈现图元，可根据区域或业务分类管理图元
 * @constructor
 * @extends ht.Node
 */
```

ht.Tab

```
/**
 * 页签，用于加入 TabView 页签组件
 * @constructor
 * @extends ht.Data
 */
ht.Tab = function() {};

/**
 * 获取页签被选中时呈现的视图组件
 * @return {HTMLElement}
```

```
 */
getView = function(){};

/**
 * 页签是否可被关闭
 * @return {Boolean}
 */
isClosable = function(){};

/**
 * 页签是否被禁用
 * @return {Boolean}
 */
isDisabled = function(){};

/**
 * 设置页签是否可被关闭
 * @param {Boolean} v
 */
setClosable = function(v){};

/**
 * 设置页签是否被禁用
 * @param {Boolean} v
 */
setDisabled = function(v){};

/**
 * 设置页签被选中时呈现的视图组件
 * @param {HTMLElement} v
 */
setView = function(v){};
```

ht.Column

```
/**
 * 列数据，用于定义表格组件的列信息，包含列名称、类型以及自定义渲染和编辑单元格等信息
 * @constructor
 * @extends ht.Data
 */
ht.Column = function() {};

/**
 * 将要显示的值传入此方法格式化处理并返回，一般用于将数字转换更易读的文本格式，可重载自
```

```
定义
 * @param {Object} value 格式化之前值
 * @return {Object} 格式化之后的值
 */
formatValue = function(value) {};

/**
 * 获取列的属性类型，值列表如下：<br>
 * <ul>
 * <li>null: 默认类型，如 name 为 age，采用 getAge()和 setAge(98)的 get/set 或 is/set 方式存取</li>
 * <li>style: 如 name 为 age，采用 getStyle('age')和 setStyle('age', 98)的方式存取</li>
 * <li>field：如 name 为 age，采用 data.age 和 data.age = 98 的方式存取</li>
 * <li>attr：如 name 为 age，采用 getAttr('age')和 setAttr('age', 98)的方式存取</li>
 * </ul>
 * @return {String|null}
 */
getAccessType = function() {};

/**
 * 获取文字的水平对齐方式，可用值有 left|right|center，默认为 left
 * @return {String}
 */
getAlign = function() {};

/**
 * 获取表头文字的颜色
 * @return {color}
 */
getColor = function() {};

/**
 * 获取颜色选择器配置
 * @see {@link ht.Column#setColorPicker setColorPicker}
 * @return {Object}
 */
getColorPicker = function() {};

/**
 * 当此列使用下拉列表作为编辑器时，此方法返回下拉列表的所有枚举 icon 数组
 * @return {Array}
 */
getEnumIcons = function() {};
```

```
/**
 * 当此列使用下拉列表作为编辑器时，此方法返回下拉列表的所有枚举文字数组
 * @return {Array}
 */
getEnumLabels = function() {};

/**
 * 当此列使用下拉列表作为编辑器时，此方法返回下拉列表的最大高度(超出使用滚动条)
 * @return {Array}
 */
getEnumMaxHeight = function() {};

/**
 * 当此列使用下拉列表作为编辑器时，此方法返回下拉列表的值数组
 * @return {Array}
 */
getEnumValues = function() {};

/**
 * 获取自定义的单元格编辑器
 * @return {Function}
 */
getItemEditor = function() {};

/**
 * 获取拉条配置
 * @see {@link ht.Column#setSlider setSlider}
 * @return {Object}
 */
getSlider = function() {};

/**
 * 获取排序函数
 * @return {Function}
 */
getSortFunc = function() {};

/**
 * 获取排序状态<br>
 * <ul>
 * <li>asc: 升序</li>
 * <li>desc: 降序</li>
 * </ul>
 * @return {String}
```

```
 */
getSortOrder = function() {};

/**
 * 获取 toolTip 文字
 * @param {ht.Data} data 数据元素
 * @param {ht.widget.TableView} tableView 视图对象
 * @return {String}
 */
getToolTip = function(data, tableView) {};

/**
 * 获取值类型，值类型用于提示组件提供合适的 renderer 渲染，合适的编辑控件，及改变值时必要的类型转换<br>
 * <ul>
 * <li>null：默认类型，显示为文本方式</li>
 * <li>string：字符串类型，显示为文本方式</li>
 * <li>boolean：布尔类型，显示为勾选框</li>
 * <li>color：颜色类型，以填充背景色的方式显示</li>
 * <li>int：整型类型，文本编辑器改变值时自动通过 parseInt 进行转换</li>
 * <li>number：浮点数类型，文本编辑器改变值时自动通过 parseFloat 转换</li>
 * </ul>
 * @return {String}
 */
getValueType = function() {};

/**
 * 获取列宽度
 * @return {Number}
 */
getWidth = function() {};

/**
 * 判断该列是否允许多选时批量编辑，默认为 true
 * @return {Boolean}
 */
isBatchEditable = function() {};

/**
 * 判断此列是否可编辑
 * @return {Boolean}
 */
isEditable = function() {};
```

```
/**
 * 判断属性是否可为空字符串，可避免输入空字符串，空字符串转换成 undefined。默认为 false
 * @return {Boolean}
 */
isEmptiable = function() {};

/**
 * 枚举下拉编辑器是否允许可输入，默认为 false
 * @return {Boolean}
 */
isEnumEditable = function() {};

/**
 * 判断值匹配时是否采用严格的===进行比较，默认为 true，若为 false 则采用==进行比较
 * @return {Boolean}
 */
isEnumStrict = function() {};

/**
 * 判断属性是否可为空，默认为 true，设置为 false 可避免输入 null 或 undefined
 * @return {Boolean}
 */
isNullable = function() {};

/**
 * 判断当前列是否可排序
 * @return {Boolean}
 */
isSortable = function() {};

/**
 * 判断当前列是否是否可见
 * @return {Boolean}
 */
isVisible = function() {};

/**
 * 设置列的属性类型，可选值如下：<br>
 * <ul>
 * <li>null: 默认类型，如 name 为 age，采用 getAge()和 setAge(98)的 get/set 或 is/set 方式存取</li>
 * <li>style: 如 name 为 age，采用 getStyle('age')和 setStyle('age', 98)的方式存取</li>
 * <li>field：如 name 为 age，采用 data.age 和 data.age = 98 的方式存取</li>
 * <li>attr：如 name 为 age，采用 getAttr('age')和 setAttr('age', 98)的方式存取</li>
```

```
 * </ul>
 * @param {String|null} accessType
 */
setAccessType = function(accessType) {};

/**
 * 设置文字的水平对齐方式，可用值有 left|right|center，默认为 left
 * @param {String} align 对齐方式
 */
setAlign = function(align) {};

/**
 * 设置该列是否允许多选时批量编辑，默认为 true
 * @param {Boolean} v
 */
setBatchEditable = function(v) {};

/**
 * 设置表头文字的颜色
 * @param {color} color
 */
setColor = function(color) {};

/**
 * 设置颜色选择器配置，需要引入 form 插件，设置此属性后，此列将使用颜色选择器作为单元格编辑器
 * @param {Object} v 颜色选择器配置，如{background: 'red'}可以指定颜色选择器背景为红色，如果要使用默认配置，使用空对象{}即可
 */
setColorPicker = function(v) {};

/**
 * 设置此列是否可编辑
 * @param {Boolean} editable
 */
setEditable = function(editable) {};

/**
 * 设置属性是否可为空字符串，可避免输入空字符串，空字符串转换成 undefined。默认为 false
 * @param {Boolean} emptiable
 */
setEmptiable = function(emptiable) {};

/**
```

```
 * 设置枚举列表，此列自动采用下拉列表作为单元格编辑器
 * @param {Object|Array} v
 * @example //示例，参数依次表示：值，文字、icon
 * column.setEnum([1,2,3], ['C','C++','JS'], ['c_icon', 'c++_icon', 'js_icon']);
 * //也可以使用对象的方式：
 * column.setEnum({values:[1,2,3], labels:['C','C++','JS'], icons:['c_icon', 'c++_icon',
'js_icon']});
 */
setEnum = function(v) {};

/**
 * 设置自定义的单元格编辑器
 * @param {Function} editor
 */
setItemEditor = function(editor) {};

/**
 * 设置属性是否可为空，默认为 true，设置为 false 可避免输入 null 或 undefined
 * @param {Boolean} nullable
 */
setNullable = function(nullable) {};

/**
 * 设置拉条配置，需要引入 form 插件，设置此属性后，此列将使用拉条作为单元格编辑器
 * @param {Object} v 拉条配置，如{background: 'red'}可以指定拉条背景为红色，如果要使用默
认配置，使用空对象{}即可
 */
setSlider = function(v) {};

/**
 * 设置当前列是否可排序
 * @param {Boolean} nullable
 */
setSortable = function(nullable) {};

/**
 * 设置排序函数
 * @param {Function} func
 */
setSortFunc = function(func) {};

/**
 * 设置排序状态<br>
 * <ul>
```

```
 * <li>asc: 升序</li>
 * <li>desc: 降序</li>
 * </ul>
 * @param {String} sortOrder
 */
setSortOrder = function(sortOrder) {};

/**
 * 设置值类型，值类型用于提示组件提供合适的 renderer 渲染，合适的编辑控件，及改变值时必要的类型转换<br>
 * <ul>
 * <li>null：默认类型，显示为文本方式</li>
 * <li>string：字符串类型，显示为文本方式</li>
 * <li>boolean：布尔类型，显示为勾选框</li>
 * <li>color：颜色类型，以填充背景色的方式显示</li>
 * <li>int：整型类型，文本编辑器改变值时自动通过 parseInt 进行转换</li>
 * <li>number：浮点数类型，文本编辑器改变值时自动通过 parseFloat 转换</li>
 * </ul>
 * @param {String|null} type
 */
setValueType = function(type) {};

/**
 * 设置当前列是否是否可见
 * @param {Boolean} v
 */
setVisible = function(v) {};

/**
 * 设置列宽度
 * @param {Number} v
 */
setWidth = function(v) {};

/**
 * 根据 value 查找对应的枚举 icon
 * @param {Object} value 枚举值
 * @return {String}
 */
toEnumIcon = function(value) {};

/**
 * 根据 value 查找对应的枚举 label 文字
 * @param {Object} value 枚举值
```

```
 * @return {String}
 */
toEnumLabel = function(value) {};
```

ht.Property

```
/**
 * 属性数据，指定 PropertyView 属性组件要显示的属性
 * @constructor
 * @extends ht.Data
 */
ht.Property = function() {};

/**
 * 绘制属性值，可重载自定义，如果返回值为 HTML 元素，则使用 HTML 元素当作 Renderer
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Property} property 属性对象
 * @param {Object} value 值
 * @param {Number} rowIndex 行索引
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} w 绘制的宽度
 * @param {Number} w 绘制的宽度
 * @param {ht.Data} data 数据元素
 * @param {ht.widget.PropertyView} view 视图组件
 * @return {HTMLElement}
 */
drawPropertyValue = function (g, property, value, rowIndex, x, y, w, h, data, view){};


/**
 * 将要显示的值传入此方法格式化处理并返回，一般用于将数字转换更易读的文本格式，可重载自定义
 * @param {Object} value 格式化之前值
 * @return {Object} 格式化之后的值
 */
formatValue = function(value) {};

/**
 * 获取属性类型，值列表如下：<br>
 * <ul>
 * <li>null: 默认类型，如 name 为 age，采用 getAge()和 setAge(98)的 get/set 或 is/set 方式存
```

```
取</li>
 * <li>style: 如 name 为 age，采用 getStyle('age')和 setStyle('age', 98)的方式存取</li>
 * <li>field：如 name 为 age，采用 data.age 和 data.age = 98 的方式存取</li>
 * <li>attr：如 name 为 age，采用 getAttr('age')和 setAttr('age', 98)的方式存取</li>
 * </ul>
 * @return {String|null}
 */
getAccessType = function() {};

/**
 * 获取文字的水平对齐方式，可用值有 left|right|center，默认为 left
 * @return {String}
 */
getAlign = function() {};

/**
 * 获取属性分类名称
 * @return {String}
 */
getCategoryName = function() {};

/**
 * 获取属性名文字的颜色
 * @return {color}
 */
getColor = function() {};

/**
 * 获取颜色选择器配置
 * @see {@link ht.Property#setColorPicker setColorPicker}
 * @return {Object}
 */
getColorPicker = function() {};

/**
 * 当属性使用下拉列表作为编辑器时，此方法返回下拉列表的所有枚举 icon 数组
 * @return {Array}
 */
getEnumIcons = function() {};

/**
 * 当属性使用下拉列表作为编辑器时，此方法返回下拉列表的所有枚举文字数组
 * @return {Array}
 */
```

```
getEnumLabels = function() {};

/**
 * 当属性使用下拉列表作为编辑器时，此方法返回下拉列表的最大高度(超出使用滚动条)
 * @return {Array}
 */
getEnumMaxHeight = function() {};

/**
 * 当属性使用下拉列表作为编辑器时，此方法返回下拉列表的值数组
 * @return {Array}
 */
getEnumValues = function() {};

/**
 * 获取自定义的单元格编辑器
 * @return {Function}
 */
getItemEditor = function() {};

/**
 * 获取拉条配置
 * @see {@link ht.Property#setSlider setSlider}
 * @return {Object}
 */
getSlider = function() {};

/**
 * 获取 toolTip 文字
 * @param {ht.Data} data 数据元素
 * @param {Boolean} isValue 是否是属性值
 * @param {ht.widget.PropertyView} propertyView 视图对象
 * @return {String}
 */
getToolTip = function(data, isValue, propertyView) {};

/**
 * 获取值类型，值类型用于提示组件提供合适的 renderer 渲染，合适的编辑控件，及改变值时必要的类型转换<br>
 * <ul>
 * <li>null：默认类型，显示为文本方式</li>
 * <li>string：字符串类型，显示为文本方式</li>
 * <li>boolean：布尔类型，显示为勾选框</li>
 * <li>color：颜色类型，以填充背景色的方式显示</li>
```

```
 * <li>int：整型类型，文本编辑器改变值时自动通过 parseInt 进行转换</li>
 * <li>number：浮点数类型，文本编辑器改变值时自动通过 parseFloat 转换</li>
 * </ul>
 * @return {String}
 */
getValueType = function() {};

/**
 * 判断是否允许多选时批量编辑，默认为 true
 * @return {Boolean}
 */
isBatchEditable = function() {};

/**
 * 判断属性是否可编辑
 * @return {Boolean}
 */
isEditable = function() {};

/**
 * 判断属性是否可为空字符串，可避免输入空字符串，空字符串转换成 undefined。默认为 false
 * @return {Boolean}
 */
isEmptiable = function() {};

/**
 * 枚举下拉编辑器是否允许可输入，默认为 false
 * @return {Boolean}
 */
isEnumEditable = function() {};

/**
 * 判断值匹配时是否采用严格的===进行比较，默认为 true，若为 false 则采用==进行比较
 * @return {Boolean}
 */
isEnumStrict = function() {};

/**
 * 判断属性是否可为空，默认为 true，设置为 false 可避免输入 null 或 undefined
 * @return {Boolean}
 */
isNullable = function() {};

/**
```

```
 * 设置属性类型，可选值如下：<br>
 * <ul>
 * <li>null: 默认类型，如 name 为 age，采用 getAge()和 setAge(98)的 get/set 或 is/set 方式存取</li>
 * <li>style: 如 name 为 age，采用 getStyle('age')和 setStyle('age', 98)的方式存取</li>
 * <li>field：如 name 为 age，采用 data.age 和 data.age = 98 的方式存取</li>
 * <li>attr：如 name 为 age，采用 getAttr('age')和 setAttr('age', 98)的方式存取</li>
 * </ul>
 * @param {String|null} accessType
 */
setAccessType = function(accessType) {};

/**
 * 设置文字的水平对齐方式，可用值有 left|right|center，默认为 left
 * @param {String} align 对齐方式
 */
setAlign = function(align) {};

/**
 * 设置是否允许多选时批量编辑，默认为 true
 * @param {Boolean} v
 */
setBatchEditable = function(v) {};

/**
 * 设置属性分类名称
 * @param {String} name
 */
setCategoryName = function(name) {};

/**
 * 设置属性名文字的颜色
 * @param {color} color
 */
setColor = function(color) {};

/**
 * 设置颜色选择器配置，需要引入 form 插件，设置后将使用颜色选择器作为属性编辑器
 * @param {Object} v 颜色选择器配置，如{background: 'red'}可以指定颜色选择器背景为红色，如果要使用默认配置，使用空对象{}即可
 */
setColorPicker = function(v) {};

/**
```

```
 * 设置属性是否可编辑
 * @param {Boolean} editable
 */
setEditable = function(editable) {};

/**
 * 设置属性是否可为空字符串，可避免输入空字符串，空字符串转换成 undefined。默认为 false
 * @param {Boolean} emptiable
 */
setEmptiable = function(emptiable) {};

/**
 * 设置枚举列表，自动采用下拉列表作为属性编辑器
 * @param {Object|Array} v
 * @example //示例，参数依次表示：值，文字、icon
 * property.setEnum([1,2,3], ['C','C++','JS'], ['c_icon', 'c++_icon', 'js_icon']);
 * //也可以使用对象的方式：
 * property.setEnum({values:[1,2,3], labels:['C','C++','JS'], icons:['c_icon', 'c++_icon',
'js_icon']});
 */
setEnum = function(v) {};

/**
 * 设置自定义的属性编辑器
 * @param {Function} editor
 */
setItemEditor = function(editor) {};

/**
 * 设置属性是否可为空，默认为 true，设置为 false 可避免输入 null 或 undefined
 * @param {Boolean} nullable
 */
setNullable = function(nullable) {};

/**
 * 设置拉条配置，需要引入 form 插件，设置后将使用拉条作为属性编辑器
 * @param {Object} v 拉条配置，如{background: 'red'}可以指定拉条背景为红色，如果要使用默
认配置，使用空对象{}即可
 */
setSlider = function(v) {};

/**
 * 设置值类型，值类型用于提示组件提供合适的 renderer 渲染，合适的编辑控件，及改变值时必要
的类型转换<br>
```

```
 * <ul>
 * <li>null：默认类型，显示为文本方式</li>
 * <li>string：字符串类型，显示为文本方式</li>
 * <li>boolean：布尔类型，显示为勾选框</li>
 * <li>color：颜色类型，以填充背景色的方式显示</li>
 * <li>int：整型类型，文本编辑器改变值时自动通过 parseInt 进行转换</li>
 * <li>number：浮点数类型，文本编辑器改变值时自动通过 parseFloat 转换</li>
 * </ul>
 * @param {String|null} type
 */
setValueType = function(type) {};

/**
 * 根据 value 查找对应的枚举 icon
 * @param {Object} value 枚举值
 * @return {String}
 */
toEnumIcon = function(value) {};

/**
 * 根据 value 查找对应的枚举 label 文字
 * @param {Object} value 枚举值
 * @return {String}
 */
toEnumLabel = function(value) {};
```

ht.Default

```
ht.Default = {};

/**
 * 指定组件基准 CSS 的 ZIndex 值，改值仅在将 HT 与其他第三方组件混合使用时根据需要设置"
 * @type Number
 */
ht.Default.baseZIndex = {};

/**
 * 判断是否为触屏可 Touch 方式交互，HT 系统一般会自动判断，对于极少数 HT 无法正确识别的系统下，可以通过配置强制指定
 * @type boolean
 */
ht.Default.isTouchable = {};
```

```
/**
 * 设备像素比，HT 系统自动取至 window.devicePixelRatio，某些特性情况需要为 mobile 应用牺牲精度节省内存时可以强制设置为较小值
 * @type Number
 */
ht.Default.devicePixelRatio = {};

/**
 * 组件初次加载时界面宽高值可能会为 0，HT 会自动尝试等待下次延迟刷新，该参数指定尝试次数，一般无需改动
 * @default 3
 * @type Number
 */
ht.Default.reinvalidateCount = {};

/**
 * 进行框选判断时为了避免内存占用过大，HT 会根据最大面积限制进行缩放判断，该参数一般无需改动
 * @default 3000
 * @type Number
 */
ht.Default.hitMaxArea = {};

/**
 * 决定 Data 元素被选中时，组件是否自动滚动到 Data 元素可见位置
 * @default true
 * @type boolean
 */
ht.Default.autoMakeVisible = {};

/**
 * 决定组件的滚动条默认是否自动隐藏，true 为自动显示和隐藏，false 则需要滚动时一直显示不会自动隐藏
 * @default true
 * @type boolean
 */
ht.Default.autoHideScrollBar = {};

/**
 * 组件无效时的透明度
 * @type Number
 */
ht.Default.disabledOpacity = {};
```

```
/**
 * 组件无效时的背景色
 * @type color
 */
ht.Default.disabledBackground = {};

/**
 * 组件的 ToolTip 显示的延迟间隔
 * @default 800
 * @type Number
 */
ht.Default.toolTipDelay = {};

/**
 * 组件的 ToolTip 显示的情况下，如果鼠标移动到新的位置时，ToolTip 是否实时持续跟进
 * @default false
 * @type boolean
 */
ht.Default.toolTipContinual = {};

/**
 * 线条末端线帽的样式，可选值为 butt|round|square
 * @default butt
 * @type String
 */
ht.Default.lineCap = {};

/**
 * 当两条线交汇时创建边角的类型，可选参数为：bevel|round|miter
 * @default round
 * @type String
 */
ht.Default.lineJoin = {};

/**
 * 默认图片的渐进色类型
 * @default linear.northeast
 * @type String
 */
ht.Default.imageGradient = {};

/**
 * 连线或多边形等图形的默认虚线样式
 * @type Array
```

```
 */
ht.Default.dashPattern = {};

/**
 * 默认动画效果函数
 * @default function (m){return m*m}
 * @type Function
 */
ht.Default.animEasing = {};

/**
 * 默认文字颜色
 * @default #000
 * @type color
 */
ht.Default.labelColor = {};

/**
 * 选中状态下文字颜色
 * @default #fff
 * @type color
 */
ht.Default.labelSelectColor = {};

/**
 * 默认文字字体
 * @default 12px arial, sans-serif
 * @type font
 */
ht.Default.labelFont = {};

/**
 * 默认文字字体
 * @default 12px arial, sans-serif
 * @type font
 */
ht.Default.labelFont = {};

/**
 * 通用组件缩进，例如树组件每一层的缩进
 * @default 20
 * @type Number
 */
```

```
ht.Default.widgetIndent = {};

/**
 * 通用组件行高，例如表格每行行高
 * @default 20
 * @type Number
 */
ht.Default.widgetRowHeight = {};

/**
 * 通用组件抬头高度，例如 TabView，TableHeader 和 Toolbar 等的头部高度
 * @default 22
 * @type Number
 */
ht.Default.widgetHeaderHeight = {};

/**
 * AccordionView 和 TabView 等组件的 Title 默认高度
 * @default 24
 * @type Number
 */
ht.Default.widgetTitleHeight = {};

/**
 * 滚动条默认颜色
 * @default rgba(0,0,0,0.35)
 * @type color
 */
ht.Default.scrollBarColor = {};

/**
 * 滚动条默认宽度
 * @default 7
 * @type Number
 */
ht.Default.scrollBarSize = {};

/**
 * 滚动条默认的隐藏间隔毫秒数
 * @default 1000
 * @type Number
 */
ht.Default.scrollBarTimeout = {};
```

```
/**
 * 滚动条默认最小长度
 * @default 20
 * @type Number
 */
ht.Default.scrollBarMinLength = {};

/**
 * 滚动条起作用区域默认大小
 * @default 16
 * @type Number
 */
ht.Default.scrollBarInteractiveSize = {};

/**
 * 默认缩放步进
 * @default 1.3
 * @type Number
 */
ht.Default.zoomIncrement = {};

/**
 * 默认滚轮缩放步进
 * @default 1.05
 * @type Number
 */
ht.Default.scrollZoomIncrement = {};

/**
 * 默认双指触屏 Touch 方式缩放步进
 * @default 1.08
 * @type Number
 */
ht.Default.pinchZoomIncrement = {};

/**
 * 默认最大缩放倍数
 * @default 20
 * @type Number
 */
ht.Default.zoomMax = {};

/**
 * 默认最小缩放倍数
```

```
 * @default 0.01
 * @type Number
 */
ht.Default.zoomMin = {};

/**
 * 默认曲线分段微分数
 * @default 12
 * @type Number
 */
ht.Default.segmentResolution = {};

/**
 * 默认模型分段微分数
 * @default 24
 * @type Number
 */
ht.Default.shapeResolution = {};

/**
 * 默认模型边数
 * @default 24
 * @type Number
 */
ht.Default.shapeSide = {};

/**
 * ToolTip 的文字颜色
 * @default #000
 * @type color
 */
ht.Default.toolTipLabelColor = {};

/**
 * ToolTip 的文字字体
 * @default 12px arial, sans-serif
 * @type String
 */
ht.Default.toolTipLabelFont = {};

/**
 * ToolTip 的文字字体
 * @default 12px arial, sans-serif
 * @type String
```

```
 */
ht.Default.toolTipLabelFont = {};

/**
 * ToolTip 的背景颜色
 * @default #FFFFE0
 * @type color
 */
ht.Default.toolTipBackground = {};

/**
 * ToolTip 的阴影颜色
 * @default rgba(0,0,0,0.35)
 * @type color
 */
ht.Default.toolTipShadowColor = {};

/**
 * 矢量组件 comp 嵌套堆栈，矢量组件 comp 可嵌套定义，通过改参数能得到当前嵌套层次信息
 * @type Array
 */
ht.Default.compStack = {};

/**
 * 此函数返回连线组的代理连线，edges 为 ht.List 类型的 ht.Edge 对象数组，默认返回
edges.get(0)，可重载自定义规则
 * @type Function
 */
ht.Default.edgeGroupAgentFunc = {};

/**
 *GraphView 组件中拖动图元到边缘时会自动滚动，该参数决定开始自动滚动的区域范围，设置为 0
或负数则代表关闭自动滚动功能
 * @default 16
 * @type Number
 */
ht.Default.graphViewAutoScrollZone = {};

/**
 * 决定 GraphView 组件按空格键是否允许复位，复位调用了 GraphView#reset()函数，该函数默认
会调用 setZoom(1)和 setTranslate(0, 0)
 * @default true
 * @type Boolean
 */
```

```
ht.Default.graphViewResettable = {};

/**
 * 决定 GraphView 组件是否允许手抓图操作
 * @default true
 * @type Boolean
 */
ht.Default.graphViewPannable = {};

/**
 * 决定 GraphView 组件是否允许按 Ctrl 键进行框选操作
 * @default true
 * @type Boolean
 */
ht.Default.graphViewRectSelectable = {};

/**
 * 决定 GraphView 组件是否显示滚动条
 * @default true
 * @type Boolean
 */
ht.Default.graphViewScrollBarVisible = {};

/**
 * GraphView 组件框选边框颜色
 * @default #2C3E50
 * @type color
 */
ht.Default.graphViewRectSelectBorderColor = {};

/**
 * GraphView 组件框选背景颜色
 * @default rgba(0,0,0,0.35)
 * @type color
 */
ht.Default.graphViewRectSelectBackground = {};

/**
 * GraphView 组件编辑点大小
 * @default 7
 * @type Number
 */
ht.Default.graphViewEditPointSize = {};
```

```
/**
 * GraphView 组件编辑点边框颜色
 * @default #2C3E50
 * @type color
 */
ht.Default.graphViewEditPointBorderColor = {};

/**
 * GraphView 组件编辑点背景颜色
 * @default #D9D9D9
 * @type color
 */
ht.Default.graphViewEditPointBackground = {};

/**
 * Graph3dView 组件初始化 WebGL 上下文参数，一般无需改动
 * @default null
 * @type Object
 */
ht.Default.graph3dViewAttributes = {};

/**
 * Graph3dView 组件是否为第一人称交互方式
 * @default false
 * @type Boolean
 */
ht.Default.graph3dViewFirstPersonMode = {};

/**
 * Graph3dView 组件在第一人称交互方式时，鼠标是否能漫游
 * @default true
 * @type Boolean
 */
ht.Default.graph3dViewMouseRoamable = {};

/**
 * Graph3dView 组件键盘控制移动的步进
 * @default 15
 * @type Number
 */
ht.Default.graph3dViewMoveStep = {};

/**
 * Graph3dView 组件键盘控制旋转的步进
```

```
 * @default 0.05235987755982988
 * @type Number
 */
ht.Default.graph3dViewRotateStep = {};

/**
 * Graph3dView 组件是否允许按 Shift 键进行手抓图平移
 * @default true
 * @type Boolean
 */
ht.Default.graph3dViewPannable = {};

/**
 * Graph3dView 组件是否允许进行旋转中心或方位操作
 * @default true
 * @type Boolean
 */
ht.Default.graph3dViewRotatable = {};

/**
 * Graph3dView 组件是否允许前进后退操作
 * @default true
 * @type Boolean
 */
ht.Default.graph3dViewWalkable = {};

/**
 *Graph3dView 组件是否允许按空格键复位，复位调用了 Graph3dView#reset()函数，该函数会重置
Graph3dView 的 eye|center|up 三个参数
 * @default true
 * @type Boolean
 */
ht.Default.graph3dViewResettable = {};

/**
 * Graph3dView 组件是否允许缩放
 * @default true
 * @type Boolean
 */
ht.Default.graph3dViewZoomable = {};

/**
 * Graph3dView 组件是否允许框选
 * @default true
```

```
 * @type Boolean
 */
ht.Default.graph3dViewRectSelectable = {};

/**
 * Graph3dView 组件框选背景
 * @default rgba(0,0,0,0.35)
 * @type color
 */
ht.Default.graph3dViewRectSelectBackground = {};

/**
 * Graph3dView 组件是否允许显示 xz 面网格
 * @default false
 * @type Boolean
 */
ht.Default.graph3dViewGridVisible = {};

/**
 * Graph3dView 组件显示 xz 面的网格行列数
 * @default 50
 * @type Number
 */
ht.Default.graph3dViewGridSize = {};

/**
 * Graph3dView 组件显示 xz 面的网格行列间距
 * @default 50
 * @type Number
 */
ht.Default.graph3dViewGridGap = {};

/**
 * Graph3dView 组件显示 xz 面的网格线颜色
 * @default [0.4,0.75,0.85,1]
 * @type color
 */
ht.Default.graph3dViewGridColor = {};

/**
 * Graph3dView 组件原点 x|y|z 三个轴线是否可见
 * @default false
 * @type Boolean
 */
```

```
ht.Default.graph3dViewOriginAxisVisible = {};

/**
 * Graph3dView 组件屏幕中心点 x|y|z 三个轴线是否可见
 * @default false
 * @type Boolean
 */
ht.Default.graph3dViewCenterAxisVisible = {};

/**
 * Graph3dView 组件显示 x 轴线颜色
 * @default [1,0,0,1]
 * @type color
 */
ht.Default.graph3dViewAxisXColor = {};

/**
 * Graph3dView 组件显示 y 轴线颜色
 * @default [0,1,0,1]
 * @type color
 */
ht.Default.graph3dViewAxisYColor = {};

/**
 * Graph3dView 组件显示 z 轴线颜色
 * @default [0,0,1,1]
 * @type color
 */
ht.Default.graph3dViewAxisZColor = {};

/**
 * Graph3dView 组件在编辑状态图元拉伸标识颜色
 * @default [1,1,0,1]
 * @type color
 */
ht.Default.graph3dViewEditSizeColor = {};

/**
 * Graph3dView 组件是否显示为正交投影方式
 * @default false
 * @type Boolean
 */
ht.Default.graph3dViewOrtho = {};
```

```
/**
 * Graph3dView 组件正交投影方式下屏幕宽度内显示的逻辑宽度值
 * @default 2000
 * @type Number
 */
ht.Default.graph3dViewOrthoWidth = {};

/**
 * Graph3dView 组件在透视投影方式下的 y 轴张角弧度（Field of view）
 * @default 0.7853981633974483
 * @type Number
 */
ht.Default.graph3dViewFovy = {};

/**
 * Graph3dView 组件投影呈现内容的最近距离，该值在可接受的范围内尽量设置较大值有利于呈现精度
 * @default 10
 * @type Number
 */
ht.Default.graph3dViewNear = {};

/**
 * Graph3dView 组件投影呈现内容的最远距离，该值可根据场景最远范围进行调节设置
 * @default 10000
 * @type Number
 */
ht.Default.graph3dViewFar = {};

/**
 * Graph3dView 组件投影呈现时，眼睛观察点所在位置
 * @default [0,300,1000]
 * @type Array
 */
ht.Default.graph3dViewEye = {};

/**
 * Graph3dView 组件投影呈现时，眼睛最终锁定的目标中心位置
 * @default [0,0,0]
 * @type Array
 */
ht.Default.graph3dViewCenter = {};

/**
```

```
 * Graph3dView 组件投影呈现时，摄像镜头垂直朝上方向
 * @default [0,1,-1e-7]
 * @type Array
 */
ht.Default.graph3dViewUp = {};

/**
 * 头灯影响范围，默认为`0`代表可照射到无穷远处，如果设置了值则光照射效果随物体远离光影而衰减
 * @default 0
 * @type Number
 */
ht.Default.graph3dViewHeadlightRange = {};

/**
 * 头灯影响范围，默认为`0`代表可照射到无穷远处，如果设置了值则光照射效果随物体远离光影而衰减
 * @default 0
 * @type Number
 */
ht.Default.graph3dViewHeadlightRange = {};

/**
 * 头灯颜色
 * @default [1,1,1,1]
 * @type Array
 */
ht.Default.graph3dViewHeadlightColor = {};

/**
 * 头灯强度，默认为 1，大于 1 增强，小于 1 减弱
 * @default 1
 * @type Number
 */
ht.Default.graph3dViewHeadlightIntensity = {};

/**
 * 是否关闭头灯效果
 * @default false
 * @type Boolean
 */
ht.Default.graph3dViewHeadlightDisabled = {};

/**
```

```
 * 是否关闭雾化效果
 * @default true
 * @type Boolean
 */
ht.Default.graph3dViewFogDisabled = {};

/**
 * 雾颜色
 * @default white
 * @type color
 */
ht.Default.graph3dViewFogColor = {};

/**
 * 代表从该距离起物体开始受雾效果影响
 * @default 1
 * @type Number
 */
ht.Default.graph3dViewFogNear = {};

/**
 * 代表从该距离之后物体完全看不清
 * @default 2000
 * @type Number
 */
ht.Default.graph3dViewFogFar = {};

/**
 * 折叠组件展开状态图标
 * @type String
 */
ht.Default.accordionViewExpandIcon = {};

/**
 * 折叠组件关闭状态图标
 * @type String
 */
ht.Default.accordionViewCollapseIcon = {};

/**
 * 折叠组件文字颜色
 * @default #FFF
 * @type color
 */
```

```
ht.Default.accordionViewLabelColor = {};

/**
 * 折叠组件文字字体
 * @default 12px arial, sans-serif
 * @type String
 */
ht.Default.accordionViewLabelFont = {};

/**
 * 折叠组件抬头背景
 * @default #2C3E50
 * @type color
 */
ht.Default.accordionViewTitleBackground = {};

/**
 * 折叠组件选中背景
 * @default #1ABC9C
 * @type color
 */
ht.Default.accordionViewSelectBackground = {};

/**
 * 折叠组件选中宽度
 * @default 3
 * @type Number
 */
ht.Default.accordionViewSelectWidth = {};

/**
 * 折叠组件分隔条颜色
 * @type color
 */
ht.Default.accordionViewSeparatorColor = {};

/**
 * 分割组件分隔条宽度
 * @default 1
 * @type Number
 */
ht.Default.splitViewDividerSize = {};

/**
```

```
 * 分割组件分隔条背景
 * @default #2C3E50
 * @type color
 */
ht.Default.splitViewDividerBackground = {};

/**
 * 分割组件分隔条拖拽过程透明度
 * @default 0.5
 * @type Number
 */
ht.Default.splitViewDragOpacity = {};

/**
 * 分割组件展开合并图标
 * @type String
 */
ht.Default.splitViewToggleIcon = {};

/**
 * 属性组件文字颜色
 * @default #000
 * @type color
 */
ht.Default.propertyViewLabelColor = {};

/**
 * 属性组件文字选中颜色
 * @default #FFF
 * @type color
 */
ht.Default.propertyViewLabelSelectColor = {};

/**
 * 属性组件文字字体
 * @default 12px arial, sans-serif
 * @type String
 */
ht.Default.propertyViewLabelFont = {};

/**
 * 属性组件展开图标
 * @type String
 */
```

```
ht.Default.propertyViewExpandIcon = {};

/**
 * 属性组件合并图标
 * @type String
 */
ht.Default.propertyViewCollapseIcon = {};

/**
 * 属性组件背景
 * @default #ECF0F1
 * @type color
 */
ht.Default.propertyViewBackground = {};

/**
 * 属性组件行线是否可见
 * @default true
 * @type Boolean
 */
ht.Default.propertyViewRowLineVisible = {};

/**
 * 属性组件列线是否可见
 * @default true
 * @type Boolean
 */
ht.Default.propertyViewColumnLineVisible = {};

/**
 * 属性组件行线颜色
 * @default #D9D9D9
 * @type color
 */
ht.Default.propertyViewRowLineColor = {};

/**
 * 属性组件列线颜色
 * @default #D9D9D9
 * @type color
 */
ht.Default.propertyViewColumnLineColor = {};

/**
```

```
 * 属性组件选中背景色
 * @default #1ABC9C
 * @type color
 */
ht.Default.propertyViewSelectBackground = {};

/**
 * 列表组件文字颜色
 * @default #000
 * @type color
 */
ht.Default.listViewLabelColor = {};

/**
 * 列表组件文字选中颜色
 * @default #FFF
 * @type color
 */
ht.Default.listViewLabelSelectColor = {};

/**
 * 列表组件文字字体
 * @default 12px arial, sans-serif
 * @type String
 */
ht.Default.listViewLabelFont = {};

/**
 * 列表组件行线是否可见
 * @default false
 * @type Boolean
 */
ht.Default.listViewRowLineVisible = {};

/**
 * 列表组件行线颜色
 * @default #D9D9D9
 * @type color
 */
ht.Default.listViewRowLineColor = {};

/**
 * 列表组件选中背景色
 * @default #1ABC9C
```

```
 * @type color
 */
ht.Default.listViewSelectBackground = {};

/**
 * 树组件文字颜色
 * @default #000
 * @type color
 */
ht.Default.treeViewLabelColor = {};

/**
 * 树组件文字选中颜色
 * @default #FFF
 * @type color
 */
ht.Default.treeViewLabelSelectColor = {};

/**
 * 树组件文字字体
 * @default 12px arial, sans-serif
 * @type String
 */
ht.Default.treeViewLabelFont = {};

/**
 * 树组件展开状态图标
 * @type String
 */
ht.Default.treeViewExpandIcon = {};

/**
 * 树组件关闭状态图标
 * @type String
 */
ht.Default.treeViewCollapseIcon = {};

/**
 * 树组件行线是否可见
 * @default false
 * @type Boolean
 */
ht.Default.treeViewRowLineVisible = {};
```

```
/**
 * 树组件行线颜色
 * @default #D9D9D9
 * @type color
 */
ht.Default.treeViewRowLineColor = {};

/**
 * 树组件选中背景色
 * @default #1ABC9C
 * @type color
 */
ht.Default.treeViewSelectBackground = {};

/**
 * 表格组件文字颜色
 * @default #000
 * @type color
 */
ht.Default.tableViewLabelColor = {};

/**
 * 表格组件文字选中颜色
 * @default #FFF
 * @type color
 */
ht.Default.tableViewLabelSelectColor = {};

/**
 * 表格组件文字字体
 * @default 12px arial, sans-serif
 * @type String
 */
ht.Default.tableViewLabelFont = {};

/**
 * 表格组件行线是否可见
 * @default true
 * @type Boolean
 */
ht.Default.tableViewRowLineVisible = {};

/**
 * 表格组件列线是否可见
```

```
 * @default true
 * @type Boolean
 */
ht.Default.tableViewColumnLineVisible = {};

/**
 * 表格组件行线颜色
 * @default #D9D9D9
 * @type color
 */
ht.Default.tableViewRowLineColor = {};

/**
 * 表格组件列线颜色
 * @default #D9D9D9
 * @type color
 */
ht.Default.tableViewColumnLineColor = {};

/**
 * 表格组件选中背景色
 * @default #1ABC9C
 * @type color
 */
ht.Default.tableViewSelectBackground = {};

/**
 * 树表组件文字颜色
 * @default #000
 * @type color
 */
ht.Default.treeTableViewLabelColor = {};

/**
 * 树表组件文字选中颜色
 * @default #FFF
 * @type color
 */
ht.Default.treeTableViewLabelSelectColor = {};

/**
 * 树表组件文字字体
 * @default 12px arial, sans-serif
 * @type String
```

```
 */
ht.Default.treeTableViewLabelFont = {};

/**
 * 树表格组件展开状态图标
 * @type String
 */
ht.Default.treeTableViewExpandIcon = {};

/**
 * 树表格组件关闭状态图标
 * @type String
 */
ht.Default.treeTableViewCollapseIcon = {};

/**
 * 树表格组件行线是否可见
 * @default true
 * @type Boolean
 */
ht.Default.treeTableViewRowLineVisible = {};

/**
 * 树表格组件列线是否可见
 * @default true
 * @type Boolean
 */
ht.Default.treeTableViewColumnLineVisible = {};

/**
 * 树表格组件行线颜色
 * @default #D9D9D9
 * @type color
 */
ht.Default.treeTableViewRowLineColor = {};

/**
 * 树表格组件列线颜色
 * @default #D9D9D9
 * @type color
 */
ht.Default.treeTableViewColumnLineColor = {};

/**
```

```
 * 树表组件选中背景色
 * @default #1ABC9C
 * @type color
 */
ht.Default.treeTableViewSelectBackground = {};

/**
 * 表头组件文字颜色
 * @default #000
 * @type color
 */
ht.Default.tableHeaderLabelColor = {};

/**
 * 表头组件文字字体
 * @default 12px arial, sans-serif
 * @type String
 */
ht.Default.tableHeaderLabelFont = {};

/**
 * 表头组件列线是否可见
 * @default true
 * @type Boolean
 */
ht.Default.tableHeaderColumnLineVisible = {};

/**
 * 表头组件列线颜色
 * @default #D9D9D9
 * @type color
 */
ht.Default.tableHeaderColumnLineColor = {};

/**
 * 表头组件背景
 * @default #ECF0F1
 * @type color
 */
ht.Default.tableHeaderBackground = {};

/**
 * 表头组件移动状态背景
 * @default rgba(0,0,0,0.35)
```

```
 * @type color
 */
ht.Default.tableHeaderMoveBackground = {};

/**
 * 表头组件插入状态颜色
 * @default #1ABC9C
 * @type color
 */
ht.Default.tableHeaderInsertColor = {};

/**
 * 表头组件降序图标
 * @type String
 */
ht.Default.tableHeaderSortDescIcon = {};

/**
 * 表头组件升序图标
 * @type String
 */
ht.Default.tableHeaderSortAscIcon = {};

/**
 * 页签组件间距
 * @default 1
 * @type Number
 */
ht.Default.tabViewTabGap = {};

/**
 * 页签组件文字颜色
 * @default #FFF
 * @type color
 */
ht.Default.tabViewLabelColor = {};

/**
 * 页签组件文字字体
 * @default 12px arial, sans-serif
 * @type String
 */
ht.Default.tabViewLabelFont = {};
```

```
/**
 * 页签组件背景
 * @default #2C3E50
 * @type color
 */
ht.Default.tabViewTabBackground = {};

/**
 * 页签组件选中宽度
 * @default 3
 * @type Number
 */
ht.Default.tabViewSelectWidth = {};

/**
 * 页签组件选中背景
 * @default #1ABC9C
 * @type color
 */
ht.Default.tabViewSelectBackground = {};

/**
 * 页签组件移动状态背景
 * @default rgba(0,0,0,0.35)
 * @type color
 */
ht.Default.tabViewMoveBackground = {};

/**
 * 页签组件插入状态颜色
 * @default #1ABC9C
 * @type color
 */
ht.Default.tabViewInsertColor = {};

/**
 * 工具条组件文字颜色
 * @default #000
 * @type color
 */
ht.Default.toolbarLabelColor = {};

/**
 * 工具条组件文字选中颜色
```

```
 * @default #FFF
 * @type color
 */
ht.Default.toolbarLabelSelectColor = {};

/**
 * 工具条组件文字字体
 * @default 12px arial, sans-serif
 * @type String
 */
ht.Default.toolbarLabelFont = {};

/**
 * 工具条组件背景
 * @default #ECF0F1
 * @type color
 */
ht.Default.toolbarBackground = {};

/**
 * 工具条组件选中背景色
 * @default #1ABC9C
 * @type color
 */
ht.Default.toolbarSelectBackground = {};

/**
 * 工具条组件 Item 的间距
 * @default 8
 * @type Number
 */
ht.Default.toolbarItemGap = {};

/**
 * 工具条组件的分隔条颜色
 * @default #868686
 * @type color
 */
ht.Default.toolbarSeparatorColor = {};


/**
 * 数字类型监听器，该监听器将使得 input 文本输入框只允许输入数学相关字符
 * @type Function
```

```
 * @example
 * numberListener: (function(){
 *      var map = {
 *          46: 1,
 *          8: 1,
 *          9: 1,
 *          27: 1,
 *          13: 1,
 *          109: 1,
 *          110: 1,
 *          189: 1,
 *          190: 1
 *      };
 *      return function(e){
 *          var keyCode = e.keyCode;
 *          if(map[keyCode] || (keyCode === 65 && Default.isCtrlDown(e)) || (keyCode >= 35
&& keyCode <= 39)){
 *              return;
 *          }
 *          if ((e.shiftKey || (keyCode < 48 || keyCode > 57)) && (keyCode < 96 || keyCode >
105)) {
 *              e.preventDefault();
 *          }
 *      };
 *  })()
 */
ht.Default.numberListener = {}

/**
 * 无法加载图片资源时会调用该函数，默认访问空，可自定义返回一个默认的图片
 * @type Function
 * @example
 * ht.Default.handleImageLoaded = function(name, url) {
 *
 * }
 */
ht.Default.handleUnfoundImage = {}

/**
 * 图片在加载之后的回调函数
 * @type Function
 * @example
 * ht.Default.handleImageLoaded = function(name, img) {
 *
```

```
 * }
 */
ht.Default.handleImageLoaded = {}

/**
 * 默认排序函数
 * @type Function
 * @example
 * ht.Default.sortFunc = function(v1, v2) {
 *
 * }
 */
ht.Default.sortFunc = {}

/**
 * 获得 HT 的版本号
 * @return {String} 版本号
 */
ht.Default.getVersion = function() {}

/**
 * 注册矢量组件类型
 * @param {String} type 类型名称
 * @param {Function} func 绘制函数，例：function(g, rect, comp, data, view){}
 */
ht.Default.setCompType = function(type, func) {}

/**
 * 获得注册的矢量组件类型
 * @param {String} type 类型名称
 */
ht.Default.getCompType = function(type) {}

/**
 * 阻止事件的默认行为，常用于屏蔽触屏上默认 DoubleTap 缩放等行为
 * @param {Event} e 事件对象
 */
ht.Default.preventDefault = function(e) {}

/**
 * 获取当前窗口 left|top|width|height 的参数信息
 *
 * 返回的对象格式如下：
 * @example {
```

```
 *   left: 0,
 *   top: 0,
 *   width: 1280
 *   height: 768
 * }
 */
ht.Default.getWindowInfo = function() {}

/**
 * 判断目前是否处于拖拽状态
 * @return {boolean} 是否处于拖拽状态
 */
ht.Default.isDragging = function() {}

/**
 * 判断是否鼠标左键被按下
 * @param {Event} e 事件对象
 * @return {boolean} 鼠标左键是否被按下
 */
ht.Default.isLeftButton = function(e) {}

/**
 * 获取当前 Touch 手指个数
 * @param {Event} e 事件对象
 * @return {Number} Touch 手指个数
 */
ht.Default.getTouchCount = function(e) {}

/**
 * 判断是否为双击事件
 * @param {Event} e 事件对象
 * @return {boolean} 是否是双击
 */
ht.Default.isDoubleClick = function(e) {}

/**
 * 判断 Shift 键是否被按下
 * @param {Event} e 事件对象
 * @return {boolean} Shift 键是否被按下
 */
ht.Default.isShiftDown = function(e) {}

/**
 * 判断 Ctrl 键是否被按下
```

```
 * @param {Event} e 事件对象
 * @return {boolean} Ctrl 键是否被按下
 */
ht.Default.isCtrlDown = function(e) {}

/**
 * 返回 client 属性坐标
 * @param {Event} e 事件对象
 * @return {Object} client 坐标对象
 * @example //返回值示例:
 * {
 *    x: 100,
 *    y: 100
 * }
 */
ht.Default.getClientPoint = function(e) {}

/**
 * 返回 page 属性坐标
 * @param {Event} e 事件对象
 * @return {Object} page 坐标对象
 * @example //返回值示例:
 * {
 *    x: 100,
 *    y: 100
 * }
 */
ht.Default.getPagePoint = function(e) {}

/**
 * 注册图片
 * @param {String} name 图片名
 * @param {Number} [width] 图片宽度
 * @param {Number} [height] 图片高度
 * @param {HTMLImageElement|HTMLCanvasElement|String|Object} img img、canvas 对象或图片
url 或 base64 字符串或矢量对象
 */
ht.Default.setImage = function(name, width, height, img) {}

/**
 * 获得已注册的图片
 * @param {String} name 图片名
 * @param {color} [color] 染色
 * @return {HTMLImageElement|HTMLCanvasElement|Object} 返回已经注册的图片
```

```
 */
ht.Default.getImage = function(name, color) {}

/**
 * 获取全局下一个 id 编号
 * @return {Number} id
 */
ht.Default.getId = function() {}

/**
 * 获取全局下一个 id 编号
 * @param {Function} func 回调函数
 * @param {Object} scope 函数域
 * @param {Array} args 函数参数列表
 * @param {Number} delay 延迟时间(毫秒)
 */
ht.Default.callLater = function(func, scope, args, delay) {}

/**
 * 传入一个对象参数，以浅拷贝的方式返回一个新的复制对象
 * @param {Object} obj 要复制的对象
 * @return {Object} 新复制的对象
 */
ht.Default.clone = function(obj) {}

/**
 * 返回所有 HT 预定义类的 json 结构信息，key 为类全路径名，value 为类声明(构造函数)
 * @return {Object} 类结构信息
 */
ht.Default.getClassMap = function() {}

/**
 * 传入全路径类字符串名称，返回类定义(构造函数)
 * @param {String} name 类名
 * @return {Function} 类定义(构造函数)
 */
ht.Default.getClass = function(name) {}

/**
 * 定义类
 * @param {String|Object} className 类名，如果为字符串，自动注册到 HT 的 classMap 中，一般使用函数(构造函数)即可
 * @param {Object} superClass 要继承的父类
 * @param {Object} methods 方法和变量声明
```

```
 * @example
 * function MyData() {
 *  MyData.superClass.constructor.call(this);
 * }
 * ht.Default.def(MyData, ht.Data, {
 *  sayHello: function() {
 *      console.log('hello');
 *  }
 * });
 */
ht.Default.def = function(className, superClass, methods) {}

/**
 * 启动动画
 * @param {Object} paramObj 动画配置对象，请参考入门手册中的动画属性
 * @example
 * ht.Default.startAnim({
 *  frames: 60,
 *  interval: 16,
 *  finishFunc: function() {
 *      console.log('finish');
 *  },
 *  action: function(t) {
 *      console.log(t);
 *  }
 * });
 */
ht.Default.startAnim = function(paramObj) {}

/**
 * 计算文字的尺寸
 * @param {String} font 文字字体，如:12px Arial
 * @param {String} text 文字内容
 * @return {Object} 文字尺寸
 * @example //返回值示例:
 * {
 *  width: 100,
 *  height: 100
 * }
 *
 */
ht.Default.getTextSize = function(font, text) {}

/**
```

```
 * 绘制文字
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {String} value 文字内容
 * @param {String} font 文字字体
 * @param {color} color 文字颜色
 * @param {Number} x 绘制开始的 x 坐标
 * @param {Number} y 绘制开始的 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 * @param {String} align 文字水平对齐方式，可选值为 left|center|right
 * @param {String} vAlign 文字垂直对齐方式，可选值为 top|middle|bottom
 * @example ht.Default.drawText(g, 'Hello, HT', '12px Arial', 0, 0, 200, 100, 'center',
'middle');
 */
ht.Default.drawText = function(g, value, font, color, x, y, width, height, align, vAlign) {}

/**
 * 获取两点之间距离，或向量长度
 *@param {Object|Array} p1 第一个点的坐标(格式：{x: x, y: y})或第一个向量(格式：[x, y, z])
 *@param {Object|Array} p2 第二个点的坐标(格式：{x: x, y: y})或第二个向量(格式：[x, y, z])
 * @return {Number} 距离
 * @example //二维两点距离
 * var distance = ht.Default.getDistance({x: 0, y: 0}, {x: 10, y: 0});// distance equals
10
 * //三维两点距离
 * var distance3d = ht.Default.getDistance([0, 0, 0], [0, 10, 0]);// distance3d equals 10
 */
ht.Default.getDistance = function(p1, p2) {}

/**
 * 返回比 color 更亮的颜色
 * @param {color} color 原始颜色
 * @param {Number} factor 变化因子，默认为 40，允许值 0~100
 * @example
 * var newColor = ht.Default.brighter('#f00');
 * @return {color} 新的颜色
 */
ht.Default.brighter = function(color, factor) {}

/**
 * 返回比 color 更暗的颜色
 * @param {color} color 原始颜色
 * @param {Number} factor 变化因子，默认为 40，允许值 0~100
 * @return {color} 新的颜色
```

```
 * @example
 * var newColor = ht.Default.darker('#f00');
 */
ht.Default.darker = function(color, factor) {}

/**
 * 将点组合成矩形
 * @param {Object} p1 第一个点或点数组
 * @param {Object} p2 第二个点
 * @return {Object} 组合的矩形
 * @example //组合两点：
 * var rect = ht.Default.unionPoint({x: 0, y: 0}, {x: 100, y: 100});
 * //rect 结果：
 * {
 *   x: 0,
 *   y: 0,
 *   width: 100,
 *   height: 100
 * }
 * 组合多点：
 * var rect = ht.Default.unionPoint([{x: 0, y: 0}, {x: 50, y: 50}, {x: 100, y: 100}]);
 * //rect 结果：
 * {
 *   x: 0,
 *   y: 0,
 *   width: 100,
 *   height: 100
 * }
 */
ht.Default.unionPoint = function(p1, p2) {}

/**
 * 将两个矩形区域 union 融合成新的矩形区域
 * @param {Object} rect1 第一个矩形区域
 * @param {Object} rect2 第二个矩形区域
 * @return {Object} 新的矩形区域
 * @example var rect = ht.Default.unionRect(
 * {x: 0, y: 0, width: 100, height: 100},
 * {x: 0, y: 0, width: 200, height: 200});
 * //rect 结果：
 * {
 *   x: 0,
 *   y: 0,
 *   width: 200,
```

```
 *  height: 200
 * }
 */
ht.Default.unionRect = function(rect1, rect2) {}

/**
 * 判断 point 是否包含在 rect 的矩形区域里
 * @param {Object} rect 矩形
 * @param {Object} point 点
 * @return {Boolean} 矩形是否包含点
 * @example ht.Default.containsPoint({x: 0, y: 0, width: 100, height: 100}, {x: 50, y:
50})//true
 */
ht.Default.containsPoint = function(rect, point) {}

/**
 * 判断矩形区域 rect1 是否包含矩形区域 rect2
 * @param {Object} rect1 矩形 1
 * @param {Object} rect2 矩形 2
 * @return {Boolean} 矩形 1 是否包含矩形 2
 *@example ht.Default.containsRect({x: 0, y: 0, width: 100, height: 100}, {x: 0, y: 0, width:
10, height: 10})//true
 */
ht.Default.containsRect = function(rect1, rect2) {}

/**
 * 判断矩形区域 rect1 和矩形区域 rect2 是否相交
 * @param {Object} rect1 矩形 1
 * @param {Object} rect2 矩形 2
 * @return {Boolean} 两个矩形是否相交
 * @example ht.Default.intersectsRect({x: 0, y: 0, width: 100, height: 100}, {x: 0, y: 0,
width: 200, height: 200})//true
 */
ht.Default.intersectsRect = function(rect1, rect2) {}

/**
 * 获得两个矩形的相交区域
 * @param {Object} rect1 矩形 1
 * @param {Object} rect2 矩形 2
 * @return {Object} 相交的矩形区域，如果没有相交，返回 null
 * @example var rect = ht.Default.intersection({x: 0, y: 0, width: 100, height: 100}, {x:
50, y: 50, width: 200, height: 200})
 * //rect 结果：
 * {
```

```
 *   x: 50,
 *   y: 50,
 *   width: 50,
 *   height: 50
 * }
 */
ht.Default.intersection = function(rect1, rect2) {}

/**
 * 改变 rect 大小，上下左右分别扩展 extend 的大小
 * @param {Object} rect 原始矩形
 * @param {Number} extend 扩展大小
 * @example var rect = {x: 0, y: 0, width: 100, height: 100};
 * ht.Default.grow(rect, 2)
 * //rect 结果：
 * {
 *   x: -2,
 *   y: -2,
 *   width: 104,
 *   height: 104
 * }
 */
ht.Default.grow = function(rect, extend) {}

/**
 * 获取交互点的逻辑坐标，使用视图对象上的此方法更为便捷
 * @param {Event} e 事件对象
 * @param {Object} view 视图对象
 * @param {Number} translateX 水平偏移
 * @param {Number} translateY 垂直偏移
 * @param {Number} zoomX 水平缩放
 * @param {Number} zoomY 垂直缩放
 * @return {Object} 逻辑点坐标
 */
ht.Default.getLogicalPoint = function(e, view, translateX, translateY, zoomX, zoomY) {}

/**
 * 删除指定的 DOM 对象
 * @param {Element} domElement DOM 对象
 * @return {Boolean} 操作是否成功
 */
ht.Default.removeHTML = function(domElement) {}

/**
```

```
 * 返回 ToolTip 的相应 div 对象，可获取进行风格自定义
 * @return {Element} ToolTip 相应的 div 对象
 */
ht.Default.getToolTipDiv = function() {}

/**
 * 判断 ToolTip 是否正在显示状态
 * @return {Boolean} ToolTip 是否显示
 */
ht.Default.isToolTipShowing = function() {}

/**
 * 隐藏正在显示的 ToolTip
 */
ht.Default.hideToolTip = function() {}

/**
 * 显示 ToolTip
 * @param {Event|Object} eventOrPoint 鼠标事件对象或点坐标
 * @param {String} innerHTML ToolTip 的内容
 */
ht.Default.showToolTip = function(eventOrPoint, innerHTML) {}

/**
 * 启动拖拽，自定义交互时可能用到
 * @param {Object} interactor 交互器
 * @param {Event} e 事件对象
 */
ht.Default.startDragging = function(interactor, e) {}

/**
 * 获得所有注册图片的信息对象
 * @return {Object} 已注册图片的映射表
 */
ht.Default.getImageMap = function() {}

/**
 * 将不连续曲线转化成 Graph3dView#setBoundaries(bs)需要的参数格式
 * @param {Array} points 曲线上的点坐标数组
 * @param {Array} segmets 曲线上的线段类型数组
 * @param {Number} resolution 曲线微分数
 * @return {Array} 适合 Graph3dView#setBoundaries(bs)需要的参数格式
 */
```

```
ht.Default.toBoundaries = function(points, segments, resolution) {}

/**
 * 返回当前键盘按键信息，key 为键的 keyCode，如果按下则值为 true
 * @return {Object} 键盘按键信息
 */
ht.Default.getCurrentKeyCodeMap = function() {}

/**
 * 以 x,y 为中心绘制 img 图片
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {HTMLImageElement|HTMLCanvasElement|Object} img 要绘制的图片(img 对象、canvas
对象或矢量对象)
 * @param {Number} x 中心点 x 坐标
 * @param {Number} y 中心点 y 坐标
 * @param {ht.Data} data 要绑定的 Data 对象
 * @param {Object} view 要绑定的视图对象
 * @param {color} color 染色
 */
ht.Default.drawCenterImage = function(g, img, x, y, data, view, color) {}

/**
 * 在矩形位置内绘制图片
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {HTMLImageElement|HTMLCanvasElement|Object} img 要绘制的图片(img 对象、canvas
对象或矢量对象)
 * @param {String} stretch 拉伸类型(uniform/centerUniform/fill)
 * @param {Number} x 矩形左上角 x 坐标
 * @param {Number} y 矩形左上角 y 坐标
 * @param {Number} w 矩形宽度
 * @param {Number} h 矩形高度
 * @param {ht.Data} data 要绑定的 Data 对象
 * @param {Object} view 要绑定的视图对象
 * @param {color} color 染色
 */
ht.Default.drawStretchImage = function(g, img, stretch, x, y, w, h, data, view, color) {}

/**
 * 将图片转换成 Canvas 对象
 * @param {HTMLImageElement|Object} image 要转换的图片(img 对象或矢量对象)
 * @param {Number} width 新 canvas 的宽度
 * @param {Number} height 新 canvas 的高度
 * @param {String} stretch 拉伸类型(uniform/centerUniform/fill)
 * @param {ht.Data} data 要绑定的 Data 对象
```

```
 * @param {Object} view 要绑定的视图对象
 * @param {color} color 染色
 * @return {HTMLCanvasElement} canvas 对象
 */
ht.Default.toCanvas = function(image, width, height, stretch, data, view, color) {}

/**
 * 创建 DOM 对象
 * @param {String} tagName DOM 类型(如 div、a、span 等)
 * @param {String} borderColor 边框颜色
 * @param {String} font 字体
 * @param {String} value 内容
 * @return {Element} DOM 对象
 */
ht.Default.createElement = function(tagName, borderColor, font, value) {}

/**
 * 判断交互事件所处位置是否在 View 组件之上，一般用于 Drog And Drop 的拖拽操作判断
 * @param {Event} event 事件对象
 * @param {Object} view 视图对象
 * @return {Boolean}
 */
ht.Default.containedInView = function(event, view) {}

/**
 * 判断目前系统是否处于隔离状态，处于隔离状态时 host 吸附和 Group 组等图元间的联动关系将会被停止
 * @return {Boolean} 是否处于隔离状态
 */
ht.Default.isIsolating = function() {}

/**
 * 设置系统的隔离状态，处于隔离状态时 host 吸附和 Group 组等图元间的联动关系将会被停止
 * @param {Boolean} isolating 新的状态
 */
ht.Default.setIsolating = function(isolating) {}

/**
 * 将颜色转换为 rgba 格式
 * @param {color} color 旧格式的颜色
 * @return {Array} rgba 格式的颜色
 */
ht.Default.toColorData = function(color) {}
```

```
/**
 * 绘制图片
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {HTMLImageElement|HTMLCanvasElement|Object} image 要绘制的图片(img 对象、canvas 对象或矢量对象)
 * @param {Number} x 绘制范围左上角 x 坐标
 * @param {Number} y 绘制范围左上角 y 坐标
 * @param {Number} width 绘制范围宽度
 * @param {Number} height 绘制范围高度
 * @param {ht.Data} data 要绑定的 Data 对象
 * @param {Object} view 要绑定的视图对象
 * @param {color} blendColor 染色
 */
ht.Default.drawImage = function(g, image, x, y, width, height, data, view, blendColor) {}

/**
 * 返回当前矢量组件 comp，即 ht.Default.compStack[0]，一般用于矢量值绑定 func 动态调用时使用
 * @return {Object} 矢量组件 comp
 */
ht.Default.getCurrentComp = function() {}

/**
 * 返回当前矢量组件上一层 comp，即 ht.Default.compStack[1]，一般用于矢量值绑定 func 动态调用时使用
 * @return {Object} 矢量组件 comp
 */
ht.Default.getParentComp = function() {}

/**
 * 注册连线类型
 * @param {String} type 连线类型名
 * @param {Function} func 决定连线走向的函数
 * @example ht.Default.setEdgeType('customEdge', function(edge, gap, graphView) {
 *          return {
 *            points: points;
 *            segments: segments;
 *          }
 * })
 */
ht.Default.setEdgeType = function(type, func) {}

/**
```

```
 * 获取连线类型函数
 * @param {String} type 连线类型名
 * @return {Function} 连线类型函数
 */
ht.Default.getEdgeType = function(type) {}

/**
 * 注册 3D 模型，请参考 modeling 建模手册
 * @param {String} name 模型名
 * @param {Object} model 模型内容
 */
ht.Default.setShape3dModel = function(name, model) {}

/**
 * 返回所注册的 3D 模型
 * @param {String} name 模型名
 * @return {Object} 模型
 */
ht.Default.getShape3dModel = function(name) {}

/**
 * 将一组 JSON 描述的缩放、移动和旋转等操作转换成对应的变化矩阵
 */
ht.Default.createMatrix = function(array, matrix) {}

/**
 * 将指定矢量或顶点，通过矩阵转换运算出变化后的新矢量或顶点位置
 */
ht.Default.transformVec = function(vec, matrix) {}

/**
 * 构建六面体模型，该模型的六个面显示的颜色和贴图都将一样
 */
ht.Default.createBoxModel = function() {}

/**
 * 构建圆角矩形体模型
 */
ht.Default.createRoundRectModel = function(top, bottom) {}

/**
 * 构建星形体模型
 */
```

```
ht.Default.createStarModel = function(top, bottom) {}

/**
 * 构建矩形体模型
 */
ht.Default.createRectModel = function(top, bottom) {}

/**
 * 构建三角形体模型
 */
ht.Default.createTriangleModel = function(top, bottom) {}

/**
 * 构建直角三角形体模型
 */
ht.Default.createRightTriangleModel = function(top, bottom) {}

/**
 * 构建平行四边形体模型
 */
ht.Default.createParallelogramModel = function(top, bottom) {}

/**
 * 构建梯形体模型
 */
ht.Default.createTrapezoidModel = function(top, bottom) {}

/**
 * 构建光滑球体模型
 */
ht.Default.createSmoothSphereModel = function(hResolution, vResolution, hStart,
hArc, vStart, vArc, radius) {}

/**
 * 构建球体模型
 */
ht.Default.createSphereModel = function(side, sideFrom, sideTo, from, to, resolution)
{}

/**
 * 构建光滑圆锥体模型
 */
```

```
ht.Default.createSmoothConeModel = function(bottom, resolution, start, arc, radius)
{}

/**
 * 构建圆锥体模型
 */
ht.Default.createConeModel = function(side, sideFrom, sideTo, from, to, bottom) {}

/**
 * 构建光滑圆柱体模型
 */
ht.Default.createSmoothCylinderModel = function(resolution, top, bottom,
topRadius, bottomRadius, start, arc, height) {}

/**
 * 构建圆柱体模型
 */
ht.Default.createCylinderModel = function(side, sideFrom, sideTo, from, to, top,
bottom) {}

/**
 * 构建光滑圆环体模型
 */
ht.Default.createSmoothTorusModel = function(radius, tubeRadius, hResolution,
vResolution, start, arc) {}

/**
 * 构建圆环体模型
 */
ht.Default.createTorusModel = function(side, sideFrom, sideTo, from, to, radius,
resolution) {}

/**
 * 根据 xz 平面多边形，挤压形成 3D 模型
 */
ht.Default.createExtrusionModel = function(array, segments, top, bottom, resolution,
repeatUVLength, tall, elevation) {}

/**
 * 根据 xy 平面的曲线，环绕一周形成光滑 3D 模型
 */
ht.Default.createSmoothRingModel = function(array, segments, vResolution, start,
arc, hResolution) {}
```

```
/**
 * 根据 xy 平面的曲线，环绕一周形成 3D 模型
 */
ht.Default.createRingModel = function(array, segments, resolution, top, bottom, side, sideFrom, sideTo, from, to) {}

/**
 * 注册 3d 图元的批量信息，用于优化大数据量图元绘制性能，详细用法请参考批量手册
 * @param {String} name 批量名
 * @param {Object} batchInfo 批量信息
 */
ht.Default.setBatchInfo = function(name, batchInfo) {}

/**
 * 获取已注册的批量信息
 * @param {String} name 批量名
 * @return {Oject} 批量信息
 */
ht.Default.getBatchInfo = function(name) {}
```

ht.graph.GraphView

```
/**
 * 拓扑图形组件 ht.graph.GraphView 是 HT 框架中 2D 功能最丰富的组件,其相关类库都在 ht.graph 包下。<br>
 * GraphView 具有基本图形的呈现和编辑、拓扑节点连线及自动布局功能;
 * 封装了电力和电信等行业预定义对象，具有动画渲染等特效， 因此其应用面很广泛，可作为监控领域的绘图工具和人机界面,或作为一般性的图形化编辑工具,或扩展成工作流和组织图等企业应用。
 * @param {ht.DataModel} dataModel 绑定的数据模型
 * @constructor
 */
ht.graph.GraphView = function(dataModel) {};

/**
 * 增加底层 Painter<br>
 * 拓扑组件上提供 Painter 接口，开发者可以使用 Canvas 的画笔对象自由绘制任意形状，底层 Painter 绘制在拓扑最下面
 * @param {Function} painter Painter 类
 * @example //Painter 示例:
 * function MyBottomPainter() {
 * }
 * ht.Default.def(MyBottomPainter, Object, {
```

```
 *  draw: function(g) {
 *      g.save();
 *      //draw...
 *      g.restore();
 *  }
 * });
 * graphView.addBottomPainter(MyBottomPainter);
 */
addBottomPainter = function(painter) {}

/**
 * 增加交互事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.graph.GraphView#mi mi}
 * @example //示例:
 * graphView.addInteractorListener(function(event) {
 *      //event 格式:
 *      {
 *          //clickData, doubleClickData, clickBackground, doubleClickBackground,
 *          //beginRectSelect, betweenRectSelect, endRectSelect, beginMove, betweenMove,
endMove,
 *          //beginPan, betweenPan, endPan, beginEditRect, betweenEditRect, endEditRect,
beginEditPoint, betweenEditPoint, endEditPoint
 *          //beginEditRotation, betweenEditRotation, endEditRotation, moveLeft,
moveRight, moveUp, moveDown, toggleNote, toggleNote2
 *          kind: 'clickData',//事件类型
 *          data: data,//事件相关的数据元素
 *          part: "part",//事件的区域,icon、label 等
 *          event: e//html 原生事件
 *      }
 * });
 */
addInteractorListener = function(listener, scope, ahead) {}

/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.graph.GraphView#mp mp}
 */
```

```
addPropertyChangeListener = function (listener, scope, ahead){};

/**
 * 增加顶层 Painter<br>
 * 拓扑组件上提供 Painter 接口，开发者可以使用 Canvas 的画笔对象自由绘制任意形状，顶层 Painter 绘制在拓扑最上面
 * @param {Function} painter Painter 类
 * @example //Painter 示例：
 * function MyTopPainter() {
 * }
 * ht.Default.def(MyTopPainter, Object, {
 *   draw: function(g) {
 *         g.save();
 *         //draw...
 *         g.restore();
 *   }
 * });
 * graphView.addTopPainter(MyTopPainter);
 */
addTopPainter = function (painter){};

/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 传入即将设置的水平平移值，返回最终设置值，可重载限制水平平移范围
 * @param {Number} value 原始水平平移值
 * @return {Number} 新的水平平移值
 */
adjustTranslateX = function (value){};

/**
 * 传入即将设置的垂直平移值，返回最终设置值，可重载限制垂直平移范围
 * @param {Number} value 原始垂直平移值
 * @return {Number} 新的垂直平移值
 */
adjustTranslateY = function (value){};

/**
```

```
 * 传入即将修改的缩放值，返回最终运行设置的缩放值，可重载进行自定义
 * @param {Number} value 原始缩放值
 * @return {Number} 最终缩放值
 */
adjustZoom = function (value){};

/**
 * 关闭 ToolTip 功能
 */
disableToolTip = function (){};

/**
 * 获取或设置数据模型，没有参数时相当于{@link ht.graph.GraphView#getDataModel
getDataModel}，有参数时相当于{@link ht.graph.GraphView#setDataModel setDataModel}
 * @param {ht.DataModel} [dataModel] 数据模型
 * @return {ht.DataModel} dataModel
 */
dm = function (dataModel){};

/**
 * 提供一个回调函数遍历此拓扑中的图元，与 DataModel 上的 each 方法不同，此方法还考虑了拓扑
中的 Layer，从低 Layer 向高 Layer 遍历
 * @param {Function} func 遍历函数
 * @param {Object} [scope] 函数域
 * @example graphView.each(function(data) {
 *   console.log(data);
 * });
 */
each = function (func, scope){};

/**
 * 启用 ToolTip
 */
enableToolTip = function (){};

/**
 * 缩放平移整个拓扑以展示所有的图元
 * @param {Boolean} [anim] 是否使用动画
 * @param {Number} [padding] 缩放后图元区域与拓扑边缘的距离，默认为 20
 * @param {Boolean} [notZoomIn] 是否将最小缩放值限定为 1
 */
fitContent = function (anim, padding, notZoomIn){};

/**
```

```
 * 缩放平移整个拓扑以展示参数 Data
 * @param {ht.Data} data 要显示的 data
 * @param {Boolean} [anim] 是否使用动画
 * @param {Number} [padding] 缩放后图元区域与拓扑边缘的距离，默认为 20
 * @param {Boolean} [notZoomIn] 是否将最小缩放值限定为 1
 * @param {Boolean} [retry] 当拓扑状态异常时，是否延时重试 fitData，默认为 true
 */
fitData = function (data, anim, padding, notZoomIn, retry){};

/**
 * 缩放平移整个拓扑以展示矩形范围内的图元
 * @param {Object} rect 矩形范围
 * @param {Boolean} [anim] 是否使用动画
 * @param {Boolean} [notZoomIn] 是否将最小缩放值限定为 1
 */
fitRect = function (rect, anim, notZoomIn){};

/**
 * 获取自动滚动区域，当鼠标距离拓扑边缘小于这个值时，拓扑自动滚动(调整 translateX 或 translateY)
 * @return {Number} 自动滚动区域
 */
getAutoScrollZone = function (){};

/**
 * 获取图元 body 的染色，可重载此方法返回自定义的颜色
 * @param {ht.Data} data 要染色的图元
 * @return {color} 最终颜色，默认为 data.s('body.color')
 */
getBodyColor = function (data){};

/**
 * 获取图元边框颜色，可重载此方法返回自定义的颜色
 * @param {ht.Data} data 要染色的图元
 * @return {color} 最终颜色，默认为 data.s('border.color')
 */
getBorderColor = function (data){};

/**
 * 获取 Group 内 child 的尺寸范围，这个尺寸参与计算 Group 的大小
 * @param {ht.Data} child 子节点
 * @return {Object} 子节点的尺寸范围
 */
```

```
getBoundsForGroup = function (child){};

/**
 * 获取拓扑的画布
 * @return {HTMLCanvasElement} 画布
 */
getCanvas = function (){};

/**
 * 获取拓扑中所有图元占用的矩形区域
 * @return {Object} 内容区域
 */
getContentRect = function (){};

/**
 * 获取当前子网
 * @return {ht.SubGraph} 子网对象
 */
getCurrentSubGraph = function (){};

/**
 * 传入逻辑坐标点或者交互 event 事件参数，返回当前点下的图元，filter 可进行过滤
 * @param {Object|Event} pointOrEvent 逻辑坐标点或交互事件对象(如鼠标事件对象)
 *@param {Functoin}[filter] 过滤器函数，传入 data,自定义逻辑返回 true 或 false 判断此 data 是否可被 getDataAt 返回
 * @param {Number} [range] 扩大点范围
 * @return {ht.Data} 点下的图元
 */
getDataAt = function (pointOrEvent, filter, range){};

/**
 * 获取绑定的数据模型
 * @return {ht.DataModel} 数据模型
 */
getDataModel = function (){};

/**
 * 获取逻辑坐标区域内的图元
 * @param {rect} rect 逻辑坐标区域
 *@param {Boolean}[intersects] 指定相交选中还是包含选中，true 表示相交，false 表示包含。
 *@param {Boolean}[selectable] 是否只返回可被选中的图元，可否被选中通过 isSelectable 判断
 * @return {ht.List}
 */
```

```
getDatasInRect = function (rect, intersects, selectable){};

/**
 * 获取图元的 UI 类
 * @param {ht.Data} data 图元
 * @return {Object}
 */
getDataUI = function (data){};

/**
 * 获取图元 UI 的绘制范围
 * @param {ht.Data} data 图元
 * @return {Object}
 */
getDataUIBounds = function (data){};

/**
 * 获取编辑过滤器函数
 * @return {Function}
 */
getEditableFunc = function (){};

/**
 * 获取编辑交互器
 * @return {ht.graph.EditInteractor|ht.graph.XEditInteractor}
 */
getEditInteractor = function (){};

/**
 * 获取编辑交互器中编辑点的背景色
 * @return {color}
 */
getEditPointBackground = function (){};

/**
 * 获取编辑交互器中编辑点的边框颜色
 * @return {color}
 */
getEditPointBorderColor = function (){};

/**
 * 获取编辑交互器中编辑点的尺寸
 * @return {Object}
 */
```

```
getEditPointSize = function (){};

/**
 * 获取拓扑组件的布局高度
 * @return {Number}
 */
getHeight = function (){};

/**
 * 传入逻辑坐标点或者交互 event 事件参数、图元对象，判断当前点下的 icon 信息
 * @param {Object|Event} pointOrEvent 逻辑坐标点或交互事件对象(如鼠标事件对象)
 * @example //返回值示例：
 * {
 *  data: data,//相关数据元素
 *  key: 'key',//styleIcon 名
 *  index: 0,//styleIcon 中第几个 icon
 *  name: 'name'//styleIcon 中相应 icon 的名字
 * }
 *
 * @return {Object}
 */
getIconInfoAt = function (pointOrEvent, data){};

/**
 * 获取交互器
 * @return {ht.List}
 */
getInteractors = function (){};

/**
 * 获取图元的 label，用于在拓扑上显示文字信息，可重载返回自定义文字
 * @param {ht.Data} data 图元
 * @return {String} 图元 label 文字，默认返回 data.s('label')||data.getName();
 */
getLabel = function (data){};

/**
 * 获取图元的第二个 label，用于在拓扑上显示文字，可重载返回自定义文字
 * @param {ht.Data} data 图元
 * @return {String} 图元第二个 label 的文字，默认返回 data.s('label2')
 */
getLabel2 = function (data){};

/**
```

```
 * 获取图元的第二个 label 的背景色，可重载返回自定义颜色
 * @param {ht.Data} data 图元
 * @return {color} 图元第二个 label 的背景色，默认返回 data.s('label2.background')
 */
getLabel2Background = function (data){};

/**
 * 获取图元的第二个 label 的文字颜色，可重载返回自定义颜色
 * @param {ht.Data} data 图元
 * @return {color} 图元第二个 label 的文字颜色，默认返回 data.s('label2.color')
 */
getLabel2Color = function (data){};

/**
 * 获取图元 label 的背景色，可重载返回自定义颜色
 * @param {ht.Data} data 图元
 * @return {color} 图元 label 的背景色，默认返回 data.s('label.background')
 */
getLabelBackground = function (data){};

/**
 * 获取图元 label 的文字颜色，可重载返回自定义颜色
 * @param {ht.Data} data 图元
 * @return {color} 图元 label 的文字颜色，默认返回 data.s('label.color')
 */
getLabelColor = function (data){};

/**
 * 获取拓扑中已定义的层
 * @return {Array}
 */
getLayers = function (){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为拓扑中的逻辑坐标
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.graph.GraphView#lp lp}
 */
getLogicalPoint = function (event){};

/**
 * 获取移动过滤器函数
 * @return {Function}
```

```
 */
getMovableFunc = function (){};

/**
 * 获取图元的 note，用于在拓扑上显示标注信息，可重载返回自定义文字
 * @param {ht.Data} data 图元
 * @return {String} 图元 note 文字，默认返回 data.s('note')
 */
getNote = function (){data};

/**
 * 获取图元的第二个 note，用于在拓扑上显示标注信息，可重载返回自定义文字
 * @param {ht.Data} data 图元
 * @return {String} 图元第二个 note 文字，默认返回 data.s('note2')
 */
getNote2 = function (){data};

/**
 * 获取图元的第二个 note 的背景色，可重载返回自定义颜色
 * @param {ht.Data} data 图元
 * @return {color} 图元第二个 note 的背景色，默认返回 data.s('note2.background')
 */
getNote2Background = function (){data};

/**
 * 获取图元 note 的文字颜色，可重载返回自定义颜色
 * @param {ht.Data} data 图元
 * @return {color} 图元 note 的文字颜色，默认返回 data.s('note.background')
 */
getNoteBackground = function (data){};

/**
 * 获取图元的透明度，可重载返回自定义透明度
 * @param {ht.Data} data 图元
 * @return {Number} 图元透明度，默认返回 data.s('opacity')
 */
getOpacity = function (data){};

/**
 * 获取点编辑(Shape、Edge 等)过滤器函数
 * @return {Function}
 */
getPointEditableFunc = function (){};
```

```
/**
 * 获取大小编辑过滤器函数
 * @return {Function}
 */
getRectEditableFunc = function (){};

/**
 * 获取框选选择框的背景色
 * @return {color}
 */
getRectSelectBackground = function (){};

/**
 * 获取框选选择框的边框颜色
 * @return {color}
 */
getRectSelectBorderColor = function (){};

/**
 * 获取旋转编辑过滤器函数
 * @return {Function}
 */
getRotationEditableFunc = function (){};

/**
 * 获取图元编辑时的旋转控制点坐标
 * @param {ht.Data} data 图元
 * @return {Object} 旋转控制点坐标
 */
getRotationPoint = function (data){};

/**
 * 获取滚动条颜色
 * @return {color}
 */
getScrollBarColor = function (){};

/**
 * 获取滚动条宽度
 * @return {Number}
 */
getScrollBarSize = function (){};

/**
```

```
 * 获取拓扑的滚动区域，即 contentRect + viewRect
 * @return {Object} 矩形区域
 */
getScrollRect = function (){};

/**
 * 获取选择过滤器函数
 * @return {Function}
 */
getSelectableFunc = function (){};

/**
 * 传入逻辑坐标点或者交互 event 事件参数，返回当前点下已选中的图元
 * @param {Object|Event} pointOrEvent 逻辑坐标点或交互事件对象(如鼠标事件对象)
 * @return {ht.Data}
 */
getSelectedDataAt = function (pointOrEvent){};

/**
 * 获取选中模型
 * @return {ht.SelectionModel}
 * @see {@link ht.graph.GraphView#sm sm}
 */
getSelectionModel = function (){};

/**
 * 获取 ToolTip 文字，可重载返回自定义的 toolTip 文字
 * @param {Event} e 鼠标或 Touch 事件对象
 * @return {String} toolTip 文字，默认取出鼠标下的图元，然后返回其 getToolTip()
 */
getToolTip = function (e){};

/**
 * 获取水平平移值
 * @return {Number} 水平平移值
 * @see {@link ht.graph.GraphView#tx tx}
 */
getTranslateX = function (){};

/**
 * 获取垂直平移值
 * @return {Number} 垂直平移值
 * @see {@link ht.graph.GraphView#ty ty}
 */
```

```
getTranslateY = function (){};

/**
 * 获取拓扑组件的根层 div
 * @return {HTMLDivElement}
 */
getView = function (){};

/**
 * 获取拓扑组件中可见区域的逻辑尺寸
 * @return {Object}
 */
getViewRect = function (){};

/**
 * 获取可见过滤器函数
 * @return {Function}
 */
getVisibleFunc = function (){};

/**
 * 获取拓扑组件的布局宽度
 * @return {Number}
 */
getWidth = function (){};

/**
 * 获取拓扑整体缩放值
 * @return {Number}
 */
getZoom = function (){};

/**
 * 无效拓扑，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.graph.GraphView#iv iv}
 */
invalidate = function (delay){};

/**
 * 无效拓扑中的所有图元
 */
invalidateAll = function (){};
```

```
/**
 * 无效拓扑中的图元
 * @param {ht.Data} data 要无效的图元
 */
invalidateData = function (data){};

/**
 * 无效选中模型中的图元
 */
invalidateSelection = function (){};

/**
 * 是否自动隐藏滚动条
 * @return {Boolean}
 */
isAutoHideScrollBar = function (){};

/**
 * 选中图元时，是否自动平移拓扑以确保该图元出现在可见区域内
 * @return {Boolean}
 */
isAutoMakeVisible = function (){};


/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
isDisabled = function (){};

/**
 * 判断图元是否可被编辑
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isEditable = function (data){};

/**
 * 图元编辑点是否可见，默认当拓扑缩放值大于 0.15 时可见
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isEditVisible = function (data){};
```

```
/**
 * 判断图元 label 是否可见，默认当拓扑缩放值大于 0.15 时可见
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isLabelVisible = function (data){};

/**
 * 判断图元是否可移动
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isMovable = function (data){};

/**
 * 判断图元 note 是否可见，默认当拓扑缩放值大于 0.15 时可见
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isNoteVisible = function (data){};

/**
 * 拓扑是否可以通过鼠标拖拽进行平移操作
 * @return {Boolean}
 */
isPannable = function (){};

/**
 * 判断图元(Shape、Edge 等)的点是否可编辑
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isPointEditable = function (data){};

/**
 * 判断图元大小是否可编辑
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isRectEditable = function (data){};

/**
 * 判断拓扑上是否允许框选操作
 * @return {Boolean}
```

```
 */
isRectSelectable = function (){};

/**
 * 判断拓扑上是否允许通过空格将拓扑的缩放和平移值复位
 * @return {Boolean}
 */
isResettable = function (){};

/**
 * 判断图元是否可编辑旋转
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isRotationEditable = function (data){};

/**
 * 判断拓扑滚动条是否可见
 * @return {Boolean}
 */
isScrollBarVisible = function (){};

/**
 * 判断图元是否可被选中
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isSelectable = function (data){};

/**
 * 判断图元是否被选中
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isSelected = function (data){};

/**
 * 根据 id 判断图元是否被选中
 * @param {String|Number} id 图元 id
 * @return {Boolean}
 */
isSelectedById = function (id){};

/**
```

```
 * 当前拓扑是否共享选中模型
 * @return {Boolean}
 */
isSelectionModelShared = function (){};

/**
 * 拓扑中的图元的选中边框是否可见，默认当拓扑缩放值大于 0.15 时可见
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isSelectVisible = function (data){};

/**
 * 判断图元是否可见
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isVisible = function (data){};

/**
 * 无效拓扑，并调用延时刷新，{@link ht.graph.GraphView#invalidate invalidate}的缩写
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.graph.GraphView#invalidate invalidate}
 */
iv = function (delay){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为拓扑中的逻辑坐标，{@link
ht.graph.GraphView#getLogicalPoint getLogicalPoint}的缩写
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.graph.GraphView#getLogicalPoint getLogicalPoint}
 */
lp = function (event){};

/**
 * 平移拓扑以确保该图元在可见区域内
 * @param {ht.Data} data 图元
 */
makeVisible = function (data){};

/**
 * 增加交互事件监听器，{@link ht.graph.GraphView#addInteractorListener
addInteractorListener}的缩写
```

```
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.graph.GraphView#addInteractorListener addInteractorListener}
 * @example //示例:
 * graphView.mi(function(event) {
 *       //event 格式:
 *       {
 *           //clickData, doubleClickData, clickBackground, doubleClickBackground,
 *           //beginRectSelect, betweenRectSelect, endRectSelect, beginMove, betweenMove,
endMove,
 *           //beginPan, betweenPan, endPan, beginEditRect, betweenEditRect, endEditRect,
beginEditPoint, betweenEditPoint, endEditPoint
 *           //beginEditRotation, betweenEditRotation, endEditRotation, moveLeft,
moveRight, moveUp, moveDown, toggleNote, toggleNote2
 *           kind: 'clickData',//事件类型
 *           data: data,//事件相关的数据元素
 *           part: "part",//事件的区域,icon、label 等
 *           event: e//html 原生事件
 *       }
 * });
 */
mi = function (listener, scope, ahead){};

/**
 * 移动选中模型中图元的位置
 * @param {Number} offsetX 水平移动值
 * @param {Number} offsetY 垂直移动值
 */
moveSelection = function (offsetX, offsetY){};

/**
 * 增加自身属性变化事件监听器，{@link ht.graph.GraphView#addPropertyChangeListener
addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.graph.GraphView#addPropertyChangeListener addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};

/**
 * 自动布局动画结束后时回调，可重载做后续处理
 */
```

```
onAutoLayoutEnded = function (){};

/**
 * 单击拓扑背景时回调，可重载做后续处理
 * @param {Event} event 事件对象
 */
onBackgroundClicked = function (event){};

/**
 * 双击拓扑背景时回调，默认调用 upSubGraph()进入上一层子网，可重载改变默认逻辑或做后续处理
 * @param {Event} event 事件对象
 */
onBackgroundDoubleClicked = function (event){};

/**
 * 当前子网变化时回调，默认实现调用 reset()恢复默认缩放和平移值，可重载改变默认逻辑或做后续处理
 * @param {Event} event 事件对象
 */
onCurrentSubGraphChanged = function (event){};

/**
 * 图元被点击时回调，可重载做后续处理
 * @param {ht.Data} data 被点击的图元
 * @param {Event} e 事件对象
 */
onDataClicked = function (data, e){};

/**
 * 图元被双击时回调，可重载做后续处理
 * @param {ht.Data} data 双击的图元
 * @param {Event} e 事件对象
 */
onDataDoubleClicked = function (data, e){};

/**
 * 连线图元被双击时回调，默认调用 edge.toggle()，可重载改变默认逻辑或做后续处理
 * @param {ht.Edge} edge 连线
 * @param {Event} e 事件对象
 */
onEdgeDoubleClicked = function (edge, e){};

/**
```

```
 * 组类型图元被双击时回调，默认实现调用 group.toggle()，可重载改变默认逻辑或做后续处理
 * @param {ht.Group} group Group 对象
 * @param {Event} e 事件对象
 */
onGroupDoubleClicked = function (group, e){};

/**
 * 移动图元位置结束时回调，可重载做后续处理
 */
onMoveEnded = function (){};

/**
 * 手抓图平移拓扑图结束时回调，可重载做后续处理
 */
onPanEnded = function (){};

/**
 * 触屏进行双指缩放结束时回调，可重载做后续处理
 */
onPinchEnded = function (){};

/**
 * 框选结束时回调，可重载做后续处理
 */
onRectSelectEnded = function (){};

/**
 * 选中变化时回调，默认实现会使得该选中图元出现在拓扑图上的可见范围
 * @param {Event} event 选中变化事件对象
 */
onSelectionChanged = function (event){};

/**
 * 子网图元被双击时回调，默认实现进入子网
 * @param {ht.SubGraph} subGraph 子网对象
 * @param {Event} event 事件对象
 */
onSubGraphDoubleClicked = function (subGraph, event){};

/**
 * 平移动画结束时回调，可重载做后续处理
 */
onTranslateEnded = function (){};
```

```
/**
 * 图元可见状态发生变化时回调，可重载做后续处理
 * @param {ht.Data} data 图元
 * @param {Boolean} newVisible 新的可见状态
 */
onVisibleChanged = function (data, newVisible){};

/**
 * 缩放动画结束时回调
 */
onZoomEnded = function (){};

/**
 * 判断图元是否在矩形范围内
 * @param {ht.Data} data 图元
 * @param {Object} rect 矩形
 * @return {Boolean}
 */
rectContains = function (data, rect){};

/**
 * 判断图元与矩形范围是否相交
 * @param {ht.Data} data 图元
 * @param {Object} rect 矩形
 * @return {Boolean}
 */
rectIntersects = function (data, rect){};

/**
 * 重绘拓扑，rect 参数为空时重绘拓扑中的所有图元，否则重绘矩形范围内的图元
 * @param {Object} [rect] 矩形范围
 */
redraw = function (rect){};

/**
 * 删除底层 Painter
 * @param {Object} painter Painter 类
 */
removeBottomPainter = function (painter){};

/**
 * 删除交互事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
```

```
 * @see {@link ht.graph.GraphView#umi umi}
 */
removeInteractorListener = function (listener, scope){};

/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除所有选中的图元
 */
removeSelection = function (){};

/**
 * 删除顶层 Painter
 * @param {Object} painter Painter 类
 */
removeTopPainter = function (painter){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 重置拓扑状态，将 zoom 设为 1，translate 设为 0
 */
reset = function (){};

/**
 * 提供一个回调函数倒序遍历此拓扑中的图元，与 DataModel 上的 each 方法不同，此方法还考虑了拓扑中的 Layer，从高 Layer 向低 Layer 遍历
 * @param {Function} func 遍历函数
 * @param {Object} [scope] 函数域
 * @example graphView.reverseEach(function(data) {
 *   console.log(data);
 * });
 */
```

```
reverseEach = function (func, scope){};

/**
 * 选中拓扑中所有图元
 */
selectAll = function (){};

/**
 * 设置是否自动隐藏滚动条
 * @param {Boolean} v
 */
setAutoHideScrollBar = function (v){};


/**
 * 设置当选中图元时，是否自动平移拓扑以确保该图元出现在可见区域内
 * @param {Boolean} v
 */
setAutoMakeVisible = function (v){};


/**
 * 设置自动滚动区域大小，当鼠标距离拓扑边缘小于这个值时，拓扑自动滚动(调整 translateX 或 translateY)
 * @param {Boolean} v
 */
setAutoScrollZone = function (v){};

/**
 * 设置当前子网
 * @param {ht.SubGraph} subGraph 子网对象
 */
setCurrentSubGraph = function (subGraph){};

/**
 * 设置绑定的数据模型
 * @return {ht.DataModel} 数据模型
 */
setDataModel = function (dataModel){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
```

```
 */
setDisabled = function (value, iconUrl){};
/**
 * 设置拓扑中的图元是否可编辑
 * @param {Boolean} editable
 */
setEditable = function (editable){};

/**
 * 设置编辑过滤器函数
 * @param {Function} func 过滤器函数
 */
setEditableFunc = function (func){};

/**
 * 设置编辑交互器中编辑点的背景色
 * @param {color} color 颜色值
 */
setEditPointBackground = function (color){};

/**
 * 设置编辑交互器中编辑点的边框颜色
 * @param {color} color 颜色值
 */
setEditPointBorderColor = function (color){};

/**
 * 设置编辑交互器中编辑点的尺寸
 * @param {Number} size 编辑点尺寸
 */
setEditPointSize = function (size){};

/**
 * 设置布局高度
 * @param {Number} height 高度值
 */
setHeight = function (height){};

/**
 * 设置交互器
 * @param {ht.List} interactors 交互器对象集合
 */
setInteractors = function (interactors){};
```

```
/**
 * 定义拓扑中的层，参数为数组，数组中每个元素代表一个层，层在数组中的索引越大，在拓扑中就越靠上显示<br>
 * 注意，图元的默认 layer 是 0，因此如果定义的层中不包含 0，所有的图元默认将不可见
 * @param {Array} layers 层数组
 * @example
 * graphView.setLayers([0, 1, 'Layer2']);
 * node.setLayer(1);
 * node2.setLayer('Layer2');
 */
setLayers = function (layers){};

/**
 * 设置移动过滤器函数
 * @param {Function} func 过滤器函数
 */
setMovableFunc = function (func){};

/**
 * 设置是否可以通过鼠标拖拽进行平移操作
 * @param {Boolean} v 是否可平移
 */
setPannable = function (v){};

/**
 * 设置点编辑(Shape、Edge 等)过滤器函数
 * @param {Function} func 过滤器函数
 */
setPointEditableFunc = function (func){};

/**
 * 设置大小编辑过滤器函数
 * @param {Function} func 过滤器函数
 */
setRectEditableFunc = function (func){};

/**
 * 设置拓扑上是否允许框选操作
 * @param {Boolean} v
 */
setRectSelectable = function (v){};

/**
 * 设置框选选择框的背景色
```

```
 * @param {color} color 颜色值
 */
setRectSelectBackground = function (color){};

/**
 * 设置框选选择框的边框颜色
 * @param {color} color 颜色值
 */
setRectSelectBorder = function (color){};

/**
 * 设置拓扑上是否允许通过空格将拓扑的缩放和平移值复位
 * @param {Boolean} v
 */
setResettable = function (v){};

/**
 * 设置旋转编辑过滤器函数
 * @param {Function} func 过滤器函数
 */
setRotationEditableFunc = function (func){};

/**
 * 设置滚动条颜色
 * @param {color} color 颜色值
 */
setScrollBarColor = function (color){};

/**
 * 设置滚动条宽度
 * @param {Number} size 宽度值
 */
setScrollBarSize = function (size){};

/**
 * 设置滚动条是否可见
 * @param {Boolean} visible
 */
setScrollBarVisible = function (visible){};

/**
 * 设置选择过滤器函数
 * @param {Function} func 过滤器函数
 */
```

```
setSelectableFunc = function (func){};

/**
 * 设置拓扑是否共享选中模型
 * @param {Boolean} v
 */
setSelectionModelShared = function (v){};

/**
 * 设置拓扑水平和垂直平移值
 * @param {Number} x 水平平移值
 * @param {Number} y 垂直平移值
 * @param {Boolean} [anim] 是否使用动画
 */
setTranslate = function (x, y, anim){};

/**
 * 设置拓扑水平平移值
 * @param {Number} x 水平平移值
 */
setTranslateX = function (x){};

/**
 * 设置拓扑垂直平移值
 * @param {Number} y 垂直平移值
 */
setTranslateY = function (y){};

/**
 * 设置可见过滤器
 * @param {Function} func 过滤器函数
 */
setVisibleFunc = function (func){};

/**
 * 设置布局宽度
 * @param {Number} width 宽度值
 */
setWidth = function (width){};

/**
 * 设置拓扑缩放值
 * @param {Number} value 缩放值
 * @param {Boolean} [anim] 是否使用动画
```

```
 * @param {Object} [point] 缩放中心点的坐标
 */
setZoom = function (value, anim, point){};

/**
 * 显示滚动条
 */
showScrollBar = function (){};

/**
 * 获取选中模型，{@link ht.graph.GraphView#getSelectionModel getSelectionModel}的缩写
 * @see {@link ht.graph.GraphView#getSelectionModel getSelectionModel}
 * @return {ht.SelectionModel}
 */
sm = function (){};

/**
 * 将拓扑导出为 canvas
 * @param {color} background 背景色
 * @return {HTMLCanvasElement}
 */
toCanvas = function (background){};

/**
 * 将拓扑导出为 base64 格式字符串
 * @param {color} background 背景色
 * @return {String}
 */
toDataURL = function (background){};

/**
 * 在当前值基础上增加水平和垂直平移值
 * @param {Number} x 新增的水平平移值
 * @param {Number} y 新增的垂直平移值
 * @param {Boolean} [anim] 是否使用动画
 */
translate = function (x, y, anim){};

/**
 * 获取或设置水平平移值，没有参数时相当于{@link ht.graph.GraphView#getTranslateX getTranslateX}，有参数时相当于{@link ht.graph.GraphView#setTranslateX setTranslateX}
 * @param {Number} value 平移值
 */
```

```
tx = function (value){};

/**
 * 获取或设置垂直平移值，没有参数时相当于{@link ht.graph.GraphView#getTranslateY getTranslateY}，有参数时相当于{@link ht.graph.GraphView#setTranslateY setTranslateY}
 * @param {Number} value 平移值
 */
ty = function (value){};

/**
 * 删除交互事件监听器，{@link ht.graph.GraphView#removeInteractorListener removeInteractorListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.graph.GraphView#removeInteractorListener removeInteractorListener}
 */
umi = function (listener, scope){};

/**
 * 删除自身属性变化事件监听器，{@link ht.graph.GraphView#removePropertyChangeListener removePropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.graph.GraphView#removePropertyChangeListener removePropertyChangeListener}
 */
ump = function (listener, scope){};

/**
 * 立刻刷新拓扑
 */
validate = function (){};

/**
 * 放大拓扑
 * @param {Boolean} [anim] 是否使用动画
 * @param {Object} [point] 缩放中心点的坐标
 */
zoomIn = function (anim, point){};

/**
 * 缩小拓扑
 * @param {Boolean} [anim] 是否使用动画
 * @param {Object} [point] 缩放中心点的坐标
```

```
 */
zoomOut = function (anim, point){};

/**
 * 将拓扑缩放值改为 1
 * @param {Boolean} [anim] 是否使用动画
 * @param {Object} [point] 缩放中心点的坐标
 */
zoomReset = function (anim, point){};
```

ht.widget.AccordionView

```
/**
 * 折叠组件，用于多组件的折叠展开效果，提供水平和垂直两种布局方式
 * @constructor
 */
ht.widget.AccordionView = function() {};

/**
 * 添加组件
 * @param {String} title 组件的标题文字信息，不同组件不得相同
 * @param {Object|HTMLElement} content 组件内容，可为 HT 框架提供的组件对象，也可为原生
HTML 元素
 * @param {Boolean} expand 组件是否展开，默认为 false
 * @param {String} icon 组件标题中显示的图标
 */
add = function (title, content, expand, icon){};

/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.AccordionView#mp mp}
 */
addPropertyChangeListener = function (listener, scope, ahead){};

/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
```

```
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 清除所有组件
 */
clear = function (){};

/**
 * 合并当前展开的组件
 */
collapse = function (){};

/**
 * 根据标题找到组件并展开
 * @param {String} title 标题文字
 */
expand = function (title){};

/**
 * 获取合并图标
 * @return {String}
 */
getCollapseIcon = function (){};

/**
 * 获取当前展开组件的标题
 * @return {String}
 */
getCurrentTitle = function (){};

/**
 * 获取展开图标
 * @return {String}
 */
getExpandIcon = function (){};

/**
 * 获取布局高度
 * @return {Number}
 */
```

```
getHeight = function (){};

/**
 * 获取标题文字颜色
 * @return {color}
 */
getLabelColor = function (){};

/**
 * 获取标题文字字体
 * @return {String}
 */
getLabelFont = function (){};

/**
 * 获取布局方式，默认为 vertical 或 v，可设置为 horizontal 或 h
 * @return {String}
 */
getOrientation = function (){};

/**
 * 获取标题选中背景色
 * @return {color}
 */
getSelectBackground = function (){};

/**
 * 获取标题选中边框宽度
 * @return {Number}
 */
getSelectWidth = function (){};

/**
 * 获取分割线的颜色
 * @return {color}
 */
getSeparatorColor = function (){};

/**
 * 获取标题背景色
 * @return {color}
 */
getTitleBackground = function (){};
```

```
/**
 * 获取标题高度
 * @return {Number}
 */
getTitleHeight = function (){};

/**
 * 获取所有标题
 * @return {ht.List}
 */
getTitles = function (){};

/**
 * 获取组件的根层 div
 * @return {HTMLDivElement}
 */
getView = function (){};

/**
 * 获取布局宽度
 * @return {Number}
 */
getWidth = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.AccordionView#iv iv}
 */

invalidate = function (delay){};


/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
isDisabled = function (){};

/**
 * 判断指定的 title 是否处于展开状态
 * @param {String} title 标题文字
 * @return {Boolean}
 */
```

```
isExpanded = function (title){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.AccordionView#invalidate invalidate}
 */

iv = function (delay){};

/**
 * 增加自身属性变化事件监听器，{@link ht.widget.AccordionView#addPropertyChangeListener addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.AccordionView#addPropertyChangeListener addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};

/**
 * 合并标题时调用，可重载做后续处理
 * @param {String} title 标题
 */
onCollapsed = function (title){};

/**
 * 展开标题时调用，可重载做后续处理
 * @param {String} title 标题
 */
onExpanded = function (title){};

/**
 * 根据标题找到组件并删除
 * @param {String} title 标题
 */
remove = function (title){};

/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
```

```
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 设置合并图标
 * @param {String} icon 图标
 */
setCollapseIcon = function (icon){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
 * 设置展开图标
 * @param {String} icon 图标
 */
setExpandIcon = function (icon){};

/**
 * 设置布局高度
 * @param {Number} v 高度值
 */
setHeight = function (v){};

/**
 * 设置标题文字颜色
 * @param {color} color 颜色值
 */
setLabelColor = function (color){};

/**
 * 设置标题文字字体
 * @param {String} font 字体
 */
```

```
setLabelFont = function (font){};

/**
 * 设置布局方式，默认为 vertical 或 v，可设置为 horizontal 或 h
 * @param {String} v 布局方式
 */
setOrientation = function (v){};

/**
 * 设置标题选中背景色
 * @param {color} color 颜色值
 */
setSelectBackground = function (color){};

/**
 * 设置标题选中边框宽度
 * @param {Number} v
 */
setSelectWidth = function (v){};

/**
 * 设置分割线颜色
 * @param {color} color 颜色值
 */
setSeparatorColor = function (color){};
/**
 * 设置标题背景色
 * @param {color} color 颜色值
 */
setTitleBackground = function (color){};

/**
 * 设置标题高度
 * @param {Number} v 高度值
 */
setTitleHeight = function (v){};

/**
 * 设置布局宽度
 * @param {Number} v 宽度值
 */
setWidth = function (v){};

/**
```

```
 * 删除自身属性变化事件监听器，{@link
ht.widget.AccordionView#removePropertyChangeListener removePropertyChangeListener}的缩
写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.widget.AccordionView#removePropertyChangeListener
removePropertyChangeListener}
 */
ump = function (listener, scope){};

/**
 * 刷新组件
 */
validate = function (){};
```

ht.widget.BorderPane

```
/**
 * 边框面板是一种组件布局容器，可在上、下、左、右、中的五个区域位置摆放子组件，
 * 子组件可为 HT 框架提供的组件，也可为 HTML 元素，子组件以 position 为 absolute 方式进行绝
对定位。
 * @constructor
 */
ht.widget.BorderPane = function() {};

/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.BorderPane#mp mp}
 */
addPropertyChangeListener = function (listener, scope, ahead){};

/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
```

```
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 获取底部组件高度
 * @return {Number}
 */
getBottomHeight = function (){};

/**
 * 获取底部组件
 * @return {Object|HTMLElement}
 */
getBottomView = function (){};

/**
 * 获取中间组件
 * @return {Object|HTMLElement}
 */
getCenterView = function (){};

/**
 * 获取布局高度
 * @return {Number}
 */
getHeight = function (){};

/**
 * 获取左侧组件
 * @return {Object|HTMLElement}
 */
getLeftView = function (){};

/**
 * 获取左侧组件宽度
 * @return {Number}
 */
getLeftWidth = function (){};

/**
 * 获取右侧组件
 * @return {Object|HTMLElement}
 */
```

```
getRightView = function (){};

/**
 * 获取右侧组件宽度
 * @return {Number}
 */
getRightWidth = function (){};

/**
 * 获取顶部组件高度
 * @return {Number}
 */
getTopHeight = function (){};

/**
 * 获取顶部组件
 * @return {Object|HTMLElement}
 */
getTopView = function (){};

/**
 * 获取组件的根层 div
 * @return {HTMLDivElement}
 */
getView = function (){};

/**
 * 获取布局宽度
 * @return {Number}
 */
getWidth = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.BorderPane#iv iv}
 */
invalidate = function (delay){};


/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
```

```
isDisabled = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.BorderPane#invalidate invalidate}
 */
iv = function (delay){};

/**
 * 增加自身属性变化事件监听器，{@link ht.widget.BorderPane#addPropertyChangeListener addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.BorderPane#addPropertyChangeListener addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};

/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 设置底部组件高度
 * @param {Number} v
 */
setBottomHeight = function (v){};

/**
 * 设置底部组件
 * @param {Object|HTMLElement} v
 */
```

```
setBottomView = function (v){};

/**
 * 设置中间组件
 * @param {Object|HTMLElement} v
 */
setCenterView = function (v){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
 * 设置布局高度
 * @param {Number} v
 */
setHeight = function (v){};

/**
 * 设置左侧组件
 * @param {Object|HTMLElement} v
 */
setLeftView = function (v){};

/**
 * 设置左侧组件宽度
 * @param {Number} v
 */
setleftWidth = function (v){};

/**
 * 设置右侧组件
 * @param {Object|HTMLElement} v
 */
setRightView = function (v){};

/**
 * 设置右侧组件宽度
 * @param {Number} v
 */
```

```
setRightWidth = function (v){};

/**
 * 设置顶部组件高度
 * @param {Number} v
 */
setTopHeight = function (v){};

/**
 * 设置顶部组件
 * @param {Object|HTMLElement} v
 */
setTopView = function (v){};

/**
 * 设置布局宽度
 * @param {Number} v
 */
setWidth = function (v){};

/**
 * 删除自身属性变化事件监听器，{@link ht.widget.BorderPane#removePropertyChangeListener removePropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.widget.BorderPane#removePropertyChangeListener removePropertyChangeListener}
 */
ump = function (listener, scope){};

/**
 * 刷新组件
 */
validate = function (){};
```

ht.widget.ListView

```
/**
 * 列表组件类，用列表的方式呈现 DataModel 中的数据
 * @param {ht.DataModel} dataModel 绑定的数据模型
 * @constructor
```

```
 */
ht.widget.ListView = function(dataModel) {};

/**
 * 增加底层 Painter<br>
 * 组件上提供 Painter 接口，开发者可以使用 Canvas 的画笔对象自由绘制任意形状，底层 Painter 绘制在组件最下面
 * @param {Function} painter Painter 类
 * @example //Painter 示例：
 * function MyBottomPainter() {
 * }
 * ht.Default.def(MyBottomPainter, Object, {
 *  draw: function(g) {
 *      g.save();
 *      //draw...
 *      g.restore();
 *  }
 * });
 * listView.addBottomPainter(MyBottomPainter);
 */
addBottomPainter = function(painter) {}

/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.ListView#mp mp}
 */
addPropertyChangeListener = function (listener, scope, ahead){};


/**
 * 增加顶层 Painter<br>
 * 组件上提供 Painter 接口，开发者可以使用 Canvas 的画笔对象自由绘制任意形状，顶层 Painter 绘制在组件最上面
 * @param {Function} painter Painter 类
 * @example //Painter 示例：
 * function MyTopPainter() {
 * }
 * ht.Default.def(MyTopPainter, Object, {
 *  draw: function(g) {
 *      g.save();
 *      //draw...
```

```
 *        g.restore();
 *   }
 * });
 * listView.addTopPainter(MyTopPainter);
 */
addTopPainter = function (painter){};


/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 关闭 ToolTip 功能
 */
disableToolTip = function (){};

/**
 * 获取或设置数据模型，没有参数时相当于{@link ht.widget.ListView#getDataModel getDataModel}，有参数时相当于{@link ht.widget.ListView#setDataModel setDataModel}
 * @param {ht.DataModel} [dataModel] 数据模型
 * @return {ht.DataModel} dataModel
 */
dm = function (dataModel){};

/**
 * 绘制图标，可重载自定义
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 */
drawIcon = function (g, data, x, y, width, height){};

/**
 * 绘制文本，可重载自定义，label 一般绘制在最后因此没有 width 参数限制
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
```

```
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} height 绘制的高度
 */
drawLabel = function (g, data, x, y, height){};

/**
 * 绘制行内容，可重载自定义，默认调用 drawIcon 和 drawLabel，如果返回值为 HTML 元素，则使用 HTML 元素当作 Renderer
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
 * @param {Boolean} selected 数据元素是否选中
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 * @return {HTMLElement}
 */
drawRow = function (g, data, selected, x, y, width, height){};

/**
 * 绘制行背景色，默认仅在选中该行时填充选中背景色，可重载自定义
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
 * @param {Boolean} selected 数据元素是否选中
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 */
drawRowBackground = function (g, data, selected, x, y, width, height){};

/**
 * 启用 ToolTip
 */
enableToolTip = function (){};

/**
 * 获取数据元素 icon 的背景色，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {color} 颜色值，默认返回 data.s('body.color')
 */
getBodyColor = function (data){};
```

```
/**
 * 获取数据元素 icon 的边框色，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {color} 颜色值，默认返回 data.s('border.color')
 */
getBorderColor = function (data){};

/**
 * 返回数据元素对应的 check 图标，可重载自定义 check 图标，该函数在 checkMode 模式下有效
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
getCheckIcon = function (data){};


/**
 * 传入逻辑坐标点或者交互 event 事件参数，返回当前点下的数据元素
 * @param {Object|Event} pointOrEvent 逻辑坐标点或交互事件对象(如鼠标事件对象)
 * @return {ht.Data} 点下的数据元素
 */
getDataAt = function (pointOrEvent){};

/**
 * 获取绑定的数据模型
 * @return {ht.DataModel} 数据模型
 */
getDataModel = function (){};

/**
 * 获取当前可见区域的结束行索引
 * @return {Number}
 */
getEndRowIndex = function (){};

/**
 * 在 checkMode 模式下数据除了被选中有 check 状态外，还可以有被点击行的 focus 状态<br>
 * 此方法获取 focus 数据元素
 * @return {ht.Data}
 */
getFocusData = function (){};

/**
 * 获取布局高度
 * @return {Number}
```

```
 */
getHeight = function (){};

/**
 * 获取 data 对象对应的 icon 图标，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
getIcon = function (data){};

/**
 * 返回 data 对象对应的图标宽度，默认如果有图标则以 indent 值为宽度，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {Number}
 */
getIconWidth = function (data){};

/**
 * 获取 indent 缩进，该值一般当作图标的宽度
 * @param {ht.Data} data 数据元素
 * @return {Number}
 */
getIndent = function (data){};

/**
 * 获取 data 对象显示的文字，默认返回 data.toLabel()，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
getLabel = function (data){};

/**
 * 获取 data 对象的文本颜色，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {color}
 */
getLabelColor = function (data){};

/**
 * 获取 data 对象的文本字体，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
```

```
getLabelFont = function (data){};

/**
 * 获取选中文本的颜色
 * @return {color}
 */
getLabelSelectColor = function (){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.ListView#lp lp}
 */
getLogicalPoint = function (event){};

/**
 * 获取当前显示的 Data 对象集合，该集合已被排序和过滤
 * @return {ht.List}
 */
getRowDatas = function (){};

/**
 * 获取行高
 * @return {Number}
 */
getRowHeight = function (){};

/**
 * 获取 data 对象所在的行索引
 * @param {ht.Data} data 数据元素
 * @return {Number}
 */
getRowIndex = function (data){};

/**
 * 获取行线颜色
 * @return {color}
 */
getRowLineColor = function (){};

/**
 * 返回当前可见行总行数
 * @return {Number}
```

```
 */
getRowSize = function (){};

/**
 * 获取滚动条颜色
 * @return {color}
 */
getScrollBarColor = function (){};

/**
 * 获取滚动条宽度
 * @return {Number}
 */
getScrollBarSize = function (){};

/**
 * 获取选择过滤器函数
 * @return {Function}
 */
getSelectableFunc = function (){};

/**
 * 获取行选中背景颜色
 * @return {color}
 */
getSelectBackground = function (){};

/**
 * 获取选中模型
 * @return {ht.SelectionModel}
 * @see {@link ht.widget.ListView#sm sm}
 */
getSelectionModel = function (){};

/**
 * 获取排序函数
 * @return {Function}
 */
getSortFunc = function (){};

/**
 * 获取当前可见区域的起始行索引
 * @return {Number}
 */
```

```
getStartRowIndex = function (){};

/**
 * 获取 ToolTip 文字，可重载返回自定义的 toolTip 文字
 * @param {Event} e 鼠标或 Touch 事件对象
 * @return {String} toolTip 文字，默认取出鼠标下的图元，然后返回其 getToolTip()
 */
getToolTip = function (e){};

/**
 * 获取垂直平移值
 * @return {Number} 垂直平移值
 * @see {@link ht.widget.ListView#ty ty}
 */
getTranslateY = function (){};

/**
 * 获取组件的根层 div
 * @return {HTMLDivElement}
 */
getView = function (){};

/**
 * 获取组件中可见区域的逻辑尺寸
 * @return {Object}
 */
getViewRect = function (){};

/**
 * 获取可见过滤器函数
 * @return {Function}
 */
getVisibleFunc = function (){};

/**
 * 获取布局宽度
 * @return {Function}
 */
getWidth = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.ListView#iv iv}
```

```
 */
invalidate = function (delay){};

/**
 * 无效数据元素
 * @param {ht.Data} data 要无效的数据元素
 */
invalidateData = function (data){};

/**
 * 无效模型，最彻底的刷新方式
 * @see {@link ht.widget.ListView#ivm ivm}
 */
invalidateModel = function (){};

/**
 * 是否自动隐藏滚动条
 * @return {Boolean}
 */
isAutoHideScrollBar = function (){};

/**
 * 选中数据元素时，是否自动平移组件以确保该元素出现在可见区域内
 * @return {Boolean}
 */
isAutoMakeVisible = function (){};

/**
 * 是否是 check 模式
 * @return {Boolean}
 */
isCheckMode = function (){};

/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
isDisabled = function (){};

/**
 * 获取行线是否可见，默认为 true
 * @return {Boolean}
 */
```

```
isRowLineVisible = function (){};

/**
 * 判断 data 对象是否可被选中
 * @param {ht.Data} data 数据元素
 * @return {Boolean}
 */
isSelectable = function (data){};

/**
 * 判断 data 对象是否被选中
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isSelected = function (data){};

/**
 * 根据 id 判断 data 对象是否被选中
 * @param {String|Number} id 数据元素 id
 * @return {Boolean}
 */
isSelectedById = function (id){};

/**
 * 当前组件是否共享选中模型
 * @return {Boolean}
 */
isSelectionModelShared = function (){};

/**
 * 判断数据元素是否可见
 * @param {ht.Data} data 数据元素
 * @return {Boolean}
 */
isVisible = function (data){};

/**
 * 无效组件，并调用延时刷新，{@link ht.widget.ListView#invalidate invalidate}的缩写
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.ListView#invalidate invalidate}
 */
iv = function (delay){};

/**
```

```
 * 无效模型，重新构造内部的 rows 数据，最彻底的刷新方式，{@link
ht.widget.ListView#invalidateModel invalidateModel}的缩写
 * @see {@link ht.widget.ListView#invalidateModel invalidateModel}
 */
ivm = function (){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标，{@link
ht.widget.ListView#getLogicalPoint getLogicalPoint}的缩写
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.ListView#getLogicalPoint getLogicalPoint}
 */
lp = function (event){};

/**
 * 平移组件以确保数据元素在可见区域内
 * @param {ht.Data} data 数据元素
 */
makeVisible = function (data){};

/**
 * 增加自身属性变化事件监听器，{@link ht.widget.ListView#addPropertyChangeListener
addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.ListView#addPropertyChangeListener addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};

/**
 * 数据元素被点击时回调，可重载做后续处理
 * @param {ht.Data} data 被点击的数据元素
 * @param {Event} e 事件对象
 */
onDataClicked = function (data, e){};

/**
 * 数据元素被双击时回调，可重载做后续处理
 * @param {ht.Data} data 双击的数据元素
 * @param {Event} e 事件对象
 */
```

```
onDataDoubleClicked = function (data, e){};

/**
 * 平移动画结束时回调，可重载做后续处理
 */
onTranslateEnded = function (){};

/**
 * 重绘组件中所有行，仅次于 invalidateModel 的彻底刷新方式
 */
redraw = function (){};

/**
 * 删除底层 Painter
 * @param {Object} painter Painter 类
 */
removeBottomPainter = function (painter){};

/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除所有选中的图元
 */
removeSelection = function (){};

/**
 * 删除顶层 Painter
 * @param {Object} painter Painter 类
 */
removeTopPainter = function (painter){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
```

```
 * 平移(滚动)组件至指定的行号
 * @param {Number} index 行号
 */
scrollToIndex = function (index){};

/**
 * 选中所有数据元素
 */
selectAll = function (){};

/**
 * 设置是否自动隐藏滚动条
 * @param {Boolean} v
 */
setAutoHideScrollBar = function (v){};

/**
 * 设置当选中数据元素，是否自动平移(滚动)组件以确保该数据元素出现在可见区域内
 * @param {Boolean} v
 */
setAutoMakeVisible = function (v){};

/**
 * 设置 check 模式
 * @param {Boolean} v
 */
setCheckMode = function (v){};

/**
 * 设置绑定的数据模型
 * @return {ht.DataModel} 数据模型
 */
setDataModel = function (dataModel){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
 * 在 checkMode 模式下数据除了被选中有 check 状态外，还可以有被点击行的 focus 状态<br>
 * 此方法设置 focus 的数据元素
```

```
 * @param {ht.Data} data 数据元素
 */
setFocusData = function (data){};

/**
 * 在 checkMode 模式下数据除了被选中有 check 状态外，还可以有被点击行的 focus 状态<br>
 * 此方法设置 focus 的数据元素
 * @param {String|Number} id 数据元素的 id
 */
setFocusDataById = function (id){};

/**
 * 设置布局高度
 * @param {Number} v 高度值
 */
setHeight = function (v){};

/**
 * 设置 indent 缩进，该值一般当作图标的宽度
 * @param {Number} v
 */
setIndent = function (v){};

/**
 * 设置行 label 文字颜色
 * @param {color} v
 */
setLabelColor = function (v){};

/**
 * 设置行 label 文字字体
 * @param {String} v
 */
setLabelFont = function (v){};

/**
 * 设置行 label 文字选中颜色
 * @param {color} v
 */
setLabelSelectColor = function (v){};

/**
 * 设置行高
 * @param {Number} v
```

```
 */
setRowHeight = function (v){};

/**
 * 设置行线颜色
 * @param {color} color
 */
setRowLineColor = function (color){};

/**
 * 设置行线是否可见
 * @param {Boolean} v
 */
setRowLineVisible = function (v){};

/**
 * 设置滚动条颜色
 * @param {color} color 颜色值
 */
setScrollBarColor = function (color){};

/**
 * 设置滚动条宽度
 * @param {Number} size 宽度值
 */
setScrollBarSize = function (size){};

/**
 * 设置选择过滤器函数
 * @param {Function} func 过滤器函数
 */
setSelectableFunc = function (func){};

/**
 * 设置行选中背景颜色
 * @param {color} color
 */
setSelectBackground = function (color){};

/**
 * 设置组件是否共享选中模型
 * @param {Boolean} v
 */
```

```
setSelectionModelShared = function (v){};

/**
 * 设置排序函数
 * @param {Function} func
 */
setSortFunc = function (func){};

/**
 * 设置垂直平移值(水平平移值无效)
 * @param {Number} x 水平平移值，此参数无效
 * @param {Number} y 垂直平移值
 * @param {Boolean} anim 是否使用动画
 */
setTranslate = function (x, y, anim){};

/**
 * 设置垂直平移值
 * @param {Number} v 垂直平移值
 */
setTranslateY = function (v){};

/**
 * 设置可见过滤器
 * @param {Function} func 过滤器函数
 */
setVisibleFunc = function (func){};

/**
 * 设置布局宽度
 * @param {Function} func 过滤器函数
 */
setWidth = function (v){};

/**
 * 显示垂直滚动条
 */
showVBar = function (){};

/**
 * 获取选中模型，{@link ht.widget.ListView#getSelectionModel getSelectionModel}的缩写
 * @see {@link ht.widget.ListView#getSelectionModel getSelectionModel}
 * @return {ht.SelectionModel}
 */
```

```
sm = function (){};

/**
 * 在当前值基础上增加垂直平移值(水平无效)
 * @param {Number} x 新增的水平平移值，此参数无效
 * @param {Number} y 新增的垂直平移值
 * @param {Boolean} [anim] 是否使用动画
 */
translate = function (x, y, anim){};

/**
 * 获取或设置垂直平移值，没有参数时相当于{@link ht.widget.ListView#getTranslateY getTranslateY}，有参数时相当于{@link ht.widget.ListView#setTranslateY setTranslateY}
 * @param {Number} value 平移值
 */
ty = function (value){};

/**
 * 删除自身属性变化事件监听器，{@link ht.widget.ListView#removePropertyChangeListener removePropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.widget.ListView#removePropertyChangeListener removePropertyChangeListener}
 */
ump = function (listener, scope){};

/**
 * 立刻刷新组件
 */
validate = function (){};
```

ht.widget.TreeView

```
/**
 * 树形组件，以树形的方式呈现 DataModel 中 Data 数据的父子关系
 * @param {ht.DataModel} dataModel 绑定的数据模型
 * @constructor
 */
ht.widget.TreeView = function(dataModel) {};

/**
 * 增加底层 Painter<br>
```

```
 * 组件上提供 Painter 接口，开发者可以使用 Canvas 的画笔对象自由绘制任意形状，底层 Painter 绘制在组件最下面
 * @param {Function} painter Painter 类
 * @example //Painter 示例：
 * function MyBottomPainter() {
 * }
 * ht.Default.def(MyBottomPainter, Object, {
 *   draw: function(g) {
 *       g.save();
 *       //draw...
 *       g.restore();
 *   }
 * });
 * treeView.addBottomPainter(MyBottomPainter);
 */
addBottomPainter = function(painter) {}

/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.TreeView#mp mp}
 */
addPropertyChangeListener = function (listener, scope, ahead){};


/**
 * 增加顶层 Painter<br>
 * 组件上提供 Painter 接口，开发者可以使用 Canvas 的画笔对象自由绘制任意形状，顶层 Painter 绘制在组件最上面
 * @param {Function} painter Painter 类
 * @example //Painter 示例：
 * function MyTopPainter() {
 * }
 * ht.Default.def(MyTopPainter, Object, {
 *   draw: function(g) {
 *       g.save();
 *       //draw...
 *       g.restore();
 *   }
 * });
 * treeView.addTopPainter(MyTopPainter);
 */
```

```
addTopPainter = function (painter){};


/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 合并 data 对象
 * @param {ht.Data} data 数据元素
 */
collapse = function (data){};

/**
 * 合并所有对象
 */
collapseAll = function (){};

/**
 * 关闭 ToolTip 功能
 */
disableToolTip = function (){};

/**
 * 获取或设置数据模型，没有参数时相当于{@link ht.widget.TreeView#getDataModel
getDataModel}，有参数时相当于{@link ht.widget.TreeView#setDataModel setDataModel}
 * @param {ht.DataModel} [dataModel] 数据模型
 * @return {ht.DataModel} dataModel
 */
dm = function (dataModel){};

/**
 * 绘制图标，可重载自定义
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 */
```

```
drawIcon = function (g, data, x, y, width, height){};

/**
 * 绘制文本，可重载自定义，label 一般绘制在最后因此没有 width 参数限制
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} height 绘制的高度
 */
drawLabel = function (g, data, x, y, height){};

/**
 * 绘制行内容，可重载自定义，默认调用 drawIcon 和 drawLabel，如果返回值为 HTML 元素，则使用 HTML 元素当作 Renderer
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
 * @param {Boolean} selected 数据元素是否选中
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 * @return {HTMLElement}
 */
drawRow = function (g, data, selected, x, y, width, height){};

/**
 * 绘制行背景色，默认仅在选中该行时填充选中背景色，可重载自定义
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
 * @param {Boolean} selected 数据元素是否选中
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 */
drawRowBackground = function (g, data, selected, x, y, width, height){};

/**
 * 启用 ToolTip
 */
enableToolTip = function (){};

/**
```

```
 * 展开 data 对象
 * @param {ht.Data} data 数据元素
 */
expand = function (data){};

/**
 * 展开所有对象
 */
expandAll = function (){};

/**
 * 获取数据元素 icon 的背景色，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {color} 颜色值，默认返回 data.s('body.color')
 */
getBodyColor = function (data){};

/**
 * 获取数据元素 icon 的边框色，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {color} 颜色值，默认返回 data.s('border.color')
 */
getBorderColor = function (data){};

/**
 * 返回数据元素对应的 check 图标，可重载自定义 check 图标，该函数在 checkMode 模式下有效
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
getCheckIcon = function (data){};

/**
 * 获取 check 模式
 * <ul>
 * <li>null：默认值，不启用 check 选择模式</li>
 * <li>default：check 模式的默认选择方式，即单击选中或取消选中，只影响当前点击中的 data 对象</li>
 * <li>children：该 check 模式将同时影响点击中的 data 对象，以及其孩子对象</li>
 * <li>descendant：该 check 模式将同时影响点击中的 data 对象，以及其所有子孙对象</li>
 * <li>all：该 check 模式将同时影响点击中的 data 对象，以及其所有父辈和子孙对象</li>
 * </ul>
 * @return {String}
 */
```

```
getCheckMode = function (){};

/**
 * 获取 toggle 的关闭图标
 * @return {String}
 */
getCollapseIcon = function (){};

/**
 * 传入逻辑坐标点或者交互 event 事件参数，返回当前点下的数据元素
 * @param {Object|Event} pointOrEvent 逻辑坐标点或交互事件对象(如鼠标事件对象)
 * @return {ht.Data} 点下的数据元素
 */
getDataAt = function (pointOrEvent){};

/**
 * 获取绑定的数据模型
 * @return {ht.DataModel} 数据模型
 */
getDataModel = function (){};

/**
 * 获取当前可见区域的结束行索引
 * @return {Number}
 */
getEndRowIndex = function (){};

/**
 * 获取 toggle 的展开图标
 * @return {String}
 */
getExpandIcon = function (){};

/**
 * 在 checkMode 模式下数据除了被选中有 check 状态外，还可以有被点击行的 focus 状态<br>
 * 此方法获取 focus 数据元素
 * @return {ht.Data}
 */
getFocusData = function (){};

/**
 * 获取布局高度
 * @return {Number}
 */
```

```
getHeight = function (){};

/**
 * 获取 data 对象对应的 icon 图标，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
getIcon = function (data){};

/**
 * 返回 data 对象对应的图标宽度，默认如果有图标则以 indent 值为宽度，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {Number}
 */
getIconWidth = function (data){};

/**
 * 获取 indent 缩进，该值一般当作图标的宽度
 * @param {ht.Data} data 数据元素
 * @return {Number}
 */
getIndent = function (data){};

/**
 * 获取 data 对象显示的文字，默认返回 data.toLabel()，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
getLabel = function (data){};

/**
 * 获取 data 对象的文本颜色，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {color}
 */
getLabelColor = function (data){};

/**
 * 获取 data 对象的文本字体，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
getLabelFont = function (data){};
```

```
/**
 * 获取选中文本的颜色
 * @return {color}
 */
getLabelSelectColor = function (){};

/**
 * 获取当前 data 的缩减层次，一般结合 indent 参数用于绘制
 * @param {ht.Data} data 数据元素
 * @return {Number}
 */
getLevel = function (data){};

/**
 * 获取延迟加载器
 * @return {Object}
 */
getLoader = function (){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.TreeView#lp lp}
 */
getLogicalPoint = function (event){};

/**
 * 获取根节点，默认为空，从 DataModel#getRoots()的对象开始展示
 * @return {ht.Data}
 */
getRootData = function (){};

/**
 * 获取当前显示的 Data 对象集合，该集合已被排序和过滤
 * @return {ht.List}
 */
getRowDatas = function (){};

/**
 * 获取行高
 * @return {Number}
 */
```

```
getRowHeight = function (){};

/**
 * 获取 data 对象所在的行索引
 * @param {ht.Data} data 数据元素
 * @return {Number}
 */
getRowIndex = function (data){};

/**
 * 获取行线颜色
 * @return {color}
 */
getRowLineColor = function (){};

/**
 * 返回当前可见行总行数
 * @return {Number}
 */
getRowSize = function (){};

/**
 * 获取滚动条颜色
 * @return {color}
 */
getScrollBarColor = function (){};

/**
 * 获取滚动条宽度
 * @return {Number}
 */
getScrollBarSize = function (){};

/**
 * 获取选择过滤器函数
 * @return {Function}
 */
getSelectableFunc = function (){};

/**
 * 获取行选中背景颜色
 * @return {color}
 */
```

```
getSelectBackground = function (){};

/**
 * 获取选中模型
 * @return {ht.SelectionModel}
 * @see {@link ht.widget.TreeView#sm sm}
 */
getSelectionModel = function (){};

/**
 * 获取排序函数
 * @return {Function}
 */
getSortFunc = function (){};

/**
 * 获取当前可见区域的起始行索引
 * @return {Number}
 */
getStartRowIndex = function (){};

/**
 * 返回当前 data 对象对应的展开或合并图标，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
getToggleIcon = function (data){};

/**
 * 获取 ToolTip 文字，可重载返回自定义的 toolTip 文字
 * @param {Event} e 鼠标或 Touch 事件对象
 * @return {String} toolTip 文字，默认取出鼠标下的图元，然后返回其 getToolTip()
 */
getToolTip = function (e){};

/**
 * 获取垂直平移值
 * @return {Number} 垂直平移值
 * @see {@link ht.widget.TreeView#ty ty}
 */
getTranslateY = function (){};

/**
 * 获取组件的根层 div
```

```
 * @return {HTMLDivElement}
 */
getView = function (){};

/**
 * 获取组件中可见区域的逻辑尺寸
 * @return {Object}
 */
getViewRect = function (){};

/**
 * 获取可见过滤器函数
 * @return {Function}
 */
getVisibleFunc = function (){};

/**
 * 获取布局宽度
 * @return {Number}
 */
getWidth = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TreeView#iv iv}
 */
invalidate = function (delay){};

/**
 * 无效数据元素
 * @param {ht.Data} data 要无效的数据元素
 */
invalidateData = function (data){};

/**
 * 无效模型，最彻底的刷新方式
 * @see {@link ht.widget.TreeView#ivm ivm}
 */
invalidateModel = function (){};

/**
 * 是否自动隐藏滚动条
 * @return {Boolean}
```

```
 */
isAutoHideScrollBar = function (){};

/**
 * 选中数据元素时，是否自动平移组件以确保该元素出现在可见区域内
 * @return {Boolean}
 */
isAutoMakeVisible = function (){};

/**
 * 是否是 check 模式
 * @return {Boolean}
 */
isCheckMode = function (){};

/**
 * 判断是否允许对 parent 对象的孩子排序，默认返回 true，可重载屏蔽孩子排序
 * @param {ht.Data} parent 父元素
 * @return {Boolean}
 */
isChildrenSortable = function (parent){};

/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
isDisabled = function (){};

/**
 * 判断 data 对象是否展开
 * @param {ht.Data} data 数据元素
 * @return {Boolean}
 */
isExpanded = function (data){};

/**
 * 判断 rootData 节点是否可见
 * @return {Boolean}
 */
isRootVisible = function (){};

/**
 * 获取行线是否可见，默认为 true
 * @return {Boolean}
```

```
 */
isRowLineVisible = function (){};

/**
 * 判断 data 对象是否可被选中
 * @param {ht.Data} data 数据元素
 * @return {Boolean}
 */
isSelectable = function (data){};

/**
 * 判断 data 对象是否被选中
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isSelected = function (data){};

/**
 * 根据 id 判断 data 对象是否被选中
 * @param {String|Number} id 数据元素 id
 * @return {Boolean}
 */
isSelectedById = function (id){};

/**
 * 当前组件是否共享选中模型
 * @return {Boolean}
 */
isSelectionModelShared = function (){};

/**
 * 判断数据元素是否可见
 * @param {ht.Data} data 数据元素
 * @return {Boolean}
 */
isVisible = function (data){};

/**
 * 无效组件，并调用延时刷新，{@link ht.widget.TreeView#invalidate invalidate}的缩写
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TreeView#invalidate invalidate}
 */
iv = function (delay){};
```

```
/**
 * 无效模型，重新构造内部的 rows 数据，最彻底的刷新方式，{@link
ht.widget.TreeView#invalidateModel invalidateModel}的缩写
 * @see {@link ht.widget.TreeView#invalidateModel invalidateModel}
 */
ivm = function (){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标，{@link
ht.widget.TreeView#getLogicalPoint getLogicalPoint}的缩写
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.TreeView#getLogicalPoint getLogicalPoint}
 */
lp = function (event){};

/**
 * 平移组件以确保数据元素在可见区域内
 * @param {ht.Data} data 数据元素
 */
makeVisible = function (data){};

/**
 * 增加自身属性变化事件监听器，{@link ht.widget.TreeView#addPropertyChangeListener
addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.TreeView#addPropertyChangeListener addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};

/**
 * 合并 data 对象时调用，可重载做后续处理
 * @param {ht.Data} data 数据元素
 */
onCollapsed = function (data){};

/**
 * 数据元素被点击时回调，可重载做后续处理
 * @param {ht.Data} data 被点击的数据元素
 * @param {Event} e 事件对象
 */
```

```
onDataClicked = function (data, e){};

/**
 * 数据元素被双击时回调，可重载做后续处理
 * @param {ht.Data} data 双击的数据元素
 * @param {Event} e 事件对象
 */
onDataDoubleClicked = function (data, e){};

/**
 * 展开 data 对象时调用，可重载做后续处理
 * @param {ht.Data} data 数据元素
 */
onExpanded = function (data){};

/**
 * 平移动画结束时回调，可重载做后续处理
 */
onTranslateEnded = function (){};

/**
 * 重绘组件中所有行，仅次于 invalidateModel 的彻底刷新方式
 */
redraw = function (){};

/**
 * 删除底层 Painter
 * @param {Object} painter Painter 类
 */
removeBottomPainter = function (painter){};

/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除所有选中的图元
 */
removeSelection = function (){};

/**
```

```
 * 删除顶层 Painter
 * @param {Object} painter Painter 类
 */
removeTopPainter = function (painter){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 平移(滚动)组件至指定的行号
 * @param {Number} index 行号
 */
scrollToIndex = function (index){};

/**
 * 选中所有数据元素
 */
selectAll = function (){};

/**
 * 设置是否自动隐藏滚动条
 * @param {Boolean} v
 */
setAutoHideScrollBar = function (v){};

/**
 * 设置当选中数据元素，是否自动平移(滚动)组件以确保该数据元素出现在可见区域内
 * @param {Boolean} v
 */
setAutoMakeVisible = function (v){};

/**
 * 设置 check 模式
 * @param {String} v
 * <ul>
 * <li>null：默认值，不启用 check 选择模式</li>
 * <li>default：check 模式的默认选择方式，即单击选中或取消选中，只影响当前点击中的 data 对象</li>
 * <li>children：该 check 模式将同时影响点击中的 data 对象，以及其孩子对象</li>
 * <li>descendant：该 check 模式将同时影响点击中的 data 对象，以及其所有子孙对象</li>
```

```
 * <li>all：该 check 模式将同时影响点击中的 data 对象，以及其所有父辈和子孙对象</li>
 * </ul>
 */
setCheckMode = function (v){};

/**
 * 设置 toggle 的关闭图标
 * @param {String} v icon
 */
setCollapseIcon = function (v){};

/**
 * 设置绑定的数据模型
 * @param {ht.DataModel} dataModel 数据模型
 */
setDataModel = function (dataModel){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
 * 设置 toggle 的展开图标
 * @param {String} v icon
 */
setExpandIcon = function (v){};

/**
 * 在 checkMode 模式下数据除了被选中有 check 状态外，还可以有被点击行的 focus 状态<br>
 * 此方法设置 focus 的数据元素
 * @param {ht.Data} data 数据元素
 */
setFocusData = function (data){};

/**
 * 在 checkMode 模式下数据除了被选中有 check 状态外，还可以有被点击行的 focus 状态<br>
 * 此方法设置 focus 的数据元素
 * @param {String|Number} id 数据元素的 id
 */
setFocusDataById = function (id){};
```

```
/**
 * 设置布局高度
 * @param {Number} v 高度值
 */
setHeight = function (v){};

/**
 * 设置 indent 缩进，该值一般当作图标的宽度
 * @param {Number} v
 */
setIndent = function (v){};

/**
 * 设置行 label 文字颜色
 * @param {color} v
 */
setLabelColor = function (v){};

/**
 * 设置行 label 文字字体
 * @param {String} v
 */
setLabelFont = function (v){};

/**
 * 设置行 label 文字选中颜色
 * @param {color} v
 */
setLabelSelectColor = function (v){};

/**
 * 设置延迟加载器
 * @param {Object} v
 * @example //示例：
 * treeView.setLoader({
 *   load: function(data) {
 *       //展开此 data 时回调，可用于加载子节点
 *   },
 *   isLoaded: function(data) {
 *       //返回此 data 的子结点是否已加载
 *   }
 * });
 *
 */
```

```
setLoader = function (v){};

/**
 * 指定根节点，默认为空，从 DataModel#getRoots()的对象开始展示
 * @param {ht.Data} v
 */
setRootData = function (v){};

/**
 * 设置根节点是否可见
 * @param {Boolean} v
 */
setRootVisible = function (v){};

/**
 * 设置行高
 * @param {Number} v
 */
setRowHeight = function (v){};

/**
 * 设置行线颜色
 * @param {color} color
 */
setRowLineColor = function (color){};

/**
 * 设置行线是否可见
 * @param {Boolean} v
 */
setRowLineVisible = function (v){};

/**
 * 设置滚动条颜色
 * @param {color} color 颜色值
 */
setScrollBarColor = function (color){};

/**
 * 设置滚动条宽度
 * @param {Number} size 宽度值
 */
setScrollBarSize = function (size){};
```

```
/**
 * 设置选择过滤器函数
 * @param {Function} func 过滤器函数
 */
setSelectableFunc = function (func){};

/**
 * 设置行选中背景颜色
 * @param {color} color
 */
setSelectBackground = function (color){};

/**
 * 设置组件是否共享选中模型
 * @param {Boolean} v
 */
setSelectionModelShared = function (v){};

/**
 * 设置排序函数
 * @param {Function} func
 */
setSortFunc = function (func){};

/**
 * 设置垂直平移值(水平平移值无效)
 * @param {Number} x 水平平移值，此参数无效
 * @param {Number} y 垂直平移值
 * @param {Boolean} anim 是否使用动画
 */
setTranslate = function (x, y, anim){};

/**
 * 设置垂直平移值
 * @param {Number} v 垂直平移值
 */
setTranslateY = function (v){};

/**
 * 设置可见过滤器
 * @param {Function} func 过滤器函数
 */
setVisibleFunc = function (func){};
```

```
/**
 * 设置布局宽度
 * @param {Number} v 宽度值
 */
setWidth = function (v){};

/**
 * 显示垂直滚动条
 */
showVBar = function (){};

/**
 * 获取选中模型，{@link ht.widget.TreeView#getSelectionModel getSelectionModel}的缩写
 * @see {@link ht.widget.TreeView#getSelectionModel getSelectionModel}
 * @return {ht.SelectionModel}
 */
sm = function (){};

/**
 * 展开或合并 data 对象
 * @param {ht.Data} data 数据元素
 */
toggle = function (data){};

/**
 * 在当前值基础上增加垂直平移值(水平无效)
 * @param {Number} x 新增的水平平移值，此参数无效
 * @param {Number} y 新增的垂直平移值
 * @param {Boolean} [anim] 是否使用动画
 */
translate = function (x, y, anim){};

/**
 * 获取或设置垂直平移值，没有参数时相当于{@link ht.widget.TreeView#getTranslateY getTranslateY}，有参数时相当于{@link ht.widget.TreeView#setTranslateY setTranslateY}
 * @param {Number} value 平移值
 */
ty = function (value){};

/**
 * 删除自身属性变化事件监听器，{@link ht.widget.TreeView#removePropertyChangeListener removePropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
```

```
 * @see {@link ht.widget.TreeView#removePropertyChangeListener
removePropertyChangeListener}
 */
ump = function (listener, scope){};

/**
 * 立刻刷新组件
 */
validate = function (){};
```

ht.widget.SplitView

```
/**
 * 分割组件，用于左右或上下分割两个组件
 * @param {Object|HTMLElement} leftView 左侧或顶部组件
 * @param {Object|HTMLElement} rightView 右侧或底部组件
 * @param {Boolean} orientation 布局方式，v 上下布局，h 左右布局
 * @param {Number} position 分割条位置，0-1 之间表示百分比，大于 1 表示绝对尺寸，正数指定
左侧或顶部组件的尺寸，负数指定右侧或底部组件的尺寸
 * @constructor
 */
ht.widget.SplitView = function(leftView, rightView, orientation, position) {};

/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
```

```
 * @see {@link ht.widget.SplitView#mp mp}
 */
addPropertyChangeListener = function (listener, scope, ahead){};

/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 获取分割条背景色
 * @return {color}
 */
getDividerBackground = function (){};

/**
 * 获取分割条 DIV
 * @return {HTMLDivElement}
 */
getDividerDiv = function (){};

/**
 * 获取分割条宽度
 * @return {Number}
 */
getDividerSize = function (){};

/**
 * 获取分割条拖拽时的透明度，默认为 0.5
 * @return {Number}
 */
getDragOpacity = function (){};

/**
 * 获取布局高度
 * @return {Number}
 */
getHeight = function (){};

/**
 * 获取左侧组件
```

```
 * @return {Object|HTMLElement}
 */
getLeftView = function (){};

/**
 * 获取布局方式，v 上下布局，h 左右布局
 * @return {String}
 */
getOrientation = function (){};

/**
 * 获取分割条位置，0-1 之间表示百分比，大于 1 表示绝对尺寸，正数指定左侧或顶部组件的尺寸，
负数指定右侧或底部组件的尺寸
 * @return {Number}
 */
getPosition = function (){};

/**
 * 获取右侧组件
 * @return {Object|HTMLElement}
 */
getRightView = function (){};

/**
 * 获取 toggle 状态
 * @return {String}
 * <ul>
 * <li>normal 代表中间分割状态</li>
 * <li>cl 代表 collapse left 关闭左侧或顶部组件</li>
 * <li>cr 代表 collapse right 关闭右侧或底部组件</li>
 * </ul>
 */
getStatus = function (){};

/**
 * 获取分割条上的 toggle 图标
 * @return {String}
 */
getToggleIcon = function (){};

/**
 * 获取组件的根层 div
 * @return {HTMLDivElement}
 */
```

```
getView = function (){};

/**
 * 获取布局宽度
 * @return {Number}
 */
getWidth = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.SplitView#iv iv}
 */
invalidate = function (delay){};

/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
isDisabled = function (){};

/**
 * 获取是否允许拖拽分割条，默认为 true
 * @return {Boolean}
 */
isDraggable = function (){};

/**
 * 获取分割点是否可通过点击直接展开和关闭，默认为 true
 * @return {Boolean}
 */
isTogglable = function (){};

/**
 * 无效组件，并调用延时刷新，{@link ht.widget.SplitView#invalidate invalidate}的缩写
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.SplitView#invalidate invalidate}
 */
iv = function (delay){};

/**
 * 增加自身属性变化事件监听器，{@link ht.widget.SplitView#addPropertyChangeListener
addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
```

```
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.SplitView#addPropertyChangeListener addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};

/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
 * 设置分割条背景色
 * @param {color} background
 */
setDividerBackground = function (background){};

/**
 * 设置分割条宽度
 * @param {Number} size
 */
setDividerSize = function (size){};

/**
 * 设置是否允许拖拽分割条，默认为 true
 * @param {Boolean} draggable
 */
```

```
setDraggable = function (draggable){};

/**
 * 设置分割条拖拽时的透明度，默认为 0.5
 * @param {Number} opacity
 */
setDragOpacity = function (opacity){};

/**
 * 设置布局高度
 * @param {Number} height
 */
setHeight = function (height){};

/**
 * 设置左侧组件
 * @param {Object|HTMLElement} left
 */
setLeftView = function (left){};

/**
 * 设置布局方式，v 上下布局，h 左右布局
 * @param {String} orientation
 */
setOrientation = function (orientation){};

/**
 * 设置分割条位置，0-1 之间表示百分比，大于 1 表示绝对尺寸，正数指定左侧或顶部组件的尺寸，
负数指定右侧或底部组件的尺寸
 * @param {Number} position
 */
setPosition = function (position){};

/**
 * 设置右侧组件
 * @param {Object|HTMLElement} right
 */
setRightView = function (right){};

/**
 * 设置 toggle 状态
 * @param {String} status
 * <ul>
 * <li>normal 代表中间分割状态</li>
```

```
 * <li>cl 代表 collapse left 关闭左侧或顶部组件</li>
 * <li>cr 代表 collapse right 关闭右侧或底部组件</li>
 * </ul>
 */
setStatus = function (status){};

/**
 * 设置分割点是否可通过点击直接展开和关闭，默认为 true
 * @param {Boolean} togglable
 */
setTogglable = function (togglable){};

/**
 * 设置分割条上的 toggle 图标
 * @param {String} icon
 */
setToggleIcon = function (icon){};

/**
 * 设置布局宽度
 * @param {Number} width
 */
setWidth = function (width){};

/**
 * 删除自身属性变化事件监听器，{@link ht.widget.SplitView#removePropertyChangeListener removePropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.widget.SplitView#removePropertyChangeListener removePropertyChangeListener}
 */
ump = function (listener, scope){};

/**
 * 立刻刷新组件
 */
validate = function (){};
```

ht.widget.TabView

```
/**
 * 页签组件，以页签的方式呈现多组件，页签支持拖拽和关闭等功能
```

```
 * @constructor
 */
ht.widget.TabView = function() {};

/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.TabView#mp mp}
 */
addPropertyChangeListener = function (listener, scope, ahead){};

/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 获取指定的 Tab 对象，参数可为 Tab 的标签文字或索引
 * @param {String|Number} nameOrIndex 标签文字或索引
 */
get = function (nameOrIndex){};

/**
 * 获取组件的内容区域 Div
 * @return {HTMLDivElement}
 */
getContentDiv = function (){};

/**
 * 获取当前选中的 Tab 对象
 * @return {ht.Tab}
 */
getCurrentTab = function (){};

/**
 * 获取布局高度
 * @return {Number}
 */
```

```
getHeight = function (){};

/**
 * 获取提示插入位置颜色
 * @return {color}
 */
getInsertColor = function (){};

/**
 * 获取 tab 对象显示的文字，默认返回 tab.toLabel()，可重载自定义
 * @param {ht.Tab} tab
 * @return {String}
 */
getLabel = function (tab){};

/**
 * 获取页签文字颜色，可重载自定义
 * @return {color}
 */
getLabelColor = function (){};

/**
 * 获取页签文字字体，可重载自定义
 * @return {String}
 */
getLabelFont = function (){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.TabView#lp lp}
 */
getLogicalPoint = function (event){};

/**
 * 获取移动时的页签背景色
 * @return {color}
 */
getMoveBackground = function (){};

/**
 * 获取页签选中线条背景色
 * @return {color}
```

```
 */
getSelectBackground = function (){};

/**
 * 获取页签选中的线条宽度，默认值为 3
 * @return {Number}
 */
getSelectWidth = function (){};

/**
 * 获取页签背景色
 * @return {color}
 */
getTabBackground = function (){};

/**
 * 获取页签间隔，默认值为 1
 * @return {Number}
 */
getTabGap = function (){};

/**
 * 获取页签高度
 * @return {Number}
 */
getTabHeight = function (){};

/**
 * 获取页签模型容器，用于增删 Tab 页签
 * @return {ht.DataModel}
 */
getTabModel = function (){};

/**
 * 获取页签位置，可用值有：top|bottom|left|right|left-vertical|right-vertical，默认值为
top
 * @return {String}
 */
getTabPosition = function (){};

/**
 * 获取页签宽度，可重载自定义
 * @param {ht.Tab} tab 页签
 * @return {Number}
```

```
 */
getTabWidth = function (tab){};

/**
 * 获取页签的 div 容器
 * @return {HTMLDivElement}
 */
getTitleDiv = function (){};

/**
 * 获取水平平移(滚动)值
 * @return {Number}
 */
getTranslateX = function (){};

/**
 * 获取垂直平移(滚动)值
 * @return {Number}
 */
getTranslateY = function (){};

/**
 * 获取组件的根层 div
 * @return {HTMLDivElement}
 */
getView = function (){};

/**
 * 获取布局宽度
 * @return {Number}
 */
getWidth = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TabView#iv iv}
 */
invalidate = function (delay){};

/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
```

```
isDisabled = function (){};

/**
 * 获取页签是否可拖拽移动改变显示顺序，默认值为 true
 * @return {Boolean}
 */
isMovable = function (){};

/**
 * 无效组件，并调用延时刷新，{@link ht.widget.TabView#invalidate invalidate}的缩写
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TabView#invalidate invalidate}
 */
iv = function (delay){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标，{@link
ht.widget.TabView#getLogicalPoint getLogicalPoint}的缩写
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.TabView#getLogicalPoint getLogicalPoint}
 */
lp = function (event){};

/**
 * 增加自身属性变化事件监听器，{@link ht.widget.TabView#addPropertyChangeListener
addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.TabView#addPropertyChangeListener addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};

/**
 * 当前选中 Tab 对象变化时回调，可重载做后续处理
 * @param {ht.Tab} oldTab 旧页签
 * @param {ht.Tab} newTab 新选中的页签
 */
onTabChanged = function (oldTab, newTab){};

/**
 * 关闭 Tab 页签回调函数，可重载做后续处理
 * @param {ht.Tab} tab 被关闭的页签
```

```
 * @param {Number} index 索引
 */
onTabClosed = function (tab, index){};

/**
 * 删除指定的 Tab
 * @param {ht.Tab|Number|String} tab Tab 对象，或整数类型的索引，或页签文字
 */
remove = function (tab){};

/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 选中指定的 Tab
 * @param {ht.Tab|Number|String} tab Tab 对象，或整数类型的索引，或页签文字
 */
select = function (tab){};


/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
 * 设置布局高度
 * @param {Number} height 高度值
 */
setHeight = function (height){};
```

```
/**
 * 设置提示插入位置颜色
 * @param {color} color
 */
setInsertColor = function (color){};

/**
 * 设置页签文字颜色
 * @param {color} color
 */
setLabelColor = function (color){};

/**
 * 设置页签文字字体
 * @param {String} font
 */
setLabelFont = function (font){};

/**
 * 设置页签是否可拖拽移动改变显示顺序，默认值为 true
 * @param {Boolean} v
 */
setMovable = function (v){};

/**
 * 设置移动时的页签背景色
 * @param {color} color
 */
setMoveBackground = function (color){};

/**
 * 设置页签选中线条背景色
 * @param {color} color
 */
setSelectBackground = function (color){};

/**
 * 设置页签选中的线条宽度，默认值为 3
 * @param {Number} width
 */
setSelectWidth = function (width){};

/**
 * 设置页签背景色
```

```
 * @param {color} color
 */
setTabBackground = function (color){};

/**
 * 设置页签间隔，默认值为 1
 * @param {Number} v
 */
setTabGap = function (v){};

/**
 * 设置页签高度
 * @param {Number} v
 */
setTabHeight = function (v){};

/**
 * 设置页签位置，可用值有：top|bottom|left|right|left-vertical|right-vertical，默认值为 top
 * @param {String} v
 */
setTabPosition = function (v){};

/**
 * 设置组件水平平移(滚动)值
 * @param {Number} x 水平平移(滚动)值
 */
setTranslateX = function (x){};

/**
 * 设置组件垂直平移(滚动)值
 * @param {Number} y 垂直平移(滚动)值
 */
setTranslateY = function (y){};

/**
 * 设置布局宽度
 * @param {Number} width
 */
setWidth = function (width){};

/**
 * 获取或设置水平平移(滚动)值，没有参数时相当于{@link ht.widget.TabView#getTranslateX getTranslateX}，有参数时相当于{@link ht.widget.TabView#setTranslateX setTranslateX}
```

```
 * @param {Number} value 平移(滚动)值
 */
tx = function (value){};

/**
 * 获取或设置垂直平移(滚动)值，没有参数时相当于{@link ht.widget.TabView#getTranslateY getTranslateY}，有参数时相当于{@link ht.widget.TabView#setTranslateY setTranslateY}
 * @param {Number} value 平移(滚动)值
 */
ty = function (value){};

/**
 * 删除自身属性变化事件监听器，{@link ht.widget.TabView#removePropertyChangeListener removePropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.widget.TabView#removePropertyChangeListener removePropertyChangeListener}
 */
ump = function (listener, scope){};

/**
 * 立刻刷新组件
 */
validate = function (){};
```

ht.widget.Toolbar

```
/**
 * 工具条组件，提供按钮等组件的水平摆放功能
 * @param {Array} items 配置 json，详细内容可以参考 Toolbar 手册
 * @constructor
 */
ht.widget.Toolbar = function(items) {};

/**
 * 在指定 index 位置插入新元素，index 为空代表插入到最后
 * @param {Object} item 监听器函数
 * @param {Number} [index] 监听器函数域
 */
addItem = function (item, index){};

/**
```

```
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.Toolbar#mp mp}
 */
addPropertyChangeListener = function (listener, scope, ahead){};

/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 关闭 ToolTip 功能
 */
disableToolTip = function (){};

/**
 * 绘制元素，并返回该元素所占的宽度值
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {Object} item 元素
 * @param {Number} x x 坐标
 * @param {Number} height 绘制的高度
 * @return {Number} 宽度值
 */
drawItem = function (g, item, x, height){};

/**
 * 启用 ToolTip
 */
enableToolTip = function (){};

/**
 * 获取布局高度
 * @return {Number}
 */
getHeight = function (){};

/**
 * 获取指定 id 对应的元素，id 值为 item 元素上的 id 属性定义
```

```
 * @param {Object} id
 * @return {Object}
 */
getItemById = function (id){};

/**
 * 获取元素之间的间距
 * @return {Number}
 */
getItemGap = function (){};

/**
 * 获取工具条元素数组
 * @return {Array}
 */
getItems = function (){};

/**
 * 获取文本颜色，可重载自定义
 * @param {Object} item
 * @return {color}
 */
getLabelColor = function (item){};

/**
 * 获取文本字体，可重载自定义
 * @return {String}
 */
getLabelFont = function (){};

/**
 * 获取文本选中颜色
 * @return {color}
 */
getLabelSelectColor = function (){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.Toolbar#lp lp}
 */
getLogicalPoint = function (e){};
```

```
/**
 * 获取选中元素的背景色，可重载自定义
 * @return {color}
 */
getSelectBackground = function (){};

/**
 * 获取分割条颜色
 * @return {color}
 */
getSeparatorColor = function (){};

/**
 * 获取 ToolTip 文字，可重载返回自定义的 toolTip 文字
 * @param {Event} e 鼠标或 Touch 事件对象
 * @return {String} toolTip 文字，默认取出鼠标下的元素，然后返回其 toolTip
 */
getToolTip = function (e){};

/**
 * 获取水平平移(滚动)值
 * @return {Number}
 */
getTranslateX = function (){};

/**
 * 根据 id 获取对应 item 元素值，比如 input 的值
 * @param {Object} id 元素 id
 * @return {Object}
 * @see {@link ht.widget.Toolbar#v v}
 */
getValue = function (id){};

/**
 * 获取组件的根层 div
 * @return {HTMLDivElement}
 */
getView = function (){};

/**
 * 获取布局宽度
 * @return {Number}
 */
```

```
getWidth = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.Toolbar#iv iv}
 */
invalidate = function (delay){};

/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
isDisabled = function (){};

/**
 * 获取是否向右对齐排布，默认为 false
 * @return {Boolean}
 */
isStickToRight = function (){};


/**
 * 无效组件，并调用延时刷新，{@link ht.widget.Toolbar#invalidate invalidate}的缩写
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.Toolbar#invalidate invalidate}
 */
iv = function (delay){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标，{@link
ht.widget.Toolbar#getLogicalPoint getLogicalPoint}的缩写
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.Toolbar#getLogicalPoint getLogicalPoint}
 */
lp = function (event){};

/**
 * 增加自身属性变化事件监听器，{@link ht.widget.Toolbar#addPropertyChangeListener
addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
```

```
 * @see {@link ht.widget.Toolbar#addPropertyChangeListener addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};

/**
 * 重绘组件
 */
redraw = function (){};

/**
 * 删除指定元素
 * @param {Object} item
 */
removeItem = function (item){};

/**
 * 根据 id 删除指定元素
 * @param {Object} id
 */
removeItemById = function (id){};

/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
 * 设置布局高度
```

```
 * @param {Number} height 高度值
 */
setHeight = function (height){};

/**
 * 设置元素之间的间距
 * @param {Number} gap
 */
setItemGap = function (gap){};

/**
 * 设置工具条元素数组
 * @param {Array} items
 */
setItems = function (items){};

/**
 * 设置文本颜色
 * @param {color} v
 */
setLabelColor = function (v){};

/**
 * 设置文本字体
 * @param {String} v
 */
setLabelFont = function (v){};

/**
 * 设置文本选中颜色
 * @param {color} v
 */
setLabelSelectColor = function (v){};

/**
 * 设置选中元素的背景色，可重载自定义
 * @param {color} v
 */
setSelectBackground = function (v){};

/**
 * 设置分割条颜色
 * @param {color} v
 */
```

```
setSeparatorColor = function (v){};

/**
 * 设置是否向右对齐排布，默认为 false
 * @param {Boolean} v
 */
setStickToRight = function (v){};

/**
 * 设置拓扑水平平移(滚动)值
 * @param {Number} x 水平平移(滚动)值
 */
setTranslateX = function (x){};

/**
 * 根据 id 设置对应 item 元素值，比如 input 的值
 * @param {Object} id 元素 id
 * @param {Object} value 值
 * @see {@link ht.widget.Toolbar#v v}
 */
setValue = function (id, value){};

/**
 * 设置布局宽度
 * @param {Number} width 宽度
 */
setWidth = function (width){};

/**
 * 获取或设置水平平移(滚动)值，没有参数时相当于{@link ht.widget.Toolbar#getTranslateX getTranslateX}，有参数时相当于{@link ht.widget.Toolbar#setTranslateX setTranslateX}
 * @param {Number} value 平移(滚动)值
 */
tx = function (value){};

/**
 * 删除自身属性变化事件监听器，{@link ht.widget.Toolbar#removePropertyChangeListener removePropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.widget.Toolbar#removePropertyChangeListener removePropertyChangeListener}
 */
```

```
ump = function (listener, scope){};

/**
 * 根据 id 获取或设置对应 item 元素值，比如 input 的值；没有参数时相当于{@link
ht.widget.Toolbar#getValue getValue}，有参数时相当于{@link ht.widget.Toolbar#setValue
setValue}
 * @param {Object} [id] 元素 id
 * @param {Object} [value] 值
 * @return {Object}
 */
v = function (id, value){};

/**
 * 立刻刷新组件
 */
validate = function (){};
```

ht.widget.TableView

```
/**
 * 表格组件，以表格的方式呈现 DataModel 中 Data 的属性
 * @param {ht.DataModel} dataModel 绑定的数据模型
 * @constructor
 */
ht.widget.TableView = function(dataModel) {};

/**
 * 增加底层 Painter<br>
 * 组件上提供 Painter 接口，开发者可以使用 Canvas 的画笔对象自由绘制任意形状，底层 Painter
绘制在组件最下面
 * @param {Function} painter Painter 类
 * @example //Painter 示例：
 * function MyBottomPainter() {
 * }
 * ht.Default.def(MyBottomPainter, Object, {
 *   draw: function(g) {
 *       g.save();
 *       //draw...
 *       g.restore();
 *   }
 * });
 * tableView.addBottomPainter(MyBottomPainter);
```

```
 */
addBottomPainter = function(painter) {}

/**
 * 以 json 的方式配置表格中的列(新增)
 * @param {Array} columns json 列
 * @example //示例:
 * tableView.addColumns([{
 *   name: 'id',
 *   displayName: '序号'
 * },
 * {
 *   name: 'background',
 *   accessType: 'style'
 * }
 * ]);
 */
addColumns = function(columns) {}


/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.TableView#mp mp}
 */
addPropertyChangeListener = function (listener, scope, ahead){};


/**
 * 增加顶层 Painter<br>
 * 组件上提供 Painter 接口，开发者可以使用 Canvas 的画笔对象自由绘制任意形状，顶层 Painter 绘制在组件最上面
 * @param {Function} painter Painter 类
 * @example //Painter 示例:
 * function MyTopPainter() {
 * }
 * ht.Default.def(MyTopPainter, Object, {
 *   draw: function(g) {
 *       g.save();
 *       //draw...
 *       g.restore();
 *   }
```

```
 * });
 * tableView.addTopPainter(MyTopPainter);
 */
addTopPainter = function (painter){};


/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 传入即将设置的水平平移值，返回最终设置值，可重载限制水平平移范围
 * @param {Number} value 原始水平平移值
 * @return {Number} 新的水平平移值
 */
adjustTranslateX = function (value){};

/**
 * 传入即将设置的垂直平移值，返回最终设置值，可重载限制垂直平移范围
 * @param {Number} value 原始垂直平移值
 * @return {Number} 新的垂直平移值
 */
adjustTranslateY = function (value){};

/**
 * 关闭 ToolTip 功能
 */
disableToolTip = function (){};

/**
 * 获取或设置数据模型，没有参数时相当于{@link ht.widget.TableView#getDataModel
getDataModel}，有参数时相当于{@link ht.widget.TableView#setDataModel setDataModel}
 * @param {ht.DataModel} [dataModel] 数据模型
 * @return {ht.DataModel} dataModel
 */
dm = function (dataModel){};

/**
 * 绘制单元格，可重载自定义单元格渲染，如果返回值为 HTML 元素，则使用 HTML 元素当作 Renderer
 * @param {CanvasRenderingContext2D} g 画笔对象
```

```
 * @param {ht.Data} data 数据元素
 * @param {Boolean} selected 数据元素是否选中
 * @param {ht.Column} column 列信息
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 * @return {HTMLElement}
 */
drawCell = function (g, data, selected, column, x, y, width, height){};

/**
 * 绘制 check 列单元格，可重载自定义
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
 * @param {Boolean} selected 数据元素是否选中
 * @param {ht.Column} column 列信息
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 * @param {Object} view 当前的表格组件
 */
drawCheckColumnCell = function (g, data, selected, column, x, y, width, height, view){};

/**
 * 绘制行背景色，默认仅在选中该行时填充选中背景色，可重载自定义
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
 * @param {Boolean} selected 数据元素是否选中
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 */
drawRowBackground = function (g, data, selected, x, y, width, height){};

/**
 * 启用 ToolTip
 */
enableToolTip = function (){};

/**
```

```
 * 返回 data 对象对应的 check 图标，可重载自定义 check 图标，该函数在 checkMode 模式下有效
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
getCheckIcon = function (data){};

/**
 * 获取鼠标下的列
 * @param {Event} e 鼠标或 Touch 事件
 * @return {ht.Column}
 */
getColumnAt = function (e){};

/**
 * 获取列线颜色
 * @return {color}
 */
getColumnLineColor = function (){};

/**
 * 获取列模型， 列模型用于存储 Column 列对象信息
 * @return {ht.DataModel}
 */
getColumnModel = function (){};

/**
 * 默认返回 sortFunc 函数，当 sortColumn 不为空时将返回其对应的排序函数
 * @return {Function}
 */
getCurrentSortFunc = function (){};

/**
 * 传入逻辑坐标点或者交互 event 事件参数，返回当前点下的数据元素
 * @param {Object|Event} pointOrEvent 逻辑坐标点或交互事件对象(如鼠标事件对象)
 * @return {ht.Data} 点下的数据元素
 */
getDataAt = function (pointOrEvent){};

/**
 * 获取绑定的数据模型
 * @return {ht.DataModel} 数据模型
 */
getDataModel = function (){};
```

```
/**
 * 获取当前可见区域的结束行索引
 * @return {Number}
 */
getEndRowIndex = function (){};

/**
 * 在 checkMode 模式下数据除了被选中有 check 状态外，还可以有被点击行的 focus 状态<br>
 * 此方法获取 focus 数据元素
 * @return {ht.Data}
 */
getFocusData = function (){};

/**
 * 获取布局高度
 * @return {Number}
 */
getHeight = function (){};

/**
 * 获取对应的单元格文本颜色，可重载自定义
 * @param {ht.Data} data 数据元素
 * @param {ht.Column} column 列
 * @param {Object} value 值
 * @return {color}
 */
getLabelColor = function (data, column, value){};

/**
 * 获取对应的单元格文本字体，可重载自定义
 * @param {ht.Data} data 数据元素
 * @param {ht.Column} column 列
 * @param {Object} value 值
 * @return {String}
 */
getLabelFont = function (data, column, value){};

/**
 * 获取文本选中颜色
 * @return {color}
 */
getLableSelectColor = function (){};

/**
```

```
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.TableView#lp lp}
 */
getLogicalPoint = function (event){};

/**
 * 获取当前显示的 Data 对象集合，该集合已被排序和过滤
 * @return {ht.List}
 */
getRowDatas = function (){};

/**
 * 获取行高
 * @return {Number}
 */
getRowHeight = function (){};

/**
 * 获取 data 对象所在的行索引
 * @param {ht.Data} data 数据元素
 * @return {Number}
 */
getRowIndex = function (data){};

/**
 * 获取行线颜色
 * @return {color}
 */
getRowLineColor = function (){};

/**
 * 返回当前可见行总行数
 * @return {Number}
 */
getRowSize = function (){};

/**
 * 获取滚动条颜色
 * @return {color}
 */
getScrollBarColor = function (){};
```

```
/**
 * 获取滚动条宽度
 * @return {Number}
 */
getScrollBarSize = function (){};

/**
 * 获取选择过滤器函数
 * @return {Function}
 */
getSelectableFunc = function (){};

/**
 * 获取行选中背景颜色
 * @return {color}
 */
getSelectBackground = function (){};

/**
 * 获取选中模型
 * @return {ht.SelectionModel}
 * @see {@link ht.widget.TableView#sm sm}
 */
getSelectionModel = function (){};

/**
 * 获取当前排序列
 * @return {ht.Column}
 */
getSortColumn = function (){};

/**
 * 获取排序函数
 * @return {Function}
 */
getSortFunc = function (){};

/**
 * 获取排序方式
 * <ul>
 * <li>none:不允许排序</li>
 * <li>bistate:点击表头时正序和倒序两种方式之间切换</li>
 * <li>tristate:点击表头时正序、倒序、不排序三种方式之间切换</li>
 * </ul>
```

```
 * @return {String}
 */
getSortMode = function (){};

/**
 * 获取当前可见区域的起始行索引
 * @return {Number}
 */
getStartRowIndex = function (){};

/**
 * 获取 ToolTip 文字，可重载返回自定义的 toolTip 文字
 * @param {Event} e 鼠标或 Touch 事件对象
 * @return {String} toolTip 文字，默认取出鼠标下的 data 和 column，然后返回
column.getToolTip(data);
 */
getToolTip = function (e){};

/**
 * 获取水平平移值
 * @return {Number} 水平平移值
 * @see {@link ht.widget.TableView#tx tx}
 */
getTranslateX = function (){};

/**
 * 获取垂直平移值
 * @return {Number} 垂直平移值
 * @see {@link ht.widget.TableView#ty ty}
 */
getTranslateY = function (){};

/**
 * 获取单元格中要显示的内容
 * @param {ht.Data} data 数据元素
 * @param {ht.Column} column 列
 * @return {Object}
 */
getValue = function (data, column){};

/**
 * 获取组件的根层 div
 * @return {HTMLDivElement}
 */
```

```
getView = function (){};

/**
 * 获取组件中可见区域的逻辑尺寸
 * @return {Object}
 */
getViewRect = function (){};

/**
 * 获取可见过滤器函数
 * @return {Function}
 */
getVisibleFunc = function (){};

/**
 * 获取布局宽度
 * @return {Number}
 */
getWidth = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TableView#iv iv}
 */
invalidate = function (delay){};

/**
 * 无效数据元素
 * @param {ht.Data} data 要无效的数据元素
 */
invalidateData = function (data){};

/**
 * 无效模型，最彻底的刷新方式
 * @see {@link ht.widget.TableView#ivm ivm}
 */
invalidateModel = function (){};

/**
 * 是否自动隐藏滚动条
 * @return {Boolean}
 */
```

```
isAutoHideScrollBar = function (){};

/**
 * 选中数据元素时，是否自动平移组件以确保该元素出现在可见区域内
 * @return {Boolean}
 */
isAutoMakeVisible = function (){};

/**
 * 是否启用批量编辑
 * @return {Boolean}
 */
isBatchEditable = function (){};

/**
 * 判断单元格是否可编辑
 * @param {ht.Data} data 数据元素
 * @param {ht.Column} column 列
 * @return {Boolean}
 */
isCellEditable = function (data, column){};

/**
 * 是否是 check 模式，默认为 false，为 true 时自动插入 checkColumn 列
 * @return {Boolean}
 */
isCheckMode = function (){};

/**
 * 获取列线是否可见，默认为 true
 * @return {Boolean}
 */
isColumnLineVisible = function (){};

/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
isDisabled = function (){};

/**
 * 返回可否编辑的总开关，默认为 false，每个 Column 列对象可再控制
 * @return {Boolean}
 */
```

```
isEditable = function (){};

/**
 * 获取行线是否可见，默认为 true
 * @return {Boolean}
 */
isRowLineVisible = function (){};

/**
 * 判断 data 对象是否可被选中
 * @param {ht.Data} data 数据元素
 * @return {Boolean}
 */
isSelectable = function (data){};

/**
 * 判断 data 对象是否被选中
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isSelected = function (data){};

/**
 * 根据 id 判断 data 对象是否被选中
 * @param {String|Number} id 数据元素 id
 * @return {Boolean}
 */
isSelectedById = function (id){};

/**
 * 当前组件是否共享选中模型
 * @return {Boolean}
 */
isSelectionModelShared = function (){};

/**
 * 判断数据元素是否可见
 * @param {ht.Data} data 数据元素
 * @return {Boolean}
 */
isVisible = function (data){};

/**
 * 无效组件，并调用延时刷新，{@link ht.widget.TableView#invalidate invalidate}的缩写
```

```
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TableView#invalidate invalidate}
 */
iv = function (delay){};

/**
 * 无效模型，重新构造内部的 rows 数据，最彻底的刷新方式，{@link
ht.widget.TableView#invalidateModel invalidateModel}的缩写
 * @see {@link ht.widget.TableView#invalidateModel invalidateModel}
 */
ivm = function (){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标，{@link
ht.widget.TableView#getLogicalPoint getLogicalPoint}的缩写
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.TableView#getLogicalPoint getLogicalPoint}
 */
lp = function (event){};

/**
 * 平移组件以确保数据元素在可见区域内
 * @param {ht.Data} data 数据元素
 */
makeVisible = function (data){};

/**
 * 增加自身属性变化事件监听器，{@link ht.widget.TableView#addPropertyChangeListener
addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.TableView#addPropertyChangeListener addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};

/**
 * 列头被点击时调用，可重载做后续处理，如远程排序功能
 * @param {ht.Column} column 列
 */
onColumnClicked = function (column){};

/**
```

```
 * 当 data 所在行被单击时回调，可重载对单击事件做响应
 * @param {ht.Data} data 数据元素
 * @param {Event} e 事件对象
 */
onDataClicked = function (data, e){};

/**
 * 当 data 所在行被双击时回调，可重载对双击事件做响应
 * @param {ht.Data} data 数据元素
 * @param {Event} e 事件对象
 */
onDataDoubleClicked = function (data, e){};

/**
 * 平移动画结束时回调，可重载做后续处理
 */
onTranslateEnded = function (){};

/**
 * 重绘组件中所有行，仅次于 invalidateModel 的彻底刷新方式
 */
redraw = function (){};

/**
 * 删除底层 Painter
 * @param {Object} painter Painter 类
 */
removeBottomPainter = function (painter){};

/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除所有选中的图元
 */
removeSelection = function (){};

/**
 * 删除顶层 Painter
 * @param {Object} painter Painter 类
```

```
 */
removeTopPainter = function (painter){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 平移(滚动)组件至指定的行号
 * @param {Number} index 行号
 */
scrollToIndex = function (index){};

/**
 * 选中所有数据元素
 */
selectAll = function (){};

/**
 * 设置是否自动隐藏滚动条
 * @param {Boolean} v
 */
setAutoHideScrollBar = function (v){};

/**
 * 设置当选中数据元素，是否自动平移(滚动)组件以确保该数据元素出现在可见区域内
 * @param {Boolean} v
 */
setAutoMakeVisible = function (v){};

/**
 * 设置是否启用批量编辑
 * @param {Boolean} v
 */
setBatchEditable = function (v){};

/**
 * 设置是否为 check 模式，默认为 false，为 true 时自动插入 checkColumn 列
 * @param {Boolean} v
 */
```

```
setCheckMode = function (v){};

/**
 * 设置列线颜色
 * @param {color} color
 */
setColumnLineColor = function (color){};

/**
 * 设置列线是否可见
 * @param {Boolean} v
 */
setColumnLineVisible = function (v){};

/**
 * 以 json 的方式配置表格中的列(设置)
 * @param {Array} columns json 列
 * @example //示例：
 * tableView.setColumns([{
 *   name: 'id',
 *   displayName: '序号'
 * },
 * {
 *   name: 'background',
 *   accessType: 'style'
 * }
 * ]);
 */
setColumns = function (v){};

/**
 * 设置绑定的数据模型
 * @param {ht.DataModel} dataModel 数据模型
 */
setDataModel = function (dataModel){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
```

```
 * 设置可否编辑的总开关，默认为 false，每个 Column 列对象可再控制
 * @param {Boolean} editable
 */
setEditable = function (editable){};

/**
 * 在 checkMode 模式下数据除了被选中有 check 状态外，还可以有被点击行的 focus 状态<br>
 * 此方法设置 focus 的数据元素
 * @param {ht.Data} data 数据元素
 */
setFocusData = function (data){};

/**
 * 在 checkMode 模式下数据除了被选中有 check 状态外，还可以有被点击行的 focus 状态<br>
 * 此方法设置 focus 的数据元素
 * @param {String|Number} id 数据元素的 id
 */
setFocusDataById = function (id){};

/**
 * 设置布局高度
 * @param {Number} v 高度值
 */
setHeight = function (v){};

/**
 * 设置文本颜色
 * @param {color} color
 */
setLabelColor = function (color){};

/**
 * 设置文本字体
 * @param {String} font
 */
setLabelFont = function (font){};

/**
 * 设置选中文本颜色
 * @param {color} color
 */
setLabelSelectColor = function (color){};

/**
```

```
 * 设置行高
 * @param {Number} v
 */
setRowHeight = function (v){};

/**
 * 设置行线颜色
 * @param {color} color
 */
setRowLineColor = function (color){};

/**
 * 设置行线是否可见
 * @param {Boolean} v
 */
setRowLineVisible = function (v){};

/**
 * 设置滚动条颜色
 * @param {color} color 颜色值
 */
setScrollBarColor = function (color){};

/**
 * 设置滚动条宽度
 * @param {Number} size 宽度值
 */
setScrollBarSize = function (size){};

/**
 * 设置选择过滤器函数
 * @param {Function} func 过滤器函数
 */
setSelectableFunc = function (func){};

/**
 * 设置行选中背景颜色
 * @param {color} color
 */
setSelectBackground = function (color){};

/**
 * 设置组件是否共享选中模型
 * @param {Boolean} v
```

```
 */
setSelectionModelShared = function (v){};

/**
 * 设置排序列
 * @param {ht.Column} column
 */
setSortColumn = function (column){};

/**
 * 设置排序函数
 * @param {Function} func
 */
setSortFunc = function (func){};

/**
 * 设置排序方式
 * @param {String} mode
 * <ul>
 * <li>none:不允许排序</li>
 * <li>bistate:点击表头时正序和倒序两种方式之间切换</li>
 * <li>tristate:点击表头时正序、倒序、不排序三种方式之间切换</li>
 * </ul>
 */
setSortMode = function (mode){};

/**
 * 设置水平和垂直平移值
 * @param {Number} x 水平平移值
 * @param {Number} y 垂直平移值
 * @param {Boolean} [anim] 是否使用动画
 */
setTranslate = function (x, y, anim){};

/**
 * 设置水平平移值
 * @param {Number} x 水平平移值
 */
setTranslateX = function (x){};

/**
 * 设置垂直平移值
 * @param {Number} y 垂直平移值
 */
```

```
setTranslateY = function (y){};

/**
 * 设置可见过滤器
 * @param {Function} func 过滤器函数
 */
setVisibleFunc = function (func){};

/**
 * 设置布局宽度
 * @param {Number} v 宽度值
 */
setWidth = function (v){};

/**
 * 显示水平滚动条
 */
showHBar = function (){};

/**
 * 显示垂直滚动条
 */
showVBar = function (){};

/**
 * 获取选中模型，{@link ht.widget.TableView#getSelectionModel getSelectionModel}的缩写
 * @see {@link ht.widget.TableView#getSelectionModel getSelectionModel}
 * @return {ht.SelectionModel}
 */
sm = function (){};

/**
 * 在当前值基础上增加水平和垂直平移值
 * @param {Number} x 新增的水平平移值
 * @param {Number} y 新增的垂直平移值
 * @param {Boolean} [anim] 是否使用动画
 */
translate = function (x, y, anim){};

/**
 * 获取或设置水平平移值，没有参数时相当于{@link ht.widget.TableView#getTranslateX getTranslateX}，有参数时相当于{@link ht.widget.TableView#setTranslateX setTranslateX}
 * @param {Number} value 平移值
 */
```

```
tx = function (value){};

/**
 * 获取或设置垂直平移值，没有参数时相当于{@link ht.widget.TableView#getTranslateY getTranslateY}，有参数时相当于{@link ht.widget.TableView#setTranslateY setTranslateY}
 * @param {Number} value 平移值
 */
ty = function (value){};

/**
 * 删除自身属性变化事件监听器，{@link ht.widget.TableView#removePropertyChangeListener removePropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.widget.TableView#removePropertyChangeListener removePropertyChangeListener}
 */
ump = function (listener, scope){};

/**
 * 立刻刷新组件
 */
validate = function (){};
```

ht.widget.TableHeader

```
/**
 * 表头组件，常与 TableView 和 TreeTableView 结合呈现 Column 信息，并提供 Column 的排序、大小和位置变化等交互操作功能
 * @param {ht.widget.TableView|ht.widget.TreeTableView} tableView 表格组件
 * @constructor
 */
ht.widget.TableHeader = function(tableView) {};

/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.TableHeader#mp mp}
 */
addPropertyChangeListener = function (listener, scope, ahead){};
```

```
/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 绘制列头，可重载自定义
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Column} column 列信息
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 */
drawColumn = function (g, column, x, y, width, height){};

/**
 * 返回 check 图标
 * @return {String}
 */
getCheckIcon = function (){};

/**
 * 获取列线颜色
 * @return {color}
 */
getColumnLineColor = function (){};

/**
 * 获取布局高度
 * @return {Number}
 */
getHeight = function (){};

/**
 * 获取缩进，一般当作列头图标的宽度
 * @return {Number}
 */
getIndent = function (){};

/**
```

```
 * 获取移动列时可插入位置的提示颜色
 * @return {color}
 */
getInsertColor = function (){};

/**
 * 获取列头文字信息，默认返回 column.toLabel()，可重载自定义
 * @param {ht.Column} column
 * @return {String}
 */
getLabel = function (column){};

/**
 * 获取列头文字水平对齐方式，默认会考虑 column.getAlign()值，可重载自定义
 * @return {String}
 */
getLabelAlign = function (column){};

/**
 * 获取列头文字颜色，默认会返回 column.getColor()||tableHeader.getLabelColor();
 * @param {ht.Column} column 列
 * @return {color}
 */
getLabelColor = function (column){};

/**
 * 获取列头文字字体，可重载自定义
 * @param {ht.Column} column 列
 * @return {String}
 */
getLabelFont = function (column){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.TableHeader#lp lp}
 */
getLogicalPoint = function (event){};

/**
 * 获取移动列时的列头背景色
 * @return {color}
 */
```

```
getMoveBackground = function (){};

/**
 * 获取表头列升序图标
 * @return {String}
 */
getSortAscIcon = function (){};

/**
 * 获取表头列降序图标
 * @return {String}
 */
getSortDescIcon = function (){};

/**
 * 获取绑定的表格组件
 * @return {ht.widget.TableView|ht.widget.TreeTableView}
 */
getTableView = function (){};

/**
 * 获取组件的根层 div
 * @return {HTMLDivElement}
 */
getView = function (){};

/**
 * 获取布局宽度
 * @return {Number}
 */
getWidth = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TableHeader#iv iv}
 */
invalidate = function (delay){};

/**
 * 获取列线是否可见，默认为 true
 * @return {Boolean}
 */
```

```
isColumnLineVisible = function (){};

/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
isDisabled = function (){};

/**
 * 获取列顺序是否允许移动改变，默认为 true
 * @return {Boolean}
 */
isMovable = function (){};

/**
 * 获取列宽是否允许改变，默认为 true
 * @return {Boolean}
 */
isResizable = function (){};

/**
 * 无效组件，并调用延时刷新，{@link ht.widget.TableHeader#invalidate invalidate}的缩写
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TableHeader#invalidate invalidate}
 */
iv = function (delay){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标，{@link
ht.widget.TableHeader#getLogicalPoint getLogicalPoint}的缩写
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.TableHeader#getLogicalPoint getLogicalPoint}
 */
lp = function (event){};

/**
 * 增加自身属性变化事件监听器，{@link ht.widget.TableHeader#addPropertyChangeListener
addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.TableHeader#addPropertyChangeListener addPropertyChangeListener}
 */
```

```
mp = function (listener, scope, ahead){};

/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 设置 check 图标
 * @param {String} icon
 */
setCheckIcon = function (icon){};

/**
 * 设置列线颜色
 * @param {color} color
 */
setColumnLineColor = function (color){};

/**
 * 设置列线是否可见
 * @param {Boolean} v
 */
setColumnLineVisible = function (v){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
 * 设置布局高度
 * @param {Number} v 高度值
```

```
 */
setHeight = function (v){};

/**
 * 设置缩进，一般当作列头图标的宽度
 * @param {Number} v
 */
setIndent = function (v){};

/**
 * 设置移动列时可插入位置的提示颜色
 * @param {color} color
 */
setInsertColor = function (color){};

/**
 * 设置列头文本颜色
 * @param {color} color
 */
setLabelColor = function (color){};

/**
 * 设置列头文本字体
 * @param {String} font
 */
setLabelFont = function (font){};

/**
 * 设置列顺序是否允许移动改变，默认为 true
 * @param {Boolean} movable
 */
setMovable = function (movable){};

/**
 * 设置移动列时的列头背景色
 * @param {color} color
 * @return {color}
 */
setMoveBackground = function (color){};

/**
 * 设置列宽是否允许改变，默认为 true
 * @param {Boolean} v
 */
```

```
setResizable = function (v){};

/**
 * 设置表头列升序图标
 * @param {String} icon
 */
setSortAscIcon = function (icon){};

/**
 * 设置表头列降序图标
 * @param {String} icon
 */
setSortDescIcon = function (icon){};

/**
 * 设置布局宽度
 * @param {Number} v 宽度值
 */
setWidth = function (v){};

/**
 * 删除自身属性变化事件监听器，{@link ht.widget.TableHeader#removePropertyChangeListener removePropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.widget.TableHeader#removePropertyChangeListener removePropertyChangeListener}
 */
ump = function (listener, scope){};

/**
 * 立刻刷新组件
 */
validate = function (){};
```

ht.widget.TablePane

```
/**
 * 表格面板，组合了 TableHeader 和 TableView 两个子组件
 * @param {ht.widget.TableView} tableView 绑定的表格组件
 * @constructor
 */
```

```
ht.widget.TablePane = function(tableView) {};

/**
 * 以 json 的方式配置表格中的列(新增)，内部调用 tableView 的 addColumns 方法
 * @param {Array} columns json 列
 * @example //示例：
 * tablePane.addColumns([{
 *   name: 'id',
 *   displayName: '序号'
 * },
 * {
 *   name: 'background',
 *   accessType: 'style'
 * }
 * ]);
 */
addColumns = function(columns) {}

/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 获取列模型， 列模型用于存储 Column 列对象信息，内部调用 tableView 的 getColumnModel 方法
 * @return {ht.DataModel}
 */
getColumnModel = function (){};

/**
 * 获取绑定的数据模型，内部调用 tableView 的 getDataModel 方法
 * @return {ht.DataModel} 数据模型
 */
getDataModel = function (){};

/**
 * 获取布局高度
 * @return {Number}
 */
getHeight = function (){};
```

```
/**
 * 获取表头组件
 * @return {ht.widget.TableHeader}
 */
getTableHeader = function (){};

/**
 * 获取表格组件
 * @return {ht.widget.TableView}
 */
getTableView = function (){};

/**
 * 获取组件的根层 div
 * @return {HTMLDivElement}
 */
getView = function (){};

/**
 * 获取布局宽度
 * @return {Number}
 */
getWidth = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TablePane#iv iv}
 */
invalidate = function (delay){};

/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
isDisabled = function (){};

/**
 * 无效组件，并调用延时刷新，{@link ht.widget.TablePane#invalidate invalidate}的缩写
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TablePane#invalidate invalidate}
 */
iv = function (delay){};
```

```
/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 以 json 的方式配置表格中的列(设置)，内部调用 tableView 的 setColumns 方法
 * @param {Array} columns json 列
 * @example //示例：
 * tablePane.setColumns([{
 *  name: 'id',
 *  displayName: '序号'
 * },
 * {
 *  name: 'background',
 *  accessType: 'style'
 * }
 * ]);
 */
setColumns = function (v){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
 * 设置布局高度
 * @param {Number} v 高度值
 */
setHeight = function (v){};

/**
 * 设置布局宽度
 * @param {Number} v 宽度值
 */
setWidth = function (v){};

/**
 * 立刻刷新组件
```

```
 */
validate = function (){};
```

ht.widget.TreeTableView

```
/**
 * 树表组件，以树形和表格的方式组合呈现 DataModel 中 Data 的父子及属性信息
 * @param {ht.DataModel} dataModel 绑定的数据模型
 * @constructor
 */
ht.widget.TreeTableView = function(dataModel) {};

/**
 * 增加底层 Painter<br>
 * 组件上提供 Painter 接口，开发者可以使用 Canvas 的画笔对象自由绘制任意形状，底层 Painter 绘制在组件最下面
 * @param {Function} painter Painter 类
 * @example //Painter 示例：
 * function MyBottomPainter() {
 * }
 * ht.Default.def(MyBottomPainter, Object, {
 *   draw: function(g) {
 *       g.save();
 *       //draw...
 *       g.restore();
 *   }
 * });
 * treeTableView.addBottomPainter(MyBottomPainter);
 */
addBottomPainter = function(painter) {}

/**
 * 以 json 的方式配置表格中的列(新增)
 * @param {Array} columns json 列
 * @example //示例：
 * treeTableView.addColumns([{
 *   name: 'id',
 *   displayName: '序号'
 * },
 * {
 *   name: 'background',
 *   accessType: 'style'
 * }
```

```
 * ]);
 */
addColumns = function(columns) {}


/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.TreeTableView#mp mp}
 */
addPropertyChangeListener = function (listener, scope, ahead){};


/**
 * 增加顶层 Painter<br>
 * 组件上提供 Painter 接口，开发者可以使用 Canvas 的画笔对象自由绘制任意形状，顶层 Painter 绘制在组件最上面
 * @param {Function} painter Painter 类
 * @example //Painter 示例：
 * function MyTopPainter() {
 * }
 * ht.Default.def(MyTopPainter, Object, {
 *  draw: function(g) {
 *      g.save();
 *      //draw...
 *      g.restore();
 *  }
 * });
 * treeTableView.addTopPainter(MyTopPainter);
 */
addTopPainter = function (painter){};


/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
```

```
 * 传入即将设置的水平平移值，返回最终设置值，可重载限制水平平移范围
 * @param {Number} value 原始水平平移值
 * @return {Number} 新的水平平移值
 */
adjustTranslateX = function (value){};

/**
 * 传入即将设置的垂直平移值，返回最终设置值，可重载限制垂直平移范围
 * @param {Number} value 原始垂直平移值
 * @return {Number} 新的垂直平移值
 */
adjustTranslateY = function (value){};

/**
 * 合并 data 对象
 * @param {ht.Data} data 数据元素
 */
collapse = function (data){};

/**
 * 合并所有对象
 */
collapseAll = function (){};

/**
 * 关闭 ToolTip 功能
 */
disableToolTip = function (){};

/**
 * 获取或设置数据模型，没有参数时相当于{@link ht.widget.TreeTableView#getDataModel
getDataModel}，有参数时相当于{@link ht.widget.TreeTableView#setDataModel setDataModel}
 * @param {ht.DataModel} [dataModel] 数据模型
 * @return {ht.DataModel} dataModel
 */
dm = function (dataModel){};

/**
 * 绘制单元格，可重载自定义单元格渲染，如果返回值为 HTML 元素，则使用 HTML 元素当作 Renderer
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
 * @param {Boolean} selected 数据元素是否选中
 * @param {ht.Column} column 列信息
 * @param {Number} x 左上角 x 坐标
```

```
 * @param {Number} y 左上角 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 * @return {HTMLElement}
 */
drawCell = function (g, data, selected, column, x, y, width, height){};

/**
 * 绘制图标，可重载自定义
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 */
drawIcon = function (g, data, x, y, width, height){};

/**
 * 绘制文本，可重载自定义，label 一般绘制在最后因此没有 width 参数限制
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} height 绘制的高度
 */
drawLabel = function (g, data, x, y, height){};
/**
 * 绘制行背景色，默认仅在选中该行时填充选中背景色，可重载自定义
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Data} data 数据元素
 * @param {Boolean} selected 数据元素是否选中
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} width 绘制的宽度
 * @param {Number} height 绘制的高度
 */
drawRowBackground = function (g, data, selected, x, y, width, height){};

/**
 * 启用 ToolTip
 */
enableToolTip = function (){};
```

```
/**
 * 展开 data 对象
 * @param {ht.Data} data 数据元素
 */
expand = function (data){};

/**
 * 展开所有对象
 */
expandAll = function (){};

/**
 * 获取数据元素 icon 的背景色，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {color} 颜色值，默认返回 data.s('body.color')
 */
getBodyColor = function (data){};

/**
 * 获取数据元素 icon 的边框色，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {color} 颜色值，默认返回 data.s('border.color')
 */
getBorderColor = function (data){};

/**
 * 返回 data 对象对应的 check 图标，可重载自定义 check 图标，该函数在 checkMode 模式下有效
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
getCheckIcon = function (data){};

/**
 * 获取 check 模式
 * <ul>
 * <li>null：默认值，不启用 check 选择模式</li>
 * <li>default：check 模式的默认选择方式，即单击选中或取消选中，只影响当前点击中的 data 对象</li>
 * <li>children：该 check 模式将同时影响点击中的 data 对象，以及其孩子对象</li>
 * <li>descendant：该 check 模式将同时影响点击中的 data 对象，以及其所有子孙对象</li>
 * <li>all：该 check 模式将同时影响点击中的 data 对象，以及其所有父辈和子孙对象</li>
 * </ul>
 * @return {String}
 */
```

```
getCheckMode = function (){};

/**
 * 获取 toggle 的关闭图标
 * @return {String}
 */
getCollapseIcon = function (){};

/**
 * 获取鼠标下的列
 * @param {Event} e 鼠标或 Touch 事件
 * @return {ht.Column}
 */
getColumnAt = function (e){};

/**
 * 获取列线颜色
 * @return {color}
 */
getColumnLineColor = function (){};

/**
 * 获取列模型， 列模型用于存储 Column 列对象信息
 * @return {ht.DataModel}
 */
getColumnModel = function (){};

/**
 * 默认返回 sortFunc 函数，当 sortColumn 不为空时将返回其对应的排序函数
 * @return {Function}
 */
getCurrentSortFunc = function (){};

/**
 * 传入逻辑坐标点或者交互 event 事件参数，返回当前点下的数据元素
 * @param {Object|Event} pointOrEvent 逻辑坐标点或交互事件对象(如鼠标事件对象)
 * @return {ht.Data} 点下的数据元素
 */
getDataAt = function (pointOrEvent){};

/**
 * 获取绑定的数据模型
 * @return {ht.DataModel} 数据模型
 */
```

```
getDataModel = function (){};

/**
 * 获取当前可见区域的结束行索引
 * @return {Number}
 */
getEndRowIndex = function (){};

/**
 * 获取 toggle 的展开图标
 * @return {String}
 */
getExpandIcon = function (){};


/**
 * 在 checkMode 模式下数据除了被选中有 check 状态外，还可以有被点击行的 focus 状态<br>
 * 此方法获取 focus 数据元素
 * @return {ht.Data}
 */
getFocusData = function (){};

/**
 * 获取布局高度
 * @return {Number}
 */
getHeight = function (){};

/**
 * 获取 data 对象对应的 icon 图标，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
getIcon = function (data){};

/**
 * 返回 data 对象对应的图标宽度，默认如果有图标则以 indent 值为宽度，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {Number}
 */
getIconWidth = function (data){};

/**
 * 获取 indent 缩进，该值一般当作图标的宽度
```

```
 * @param {ht.Data} data 数据元素
 * @return {Number}
 */
getIndent = function (data){};

/**
 * 获取 data 对象显示在 treeColumn 中的文字，默认返回 data.toLabel()，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
getLabel = function (data){};


/**
 * 获取对应的单元格文本颜色，可重载自定义
 * @param {ht.Data} data 数据元素
 * @param {ht.Column} column 列
 * @param {Object} value 值
 * @return {color}
 */
getLabelColor = function (data, column, value){};

/**
 * 获取对应的单元格文本字体，可重载自定义
 * @param {ht.Data} data 数据元素
 * @param {ht.Column} column 列
 * @param {Object} value 值
 * @return {String}
 */
getLabelFont = function (data, column, value){};

/**
 * 获取文本选中颜色
 * @return {color}
 */
getLableSelectColor = function (){};

/**
 * 获取当前 data 的缩减层次，一般结合 indent 参数用于绘制
 * @param {ht.Data} data 数据元素
 * @return {Number}
 */
getLevel = function (data){};
```

```
/**
 * 获取延迟加载器
 * @return {Object}
 */
getLoader = function (){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.TreeTableView#lp lp}
 */
getLogicalPoint = function (event){};

/**
 * 获取根节点，默认为空，从 DataModel#getRoots()的对象开始展示
 * @return {ht.Data}
 */
getRootData = function (){};

/**
 * 获取当前显示的 Data 对象集合，该集合已被排序和过滤
 * @return {ht.List}
 */
getRowDatas = function (){};

/**
 * 获取行高
 * @return {Number}
 */
getRowHeight = function (){};

/**
 * 获取 data 对象所在的行索引
 * @param {ht.Data} data 数据元素
 * @return {Number}
 */
getRowIndex = function (data){};

/**
 * 获取行线颜色
 * @return {color}
 */
```

```
getRowLineColor = function (){};

/**
 * 返回当前可见行总行数
 * @return {Number}
 */
getRowSize = function (){};

/**
 * 获取滚动条颜色
 * @return {color}
 */
getScrollBarColor = function (){};

/**
 * 获取滚动条宽度
 * @return {Number}
 */
getScrollBarSize = function (){};

/**
 * 获取选择过滤器函数
 * @return {Function}
 */
getSelectableFunc = function (){};

/**
 * 获取行选中背景颜色
 * @return {color}
 */
getSelectBackground = function (){};

/**
 * 获取选中模型
 * @return {ht.SelectionModel}
 * @see {@link ht.widget.TreeTableView#sm sm}
 */
getSelectionModel = function (){};

/**
 * 获取当前排序列
 * @return {ht.Column}
 */
```

```
getSortColumn = function (){};

/**
 * 获取排序函数
 * @return {Function}
 */
getSortFunc = function (){};

/**
 * 获取排序方式
 * <ul>
 * <li>none:不允许排序</li>
 * <li>bistate:点击表头时正序和倒序两种方式之间切换</li>
 * <li>tristate:点击表头时正序、倒序、不排序三种方式之间切换</li>
 * </ul>
 * @return {String}
 */
getSortMode = function (){};

/**
 * 获取当前可见区域的起始行索引
 * @return {Number}
 */
getStartRowIndex = function (){};

/**
 * 返回当前 data 对象对应的展开或合并图标，可重载自定义
 * @param {ht.Data} data 数据元素
 * @return {String}
 */
getToggleIcon = function (data){};

/**
 * 获取 ToolTip 文字，可重载返回自定义的 toolTip 文字
 * @param {Event} e 鼠标或 Touch 事件对象
 * @return {String} toolTip 文字，默认取出鼠标下的 data 和 column，然后返回
column.getToolTip(data);
 */
getToolTip = function (e){};

/**
 * 获取水平平移值
 * @return {Number} 水平平移值
 * @see {@link ht.widget.TreeTableView#tx tx}
```

```
 */
getTranslateX = function (){};

/**
 * 获取垂直平移值
 * @return {Number} 垂直平移值
 * @see {@link ht.widget.TreeTableView#ty ty}
 */
getTranslateY = function (){};

/**
 * 返回树表组件默认插入的树表列，该列显示父子关系的树层次结构
 * @return {ht.Column}
 */
getTreeColumn = function (){};

/**
 * 获取单元格中要显示的内容
 * @param {ht.Data} data 数据元素
 * @param {ht.Column} column 列
 * @return {Object}
 */
getValue = function (data, column){};

/**
 * 获取组件的根层 div
 * @return {HTMLDivElement}
 */
getView = function (){};

/**
 * 获取组件中可见区域的逻辑尺寸
 * @return {Object}
 */
getViewRect = function (){};

/**
 * 获取可见过滤器函数
 * @return {Function}
 */
getVisibleFunc = function (){};

/**
 * 获取布局宽度
```

```
 * @return {Number}
 */
getWidth = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TreeTableView#iv iv}
 */
invalidate = function (delay){};

/**
 * 无效数据元素
 * @param {ht.Data} data 要无效的数据元素
 */
invalidateData = function (data){};

/**
 * 无效模型，最彻底的刷新方式
 * @see {@link ht.widget.TreeTableView#ivm ivm}
 */
invalidateModel = function (){};

/**
 * 是否自动隐藏滚动条
 * @return {Boolean}
 */
isAutoHideScrollBar = function (){};

/**
 * 选中数据元素时，是否自动平移组件以确保该元素出现在可见区域内
 * @return {Boolean}
 */
isAutoMakeVisible = function (){};

/**
 * 是否启用批量编辑
 * @return {Boolean}
 */
isBatchEditable = function (){};

/**
 * 判断单元格是否可编辑
 * @param {ht.Data} data 数据元素
```

```
 * @param {ht.Column} column 列
 * @return {Boolean}
 */
isCellEditable = function (data, column){};

/**
 * 是否是 check 模式，默认为 false，为 true 时自动插入 checkColumn 列
 * @return {Boolean}
 */
isCheckMode = function (){};

/**
 * 判断是否允许对 parent 对象的孩子排序，默认返回 true，可重载屏蔽孩子排序
 * @param {ht.Data} parent 父元素
 * @return {Boolean}
 */
isChildrenSortable = function (parent){};

/**
 * 获取列线是否可见，默认为 true
 * @return {Boolean}
 */
isColumnLineVisible = function (){};

/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
isDisabled = function (){};

/**
 * 返回可否编辑的总开关，默认为 false，每个 Column 列对象可再控制
 * @return {Boolean}
 */
isEditable = function (){};

/**
 * 判断 data 对象是否展开
 * @param {ht.Data} data 数据元素
 * @return {Boolean}
 */
isExpanded = function (data){};

/**
```

```
 * 判断 rootData 节点是否可见
 * @return {Boolean}
 */
isRootVisible = function (){};

/**
 * 获取行线是否可见，默认为 true
 * @return {Boolean}
 */
isRowLineVisible = function (){};

/**
 * 判断 data 对象是否可被选中
 * @param {ht.Data} data 数据元素
 * @return {Boolean}
 */
isSelectable = function (data){};

/**
 * 判断 data 对象是否被选中
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isSelected = function (data){};

/**
 * 根据 id 判断 data 对象是否被选中
 * @param {String|Number} id 数据元素 id
 * @return {Boolean}
 */
isSelectedById = function (id){};

/**
 * 当前组件是否共享选中模型
 * @return {Boolean}
 */
isSelectionModelShared = function (){};

/**
 * 判断数据元素是否可见
 * @param {ht.Data} data 数据元素
 * @return {Boolean}
 */
```

```
isVisible = function (data){};

/**
 * 无效组件，并调用延时刷新，{@link ht.widget.TreeTableView#invalidate invalidate}的缩写
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TreeTableView#invalidate invalidate}
 */
iv = function (delay){};

/**
 * 无效模型，重新构造内部的 rows 数据，最彻底的刷新方式，{@link
ht.widget.TreeTableView#invalidateModel invalidateModel}的缩写
 * @see {@link ht.widget.TreeTableView#invalidateModel invalidateModel}
 */
ivm = function (){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标，{@link
ht.widget.TreeTableView#getLogicalPoint getLogicalPoint}的缩写
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.TreeTableView#getLogicalPoint getLogicalPoint}
 */
lp = function (event){};

/**
 * 平移组件以确保数据元素在可见区域内
 * @param {ht.Data} data 数据元素
 */
makeVisible = function (data){};

/**
 * 增加自身属性变化事件监听器，{@link ht.widget.TreeTableView#addPropertyChangeListener
addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.TreeTableView#addPropertyChangeListener
addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};


/**
```

```
 * 合并 data 对象时调用，可重载做后续处理
 * @param {ht.Data} data 数据元素
 */
onCollapsed = function (data){};

/**
 * 列头被点击时调用，可重载做后续处理，如远程排序功能
 * @param {ht.Column} column 列
 */
onColumnClicked = function (column){};

/**
 * 当 data 所在行被单击时回调，可重载对单击事件做响应
 * @param {ht.Data} data 数据元素
 * @param {Event} e 事件对象
 */
onDataClicked = function (data, e){};

/**
 * 当 data 所在行被双击时回调，可重载对双击事件做响应
 * @param {ht.Data} data 数据元素
 * @param {Event} e 事件对象
 */
onDataDoubleClicked = function (data, e){};

/**
 * 展开 data 对象时调用，可重载做后续处理
 * @param {ht.Data} data 数据元素
 */
onExpanded = function (data){};

/**
 * 平移动画结束时回调，可重载做后续处理
 */
onTranslateEnded = function (){};

/**
 * 重绘组件中所有行，仅次于 invalidateModel 的彻底刷新方式
 */
redraw = function (){};

/**
 * 删除底层 Painter
 * @param {Object} painter Painter 类
```

```
 */
removeBottomPainter = function (painter){};

/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除所有选中的图元
 */
removeSelection = function (){};

/**
 * 删除顶层 Painter
 * @param {Object} painter Painter 类
 */
removeTopPainter = function (painter){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 平移(滚动)组件至指定的行号
 * @param {Number} index 行号
 */
scrollToIndex = function (index){};

/**
 * 选中所有数据元素
 */
selectAll = function (){};

/**
 * 设置是否自动隐藏滚动条
 * @param {Boolean} v
 */
```

```
setAutoHideScrollBar = function (v){};

/**
 * 设置当选中数据元素，是否自动平移(滚动)组件以确保该数据元素出现在可见区域内
 * @param {Boolean} v
 */
setAutoMakeVisible = function (v){};

/**
 * 设置是否启用批量编辑
 * @param {Boolean} v
 */
setBatchEditable = function (v){};

/**
 * 设置 check 模式
 * @param {String} v
 * <ul>
 * <li>null：默认值，不启用 check 选择模式</li>
 * <li>default：check 模式的默认选择方式，即单击选中或取消选中，只影响当前点击中的 data 对象</li>
 * <li>children：该 check 模式将同时影响点击中的 data 对象，以及其孩子对象</li>
 * <li>descendant：该 check 模式将同时影响点击中的 data 对象，以及其所有子孙对象</li>
 * <li>all：该 check 模式将同时影响点击中的 data 对象，以及其所有父辈和子孙对象</li>
 * </ul>
 */
setCheckMode = function (v){};

/**
 * 设置 toggle 的关闭图标
 * @param {String} v icon
 */
setCollapseIcon = function (v){};

/**
 * 设置列线颜色
 * @param {color} color
 */
setColumnLineColor = function (color){};

/**
 * 设置列线是否可见
 * @param {Boolean} v
 */
```

```
setColumnLineVisible = function (v){};
/**
 * 以 json 的方式配置表格中的列(设置)
 * @param {Array} columns json 列
 * @example //示例：
 * treeTableView.setColumns([{
 *  name: 'id',
 *  displayName: '序号'
 * },
 * {
 *  name: 'background',
 *  accessType: 'style'
 * }
 * ]);
 */
setColumns = function (v){};

/**
 * 设置绑定的数据模型
 * @param {ht.DataModel} dataModel 数据模型
 */
setDataModel = function (dataModel){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
 * 设置可否编辑的总开关，默认为 false，每个 Column 列对象可再控制
 * @param {Boolean} editable
 */
setEditable = function (editable){};

/**
 * 设置 toggle 的展开图标
 * @param {String} v icon
 */
setExpandIcon = function (v){};

/**
 * 在 checkMode 模式下数据除了被选中有 check 状态外，还可以有被点击行的 focus 状态<br>
```

```
 * 此方法设置 focus 的数据元素
 * @param {ht.Data} data 数据元素
 */
setFocusData = function (data){};

/**
 * 在 checkMode 模式下数据除了被选中有 check 状态外，还可以有被点击行的 focus 状态<br>
 * 此方法设置 focus 的数据元素
 * @param {String|Number} id 数据元素的 id
 */
setFocusDataById = function (id){};

/**
 * 设置布局高度
 * @param {Number} v 高度值
 */
setHeight = function (v){};

/**
 * 设置 indent 缩进，该值一般当作图标的宽度
 * @param {Number} v
 */
setIndent = function (v){};

/**
 * 设置文本颜色
 * @param {color} color
 */
setLabelColor = function (color){};

/**
 * 设置文本字体
 * @param {String} font
 */
setLabelFont = function (font){};

/**
 * 设置选中文本颜色
 * @param {color} color
 */
setLabelSelectColor = function (color){};

/**
 * 设置延迟加载器
```

```
 * @param {Object} v
 * @example //示例:
 * treeTableView.setLoader({
 *  load: function(data) {
 *      //展开此 data 时回调，可用于加载子节点
 *  },
 *  isLoaded: function(data) {
 *      //返回此 data 的子结点是否已加载
 *  }
 * });
 *
 */
setLoader = function (v){};

/**
 * 指定根节点，默认为空，从 DataModel#getRoots()的对象开始展示
 * @param {ht.Data} v
 */
setRootData = function (v){};

/**
 * 设置根节点是否可见
 * @param {Boolean} v
 */
setRootVisible = function (v){};

/**
 * 设置行高
 * @param {Number} v
 */
setRowHeight = function (v){};

/**
 * 设置行线颜色
 * @param {color} color
 */
setRowLineColor = function (color){};

/**
 * 设置行线是否可见
 * @param {Boolean} v
 */
setRowLineVisible = function (v){};
```

```
/**
 * 设置滚动条颜色
 * @param {color} color 颜色值
 */
setScrollBarColor = function (color){};

/**
 * 设置滚动条宽度
 * @param {Number} size 宽度值
 */
setScrollBarSize = function (size){};

/**
 * 设置选择过滤器函数
 * @param {Function} func 过滤器函数
 */
setSelectableFunc = function (func){};

/**
 * 设置行选中背景颜色
 * @param {color} color
 */
setSelectBackground = function (color){};

/**
 * 设置组件是否共享选中模型
 * @param {Boolean} v
 */
setSelectionModelShared = function (v){};

/**
 * 设置排序列
 * @param {ht.Column} column
 */
setSortColumn = function (column){};

/**
 * 设置排序函数
 * @param {Function} func
 */
setSortFunc = function (func){};

/**
 * 设置排序方式
```

```
 * @param {String} mode
 * <ul>
 * <li>none:不允许排序</li>
 * <li>bistate:点击表头时正序和倒序两种方式之间切换</li>
 * <li>tristate:点击表头时正序、倒序、不排序三种方式之间切换</li>
 * </ul>
 */
setSortMode = function (mode){};

/**
 * 设置水平和垂直平移值
 * @param {Number} x 水平平移值
 * @param {Number} y 垂直平移值
 * @param {Boolean} [anim] 是否使用动画
 */
setTranslate = function (x, y, anim){};

/**
 * 设置水平平移值
 * @param {Number} x 水平平移值
 */
setTranslateX = function (x){};

/**
 * 设置垂直平移值
 * @param {Number} y 垂直平移值
 */
setTranslateY = function (y){};

/**
 * 设置可见过滤器
 * @param {Function} func 过滤器函数
 */
setVisibleFunc = function (func){};

/**
 * 设置布局宽度
 * @param {Number} v 宽度值
 */
setWidth = function (v){};

/**
 * 显示水平滚动条
 */
```

```
showHBar = function (){};

/**
 * 显示垂直滚动条
 */
showVBar = function (){};

/**
 * 获取选中模型，{@link ht.widget.TreeTableView#getSelectionModel getSelectionModel}的缩写
 * @see {@link ht.widget.TreeTableView#getSelectionModel getSelectionModel}
 * @return {ht.SelectionModel}
 */
sm = function (){};


/**
 * 展开或合并 data 对象
 * @param {ht.Data} data 数据元素
 */
toggle = function (data){};


/**
 * 在当前值基础上增加水平和垂直平移值
 * @param {Number} x 新增的水平平移值
 * @param {Number} y 新增的垂直平移值
 * @param {Boolean} [anim] 是否使用动画
 */
translate = function (x, y, anim){};

/**
 * 获取或设置水平平移值，没有参数时相当于{@link ht.widget.TreeTableView#getTranslateX getTranslateX}，有参数时相当于{@link ht.widget.TreeTableView#setTranslateX setTranslateX}
 * @param {Number} value 平移值
 */
tx = function (value){};

/**
 * 获取或设置垂直平移值，没有参数时相当于{@link ht.widget.TreeTableView#getTranslateY getTranslateY}，有参数时相当于{@link ht.widget.TreeTableView#setTranslateY setTranslateY}
 * @param {Number} value 平移值
 */
```

```
ty = function (value){};

/**
 * 删除自身属性变化事件监听器，{@link
ht.widget.TreeTableView#removePropertyChangeListener removePropertyChangeListener}的缩
写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.widget.TreeTableView#removePropertyChangeListener
removePropertyChangeListener}
 */
ump = function (listener, scope){};

/**
 * 立刻刷新组件
 */
validate = function (){};
```

ht.widget.TreeTablePane

```
/**
 * 树表面板，组合了 TableHeader 和 TreeTableView 两个子组件
 * @param {ht.widget.TreeTableView} treeTableView 绑定的树表组件
 * @constructor
 */
ht.widget.TreeTablePane = function(treeTableView) {};

/**
 * 以 json 的方式配置树表中的列(新增)，内部调用 treeTableView 的 addColumns 方法
 * @param {Array} columns json 列
 * @example //示例:
 * treeTablePane.addColumns([{
 *   name: 'id',
 *   displayName: '序号'
 * },
 * {
 *   name: 'background',
 *   accessType: 'style'
 * }
 * ]);
 */
addColumns = function(columns) {}

/**
```

```
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 获取列模型， 列模型用于存储 Column 列对象信息，内部调用 treeTableView 的 getColumnModel 方法
 * @return {ht.DataModel}
 */
getColumnModel = function (){};

/**
 * 获取绑定的数据模型，内部调用 treeTableView 的 getDataModel 方法
 * @return {ht.DataModel} 数据模型
 */
getDataModel = function (){};

/**
 * 获取布局高度
 * @return {Number}
 */
getHeight = function (){};

/**
 * 获取表头组件
 * @return {ht.widget.TableHeader}
 */
getTableHeader = function (){};

/**
 * 获取树表组件
 * @return {ht.widget.TreeTableView}
 */
getTableView = function (){};

/**
 * 获取组件的根层 div
 * @return {HTMLDivElement}
 */
getView = function (){};
```

```
/**
 * 获取布局宽度
 * @return {Number}
 */
getWidth = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TreeTablePane#iv iv}
 */
invalidate = function (delay){};

/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
isDisabled = function (){};

/**
 * 无效组件，并调用延时刷新，{@link ht.widget.TreeTablePane#invalidate invalidate}的缩写
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.TreeTablePane#invalidate invalidate}
 */
iv = function (delay){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 以 json 的方式配置树表中的列(设置)，内部调用 treeTableView 的 setColumns 方法
 * @param {Array} columns json 列
 * @example //示例：
 * treeTablePane.setColumns([{
 *   name: 'id',
 *   displayName: '序号'
 * },
 * {
 *   name: 'background',
 *   accessType: 'style'
```

```
 * }
 * ]);
 */
setColumns = function (v){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
 * 设置布局高度
 * @param {Number} v 高度值
 */
setHeight = function (v){};

/**
 * 设置布局宽度
 * @param {Number} v 宽度值
 */
setWidth = function (v){};

/**
 * 立刻刷新组件
 */
validate = function (){};
```

ht.widget.PropertyView

```
/**
 * 属性组件，用于显示 Data 类型对象属性，以分组、排序等方式展示属性
 * @param {ht.DataModel} dataModel 绑定的数据模型
 * @constructor
 */
ht.widget.PropertyView = function(dataModel) {};

/**
 * 增加底层 Painter<br>
 * 组件上提供 Painter 接口，开发者可以使用 Canvas 的画笔对象自由绘制任意形状，底层 Painter 绘制在组件最下面
 * @param {Function} painter Painter 类
```

```
 * @example //Painter 示例：
 * function MyBottomPainter() {
 * }
 * ht.Default.def(MyBottomPainter, Object, {
 *   draw: function(g) {
 *       g.save();
 *       //draw...
 *       g.restore();
 *   }
 * });
 * propertyView.addBottomPainter(MyBottomPainter);
 */
addBottomPainter = function(painter) {}

/**
 * 以 json 的方式配置属性(新增)
 * @param {Array} properties json 属性
 * @example //示例：
 * propertyView.addProperties([{
 *   name: 'id',
 *   displayName: '序号'
 * },
 * {
 *   name: 'background',
 *   accessType: 'style'
 * }
 * ]);
 */
addProperties = function(properties) {}


/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.PropertyView#mp mp}
 */
addPropertyChangeListener = function (listener, scope, ahead){};


/**
 * 增加顶层 Painter<br>
 * 组件上提供 Painter 接口，开发者可以使用 Canvas 的画笔对象自由绘制任意形状，顶层 Painter
```

```
绘制在组件最上面
 * @param {Function} painter Painter类
 * @example //Painter示例:
 * function MyTopPainter() {
 * }
 * ht.Default.def(MyTopPainter, Object, {
 *  draw: function(g) {
 *      g.save();
 *      //draw...
 *      g.restore();
 *  }
 * });
 * propertyView.addTopPainter(MyTopPainter);
 */
addTopPainter = function (painter){};


/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 传入即将设置的垂直平移值，返回最终设置值，可重载限制垂直平移范围
 * @param {Number} value 原始垂直平移值
 * @return {Number} 新的垂直平移值
 */
adjustTranslateY = function (value){};

/**
 * 合并分组
 * @param  {String} categoryName 分组名
 */
collapse = function (categoryName){};

/**
 * 合并所有分组
 */
collapseAll = function (){};

/**
```

```
 * 关闭 ToolTip 功能
 */
disableToolTip = function (){};

/**
 * 获取或设置数据模型，没有参数时相当于{@link ht.widget.PropertyView#getDataModel getDataModel}，有参数时相当于{@link ht.widget.PropertyView#setDataModel setDataModel}
 * @param {ht.DataModel} [dataModel] 数据模型
 * @return {ht.DataModel} dataModel
 */
dm = function (dataModel){};

/**
 * 绘制分组名，可重载自定义
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {String} name 分组名
 * @param {Number} rowIndex 行索引
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} w 绘制的宽度
 * @param {Number} h 绘制的高度
 */
drawCategoryName = function (g, name, rowIndex, x, y, w, h){};

/**
 * 绘制分组名，可重载自定义
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Property} property 属性对象
 * @param {Number} rowIndex 行索引
 * @param {Number} x 左上角 x 坐标
 * @param {Number} y 左上角 y 坐标
 * @param {Number} w 绘制的宽度
 * @param {Number} h 绘制的高度
 */
drawPropertyName = function (g, property, rowIndex, x, y, w, h){};

/**
 * 绘制属性值，可重载自定义，如果返回值为 HTML 元素，则使用 HTML 元素当作 Renderer，一般重载 Property 的 drawPropertyValue，无需重载此方法
 * @param {CanvasRenderingContext2D} g 画笔对象
 * @param {ht.Property} property 属性对象
 * @param {Object} value 值
 * @param {Number} rowIndex 行索引
 * @param {Number} x 左上角 x 坐标
```

```
 * @param {Number} y 左上角 y 坐标
 * @param {Number} w 绘制的宽度
 * @param {Number} w 绘制的宽度
 * @param {ht.Data} data 数据元素
 * @return {HTMLElement}
 */
drawPropertyValue = function (g, property, value, rowIndex, x, y, w, h, data){};

/**
 * 启用 ToolTip
 */
enableToolTip = function (){};


/**
 * 展开分组
 * @param {String} categoryName 分组名
 */
expand = function (categoryName){};

/**
 * 展开所有分组
 */
expandAll = function (){};

/**
 * 获取边框和分组行的背景颜色
 * @return {color}
 */
getBackground = function (){};

/**
 * 返回分组文本颜色，可重载自定义
 * @param {String} categoryName 分组名
 * @return {color}
 */
getCategoryColor = function (categoryName){};

/**
 * 返回分组文本字体，可重载自定义
 * @param {String} categoryName 分组名
 * @return {String}
 */
```

```
getCategoryFont = function (categoryName){};

/**
 * 获取分组合并图标
 * @return {String}
 */
getCollapseIcon = function (){};

/**
 * 获取列线颜色
 * @return {color}
 */
getColumnLineColor = function (){};

/**
 * 获取列线位置比例，默认值 0.5，允许范围为 0~1
 * @return {Number}
 */
getColumnPosition = function (){};

/**
 * 获取当前显示对象
 * @return {ht.Data}
 */
getCurrentData = function (){};

/**
 * 获取绑定的数据模型
 * @return {ht.DataModel} 数据模型
 */
getDataModel = function (){};

/**
 * 获取分组展开图标
 * @return {String}
 */
getExpandIcon = function (){};

/**
 * 获取布局高度
 * @return {Number}
 */
getHeight = function (){};
```

```
/**
 * 获取左侧缩进，左侧空间用于绘制分组 toggle 图标，以及属性提示 icon 图标
 * @return {Number}
 */
getIndent = function (data){};


/**
 * 返回属性值文本颜色，可重载自定义
 * @param {ht.Property} property 属性对象
 * @param {Object} value 值
 * @param {Number} rowIndex 行索引
 * @return {color}
 */
getLabelColor = function (property, value, rowIndex){};

/**
 * 返回属性值文本字体，可重载自定义
 * @param {ht.Property} property 属性对象
 * @param {Object} value 值
 * @param {Number} rowIndex 行索引
 * @return {String}
 */
getLabelFont = function (property, value, rowIndex){};

/**
 * 获取文本选中颜色
 * @return {color}
 */
getLableSelectColor = function (){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标
 * @param {Event} event 事件对象
 * @return {Object}
 * @see {@link ht.widget.PropertyView#lp lp}
 */
getLogicalPoint = function (event){};

/**
 * 返回 event 事件所在的行的属性信息
 * @param {Event} event 事件对象
 * @return {ht.Property}
```

```
 */
getPropertyAt = function (event){};

/**
 * 返回属性名文本颜色，可重载自定义
 * @param {ht.Property} property 属性对象
 * @param {Number} rowIndex 行索引
 * @return {color}
 */
getPropertyColor = function (property, rowIndex){};

/**
 * 返回属性名文本字体，可重载自定义
 * @param {ht.Property} property 属性对象
 * @param {Number} rowIndex 行索引
 * @return {color}
 */
getPropertyFont = function (property, rowIndex){};

/**
 * 获取 propertyModel 属性，可增删 Property 属性对象
 * @return {ht.DataModel}
 */
getPropertyModel = function (){};

/**
 * 返回显示在左列的属性名，可重载自定义
 * @return {String}
 */
getPropertyName = function (property){};

/**
 * 返回要显示的原始未过滤排序的属性集合，默认返回 propertyModel.getRoots()，可重载自定义
 * @return {ht.List}
 */
getRawProperties = function (){};

/**
 * 获取行高
 * @return {Number}
 */
getRowHeight = function (){};

/**
```

```
 * 获取 event 事件所在的行索引
 * @param {Event} event 事件对象
 * @return {Number}
 */
getRowIndexAt = function (event){};

/**
 * 返回当前显示的所有行信息的集合，集合元素为 string 类型代表分组名，{data:d,property:p}
结构对象代表属性信息
 * @return {ht.List}
 */
getRows = function (){};

/**
 * 获取滚动条颜色
 * @return {color}
 */
getScrollBarColor = function (){};

/**
 * 获取滚动条宽度
 * @return {Number}
 */
getScrollBarSize = function (){};

/**
 * 获取行选中背景颜色
 * @return {color}
 */
getSelectBackground = function (){};

/**
 * 获取选中模型
 * @return {ht.SelectionModel}
 * @see {@link ht.widget.PropertyView#sm sm}
 */
getSelectionModel = function (){};

/**
 * 获取当前选中行索引
 * @return {Number}
 */
getSelectRowIndex = function (){};
```

```
/**
 * 获取排序函数
 * @return {Function}
 */
getSortFunc = function (){};

/**
 * 获取垂直平移值
 * @return {Number} 垂直平移值
 * @see {@link ht.widget.PropertyView#ty ty}
 */
getTranslateY = function (){};

/**
 * 获取组件的根层 div
 * @return {HTMLDivElement}
 */
getView = function (){};

/**
 * 获取可见过滤器函数
 * @return {Function}
 */
getVisibleFunc = function (){};

/**
 * 获取布局宽度
 * @return {Number}
 */
getWidth = function (){};

/**
 * 无效组件，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.PropertyView#iv iv}
 */
invalidate = function (delay){};

/**
 * 无效模型，最彻底的刷新方式
 * @see {@link ht.widget.PropertyView#ivm ivm}
 */
invalidateModel = function (){};
```

```
/**
 * 是否自动隐藏滚动条
 * @return {Boolean}
 */
isAutoHideScrollBar = function (){};

/**
 * 是否启用批量编辑
 * @return {Boolean}
 */
isBatchEditable = function (){};

/**
 * 是否分组显示，默认为 true，为 false 则忽略 Property 的 categoryName 属性
 * @return {Boolean}
 */
isCategorizable = function (){};

/**
 * 获取列线是否可见，默认为 true
 * @return {Boolean}
 */
isColumnLineVisible = function (){};

/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
isDisabled = function (){};

/**
 * 返回可否编辑的总开关，默认为 false，每个 Property 属性对象可再控制
 * @return {Boolean}
 */
isEditable = function (){};

/**
 * 分组是否展开
 * @param {String} categoryName 分组名
 * @return {Boolean}
 */
isExpanded = function (categoryName){};

/**
```

```
 * 判断属性是否可编辑，可重载自定义
 * @param {ht.Property} property 属性对象
 * @return {Boolean}
 */
isPropertyEditable = function (property){};

/**
 * 获取行线是否可见，默认为 true
 * @return {Boolean}
 */
isRowLineVisible = function (){};

/**
 * 当前组件是否共享选中模型
 * @return {Boolean}
 */
isSelectionModelShared = function (){};

/**
 * 判断属性是否可见
 * @param {ht.Property} property 属性对象
 * @return {Boolean}
 */
isVisible = function (property){};

/**
 * 无效组件，并调用延时刷新，{@link ht.widget.PropertyView#invalidate invalidate}的缩写
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.widget.PropertyView#invalidate invalidate}
 */
iv = function (delay){};

/**
 * 无效模型，重新构造内部的 rows 数据，最彻底的刷新方式，{@link
ht.widget.PropertyView#invalidateModel invalidateModel}的缩写
 * @see {@link ht.widget.PropertyView#invalidateModel invalidateModel}
 */
ivm = function (){};

/**
 * 传入 HTML 事件对象，将事件坐标转换为组件中的逻辑坐标，{@link
ht.widget.PropertyView#getLogicalPoint getLogicalPoint}的缩写
 * @param {Event} event 事件对象
 * @return {Object}
```

```
 * @see {@link ht.widget.PropertyView#getLogicalPoint getLogicalPoint}
 */
lp = function (event){};

/**
 * 增加自身属性变化事件监听器，{@link ht.widget.PropertyView#addPropertyChangeListener addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.PropertyView#addPropertyChangeListener addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};

/**
 * 合并分组时调用，可重载做后续处理
 * @param {String} categoryName 分组名
 */
onCollapsed = function (categoryName){};

/**
 * 展开分组时调用，可重载做后续处理
 * @param {String} categoryName 分组名
 */
onExpanded = function (categoryName){};

/**
 * 平移动画结束时回调，可重载做后续处理
 */
onTranslateEnded = function (){};

/**
 * 重绘组件中所有行，仅次于 invalidateModel 的彻底刷新方式
 */
redraw = function (){};

/**
 * 删除底层 Painter
 * @param {Object} painter Painter 类
 */
removeBottomPainter = function (painter){};

/**
```

```
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除所有选中的图元
 */
removeSelection = function (){};

/**
 * 删除顶层 Painter
 * @param {Object} painter Painter 类
 */
removeTopPainter = function (painter){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};
/**
 * 设置是否自动隐藏滚动条
 * @param {Boolean} v
 */
setAutoHideScrollBar = function (v){};
/**
 * 设置边框和分组行的背景颜色
 * @param {color} color
 */
setBackground = function (color){};

/**
 * 设置是否启用批量编辑
 * @param {Boolean} v
 */
setBatchEditable = function (v){};


/**
 * 设置是否分组显示，默认为 true，为 false 则忽略 Property 的 categoryName 属性
 * @param {Boolean} v
```

```
 */
setCategorizable = function (v){};
/**
 * 设置分组合并图标
 * @param {String} icon
 */
setCollapseIcon = function (icon){};

/**
 * 设置列线颜色
 * @param {color} color
 */
setColumnLineColor = function (color){};

/**
 * 设置列线是否可见
 * @param {Boolean} v
 */
setColumnLineVisible = function (v){};


/**
 * 设置列线位置比例，默认值 0.5，允许范围为 0~1
 * @param {Number} v
 */
setColumnPosition = function (v){};

/**
 * 设置绑定的数据模型
 * @param {ht.DataModel} dataModel 数据模型
 */
setDataModel = function (dataModel){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
 * 设置可否编辑的总开关，默认为 false，每个 Property 属性对象可再控制
 * @param {Boolean} editable
 */
```

```
setEditable = function (editable){};

/**
 * 设置分组展开图标
 * @param {String} icon
 */
setExpandIcon = function (icon){};

/**
 * 设置布局高度
 * @param {Number} v 高度值
 */
setHeight = function (v){};

/**
 * 设置左侧缩进，左侧空间用于绘制分组 toggle 图标，以及属性提示 icon 图标
 * @param {Number} v
 */
setIndent = function (v){};

/**
 * 设置属性值文本颜色
 * @param {color} color
 */
setLabelColor = function (color){};

/**
 * 设置属性值文本字体
 * @param {String} font
 */
setLabelFont = function (font){};

/**
 * 设置选中文本颜色
 * @param {color} color
 */
setLabelSelectColor = function (color){};

/**
 * 以 json 的方式配置属性(设置)
 * @param {Array} properties json 属性
 * @example //示例:
 * propertyView.setProperties([{
 *   name: 'id',
```

```
 *  displayName: '序号'
 * },
 * {
 *  name: 'background',
 *  accessType: 'style'
 * }
 * ]);
 */
setProperties = function (properties){};

/**
 * 设置行高
 * @param {Number} v
 */
setRowHeight = function (v){};

/**
 * 设置行线颜色
 * @param {color} color
 */
setRowLineColor = function (color){};

/**
 * 设置行线是否可见
 * @param {Boolean} v
 */
setRowLineVisible = function (v){};

/**
 * 设置滚动条颜色
 * @param {color} color 颜色值
 */
setScrollBarColor = function (color){};

/**
 * 设置滚动条宽度
 * @param {Number} size 宽度值
 */
setScrollBarSize = function (size){};

/**
 * 设置行选中背景颜色
 * @param {color} color
 */
```

```
setSelectBackground = function (color){};

/**
 * 设置组件是否共享选中模型
 * @param {Boolean} v
 */
setSelectionModelShared = function (v){};

/**
 * 设置当前选中行索引
 * @param {Number} v
 */
setSelectRowIndex = function (v){};

/**
 * 设置排序函数
 * @param {Function} func
 */
setSortFunc = function (func){};

/**
 * 设置水平和垂直平移值
 * @param {Number} x 水平平移值(无效)
 * @param {Number} y 垂直平移值
 * @param {Boolean} [anim] 是否使用动画
 */
setTranslate = function (x, y, anim){};


/**
 * 设置垂直平移值
 * @param {Number} y 垂直平移值
 */
setTranslateY = function (y){};

/**
 * 设置可见属性过滤器
 * @param {Function} func 过滤器函数
 */
setVisibleFunc = function (func){};

/**
 * 设置布局宽度
 * @param {Number} v 宽度值
```

```
 */
setWidth = function (v){};

/**
 * 显示垂直滚动条
 */
showVBar = function (){};

/**
 * 获取选中模型，{@link ht.widget.PropertyView#getSelectionModel getSelectionModel}的缩写
 * @see {@link ht.widget.PropertyView#getSelectionModel getSelectionModel}
 * @return {ht.SelectionModel}
 */
sm = function (){};

/**
 * 切换分组展开合并状态
 * @param {String} categoryName 分组名
 */
toggle = function (categoryName){};

/**
 * 在当前值基础上增加水平和垂直平移值
 * @param {Number} x 新增的水平平移值(无效)
 * @param {Number} y 新增的垂直平移值
 * @param {Boolean} [anim] 是否使用动画
 */
translate = function (x, y, anim){};

/**
 * 获取或设置垂直平移值，没有参数时相当于{@link ht.widget.PropertyView#getTranslateY getTranslateY}，有参数时相当于{@link ht.widget.PropertyView#setTranslateY setTranslateY}
 * @param {Number} value 平移值
 */
ty = function (value){};

/**
 * 删除自身属性变化事件监听器，{@link ht.widget.PropertyView#removePropertyChangeListener removePropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.widget.PropertyView#removePropertyChangeListener removePropertyChangeListener}
```

```
 */
ump = function (listener, scope){};

/**
 * 立刻刷新组件
 */
validate = function (){};
```

ht.graph3d.Graph3dView

```
/**
 * 3D 渲染引擎组件，可视化呈现数据模型的三维环境场景
 * @param {ht.DataModel} dataModel 绑定的数据模型
 * @constructor
 */
ht.graph3d.Graph3dView = function(dataModel) {};

/**
 * 增加交互事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.graph3d.Graph3dView#mi mi}
 * @example //示例：
 * graph3dView.addInteractorListener(function(event) {
 *       //event 格式：
 *       {
 *           kind: 'clickData',//事件类型
 *           data: data,//事件相关的数据元素
 *           part: "part",//事件的区域,icon、label 等
 *           event: e//html 原生事件
 *       }
 * });
 */
addInteractorListener = function(listener, scope, ahead) {}

/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.graph3d.Graph3dView#mp mp}
 */
```

```
addPropertyChangeListener = function (listener, scope, ahead){};

/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 关闭 ToolTip 功能
 */
disableToolTip = function (){};

/**
 * 获取或设置数据模型，没有参数时相当于{@link ht.graph3d.Graph3dView#getDataModel getDataModel}，有参数时相当于{@link ht.graph3d.Graph3dView#setDataModel setDataModel}
 * @param {ht.DataModel} [dataModel] 数据模型
 * @return {ht.DataModel} dataModel
 */
dm = function (dataModel){};

/**
 * 启用 ToolTip
 */
enableToolTip = function (){};

/**
 * 获取截头锥体的宽高比
 * @return {Number}
 */
getAspect = function (){};

/**
 * 获取 x 轴线颜色
 * @return {color}
 */
getAxisXColor = function (){};

/**
 * 获取 y 轴线颜色
 * @return {color}
 */
```

```
getAxisYColor = function (){};

/**
 * 获取 z 轴线颜色
 * @return {color}
 */
getAxisZColor = function (){};

/**
 * 获取碰撞边界
 * @return {Array}
 */
getBoundaries = function (){};

/**
 * 获取图元最终亮度，默认为 1,大于 1 变亮，小于 1 变暗
 * @param {ht.Data} data 图元
 * @return {Number}
 */
getBrightness = function (data){};

/**
 * 获取渲染的画布
 * @return {HTMLCanvasElement} 画布
 */
getCanvas = function (){};

/**
 * 获取拓扑中心点
 * @return {Array} 中心点坐标，格式：[x, y, z]
 */
getCenter = function (){};

/**
 * 获取当前子网
 * @return {ht.SubGraph} 子网对象
 */
getCurrentSubGraph = function (){};

/**
 * 传入逻辑坐标点或者交互 event 事件参数，返回当前点下的图元
 * @param {Object|Event} pointOrEvent 逻辑坐标点或交互事件对象(如鼠标事件对象)
 * @return {ht.Data} 点下的图元
 */
```

```
getDataAt = function (pointOrEvent){};

/**
 * 传入逻辑坐标点或者交互 event 事件参数，返回当前点下的图元及 part 信息
 * @param {Object|Event} pointOrEvent 逻辑坐标点或交互事件对象(如鼠标事件对象)
 * @return {Object} 图元和 part 信息
 */
getDataInfoAt = function (pointOrEvent){};

/**
 * 获取绑定的数据模型
 * @return {ht.DataModel} 数据模型
 */
getDataModel = function (){};

/**
 * 获取矩形区域内的图元
 * @param {rect} rect 逻辑坐标区域
 * @return {ht.List}
 */
getDatasInRect = function (rect){};

/**
 * 获取编辑过滤器函数
 * @return {Function}
 */
getEditableFunc = function (){};

/**
 * 获取大小编辑控制条的颜色
 * @return {color}
 */
getEditSizeColor = function (){};

/**
 * 获取眼睛（或 Camera）所在位置，默认值为[0, 300, 1000]
 * @return {Array} 眼睛位置坐标，格式[x, y, z]
 */
getEye = function (){};

/**
 * 获取远端截面位置，默认值为 10000
 * @return {Number}
 */
```

```
getFar = function (){};

/**
 * 获取垂直方向的视觉张角弧度，默认值为 Math.PI/4
 * @return {Number}
 */
getFovy = function (){};

/**
 * 获取网格线颜色
 * @return {color}
 */
getGridColor = function (){};

/**
 * 获取网格线间距
 * @return {Number}
 */
getGridGap = function (){};

/**
 * 获取网格行列数，默认为 40
 * @return {Number}
 */
getGridSize = function (){};

/**
 * 获取拓扑组件的布局高度
 * @return {Number}
 */
getHeight = function (){};

/**
 * 获取交互器
 * @return {ht.List}
 */
getInteractors = function (){};

/**
 * 获取图元的 label，用于在拓扑上显示文字信息，可重载返回自定义文字
 * @param {ht.Data} data 图元
 * @return {String} 图元 label 文字，默认返回 data.s('label')||data.getName();
 */
```

```
getLabel = function (data){};

/**
 * 获取图元的第二个 label，用于在拓扑上显示文字，可重载返回自定义文字
 * @param {ht.Data} data 图元
 * @return {String} 图元第二个 label 的文字，默认返回 data.s('label2')
 */
getLabel2 = function (data){};

/**
 * 获取图元的第二个 label 的背景色，可重载返回自定义颜色
 * @param {ht.Data} data 图元
 * @return {color} 图元第二个 label 的背景色，默认返回 data.s('label2.background')
 */
getLabel2Background = function (data){};

/**
 * 获取图元的第二个 label 的文字颜色，可重载返回自定义颜色
 * @param {ht.Data} data 图元
 * @return {color} 图元第二个 label 的文字颜色，默认返回 data.s('label2.color')
 */
getLabel2Color = function (data){};

/**
 * 获取图元 label 的背景色，可重载返回自定义颜色
 * @param {ht.Data} data 图元
 * @return {color} 图元 label 的背景色，默认返回 data.s('label.background')
 */
getLabelBackground = function (data){};

/**
 * 获取图元 label 的文字颜色，可重载返回自定义颜色
 * @param {ht.Data} data 图元
 * @return {color} 图元 label 的文字颜色，默认返回 data.s('label.color')
 */
getLabelColor = function (data){};

/**
 * 获取移动过滤器函数
 * @return {Function}
 */
getMovableFunc = function (){};

/**
```

```
 * 获取移动漫游步进
 * @return {Number}
 */
getMoveStep = function (){};

/**
 * 获取近端截面位置，默认值为 10
 * @return {Number}
 */
getNear = function (){};

/**
 * 获取图元的 note，用于在拓扑上显示标注信息，可重载返回自定义文字
 * @param {ht.Data} data 图元
 * @return {String} 图元 note 文字，默认返回 data.s('note')
 */
getNote = function (){data};

/**
 * 获取图元的第二个 note，用于在拓扑上显示标注信息，可重载返回自定义文字
 * @param {ht.Data} data 图元
 * @return {String} 图元第二个 note 文字，默认返回 data.s('note2')
 */
getNote2 = function (){data};

/**
 * 获取图元的第二个 note 的背景色，可重载返回自定义颜色
 * @param {ht.Data} data 图元
 * @return {color} 图元第二个 note 的背景色，默认返回 data.s('note2.background')
 */
getNote2Background = function (){data};

/**
 * 获取图元 note 的文字颜色，可重载返回自定义颜色
 * @param {ht.Data} data 图元
 * @return {color} 图元 note 的文字颜色，默认返回 data.s('note.background')
 */
getNoteBackground = function (data){};

/**
 * 获取图元的透明度，可重载返回自定义透明度
 * @param {ht.Data} data 图元
 * @return {Number} 图元透明度，默认返回 data.s('opacity')
 */
```

```
getOpacity = function (data){};

/**
 * 获取正交投影宽度，默认为 2000
 * @return {Number}
 */
getOrthoWidth = function (){};

/**
 * 获取框选选择框的背景色
 * @return {color}
 */
getRectSelectBackground = function (){};

/**
 * 获取旋转编辑过滤器函数
 * @return {Function}
 */
getRotationEditableFunc = function (){};

/**
 * 获取选择过滤器函数
 * @return {Function}
 */
getSelectableFunc = function (){};

/**
 * 获取选中模型
 * @return {ht.SelectionModel}
 * @see {@link ht.graph3d.Graph3dView#sm sm}
 */
getSelectionModel = function (){};

/**
 * 获取大小编辑过滤器
 * @return {Function}
 */
getSizeEditableFunc = function (){};

/**
 * 获取组件内部的贴图映射表
 * @return {Object}
 */
```

```
getTextureMap = function (){};

/**
 * 获取 ToolTip 文字，可重载返回自定义的 toolTip 文字
 * @param {Event} e 鼠标或 Touch 事件对象
 * @return {String} toolTip 文字，默认取出鼠标下的图元，然后返回其 getToolTip()
 */
getToolTip = function (e){};

/**
 * 获取摄像头正上方向，该参数一般较少改动，默认值为[0, 1, 0]
 * @return {Array} 格式：[x, y, z]
 */
getUp = function (){};

/**
 * 获取拓扑组件的根层 div
 * @return {HTMLDivElement}
 */
getView = function (){};

/**
 * 获取可见过滤器函数
 * @return {Function}
 */
getVisibleFunc = function (){};

/**
 * 获取拓扑组件的布局宽度
 * @return {Number}
 */
getWidth = function (){};

/**
 * 定义图元立体线框效果
 * @param {ht.Data} data 数据元素
 * @example //示例:
 * g3d.getWireframe = function(data){
 * var visible = data.s('wf.visible');
 *  if(visible === true || (visible === 'selected' && this.isSelected(data))){
 *      return {
 *          color: data.s('wf.color'),
 *          width: data.s('wf.width'),
 *          short: data.s('wf.short'),
```

```
 *          mat: data.s('wf.mat')
 *      };
 *  }
 * },
 * @return {Object}
 */
getWireframe = function (data){};

/**
 * 无效拓扑，并调用延时刷新
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.graph3d.Graph3dView#iv iv}
 */
invalidate = function (delay){};

/**
 * 无效拓扑中的所有图元
 */
invalidateAll = function (){};

/**
 * 无效拓扑中的图元
 * @param {ht.Data} data 要无效的图元
 */
invalidateData = function (data){};

/**
 * 无效选中模型中的图元
 */
invalidateSelection = function (){};

/**
 * 选中图元时，是否自动平移拓扑以确保该图元出现在可见区域内
 * @return {Boolean}
 */
isAutoMakeVisible = function (){};

/**
 * 是否显示中心点轴线
 * @return {Boolean}
 */
isCenterAxisVisible = function (){};
```

```
/**
 * 组件是否处于不可用状态，处于此状态时不能进行任何操作并且会遮挡一层蒙板
 * @return {Boolean}
 */
isDisabled = function (){};

/**
 * 判断图元是否可被编辑
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isEditable = function (data){};

/**
 * 是否是第一人称模式
 * @return {Boolean}
 */
isFirstPersonMode = function (){};

/**
 * 是否显示网格
 * @return {Boolean}
 */
isGridVisible = function (){};

/**
 * 判断图元是否可移动
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isMovable = function (data){};

/**
 * 是否使用鼠标漫游，默认为 true，如果改为 false，则鼠标左键右键都不支持前进后退的操作功能，
 * 但左键可拖拽编辑图元，右键可改变视角方向，采用这样的方式一般会结合键盘 w|s|a|d 按键进行漫游操作
 * @return {Boolean}
 */
isMouseRoamable = function (){};


/**
 * 是否显示坐标原点[0,0,0]轴线
```

```
 * @return {Boolean}
 */
isOriginAxisVisible = function (){};

/**
 * 是否使用正交投影
 * @return {Boolean}
 */
isOrtho = function (){};

/**
 * 是否允许平移操作
 * @return {Boolean}
 */
isPannable = function (){};

/**
 * 是否允许框选操作
 * @return {Boolean}
 */
isRectSelectable = function (){};

/**
 * 判断是否允许通过空格将拓扑复位
 * @return {Boolean}
 */
isResettable = function (){};

/**
 * 是否可旋转
 * @return {Boolean}
 */
isRotatable = function (){};

/**
 * 获取旋转步进
 * @return {Number}
 */
getRotateStep = function (){};


/**
 * 判断图元是否可编辑旋转
 * @param {ht.Data} data 图元
```

```
 * @return {Boolean}
 */
isRotationEditable = function (data){};

/**
 * 判断图元是否可被选中
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isSelectable = function (data){};

/**
 * 判断图元是否被选中
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isSelected = function (data){};

/**
 * 根据 id 判断图元是否被选中
 * @param {String|Number} id 图元 id
 * @return {Boolean}
 */
isSelectedById = function (id){};

/**
 * 当前拓扑是否共享选中模型
 * @return {Boolean}
 */
isSelectionModelShared = function (){};

/**
 * 图元是否可编辑大小
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
isSizeEditable = function (data){};

/**
 * 判断图元是否可见
 * @param {ht.Data} data 图元
 * @return {Boolean}
 */
```

```
isVisible = function (data){};

/**
 * 是否可进退
 * @return {Boolean}
 */
isWalkable = function (){};

/**
 * 是否可缩放
 * @return {Boolean}
 */
isZoomable = function (){};


/**
 * 无效拓扑，并调用延时刷新，{@link ht.graph3d.Graph3dView#invalidate invalidate}的缩写
 * @param {Number} delay 延迟刷新的间隔事件(单位:ms)
 * @see {@link ht.graph3d.Graph3dView#invalidate invalidate}
 */
iv = function (delay){};

/**
 * 平移拓扑以确保该图元在可见区域内
 * @param {ht.Data} data 图元
 */
makeVisible = function (data){};

/**
 * 增加交互事件监听器，{@link ht.graph3d.Graph3dView#addInteractorListener
addInteractorListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.graph3d.Graph3dView#addInteractorListener addInteractorListener}
 * @example //示例:
 * graph3dView.mi(function(event) {
 *       //event 格式:
 *       {
 *           kind: 'clickData',//事件类型
 *           data: data,//事件相关的数据元素
 *           part: "part",//事件的区域,icon、label 等
 *           event: e//html 原生事件
 *       }
```

```
 * });
 */
mi = function (listener, scope, ahead){};

/**
 * 移动选中模型中图元的位置
 * @param {Number} dx x 轴方向移动值
 * @param {Number} dy y 轴方向移动值
 * @param {Number} dz z 轴方向移动值
 */
moveSelection = function (dx, dy, dz){};

/**
 * 增加自身属性变化事件监听器，{@link ht.graph3d.Graph3dView#addPropertyChangeListener addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.graph3d.Graph3dView#addPropertyChangeListener addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};

/**
 * 自动布局动画结束后时回调，可重载做后续处理
 */
onAutoLayoutEnded = function (){};

/**
 * 单击拓扑背景时回调，可重载做后续处理
 * @param {Event} event 事件对象
 */
onBackgroundClicked = function (event){};

/**
 * 双击拓扑背景时回调，默认调用 upSubGraph()进入上一层子网，可重载改变默认逻辑或做后续处理
 * @param {Event} event 事件对象
 */
onBackgroundDoubleClicked = function (event){};

/**
 * 图元被点击时回调，可重载做后续处理
 * @param {ht.Data} data 被点击的图元
```

```
 * @param {Event} e 事件对象
 * @param {Object} part 区域
 */
onDataClicked = function (data, e, part){};

/**
 * 图元被双击时回调，可重载做后续处理
 * @param {ht.Data} data 双击的图元
 * @param {Event} e 事件对象
 * @param {Object} part 区域
 */
onDataDoubleClicked = function (data, e, part){};

/**
 * 连线图元被双击时回调，默认调用 edge.toggle()，可重载改变默认逻辑或做后续处理
 * @param {ht.Edge} edge 连线
 * @param {Event} e 事件对象
 * @param {Object} part 区域
 */
onEdgeDoubleClicked = function (edge, e, part){};

/**
 * 组类型图元被双击时回调，默认实现调用 group.toggle()，可重载改变默认逻辑或做后续处理
 * @param {ht.Group} group Group 对象
 * @param {Event} e 事件对象
 * @param {Object} part 区域
 */
onGroupDoubleClicked = function (group, e, part){};

/**
 * 移动图元位置结束时回调，可重载做后续处理
 */
onMoveEnded = function (){};

/**
 * 手抓图平移拓扑图结束时回调，可重载做后续处理
 */
onPanEnded = function (){};

/**
 * 触屏进行双指缩放结束时回调，可重载做后续处理
 */
onPinchEnded = function (){};
```

```
/**
 * 框选结束时回调，可重载做后续处理
 */
onRectSelectEnded = function (){};

/**
 * 旋转结束时回调，可重载做后续处理
 */
onRotateEnded = function (){};

/**
 * 选中变化时回调，默认实现会使得该选中图元出现在拓扑图上的可见范围
 * @param {Event} event 选中变化事件对象
 */
onSelectionChanged = function (event){};

/**
 * 子网图元被双击时回调，默认实现进入子网
 * @param {ht.SubGraph} subGraph 子网对象
 * @param {Event} event 事件对象
 * @param {Object} part 区域
 */
onSubGraphDoubleClicked = function (subGraph, event, part){};

/**
 * 进退操作结束时回调，可重载做后续处理
 * @param {ht.SubGraph} subGraph 子网对象
 * @param {Event} event 事件对象
 */
onWalkEnded = function (){};

/**
 * 缩放动画结束时回调
 */
onZoomEnded = function (){};

/**
 * 上下左右四个方向的平移，本质为 eye 和 center 同时做四个方向的相同偏移量，
 * dx 左右偏移参数，dy 上下偏移参数，dx 和 dy 一般代表屏幕移动像素，
 * Graph3dView 自动会换算成合理的 3D 空间逻辑坐标偏移量。<br>
 * firstPersonMode 参数为空时则默认采用 Graph3dView#isFirstPersonMode()当前值，
 * 如果为第一人称模式调用 pan 操作，该函数会考虑 Graph3dView#getBoundaries()边界限制。
 * @param  {dx} dx x 轴方向的偏移量
 * @param  {dy} dy y 轴方向的偏移量
```

```
 * @param  {Boolean} [anim] 是否使用动画
 * @param  {Boolean} [firstPersonMode] 是否第一人称模式
 */
pan = function (dx, dy, anim, firstPersonMode){};


/**
 * 重绘拓扑
 */
redraw = function (){};

/**
 * 删除交互事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.graph3d.Graph3dView#umi umi}
 */
removeInteractorListener = function (listener, scope){};

/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removePropertyChangeListener = function (listener, scope){};

/**
 * 删除所有选中的图元
 */
removeSelection = function (){};

/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 复位函数，调用该函数将 eye、center 和 up 三个变量设置为 ht.Default 上对应的
graph3dViewCenter、graph3dViewEye 和 graph3dViewUp 初始默认值。
 */
reset = function (){};
```

```
/**
 * 上下左右四个方位旋转一定角度
 * @param {Number} leftRight 水平旋转弧度
 * @param {Number} upDown 垂直旋转弧度
 * @param {Boolean} anim 是否使用动画
 * @param {Boolean} firstPersonMode 是否第一人称模式，为空时则采用
Graph3dView#isFirstPersonMode()，该参数将影响旋转移动的参照标准，为默认非第一人称模式时，
旋转是以 center 为中心进行旋转，也就是围绕中心物体旋转，当为第一人称时旋转以 eye 为中心进
行旋转，也就是旋转眼睛朝向方向。
 */
rotate = function (leftRight, upDown, anim, firstPersonMode){};

/**
 * 选中拓扑中所有图元
 */
selectAll = function (){};

/**
 * 设置截头锥体的宽高比，该参数默认自动根据屏幕的宽高比决定，一般不需要设置。
 * @param {Number} v
 */
setAspect = function (v){};

/**
 * 设置当选中图元时，是否自动平移拓扑以确保该图元出现在可见区域内
 * @param {Boolean} v
 */
setAutoMakeVisible = function (v){};

/**
 * 设置 x 轴线颜色
 * @param {color} color
 */
setAxisXColor = function (color){};

/**
 * 设置 y 轴线颜色
 * @param {color} color
 */
setAxisYColor = function (){};

/**
 * 设置 z 轴线颜色
 * @param {color} color
```

```
 */
setAxisZColor = function (){};


/**
 * 设置碰撞边界
 * @param {Array} boundaries 边界数组
 * @example //示例:
 * g3d.setBoundaries([
 *   [
 *       p0.x, p0.y,
 *       p1.x, p1.y,
 *       p2.x, p2.y,
 *       p3.x, p3.y
 *   ],
 *   [
 *       p4.x, p4.y,
 *       p5.x, p5.y,
 *       p6.x, p6.y
 *   ]
 * ]);
 */
setBoundaries = function (boundaries){};

/**
 * 设置中心点
 * @param {Array} center 中心点坐标，格式：[x, y, z]
 */
setCenter = function (center){};

/**
 * 设置是否显示中心点轴线
 * @param {Boolean} v
 */
setCenterAxisVisible = function (v){};

/**
 * 设置当前子网
 * @param {ht.SubGraph} subGraph 子网对象
 */
setCurrentSubGraph = function (subGraph){};

/**
 * 设置绑定的数据模型
```

```
 * @return {ht.DataModel} 数据模型
 */
setDataModel = function (dataModel){};

/**
 * 设置组件是否处于不可用状态，处于不可用状态时不能进行任何操作并且会遮挡一层蒙板
 * @param {Boolean} value 是否禁用组件
 * @param {String} [iconUrl] 蒙板上显示的 icon 的路径
 */
setDisabled = function (value, iconUrl){};

/**
 * 设置拓扑中的图元是否可编辑
 * @param {Boolean} editable
 */
setEditable = function (editable){};

/**
 * 设置编辑过滤器函数
 * @param {Function} func 过滤器函数
 */
setEditableFunc = function (func){};

/**
 * 设置大小编辑控制条的颜色
 * @param {color} color
 */
setEditSizeColor = function (color){};

/**
 * 设置眼睛（或 Camera）所在位置，默认值为[0, 300, 1000]
 * @param {Array} eye 眼睛位置坐标，格式[x, y, z]
 */
setEye = function (eye){};

/**
 * 设置远端截面位置，默认值为 10000
 * @param {Number} far
 */
setFar = function (far){};

/**
 * 设置第一人称模式
 * @param {Boolean} mode
```

```
 */
setFirstPersonMode = function (mode){};

/**
 * 设置垂直方向的视觉张角弧度，默认值为 Math.PI/4
 * @param {Number} fovy
 */
setFovy = function (fovy){};

/**
 * 设置网格线颜色
 * @param {color} color
 */
setGridColor = function (color){};

/**
 * 设置网格线间距
 * @param {Number} gap
 */
setGridGap = function (gap){};

/**
 * 设置网格行列数，默认为 40
 * @param {Number} size
 */
setGridSize = function (size){};

/**
 * 设置是否显示网格
 * @param {Boolean} v
 */
setGridVisible = function (v){};

/**
 * 设置布局高度
 * @param {Number} height 高度值
 */
setHeight = function (height){};
/**
 * 设置交互器
 * @param {ht.List} interactors 交互器对象集合
 */
setInteractors = function (interactors){};
```

```
/**
 * 设置是否使用鼠标漫游，默认为 true，如果改为 false，则鼠标左键右键都不支持前进后退的操作功能，
 * 但左键可拖拽编辑图元，右键可改变视角方向，采用这样的方式一般会结合键盘 w|s|a|d 按键进行漫游操作
 * @param {Boolean} v
 */
setMouseRoamable = function (v){};

/**
 * 设置移动过滤器函数
 * @param {Function} func 过滤器函数
 */
setMovableFunc = function (func){};

/**
 * 设置移动漫游步进
 * @param {Number} v
 */
setMoveStep = function (v){};

/**
 * 设置近端截面位置，默认值为 10
 * @param {Number} v
 */
setNear = function (v){};

/**
 * 设置是否显示坐标原点[0,0,0]轴线
 * @param {Boolean} v
 */
setOriginAxisVisible = function (v){};

/**
 * 设置是否使用正交投影
 * @param {Boolean} v
 */
setOrtho = function (v){};

/**
 * 设置正交投影宽度，默认为 2000
 * @param {Number} width
 */
```

```
setOrthoWidth = function (width){};

/**
 * 设置是否允许平移操作
 * @param {Boolean} v 是否可平移
 */
setPannable = function (v){};

/**
 * 设置拓扑上是否允许框选操作
 * @param {Boolean} v
 */
setRectSelectable = function (v){};

/**
 * 设置框选选择框的背景色
 * @param {color} color 颜色值
 */
setRectSelectBackground = function (color){};

/**
 * 设置是否允许通过空格将拓扑复位
 * @param {Boolean} v
 */
setResettable = function (v){};

/**
 * 设置是否可旋转
 * @param {Boolean} v
 */
setRotatable = function (v){};

/**
 * 设置旋转步进
 * @param {Number} v
 */
setRotateStep = function (v){};

/**
 * 设置图元是否可编辑旋转过滤器
 * @param {Function} func
 */
setRotationEditableFunc = function (func){};
```

```
/**
 * 设置拓扑是否共享选中模型
 * @param {Boolean} v
 */
setSelectionModelShared = function (v){};

/**
 * 设置大小编辑过滤器
 * @param {Function} func
 */
setSizeEditableFunc = function (func){};

/**
 * 设置摄像头正上方向，该参数一般较少改动，默认值为[0, 1, 0]
 * @param {Array} up 格式：[x, y, z]
 */
setUp = function (up){};

/**
 * 设置可见过滤器
 * @param {Function} func 过滤器函数
 */
setVisibleFunc = function (func){};

/**
 * 设置是否可进退
 * @param {Boolean} walkable
 */
setWalkable = function (walkable){};

/**
 * 设置布局宽度
 * @param {Number} width
 */
setWidth = function (width){};

/**
 * 缩放操作,默认操作模式意味着 eye 离 center 的远近变化,如果在 Graph3dView#isOrtho()为 true 的正交投影情况下，缩放意味着改变 Graph3dView#setOrthoWidth(width)的可视宽度范围。
 * @param {Number} increment 步进的比例，调用 zoomIn(anim)和 zoomOut(anim)，等同于调用了 setZoom(1.3, anim)和 setZoom(1/1.3, anim)。
 * @param {Boolean} anim 是否使用动画
 */
```

```
setZoom = function (increment, anim){};

/**
 * 设置是否可缩放
 * @param {Boolean} v
 */
setZoomable = function (v){};

/**
 * 获取选中模型，{@link ht.graph3d.Graph3dView#getSelectionModel getSelectionModel}的缩写
 * @see {@link ht.graph3d.Graph3dView#getSelectionModel getSelectionModel}
 * @return {ht.SelectionModel}
 */
sm = function (){};

/**
 * 将拓扑导出为 canvas
 * @param {color} background 背景色
 * @return {HTMLCanvasElement}
 */
toCanvas = function (background){};

/**
 * 将拓扑导出为 base64 格式字符串
 * @param {color} background 背景色
 * @return {String}
 */
toDataURL = function (background){};

/**
 * 删除交互事件监听器，{@link ht.graph3d.Graph3dView#removeInteractorListener removeInteractorListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.graph3d.Graph3dView#removeInteractorListener removeInteractorListener}
 */
umi = function (listener, scope){};

/**
 * 删除自身属性变化事件监听器，{@link ht.graph3d.Graph3dView#removePropertyChangeListener removePropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
```

```
 * @see {@link ht.graph3d.Graph3dView#removePropertyChangeListener
removePropertyChangeListener}
 */
ump = function (listener, scope){};

/**
 * 立刻刷新拓扑
 */
validate = function (){};

/**
 * 同时改变 eye 和 center 的位置，也就是 eye 和 center 在两点建立的矢量方向上同时移动相同的
偏移量。
 * 如果为第一人称模式调用 walk 操作，该函数会考虑 Graph3dView#getBoundaries()边界限制
 * @param {Number} step 偏移的矢量长度值
 * @param {Boolean} anim 是否使用动画
 * @param {Boolean} firstPersonMode  是否是第一人称模式，为空时则采用
Graph3dView#isFirstPersonMode()
 */
walk = function (step, anim, firstPersonMode){};

/**
 * 相当于调用 setZoom(1.3, anim)
 * @param {Boolean} [anim] 是否使用动画
 * @see {@link ht.graph3d.Graph3dView#setZoom setZoom}
 */
zoomIn = function (anim){};

/**
 * 相当于调用 setZoom(1/1.3, anim)
 * @param {Boolean} [anim] 是否使用动画
 * @see {@link ht.graph3d.Graph3dView#setZoom setZoom}
 */
zoomOut = function (anim){};
```

ht.widget.Palette

```
/**
 * 组件面板或调色板，类似于 Toolbar，允许用户快速访问按钮或命令
 * @constructor
 */
```

```
ht.widget.Palette = function() {};

/**
 * 增加自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.Palette#mp mp}
 */
addPropertyChangeListener = function (listener, scope, ahead){};

/**
 * 监听视图事件，如布局、刷新等
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 */
addViewListener = function (listener, scope, ahead){};

/**
 * 获取按钮元素的宽度，默认为 70
 * @return {Number}
 */
getItemImageWidth = function (){};

/**
 * 设置按钮元素的宽度，默认为 70
 * @param {Number} v
 */
setItemImageWidth = function (v){};

/**
 * 获取按钮元素的高度，默认为 50
 * @return {Number}
 */
getItemImageHeight = function (){};

/**
 * 设置按钮元素的高度，默认为 50
 * @param {Number} v
 */
setItemImageHeight = function (v){};

/**
```

```
 * 获取按钮元素图片与边框的距离，默认为 4
 * @return {Number}
 */
getItemImagePadding = function (){};

/**
 * 设置按钮元素图片与边框的距离，默认为 4
 * @param {Number} v
 */
setItemImagePadding = function (v){};


/**
 * 获取按钮元素之间的间隔，默认为 10
 * @return {Number}
 */
getItemMargin = function (){};

/**
 * 设置按钮元素之间的间隔，默认为 10
 * @param {Number} v
 */
setItemMargin = function (v){};

/**
 * 获取按钮元素的布局方式
 * <ul>
 * <li>largeicons:大图标模式</li>
 * <li>smallicons:小图标模式</li>
 * <li>iconsonly:仅图标模式</li>
 * </ul>
 * @return {String}
 */
getLayout = function (){};

/**
 * 设置按钮元素的布局方式
 * @param {String} layout
 * <ul>
 * <li>largeicons:大图标模式</li>
 * <li>smallicons:小图标模式</li>
 * <li>iconsonly:仅图标模式</li>
 * </ul>
 */
```

```
setLayout = function (layout){};


/**
 * 获取或设置数据模型，没有参数时相当于{@link ht.widget.Palette#getDataModel
getDataModel}，有参数时相当于{@link ht.widget.Palette#setDataModel setDataModel}
 * @param {ht.DataModel} [dataModel] 数据模型
 * @return {ht.DataModel} dataModel
 */
dm = function (dataModel){};

/**
 * 获取绑定的数据模型
 * @return {ht.DataModel} 数据模型
 */
getDataModel = function (){};


/**
 * 设置绑定的数据模型
 * @param {ht.DataModel} dataModel 数据模型
 */
setDataModel = function (dataModel){};


/**
 * 获取组件的根层 div
 * @return {HTMLDivElement}
 */
getView = function (){};

/**
 * 重绘组件
 */
redraw = function (){};


/**
 * 删除自身属性变化事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removePropertyChangeListener = function (listener, scope){};
```

```
/**
 * 删除视图事件监听器
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 */
removeViewListener = function (listener, scope){};

/**
 * 增加自身属性变化事件监听器，{@link ht.widget.Palette#addPropertyChangeListener addPropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @param {Boolean} [ahead] 是否将当前监听器插入到监听器列表开头
 * @see {@link ht.widget.Palette#addPropertyChangeListener addPropertyChangeListener}
 */
mp = function (listener, scope, ahead){};


/**
 * 删除自身属性变化事件监听器，{@link ht.widget.Palette#removePropertyChangeListener removePropertyChangeListener}的缩写
 * @param {Function} listener 监听器函数
 * @param {Object} [scope] 监听器函数域
 * @see {@link ht.widget.Palette#removePropertyChangeListener removePropertyChangeListener}
 */
ump = function (listener, scope){};
```