Table 模块

`Table` 即数据表，可以将数据表理解成一个 `数据源` ，它负责维护数据与数据之间的联系，并不涉及 UI 展示(如字段顺序、记录顺序等，这些顺序信息保存在 View 模块中)。

通过 `Base` 获取到 `Table` 之后，就可以调用 `Table` 中的 API，可以通过 `getActiveTable` 方法来获取当前选中的数据表实例：

typescript

```
const table = await bitable.base.getActiveTable()
```

当然也可以通过数据表 `id` 或名称来获取指定的数据表实例：

typescript

```
const table = await bitable.base.getTable(tableId/tableName)
```

---

id

数据表 id。

typescript

```
id: string;
```

---

getName

获取数据表名。

typescript

```
getName: () => Promise<string>;
```

示例

typescript

```
const name = await table.getName();
```

getMeta

获取数据表元数据。

```typescript
getMeta: () => Promise<ITableMeta>;

interface ITableMeta {
  id: string;
  name: string
  isSync: boolean; // 是否同步表
}
```

示例

```typescript
const meta = await table.getMeta();
```

获取字段

isFieldExist

判断指定字段 id 判断字段是否存在。

```typescript
isFieldExist(fieldId: string): Promise<boolean>;
```

示例

```typescript
const isExist = await table.isFieldExist('fieldId');
```

getField

```typescript
getField: <T extends IField>(idOrName: string) => Promise<T>;
```

通过 `id` 或 `name` 来获取对应的 `Field(字段)`，建议传入 `Field` 类型(例如示例中的 `<IAttachmentField>`)，来获得更好的语法提示。

示例

typescript

```
const field = await table.getField<IAttachmentField>(idOrName);
```

getFieldById

通过 `id` 来获取对应的 `Field(字段)`，建议传入 `Field` 类型(例如示例中的 `<IAttachmentField>`)，来获得更好的语法提示。

typescript

```
getFieldById: <T extends IField>(id: string) => Promise<T>;
```

示例

typescript

```
const field = await table.getFieldById<IAttachmentField>(idOrName);
```

getFieldByName

通过 `name` 来获取对应的 `Field(字段)`，建议传入 `Field` 类型(例如示例中的 `<IAttachmentField>`)，来获得更好的语法提示

typescript

```
getFieldByName: <T extends IField>(name: string) => Promise<T>;
```

示例

typescript

```
const field = await table.getFieldByName<IAttachmentField>(idOrName);
```

getFieldList

获取当前 table 下所有的字段列表。

typescript

```
getFieldList: <T extends IField>() => Promise<T[]>;
```

示例

typescript

```
const fieldList = await table.getFieldList();
```

getFieldIdList

获取字段 id 列表。

WARNING

通过该方法获取的字段 id 列表是**无序的**，因为 `table` 不涉及 UI 展示层面的信息，如果需要获取有序的字段 id 列表，需要在 `View 模块` 调用 `view.getVisibleFieldIdList` 来获取有序的字段 id 列表

typescript

```
getFieldIdList(): Promise<string[]>;
```

示例

typescript

```
const fieldIdList = await table.getFieldIdList();
```

getFieldMetaById

通过 id 获取对应的字段元信息。

typescript

```
getFieldMetaById(fieldId: string): Promise<IFieldMeta>;

interface IFieldMeta {
  id: string;
  type: FieldType;
  name: string;
  isPrimary: boolean;
  description: IBaseFieldDescription;
}
```

示例

typescript

```
const fieldMeta = await table.getFieldMetaById('f_id');
// { id: 'f_id', name: 'text field', type: 1, isPrimary: true, description: { content:
```

getFieldMetaList

获取所有字段元信息。

WARNING

通过该方法获取的字段 meta 列表是**无序的**，因为 `table` 不涉及 UI 展示层面的信息，如果需要获取有序的字段 meta 列表，需要在 `View 模块` 调用 `view.getFieldMetaList` 来获取有序的字段 id 列表

typescript

```typescript
getFieldMetaList(): Promise<IFieldMeta[]>;

interface IFieldMeta {
  id: string;
  type: FieldType;
  property: IFieldProperty;
  name: string;
  isPrimary: boolean;
  description: IBaseFieldDescription;
}
```

示例

typescript

```typescript
const fieldMetaList = await table.getFieldMetaList();
```

getFieldListByType

获取当前数据表下所有指定字段类型的字段列表。

typescript

```typescript
getFieldListByType: <T extends IField>(type: FieldType) => Promise<T[]>;
```

示例

typescript

```typescript
// 获取 table 下所有的附件字段
const attachmentFieldList = await table.getFieldListByType<IAttachmentField>(FieldType
```

getFieldMetaListByType

获取当前数据表下所有指定字段类型的字段元信息列表。

typescript

```typescript
getFieldMetaListByType: <T extends IFieldMeta>(type: FieldType) => Promise<T[]>;

interface IFieldMeta {
  id: string;
  type: FieldType;
  name: string;
  property: IFieldProperty;
  isPrimary: boolean;
  description: IBaseFieldDescription;
}
```

示例

```typescript
// 获取 table 下所有的附件字段的 Meta 列表
const attachmentFieldMetaList = await table.getFieldMetaListByType<IAttachmentFieldMeta
```

新增字段

addField

新增字段，并返回对对应的字段 id。

> **TIP**
>
> `addField` 支持直接配置字段属性，但推荐在新建字段之后通过对应的字段方法修改字段属性，更简便不易出错。

```typescript
addField: (fieldConfig: IAddFieldConfig) => Promise<FieldId>;

type IAddFieldConfig = {
  type: FieldType;
  property?: FieldProperty;
  name?: string;
  description?: { // 字段描述
    content?: string;
    /** 是否禁止同步，如果为true，表示禁止同步该描述内容到表单的问题描述（只在新增、修改字段时
    disableSyncToFormDesc?: boolean;
  };
}

type FieldId = string;
```

示例

```typescript
const field = await table.addField({ type: FieldType.SingleSelect });
const singleSelectField = await table.getField<ISingleSelectField>(field);
await singleSelectField.addOption('Option1');
```

如上所示的例子，我们先新增了一个单选字段，然后再在这个字段上新增了一个选项（推荐在获取字段的时候，指定对应的类型（如 `<ISingleSelectField>` ），以获得更好的语法提示）

onFieldAdd

监听 Field 添加事件，返回一个取消监听函数。

typescript

```
onFieldAdd(callback: (ev: IEventCbCtx) => void): () => void;
```

示例

typescript

```
const off = table.onFieldAdd((event) => {
  console.log('event:', event);
})
const fieldId = await table.addField({ // 新增一个多行文本类型的字段
  type: FieldType.Text,
  name: 'field_test'
})
```

修改字段

setField

修改字段，如字段类型、字段名称和字段属性等。

> **TIP**
>
> 推荐从 `Field` 实例调用对应的字段方法来修改字段属性，更简便不易出错。

typescript

```
setField(fieldId: string, fieldConfig: ISetFieldConfig): Promise<IFieldRes>;

// 支持不传 name、type、property 等参数，不传参数时会合并原属性
type ISetFieldConfig = {
  type: FieldType;
  property?: FieldProperty;
  name?: string;
  description?: { // 字段描述
    content?: string;
    /** 是否禁止同步，如果为true，表示禁止同步该描述内容到表单的问题描述（只在新增、修改字段时
    disableSyncToFormDesc?: boolean;
  };
}
```

示例

typescript

```typescript
const field = await table.getField('f_idxxx');

const res = await table.setField(field.id, {
  name: 'modify_field_name'
})
```

onFieldModify

监听字段修改事件，返回一个取消监听函数。

typescript

```typescript
onFieldModify(callback: (ev: IEventCbCtx) => void): () => void;
```

示例

typescript

```typescript
const off = table.onFieldModify((event) => {
  console.log('field modify:', event);
})

const fieldId = await table.addField({ // 新增一个多行文本类型的字段
  type: FieldType.Text,
  name: 'field_test'
})

const fieldId = await table.setField({ // 修改字段名称
  name: 'field_modified'
})
```

删除字段

deleteField

删除指定字段。

typescript

```typescript
deleteField: (fieldOrId: string | IField) => Promise<boolean>;
```

示例

typescript

```typescript
const attachmentField = await table.addField({ FieldType.Attachment });
// 直接传递 field 实例
await table.deleteField(attachmentField)
// 或者传递 fieldId
await table.deleteField(attachmentField.id);
```

onFieldDelete

监听 Field 删除事件，返回一个取消监听函数。

typescript

```typescript
onFieldDelete(callback: (ev: IEventCbCtx) => void): () => void;
```

示例

typescript

```typescript
const off = table.onFieldDelete((event) => {
  console.log('field delete', event);
})

const fieldId = await table.addField({ // 新增多行文本类型的字段
  type: FieldType.Text,
  name: 'field_test'
})

table.deleteField(fieldId) // 删除字段
```

获取记录

getCellValue

获取指定单元格的取值。

typescript

```typescript
getCellValue(fieldId: string, recordId: string): Promise<IOpenCellValue>;
```

示例

typescript

```typescript
// 光标选中数据表中的单元格
const { fieldId, recordId } = await bitable.base.getSelection();
const cellValue = table.getCellValue(fieldId, recordId);
```

getRecordById

通过指定 id 去获取对应记录。

> **TIP**
>
> 批量获取场景下，建议使用 getRecords 方法以获得更好的性能体验

typescript

```
getRecordById(recordId: string): Promise<IRecordValue>;

type IRecordValue = {
  fields: {
    [fieldId: string]: IOpenCellValue;
  };
};
```

示例

typescript

```
const recordIdList = await table.getRecordIdList(); // 获取 recordId 列表

const recordValue = await table.getRecordById(recordIdList[0]);
```

getRecords

批量获取 record 数据。

> **WARNING**
>
> 单次获取上限 **5000** 条。

typescript

```
getRecords({ pageSize, pageToken, viewId }: IGetRecordsParams): Promise<IGetRecordsRes
```

名称	数据类型	是否必填	描述
pageSize	number	是	分页页面大小 size，最大值：5000
pageToken	string	否	分页标记，第一次请求不填，表示从头开始遍历；分页查询结果还有更多项时会同时返回新的 page_token，下次遍历可采用该 page_token 获取查询结果
viewId	string	否	视图的唯一标识符，获取指定视图下的记录

相关类型定义如下：

```typescript
interface IGetRecordsParams {
  pageSize?: number; // 获取数量，默认 5000，最大不得超过 5000
  pageToken?: string; // 分页标记，第一次请求不填，表示从头开始遍历；分页查询结果还有更多项时
  viewId?: string;  // 获取指定视图的 record
}

interface IGetRecordsResponse {
  total: number; // 记录总数
  hasMore: boolean; // 是否还有更多记录
  records: IRecord[]; // 记录列表
  pageToken?: string; // 分页标记
}

interface IRecord {
  recordId: string;
  fields: {
    [fieldId: string]: IOpenCellValue;
  };
}
```

示例

```typescript
// 首先使用 getActiveTable 方法获取了当前用户选择的 table（用户当前编辑的数据表）
const table = await bitable.base.getActiveTable();
const records = await table.getRecords({
  pageSize: 5000
})
```

getRecordIdList

获取所有记录 id 列表。

> **WARNING**
>
> 通过该方法获取的记录 id 列表是**无序的**，因为 `table` 不涉及 UI 展示层面的信息，如果需要获取有序的字段 id 列表，需要在 `View 模块` 调用 `view.getVisibleRecordIdList` 来获取有序的记录 id 列表

```typescript
getRecordIdList(): Promise<string[]>;
```

示例

typescript

```
const recordIdList = await table.getRecordIdList();
```

getRecordList

获取当前的记录列表， `Record` 模块中的相关方法可以查看 Record 模块

typescript

```
getRecordsList(): Promise<Record>;
```

typescript

```
const recordList = await table.getRecordList();
for (const record of recordList) {
  const cell = await record.getCellByField(fieldId);
  const val = await cell.getValue();
}
```

getCellAttachmentUrls

批量获取指定附件单元格中附件的 URL，参数中的 token 需要从附件字段所属的单元格中获取。(推荐通过 AttachmentField 模块去获取)

> **WARNING**
>
> 接口返回的临时链接的有效时间是 10 分钟

typescript

```
getCellAttachmentUrls(tokens: string[], fieldId: string, recordId: string): Promise<str
```

示例

typescript

```
const urls = await table.getCellAttachmentUrls(['token_1', 'token_2'], 'f_id', 'r_id')
```

getCellThumbnailUrls

批量获取指定附件单元格中**缩略图的 URL**，可指定缩略图的图片质量，参数中的 token 需要从附件字段所属的单元格中获取，该接口返回的是 `base64` 格式的字符串。

typescript

```
getCellThumbnailUrls(tokens: string[], fieldId: string, recordId: string, quality: Imag

enum ImageQuality {
  Low = 120,
```

```
  Mid = 360,
  HIGH = 720,
  MAX = 1280,
}
```

示例

typescript

```
const urls = await table.getCellThumbnailUrls(['token_1', 'token_2'], 'f_id', 'r_id',
```

getRecordShareLink

获取指定记录的分享链接，获得链接的用户，将以多维表格的权限访问。

typescript

```
getRecordShareLink(recordId: string)
```

示例

typescript

```
const recordShareLink = await table.getRecordShareLink('r_Id')
```

新增记录

addRecord

新增一条记录，新增成功后返回 `recordId` ，支持直接传递 `RecordValue` 或单元格 `Cell` 实例。

> **TIP**
>
> 批量新增场景下，建议使用 addRecords 方法以获得更好的性能体验

typescript

```
addRecord: (recordVale?: IRecordValue | ICell | ICell[]) => Promise<IRecordRes>;

type IRecordValue = {
  fields: {
    [fieldId: string]: IOpenCellValue;
  };
};

type IRecordRes = string;
```

示例

如果使用 `RecordValue` 来创建（不推荐）：

typescript

```
const field = await table.getField('多行文本'); // 选择某个多行文本字段

const res = await table.addRecord({
  fields: {
    [field.id]: 'new text field value'
  }
});
// 'recxxx' 新增的记录 id
```

更推荐通过组合 Cell 实例来插入一条记录，Cell 可以通过各个字段的 `createCell` 方法来创建，下面是一个例子:

typescript

```
const textField = await table.getField<TextField>('多行文本');

const textCell = await textField.createCell('new text field value');
const recordId = await table.addRecord(textCell);
```

addRecords

新增多条记录，新增成功后返回 `recordId` 列表。

> **WARNING**
>
> 单次新增记录上限 **5000** 条

typescript

```
addRecords: (record?: IRecordValue[] | ICell[] | Array<ICell[]>) => Promise<IRecordRes

type IRecordValue = {
  fields: {
    [fieldId: string]: IOpenCellValue;
  };
};

type IRecordRes = string;
```

示例

如果使用 `RecordValue` 来创建（不推荐）：

typescript

```typescript
const field = await table.getField('多行文本'); // 选择某个多行文本字段

const res = await table.addRecords([
  {
    fields: {
      [field.id]: 'new text field value1'
    }
  },
  {
    fields: {
      [field.id]: 'new text field value2'
    }
  },
]);
```

更推荐通过组合 Cell 实例来插入多条记录，Cell 可以通过各个字段的 `createCell` 方法来创建，下面是一个例子:

typescript

```typescript
const textField = await table.getField('多行文本'); // 选择某个多行文本字段

const textCell1 = await textField.createCell('new text field value1');
const textCell2 = await textField.createCell('new text field value1');
const recordIds = await table.addRecords([[textCell1],[textCell2]]);
```

onRecordAdd

监听 Record 添加事件，返回一个取消监听方法。

typescript

```typescript
onRecordAdd(callback: (ev: IEventCbCtx<[recordId: string]>) => void): () => void;
```

示例

typescript

```typescript
const field = await table.getField('多行文本'); // 根据字段名获取多行文本类型的字段
const off = table.onRecordAdd((event) => { // 监听字段增加事件
  console.log('record add', event);
});

const cell = field.createCell('new text field value');
table.addRecord(cell);
```

修改记录

setCellValue

修改指定单元格的值。(推荐通过 Field 来修改)

> **TIP**
>
> 批量修改场景下，建议使用 setRecords 方法以获得更好的性能体验

typescript

```
setCellValue<T extends IOpenCellValue = IOpenCellValue>(fieldId: string, recordId: str
```

示例

typescript

```
const recordIds = await table.getRecordIdList();
const field = await table.getField('多行文本');

// 修改某个多行文本类型的字段
const res = await table.setCellValue(field.id, recordIds[0], 'test setCellValue')
// true
```

setRecord

修改指定记录数据。

> **TIP**
>
> 批量修改场景下，建议使用 setRecords 方法以获得更好的性能体验

typescript

```
setRecord(recordId: string, recordValue?: IRecordValue): Promise<IRecordRes>;

type IRecordValue = {
  fields: {
    [fieldId: string]: IOpenCellValue;
  };
};
```

示例

typescript

```
const recordIds = await table.getRecordIdList(); // 获取所有记录 id
const field = await table.getField('多行文本'); // 选择多行文本字段
```

```
const res = await table.setRecord(recordIds[0], {
    fields: {
      [field.id]: 'test setRecord'
    }
})
```

setRecords

批量修改记录数据。

> **WARNING**
>
> 单次修改记录上限 5000 条

typescript

```
setRecords(records?: IRecord[]): Promise<IRecordRes[]>;

interface IRecord {
  recordId: string;
  fields: {
    [fieldId: string]: IOpenCellValue;
  };
}
```

示例

typescript

```
const recordIds = await table.getRecordIdList(); // 获取所有记录 id
const field = await table.getField('多行文本'); // 选择多行文本字段

await table.setRecords([
  {
    recordId: recordIds[0],
    fields: {
      [field.id]: 'test setRecords1'
    }
  },
  {
    recordId: recordIds[1],
    fields: {
      [field.id]: 'test setRecords2'
    }
  }
])
```

getCellString

获取单元格取值的**原始字符串形式**，如日期字段会返回具体的年月日字符串。

typescript

```
getCellString(fieldId: string, recordId: string): Promise<string>;
```

示例

typescript

```
const recordIds = await table.getRecordIdList();
const dateTimeField = await table.getField('日期');

const res = await table.getCellString(dateTimeField.id, recordIds[0]);
// 2023/10/01
```

onRecordModify

监听 Record 修改事件，返回一个取消监听方法。如果记录修改前后并未发生变化，则不会触发回调函数。

typescript

```
onRecordModify(callback: (ev: IEventCbCtx<{
  recordId: string;
  fieldIds: string[];
}>) => void): () => void;
```

示例

typescript

```
const recordIds = await table.getRecordIdList();
const field = await table.getFieldByName('多行文本')

const off = table.onRecordModify((event) => { // 监听记录修改事件
  console.log('record modify', event);
})

await table.setRecord(recordIds[0], { // 修改某条记录的多行文本字段
  fields:{
    [field.id]: 'modify value'
  }
})
```

删除记录

deleteRecord

删除指定记录。

> **TIP**
>
> 批量删除场景下，建议使用 deleteRecords 方法以获得更好的性能体验

typescript

```typescript
deleteRecord(recordId: string): Promise<boolean>;
```

示例

typescript

```typescript
const recordIdList = await table.getRecordIdList();

await table.deleteRecord(recordIdList[0]);
```

deleteRecords

批量删除记录。

> **WARNING**
>
> 单次删除记录上限 5000 条

typescript

```typescript
deleteRecords(recordIdList: string[]): Promise<boolean>;
```

示例

typescript

```typescript
const recordIdList = await table.getRecordIdList();

// 删除前100条记录
await table.deleteRecords(recordIdList.slice(0, 100));
```

onRecordDelete

监听 Record 删除事件，返回一个取消监听方法。

typescript

```typescript
onRecordDelete(callback: (ev: IEventCbCtx<[recordId: string]>) => void): () => void;
```

示例

typescript

```typescript
const off = table.onRecordDelete((event) => {
  console.log('record delete', event);
})

const recordIdList = await table.getRecordIdList();
table.deleteRecord(recordIdList[0]);
```

获取视图

View 模块相关能力请参考 视图模块。

getActiveView

> **WARNING**
>
> This method is under testing, please use the 0.3.5-alpha.4 version package for test

获取当前选择的 View 视图。

typescript

```typescript
getActiveView: () => Promise<IView>;
```

示例

typescript

```typescript
const view = await table.getActiveView();
```

isViewExist

通过 viewId 判断视图是否存在。

typescript

```typescript
isViewExist(viewId: string): Promise<boolean>;
```

示例

typescript

```typescript
const isExist = await table.isViewExist('viewId');
```

getViewById

通过 id 来获取 View 视图。

typescript

```
getViewById: (id: string) => Promise<IView>;
```

示例

typescript

```
const view = await table.getViewById(viewId);
```

getViewList

获取当前数据表的所有视图。

typescript

```
getViewList: () => Promise<IView[]>;
```

示例

typescript

```
const viewList = await table.getViewList();
```

getViewMetaById

通过 id 获取视图的元信息。

typescript

```
getViewMetaById(viewId: string): Promise<IViewMeta>;

interface IViewMeta {
  id: string;
  name: string;
  type: ViewType;
  property: IViewProperty;
}
```

示例

typescript

```
const viewMeta = await table.getViewById(viewId);
```

getViewMetaList

获取当前数据表下所有的视图元信息。

typescript

```
getViewMetaList(): Promise<IViewMeta[]>;
```

```typescript
interface IViewMeta {
  id: string;
  name: string;
  type: ViewType;
  property: IViewProperty;
}
```

示例

typescript

```typescript
const viewMetaList = await table.getViewMetaList();
```

新增视图

addView

给当前数据表添加视图。

> **WARNING**
>
> 目前仅支持设置 `ViewType` 和 `name`，推荐创建后通过 View 模块的 API 进行视图配置。

typescript

```typescript
addView(config: IAddViewConfig): Promise<IAddViewResult>;

interface IAddViewConfig {
  name?: string;
  type: ViewType;
}

interface IAddViewResult {
  viewId: string;
  index: number; // 视图顺序
}
```

示例

typescript

```typescript
await table.addView({ type: ViewType.Grid, name: 'test'});
```

修改视图

setView

修改指定视图信息。

> **WARNING**
>
> 目前仅支持设置 `ViewType` 和 `name`，推荐创建后通过 View 模块的 API 进行视图配置。

typescript

```typescript
setView(viewId: string, config: ISetViewConfig): Promise<ViewId>;

interface ISetViewConfig {
  name?: string;
}
```

示例

typescript

```typescript
await table.setView('v_id', { name: 'modified name'});
```

删除视图

deleteView

删除指定视图。

typescript

```typescript
deleteView(viewId: string): Promise<boolean>;
```

示例

typescript

```typescript
await table.deleteView('v_id');
```

Last updated: 2024/5/13 16:06