tradingview

序言

最新修订时间: 2019-12-17

本书翻译自官方 wiki：https://github.com/tradingview/charting_library/wiki

> 未申请 lincense 的同学，点击上面官方 wiki 会报 404 错误：）

TradingView 为优秀的交易技术分析金融图表，拥有丰富的技术指标库，并拥有可以直接交易的交易终端插件。

本项目地址

- 仓库：https://github.com/zlq4863947/tradingViewWikiCn
- 在线阅读：https://zlq4863947.gitbooks.io/tradingview/
- 在线阅读(国内防墙)：https://b.aitrade.ga/books/tradingview/

---

tradingview 轻量级图表插件

- 仓库：https://github.com/zlq4863947/lightweight-charts-docs-cn
- 在线阅读(国内防墙)：https://b.aitrade.ga/books/lightweight-charts-docs-cn/

视频教程

- 仓库：https://github.com/zlq4863947/proficient-tradingview
- 在线阅读：https://b.aitrade.ga/books/proficient-tradingview

开发交流 QQ 群

- tradingview 开发：313839516

更新日志

图表库更新日志

- 版本变更点

1.15 开发版 -- 20191217

- 文档更新为 Tradingview1.15 开发版

1.15 开发版 -- 20190801

- 文档更新为 Tradingview1.15 开发版

1.14 开发版 -- 20180922

- 文档更新为 Tradingview1.14 开发版

1.13 正式版 -- 20180906

- 文档更新为 Tradingview1.13 正式版
- 将术语**分辨率**以及**间隔**改为**周期**

1.13 dev -- 20180801

- 文档更新至 Tradingview 的 wiki 最新版本
- 将术语**研究**改为**指标**
- 修正文字错误及语义不通的地方

1.13 -- 20180419

- 文档更新至 Tradingview 1.13 开发库版本

1.12 -- 20180113

- 中文开发文档版本与 Tradingview 开发库版本一致

1.1 -- 20171223

- 图片文字汉化
- 链接与内容修正
- 官方 wiki 同步更新

1.0 -- 20170910

- 初版做成

0.1 ~ 0.9 -- 20??????

- 膜拜大神

膜拜

1、Charting Library 是什么

Charting Library是什么

具有开放数据API并可下载的图表控件。这是一个独立的解决方案，您可以下载，托管在您的服务器上，连接您自己的数据，并在您的网站/应用程序中免费使用。您所要做的就是:

步骤	您会得到什么
1. `git clone` 下载Charting Library并且运行它	在主机上运行的图表的例子
2. 使用我们的API将您的数据插入图表库。您可以参考这些例子	部署并运行图表库并且加载您的数据

如果您想定制您的图表，那么您可以更进一步。

- 查询图表库 定制概述 和 定制用例
- 创建自定义指标

使用实例

我们有一个公开的GitHub资源库，其中包含了集成图表库的实例。 这些实例将告诉您如何使用不同的Web框架集成图表库。

交易终端

📈 交易终端是为强大图表配备的随时可用的产品，使图表具备交易功能。阅读更多

最佳实践

阅读这篇文章会让您避免常见的错误，**节约您的时间**。

2、入门指南

2-1、图表库内容

Charting Library 图表库包可在GitHub上获得（必须获得授权才能访问GitHub上的这个私有资源）. 您可以检出最新的稳定版本(`master` branch)或最新的开发版本(`unstable` branch)。如果想要访问此资源请联系我们。

您可以通过在浏览器控制台中执行 TradingView.version() 来查看图表库版本。

图表库内容

```
+/charting_library
    + /static
    - charting_library.min.js
    - charting_library.min.d.ts
    - datafeed-api.d.ts
+ /datafeeds
    + /udf
- index.html
- mobile_black.html
- mobile_white.html
- test.html
```

- `/charting\_library` 包含全部的图表库文件。
- `/charting\_library/charting\_library.min.js` 包含外部图表库widget接口。此文件不应该被修改。
- `/charting_library/charting_library.min.d.ts` 包含TypeScript定义的widget接口
- `/charting_library/datafeed-api.d.ts` 包含TypeScript定义的datafeed接口。
- `/charting_library/datafeeds/udf/` 包含UDF-compatible 的datafeed包装类（用于实现JS API通过UDF传输数据给图表库）。例子中的datafeed包装器类实现了脉冲实时仿真数据。您可以自由编辑此文件。
- `/charting\_library/static` 文件夹中存储图表库内部资源，不适用于其他目的。
- `/index.html` 为使用Charting Library widget 的html例子。
- `/test.html` 为不同的图表库自定义功能使用的示例。
- `/mobile\*.html` 也是Widget自定义的示例。

所有内部的JS和CSS代码都被内联和缩小，以减少页面加载时间。 需要由您编辑的文件不会被缩小。

2-2、运行图表库

运行图表库

让我们在您的本地计算机上启动图表库

1. 克隆图表库资源库

克隆资源库方法介绍

2. 启动HTTP服务器

您可以设置任何HTTP服务器来侦听主机的任何空闲端口，并参考包含图表库的文件夹。 您将在下面找到有关如何启动多个最常见HTTP服务器的说明。

Python 2.x

如果您的计算机上安装了Python，则可以使用它以快捷方式启动HTTP服务器。

在图表库文件夹中执行以下命令：

```
python -m SimpleHTTPServer 9090
```

Python 3.x

```
python -m http.server 9090
```

NodeJS

首先安装 `http-server`：

```
npm install http-server -g
```

在图表库文件夹中，通过以下命令启动 `http-server`：

`http-server -p 9090`

NGINX

- 安装 NGINX
- 打开配置文件 `nginx.conf` 并在文件的 `http` 部分改为以下内容：

```
    server {
        listen       9090;
        server_name  localhost;

        location / {
            root ABSOLUTE_PATH_TO_THE_CHARTING_LIBRARY;
        }
    }
```

- 将 `ABSOLUTE_PATH_TO_THE_CHARTING_LIBRARY` 替换为图表库文件夹的绝对路径。
- 运行 `nginx`

3. 在浏览器中打开图表库

运行HTTP服务器后，您将拥有一个已准备好的图表库。

在浏览器中打开 `http://127.0.0.1:9090` 。

在线示例

您会看到图表库此处的运行示例。 此示例基于我们的示例datafeed。

重要：此数据源只是一个示例。 它包含十几个商品（来自Quandl）并仅提供DWM(日周月)。 但它支持报价。 请仅将其用于测试目的。

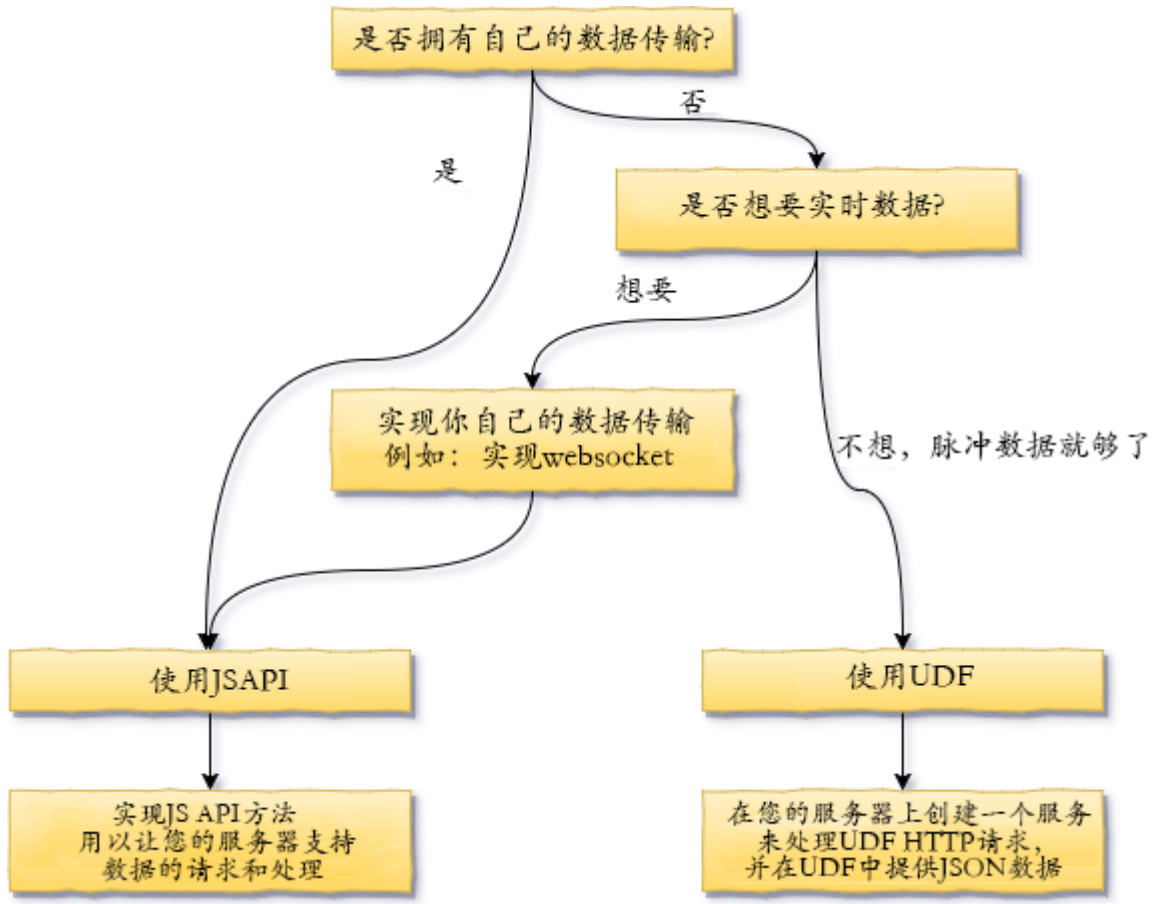
3、数据绑定

3-1、如何连接我的数据

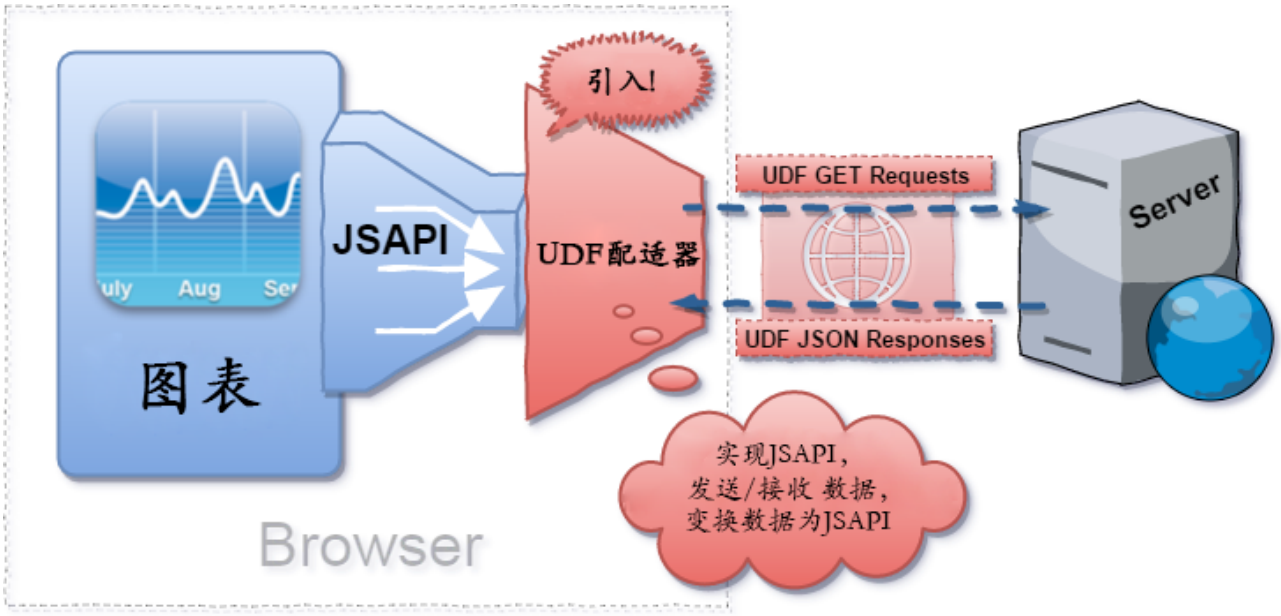
图表库并不包含市场数据，你必须提供所需格式的数据。示例使用了Quandl历史数据。图表可以用两种方式接收数据：

1. 使用推模型技术实时更新，例如通过WebSocket。这样你的图表将会自动更新价格。为了达到这个目的，你必须使用JavaScript API并且准备好自己的传输方法。
2. 使用拉模型/脉冲(pulse)/刷新为基础进行更新（如当今大多数基于Web的图表），其中图表数据每X秒更新一次（图表客户端将要求服务器模拟推模型更新），或者被用户手动重新加载。 为此，请使用UDF协议并编写自己的datafeed包装类。

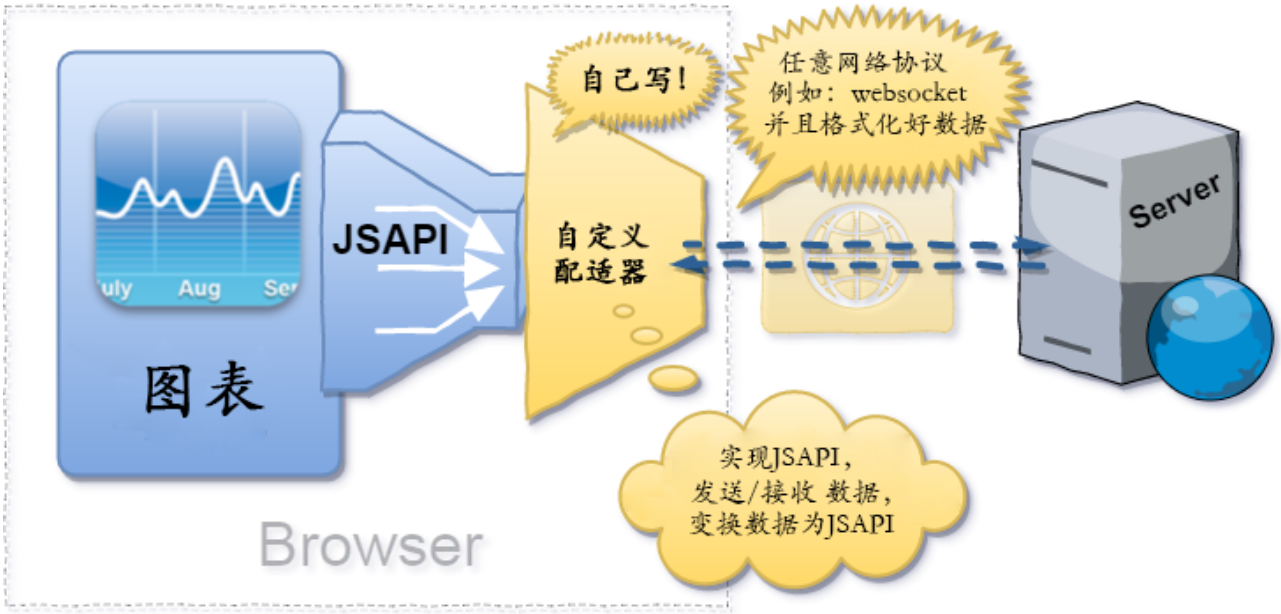
JavaScript API 或者 UDF?



UDF scheme



JSAPI scheme



上图显示了UDF和JSAPI之间的区别。 必须的图表库部分是蓝色的。 红色的部分（默认数据传输）包含在默认包中（具有未压缩的源代码），并可以被替换。 您可以看到默认数据传输实现JS API来与图表交互。此外，默认传输实现了与服务器通信的UDF协议。

1. **如果您已经准备好了数据传输**（websocket流传输，轮询或任何其他传输），或者如果您不需要流传输数据 - 请使用我们的JavaScript API，这是非常紧凑和易于实现。 您必

须使用JavaScript在数据传输和我们的图表之间创建一个小的 **客户端数据适配器**。

2. **如果您没有任何传输**，并且不需要流数据（例如，您只需要数据脉冲），那么您将不得不创建（或使用）至少一个服务器端的datafeed包装类。 您可以使用任何语言和技术来实现这一目的：您的包装类只需要支持我们的数据交换协议（我们称之为UDF），以便能够为您的图表提供数据。 您必须使用自己喜欢的语言在后端和我们的图表之间创建一个小的 **服务器端数据适配器**。

示例

UDF-compatible 实现示例。可以在github上获得, 是一个服务器端包装类的示例, 它使用Quandl数据。

JS API 实现示例（和UDF客户端同时）为图表库的一部分（请参阅 `/datafeeds/udf/` 文件夹）。

3-2、JS Api

JS Api

这是啥? 一套JS方法（以实现指定的公共接口）。
我该怎么使用它?: 您应该创建一个JS对象，它将以某种方式接收数据，并响应图表库的请求。

在图表库中实现了数据缓存（历史和商品信息）。当您创建一个实现接口的对象时，只需将它传递给图表库Widget的构造函数。

Methods

1. onReady
2. searchSymbols
3. resolveSymbol
4. getBars
5. subscribeBars
6. unsubscribeBars
7. calculateHistoryDepth
8. getMarks
9. getTimescaleMarks
10. getServerTime

💹 交易终端专属:

1. getQuotes
2. subscribeQuotes
3. unsubscribeQuotes
4. subscribeDepth
5. unsubscribeDepth

onReady(callback)

```
callback: function(configurationData)
    configurationData: object (见下文)
```

此方法可以设置图表库支持的图表配置。这些数据会影响到图表支持的功能，所以它被称为服务端定制。

图表库要求您使用回调函数来传递datafeed的 `configurationData` 参数。

configurationData是一个对象，现在支持以下属性:

exchanges

交易所对象数组。 交易所对象的结构为: `{value, name, desc}` 。

`value` 将被作为 `exchange` 参数传递给 searchSymbols。

`exchanges = []` 会导致商品查询列表中看不到交易所过滤器。 `value = ""` 来创建通配符筛选器（所有的交易所）。

symbols_types

商品类型对象数组。商品类型对象的结构为: `{name, value}` 。 `value` 将被作为 `symbolType` 参数传递给searchSymbols。

`symbolsTypes` = [] 会导致商品查询列表中看不到商品类型过滤器。 使用 `value= ""` 来创建通配符筛选器（所有的商品类型）。

supported_resolutions

支持的周期数组，周期可以是数字或字符串。 如果周期是一个数字，它被视为分钟数。 字符串可以是“*D”，“*W”，“_M”（_的意思是任何数字）。格式化详细参照:文章。

`resolutions = undefined` 或 `resolutions = []` 时，周期拥有 `widget` 中的默认内容。

例: `[1, 15, 240, "D", "6M"]` 您将在周期中得到 "1 分钟, 15 分钟, 4 小时, 1 天, 6 个月" 。

supports_marks

布尔值来标识您的 datafeed 是否支持在K线上显示标记。

supports_timescale_marks

布尔值来标识您的 datafeed 是否支持时间刻度标记。

supports_time

将此设置为 `true` 假如您的datafeed提供服务器时间（unix时间）。 它仅用于在价格刻度上显示倒计时。

futures_regex

设置后可以在商品搜索中对期货进行分组。 这个正则表达式会将期货商品分为两部分：合约种类和到期时间。

实例 regex: : `/^(.+)([12]!|[FGHJKMNQUVXZ]\d{1,2})$/` . 它将应用于类型为 `futures` 的商品图表。

searchSymbols(userInput, exchange, symbolType, onResultReadyCallback)

1. `userInput` : string，用户在商品搜索框中输入的文字。
2. `exchange` :string，请求的交易所（由用户选择）。空值表示没有指定。
3. `symbolType` : string，请求的商品类型： `index` 、 `stock` 、 `forex` 等等（由用户选择）。空值表示没有指定。
4. `onResultReadyCallback` : function(result)
    1. `result` : 数组 (见下文)

方法介绍：提供一个匹配用户搜索的商品列表。 `result` 为预期的商品 ，像下面这样:

```
[
    {
        "symbol": "<商品缩写名>",
        "full_name": "<商品全称 -- 例: BTCE:BTCUSD>",
        "description": "<商品描述>",
        "exchange": "<交易所名>",
```

```
        "ticker": "<商品代码，可选>",
        "type": "stock" | "futures" | "bitcoin" | "forex" | "index"
    }, {
        //    .....
    }
]
```

如果没有找到商品，则应该使用空数组来调用回调。查看更多关于 `ticker` 的细节 在这里

resolveSymbol(symbolName, onSymbolResolvedCallback, onResolveErrorCallback)

1. `symbolName` : string，商品名称 或 `ticker`
2. `onSymbolResolvedCallback` : function(SymbolInfo)
3. `onResolveErrorCallback` : function(reason)

方法介绍：通过商品名称解析商品信息(SymbolInfo)。

getBars(symbolInfo, resolution, from, to, onHistoryCallback, onErrorCallback, firstDataRequest)

1. `symbolInfo` :SymbolInfo 商品信息对象
2. `resolution` : string （周期）
3. `from` : unix 时间戳, 最左边请求的K线时间
4. `to` : unix 时间戳, 最右边请求的K线时间
5. `onHistoryCallback` : function(数组 `bars` , `meta` ={ *noData = false* })
    1. `bars` : Bar对象数组 `{time, close, open, high, low, volume}[]`
    2. `meta` : object `{noData = true | false, nextTime = unix time}`
6. `onErrorCallback` : function(reason：错误原因)
7. `firstDataRequest` : 布尔值，以标识是否第一次调用此商品/周期的历史记录。当设置为 `true` 时
   你可以忽略 `to` 参数（这取决于浏览器的 `Date.now()`) 并返回K线数组直到当前K线（包括它）。

方法介绍：通过日期范围获取历史K线数据。图表库希望通过 `onHistoryCallback` 仅一次调用，接收所有的请求历史。而不是被多次调用。

> 发生不断自动刷新图表问题时，请检查 `from` 和 `to` 与 `onHistoryCallback` 方法返回的K线时间范围是否一致，没有数据时请返回 `noData = true`

`nextTime` 历史中下一个K线柱的时间。 只有在请求的时间段内没有数据时，才应该被设置。

`noData` 只有在请求的时间段内没有数据时，才应该被设置。

Remark: `bar.time` 为以毫秒开始的Unix时间戳（UTC标准时区）。

Remark: `bar.time` 对于日K线预期一个交易日 (未开始交易时) 以 00:00 UTC为起点。 图表库会根据商品的交易（Session）时间进行匹配。

Remark: `bar.time` 对于月K线为这个月的第一个交易日，除去时间的部分。

subscribeBars(symbolInfo, resolution, onRealtimeCallback, subscriberUID, onResetCacheNeededCallback)

1. `symbolInfo` :object SymbolInfo
2. `resolution` : string 周期
3. `onRealtimeCallback` : function(bar)
    1. `bar` : object `{time, close, open, high, low, volume}`
4. `subscriberUID` : object
5. `onResetCacheNeededCallback` (从1.7开始): function()将在bars数据发生变化时执行

方法介绍：订阅K线数据。图表库将调用 `onRealtimeCallback` 方法以更新实时数据。

Remark: 当您调用 `onRealtimeCallback` 且K线时间等于最近一条K线时间时，那么这条最近的K线将被您传入的K线所替换。 例:

1. 最近一条K线为 `{1419411578413, 10, 12, 9, 11}`
2. 调用 `onRealtimeCallback({1419411578413, 10, 14, 9, 14})`
3. 图表库通过时间 `1419411578413` 找出K线，已存在并且是最近的那一个
4. 图表库替换K线，因此现在最近一条K线为 `{1419411578413, 10, 14, 9, 14}`

Remark 2: 是否可以更新最近的K线或追加一条新的，取决于 `onRealtimeCallback` 。 如果您调用此功能尝试更新历史记录中的一个K线时，则会收到错误消息。

Remark 3: 现在，在图表接收到数据后，没有办法改变历史上的K线。

unsubscribeBars(subscriberUID)

1. `subscriberUID` :object

方法介绍：取消订阅K线数据。在调用 `subscribeBars` 方法时,图表库将跳过与 `subscriberUID` 相同的对象。

calculateHistoryDepth(resolution, resolutionBack, intervalBack)

1. `resolution` : 请求商品的周期
2. `resolutionBack` : 期望历史周期刻度。支持的值: `D` | `M`
3. `intervalBack` : 数量

方法介绍：图表库在它要请求一些历史数据的时候会调用这个函数，让你能够覆盖所需的历史深度。

通过传递的参数，可以让您知道要获得什么样的K线数据。 以下是几个例子:

- `calculateHistoryDepth("D", "M", 12)`
  调用: 图表库请求12个月的日线数据
- `calculateHistoryDepth(60, "D", 15)`
  调用: 图表库请求15天的60分钟数据

如果你不想重写处理方法，这个函数应该返回 `undefined` 。如果你想要重写，它应该返回一个对象 `{resolutionBack, intervalBack}` 。

例子:

假设实现为

```
Datafeed.prototype.calculateHistoryDepth = function(resolution, resolution
    if (period == "1D") {
```

```
            return {
                resolutionBack: 'M',
                intervalBack: 6
            };
        }
    }
```

以上代码为当图表库将要求周期为 `1D` ，历史为6个月的深度。 在其他情况下，历史深度将具有其他默认值。

getMarks(symbolInfo, startDate, endDate, onDataCallback, resolution)

1. `symbolInfo` :SymbolInfo 商品信息对象
2. `startDate` ： unix 时间戳, 最左边请求的K线时间
3. `endDate` : unix 时间戳, 最右边请求的K线时间
4. `onDataCallback` : function(标记数字 `marks`)
5. `resolution` : string

方法介绍：图表库调用这个函数来获得可见的K线范围的标记。 图表预期每调用一次 `getMarks` 就会调用一次 `onDataCallback` 。

`mark` 为具有以下属性的对象:

- **id**: 唯一标识id 。当用户点击标记时，将传递给相应的回调:respective callback
- **time**: unix time, UTC
- **color**: `red` | `green` | `blue` | `yellow` | `{ border: '#ff0000', background: '#00ff00' }`
- **text**: 标记弹出式文字。 支持HTML
- **label**: 印在标记上的文字。单字符
- **labelFontColor**: label的文字颜色
- **minSize**: 标记的最小尺寸 (diameter, pixels)

每个K线允许几个标记（现在最多为10个）。不允许标记脱离K线。

Remark: 只有当您声明您的后端是支持标记时才会调用这个函数。supporting marks.

getTimescaleMarks(symbolInfo, startDate, endDate, onDataCallback, resolution)

1. `symbolInfo` :SymbolInfo object
2. `startDate` : unix时间戳 (UTC). Leftmost visible bar's time.
3. `endDate` : unix时间戳 (UTC). Rightmost visible bar's time.
4. `onDataCallback` : function(array of `mark` s)
5. `resolution` : string

图表库调用此函数获取可见K线范围的时间刻度标记。图表预期您每个调用getTimescaleMarks会调用一次onDataCallback。

mark为具有以下属性的对象:

- **id**: 唯一标识id 。当用户点击标记时，将传递给相应的回调:respective callback
- **time**: unix time, UTC
- **color**: `red` | `green` | `blue` | `yellow` | ... | #000000
- **label**: 印在标记上的文字。单字符
- **tooltip**: 字符串串数组。数组的每个元素都是工具提示的单独行内容。

每个K线只允许一个标记。 不允许标记脱离K线。

Remark: 只有当您声明您的后端是支持标记时才会调用这个函数。upporting marks.

getServerTime(callback)

1. `callback` : function(unixTime)

当图表需要知道服务器时间时，如果配置标志 `supports_time` 设置为 `true` ，则调用此函数。图表库预期只调用一次回调。所提供的时间没有毫秒。例子：1445324591。它是用来显示倒数的价格范围。

交易终端专属

getQuotes(symbols, onDataCallback, onErrorCallback)

1. `symbols` : 商品名称数组
2. `onDataCallback` : function(array of `data`)
    1. `data` :商品报价数据

3. `onErrorCallback` : function(reason)

当图表需要报价数据时，将调用此函数。图表库预期在收到所有请求数据时调用onDataCallback。

💹 subscribeQuotes(symbols, fastSymbols, onRealtimeCallback, listenerGUID)

1. `symbols` : 很少更新的商品数组（建议频率为每分钟一次）。这些商品在观察列表中，但它们目前不可见。
2. `fastSymbols` : 频繁更新的商品数组（一次在10秒或更快）
3. `onRealtimeCallback` : function(array of `data`)
    1. `data` :商品报价数据
4. `listenerGUID` : 监听的唯一标识符

交易终端当需要接收商品的实时报价时调用此功能。图表预期您每次要更新报价时都会调用`onRealtimeCallback` 。

💹 unsubscribeQuotes(listenerGUID)

1. `listenerGUID` : 监听的唯一标识符

交易终端当不需要再接收商品的实时报价时调用此函数。当图表库遇到`listenerGUID` 相同的对象会跳过`subscribeQuotes` 方法。

💹 subscribeDepth(symbolInfo, callback): String

1. `symbolInfo` :SymbolInfo object
2. `callback` : function(depth)
    1. `depth` : object `{snapshot, asks, bids}`
        1. `snapshot` : Boolean - 如果 `true` 时 `asks` 和 `bids` 具有全部深度，否则只包含更新的级别。
        2. `asks` : 买盘数组 `{price, volume}`
        3. `bids` : 卖盘数组 `{price, volume}`

交易终端当要接收商品的实时level 2 信息（DOM）时，调用此函数。 图表预期您每次要更新深度数据时都会调用回调。

此方法应返回唯一标识（subscriberUID），用于取消订阅数据。

💹 unsubscribeDepth(subscriberUID)

1. `subscriberUID`: String

交易终端当不希望接收此监听时调用此函数。

3-3、UDF

UDF

这是啥?: Universal Data Feed 通用数据反馈，基于HTTP协议的旨在以简单有效的方式向图表库提供数据。 **我该怎么使用它?**: 您应该创建小型的HTTP服务，让它从您的数据库中获取数据并响应图表库请求。

表式响应概念

Datafeed 响应通常可以被视为表。例如，关于交易所的商品列表的响应，可以被视为每个商品代表一行，并有存在一些列（minimal_price_movement，description，has_intraday等）。每个列可以是一个数组（因此，它将为每个表的行提供单独的值）。但是当所有表的行具有相同的列值时，可能会出现这种情况：列的值可以是单独的JSON响应。

例如:

让我们假设我们已经请求了名称为NYSE(纽约证券交易所)的商品列表。响应（伪格式）可能像这样

```
{
    symbols: ["MSFT", "AAPL", "FB", "GOOG"],
    min_price_move: 0.1,
    description: ["Microsoft corp.", "Apple Inc", "Facebook", "Google"]
}
```

如果我们试图将这个回应设想成一张表，那么它就像这样

Symbol（商品代码）	min_price_move（最小价格变动）	描述
MSFT	0.1	Microsoft corp.
AAPL	0.1	Apple Inc

FB	0.1	Facebook
GOOG	0.1	Google

API调用

Datafeed 配置数据

Request: `GET /config`

Response: 图表库期望接收与JS API调用setup()相同结构的JSON数据。 此外，还应该有2个附加属性：:

- **supports_search**: 设置这一选项为 `true` 如果你的datafed 支持商品查询和人商品解析逻辑。
- **supports_group_request**: 设置这一选项为 `true` 如果您的datafeed只提供所有商品集合的完整信息，并且无法进行商品搜索或单个商品解析。

`supports_search` 和 `supports_group_request` 两者之中有只有一个可以为 `true` 。

Remark: 如果你的datafeed 没有实现这个调用(根本不响应或发送404)，将使用默认配置。这样：:

```
{
    supports_search: false,
    supports_group_request: true,
    supported_resolutions: ["1", "5", "15", "30", "60", "1D", "1W", "1M"],
    supports_marks: false,
    supports_time: true
}
```

商品集合信息

Request: `GET /symbol_info?group=<group_name>`

1. `group_name` : string

Example: `GET /symbol_info?group=NYSE`

Response: 预期响应是具有以下列出的属性的对象。 每个属性都被视为表的一列，如上所述（请参见表式响应）。响应结构与SymbolInfo类似（但不等于），因此有关所有字段的详细信息，请参见其描述。

- **symbol**
- **description**
- **exchange-listed/exchange-traded**
- **minmovement/minmov**（注意：minmov已被弃用，并将在未来的版本中被删除）
- **minmovement2/minmov2**（注意：minmov2已被弃用，并将在未来的版本中被删除）
- **fractional**
- **pricescale**
- **has-intraday**
- **has-no-volume**
- **type**
- **ticker**
- **timezone**
- **session-regular**(mapped to `SymbolInfo.session`)
- **supported-resolutions**
- **force-session-rebuild**
- **has-daily**
- **intraday-multipliers**
- **has-fractional-volume**(obsolete)
- **volume_precision**
- **has-weekly-and-monthly**
- **has-empty-bars**

示例：以下是对datafeed的响应示例 `GET /symbol_info?group=NYSE` (数据为手工制造):

```
{
    symbol: ["AAPL", "MSFT", "SPX"],
    description: ["Apple Inc", "Microsoft corp", "S&P 500 index"],
    exchange-listed: "NYSE",
    exchange-traded: "NYSE",
```

```
    minmov: 1,
    minmov2: 0,
    pricescale: [1, 1, 100],
    has-dwm: true,
    has-intraday: true,
    has-no-volume: [false, false, true]
    type: ["stock", "stock", "index"],
    ticker: ["AAPL~0", "MSFT~0", "$SPX500"],
    timezone: “America/New_York”,
    session-regular: “0900-1600”,
}
```

Remark 1: 如果您的datafeed配置supports_group_request：true或根本没有响应配置请求，则将使用此调用。

Remark 2: 如果您的datafeed 被请求不支持的集合（如果您对请求＃1（支持的集合）的响应是正确的），则会发生404错误）。

Remark 3: 使用此模式（获取大量的商品数据）在浏览器中存储一些用户不需要的数据。 因此，如果您的商品列表有多个项目，请考虑支持商品搜索/单个商品解析。

商品解析

Request: `GET /symbols?symbol=<symbol>`

1. `symbol` : string. 商品名称或者代码.

例: `GET /symbols?symbol=AAL` , `GET /symbols?symbol=NYSE:MSFT`

Response: JSON包含的对象与SymbolInfo完全一样

Remark: 如果您的datafeed配置supports_group_request：false 和 supports_search：true，则将执行此调用。

商品检索

Request: `GET /search?query=<query>&type=<type>&exchange=<exchange>&limit=<limit>`

1. `query` : string. 用户在商品搜索编辑框中输入的文本

2. `type` : string. 您的后台支持的类型之一
3. `exchange` : string. 您的后台支持的交易所之一
4. `limit` : integer. 响应最大项目数

例: `GET /search?query=AA&type=stock&exchange=NYSE&limit=15`

Response: 响应将是调用JS API后返回的一个数组类型的商品记录

Remark: 如果您的datafeed配置supports_group_request：false 和 supports_search：true，则将执行此调用。

K线柱

Request:

```
GET /history?symbol=<ticker_name>&from=<unix_timestamp>&to=
<unix_timestamp>&resolution=<resolution>
```

1. `symbol` : 商品名称或者代码
2. `from` : unix timestamp (UTC) or leftmost required bar
3. `to` : unix timestamp (UTC) or rightmost required bar
4. `resolution` : string

例: `GET /history?symbol=BEAM~0&resolution=D&from=1386493512&to=1395133512`

Response: 响应的预期是一个对象，下面列出了一些属性。每个属性都被视为表的列，如上所述。

- **s**: 状态码。 预期值: `ok` | `error` | `no_data`
- **errmsg**: 错误消息。只在 `s = 'error'时出现`
- **t**: K线时间. unix时间戳 (UTC)
- **c**: 收盘价
- **o**: 开盘价 (可选)
- **h**: 最高价 (可选)
- **l**: 最低价(可选)
- **v**: 成交量 (可选)
- **nextTime**: 下一个K线柱的时间 如果在请求期间无数据 (状态码为 `no_data`) (可选)

Remark: bar time 对于日K线柱预期为 一个交易日 (not session start day) 以 00:00 UTC为起点。 Charting Library 会根据SymbolInfo的Session时间进行匹配。

Remark: K线时间对于月K线柱为这个月的第一个交易日，除去时间的部分。

Remark: 价格应作为数字传递，而不是使用字符串。

例:

```
{
    s: "ok",
    t: [1386493512, 1386493572, 1386493632, 1386493692],
    c: [42.1, 43.4, 44.3, 42.8]
}
```

```
{
    s: "no_data",
    nextTime: 1386493512
}
```

```
{
    s: "ok",
    t: [1386493512, 1386493572, 1386493632, 1386493692],
    c: [42.1, 43.4, 44.3, 42.8],
    o: [41.0, 42.9, 43.7, 44.5],
    h: [43.0, 44.1, 44.8, 44.5],
    l: [40.4, 42.1, 42.8, 42.3],
    v: [12000, 18500, 24000, 45000]
}
```

nextTime是怎么工作的

假设您以周期= 1观看图表，并且Library要求您以[2015年4月3日16:00 UTC + 0，2015年4月3日19:00 UTC + 0]为范围，向纽约证券交易所请求股票数据。 4月3日为受难节，交易所休假。 Library假定你会作出如下响应:

```
{
    s: "no_data",
    nextTime: 1428001140000 //2015年4月2日 18:59:00 GMT+0
}
```

因此 `nextTime` 是一个从Library的原始请求边界的左侧（在假想时间线上）的K线柱时间。

所有省略的价格将被视为等于收盘价。

标识

Request:

```
GET /marks?symbol=<ticker_name>&from=<unix_timestamp>&to=
<unix_timestamp>&resolution=<resolution>
```

1. `symbol` : symbol name or ticker.
2. `from` : unix timestamp (UTC) or leftmost visible bar
3. `to` : unix timestamp (UTC) or rightmost visible bar
4. `resolution` : string

Response: 响应预期是一个对象，下面列出了一些属性。此对象与JS API中的respective response相似，但每个属性都被视为表的列，如上所述。

```
{
    id: [array of ids],
    time: [array of times],
    color: [array of colors],
    text: [array of texts],
    label: [array of labels],
    labelFontColor: [array of label font colors],
    minSize: [array of minSizes],
}
```

Remark: 备注：如果您的datafeed在传输的配置数据中发送了supports_marks：true，则会调用此方法。

时间刻度标记

Request:

```
GET /timescale_marks?symbol=<ticker_name>&from=<unix_timestamp>&to=
<unix_timestamp>&resolution=<resolution>
```

1. `symbol` : symbol name or ticker.
2. `from` : unix timestamp (UTC) or leftmost visible bar
3. `to` : unix timestamp (UTC) or rightmost visible bar
4. `resolution` : string

Response: 响应预期为一个具有下列属性的数组对象。

1. `id` : unique identifier of a mark
2. `color` : rgba color
3. `label` : 显示在圆圈中的文字
4. `time` : unix time
5. `tooltip` : 提示文本

Remark: This call will be requested if your datafeed sent `supports_timescale_marks: true` in configuration data.

服务器时间

Request: `GET /time`

Response: 数字unix时间没有毫秒。 例: 1445324591

报价

Request: `GET /quotes?symbols=<ticker_name_1>,<ticker_name_2>,...,<ticker_name_n>`

Example: `GET /quotes?symbols=NYSE%3AAA%2CNYSE%3AF%2CNasdaqNM%3AAAAPL`

Response: Response is an object.

- **s**: status code for request. Expected values: `ok` | `error`

- **errmsg**: error message for client
- **d**:symbols data array

Example:

```
{
    "s": "ok",
    "d": [{
        "s": "ok",
        "n": "NYSE:AA",
        "v": {
            "ch": "+0.16",
            "chp": "0.98",
            "short_name": "AA",
            "exchange": "NYSE",
            "description": "Alcoa Inc. Common",
            "lp": "16.57",
            "ask": "16.58",
            "bid": "16.57",
            "open_price": "16.25",
            "high_price": "16.60",
            "low_price": "16.25",
            "prev_close_price": "16.41",
            "volume": "4029041"
        }
    }, {
        "s": "ok",
        "n": "NYSE:F",
        "v": {
            "ch": "+0.15",
            "chp": "0.89",
            "short_name": "F",
            "exchange": "NYSE",
            "description": "Ford Motor Compan",
            "lp": "17.02",
            "ask": "17.03",
            "bid": "17.02",
            "open_price": "16.74",
            "high_price": "17.08",
            "low_price": "16.74",
            "prev_close_price": "16.87",
            "volume": "7713782"
        }
    }]
}
```

构造函数

```
Datafeeds.UDFCompatibleDatafeed = function(datafeedURL, updateFrequency,
protocolVersion)
```

datafeedURL

这是一个数据服务器的URL，它将得到请求和返回数据。

updateFrequency（更新频率）

这是一个有周期的实时数据请求，datafeed将以毫秒为单位发送到服务器。 默认值为10000（10秒）。

3-4、Symbology

图表库使用您自己的数据，您自己定义商品体系。 您可以根据需要命名商品。

但有些细节你应该知道:

1. 我们自己定义的商品名称必须为此格式： `EXCHANGE:SYMBOL` 。 图表默认使用此格式
2. 如果已经有或者考虑区别不同的商品，那么你可能需要使用 `ticker` 字段。
   `ticker` 为商品唯一标识符，只用于图表的内部，您的用户将不会看到它。
   `ticker` 为 **仅在** 图表库内部使用的商品唯一标识符。 您的用户将永远无法看到它。
   只需在所有SymbolInfo对象和商品搜索结果中输入 `ticker` 值，图表库将根据这些值请求数据。

商品信息结构

这一节非常重要。 图表库用户遇到的72.2％的问题，都是由于错误的/格式错误的SymbolInfo数据引起的。

SymbolInfo是一个包含商品metadata的对象。 该对象是解析商品的结果。 SymbolInfo有以下字段：

name

商品名称。您的用户将看到它(作为一个字符串)。 此外，如果您不使用 `tickers` ，它将用于数据请求。

ticker

它是您的商品体系中此商品的唯一标识符。如果您指定此属性，则其值将用于所有数据请求， `ticker` 如果未明确指定，则被视为等于 `symbol` 。(译者注：请一定指定 `ticker` ，如果没有 `ticker` 可以将 `symbol` 赋值给 `ticker` ，未指定 `ticker` 时会发生错误。)

description

商品说明。这个商品说明将被打印在图表的标题栏中。

type

仪表的可选类型。

可能的类型是:

- `stock`
- `index`
- `forex`
- `futures`
- `bitcoin`
- `expression`
- `spread`
- `cfd`
- 或其他字符串。

session

商品交易时间。请参阅交易日细节了解更多详情。交易时段

exchange, listed_exchange

现在，这两个字段都为某个交易所的略称。将被显示在图表的图例中，以表示此商品。目前此字段不用于其他目的。

timezone

这个商品的交易所时区。我们希望以olsondb格式获取时区的名称。

支持的时区为:

- `Etc/UTC`
- `Africa/Cairo`

- `Africa/Johannesburg`
- `Africa/Lagos`
- `America/Argentina/Buenos_Aires`
- `America/Bogota`
- `America/Caracas`
- `America/Chicago`
- `America/El_Salvador`
- `America/Juneau`
- `America/Lima`
- `America/Los_Angeles`
- `America/Mexico_City`
- `America/New_York`
- `America/Phoenix`
- `America/Santiago`
- `America/Sao_Paulo`
- `America/Toronto`
- `America/Vancouver`
- `Asia/Almaty`
- `Asia/Ashkhabad`
- `Asia/Bahrain`
- `Asia/Bangkok`
- `Asia/Chongqing`
- `Asia/Dubai`
- `Asia/Ho_Chi_Minh`
- `Asia/Hong_Kong`
- `Asia/Jakarta`
- `Asia/Jerusalem`
- `Asia/Kathmandu`
- `Asia/Kolkata`
- `Asia/Kuwait`
- `Asia/Muscat`
- `Asia/Qatar`
- `Asia/Riyadh`
- `Asia/Seoul`

- `Asia/Shanghai`
- `Asia/Singapore`
- `Asia/Taipei`
- `Asia/Tehran`
- `Asia/Tokyo`
- `Atlantic/Reykjavik`
- `Australia/ACT`
- `Australia/Adelaide`
- `Australia/Brisbane`
- `Australia/Sydney`
- `Europe/Athens`
- `Europe/Belgrade`
- `Europe/Berlin`
- `Europe/Copenhagen`
- `Europe/Helsinki`
- `Europe/Istanbul`
- `Europe/London`
- `Europe/Luxembourg`
- `Europe/Madrid`
- `Europe/Moscow`
- `Europe/Paris`
- `Europe/Riga`
- `Europe/Rome`
- `Europe/Stockholm`
- `Europe/Tallinn`
- `Europe/Vilnius`
- `Europe/Warsaw`
- `Europe/Zurich`
- `Pacific/Auckland`
- `Pacific/Chatham`
- `Pacific/Fakaofo`
- `Pacific/Honolulu`
- `Pacific/Norfolk`
- `US/Mountain`

format

在价格刻度上显示标签的格式：

- `price` - 根据 `minmov`，`pricescale`，`minmove2` 和 `fractional` 的值，格式化价格的小数或分数
- `volume` - 将十进制数字格式化为千，百万或十亿

minmov(最小波动), pricescale(价格精度), minmove2, fractional(分数)

1. 最小的价格变化是由这些值决定的。
2. PriceScale 参数确定了图表价格量表上的价格线之间的周期。

这三个键有不同意义时，使用通常价格和分数价格。

通常价格

```
MinimalPossiblePriceChange（最小可能价格变动） = minmov / pricescale
```

`minmov` 数字型单位组成一个tick。例如，美国股票价格和tick有小数，并可以上下浮动+/-0.01。

分数价格

分数显示价格,1 - xx'yy（例如，133'21)或 2 - xx'yy'zz （例如，133'21'5）。

minmove2<0>

这是一个神奇的数字来格式化复杂情况下的价格。这里有一些例子:

```
1 典型的股票以0.01价格增量：minmov = 1，pricecale = 100，minmove2 = 0
2 ZBM2014（国债），1/32：minmov = 1，pricecale = 32，minmove2 = 0
3 ZCM2014（玉米），2/8：minmov = 2，pricecale = 8，minmove2 = 0
```

```
ZFM2014（5年期国债），1/32的1/4：minmov = 1，pricecale = 128，minmove2 = 4
```

has_intraday

布尔值显示商品是否具有日内（分钟）历史数据。 如果它为 `false` ，则当图表中的该商品处于活动状态时，日内周期的所有按钮将被禁用。 如果设置为 `true` ，则由datafeed直接提供的所有周期必须在intraday_multipliers数组中设定。

supported_resolutions

在这个商品的周期选择器中启用一个周期数组。 数组的每个项目都是字符串。

被datafeed支持（见datafeed配置数据）但不受当前商品支持的周期,将在周期选择器部件中禁用。 如果更改商品，新商品不支持选定的周期，则周期将切换到支持的周期列表中的第一项。 周期可用性逻辑（伪代码）：

```
resolutionAvailable  =
    resolution.isIntraday ?
       symbol.has_intraday && symbol.supports_resoluiton(resolution) :
    symbol.supports_resoluiton(resolution);
```

如果在商品信息中没有supported_resolutionsin，则所有DWM(daily, weekly, monthly)周期都可用。 如果has_intraday为true，则日内周期可用。

支持的周期也会影响可用的时间范围。 如果使用不支持的周期，则时间范围将不可用。

intraday_multipliers <[]>

这是一个包含日内周期(分钟单位)的数组，datafeed将会自行构建它。

举例来说：如果datafeed报告说它支持 ["1", "5", "15"]，但事实上股票X只有1分钟的数据，股票X将设定 intraday_multipliers = [1]，那么Charting Library将自行构建5分钟和15分钟的周期。

has_seconds

布尔值显示商品是否具有以秒为单位的历史数据。如果它为 `false` ，那么在图表中此商品处于活动状态时，所有秒的周期的按钮将被禁用。如果它为 `true` ，则由datafeed直接提供的所有周期必须在 `seconds_multipliers` 数组中设定。

seconds_multipliers <[]>

这是一个包含秒周期(以秒为单位，无小数) ，datafeed将会自行构建它。
举例来说：如果datafeed报告说它支持 ["1S", "5S", "15S"]，但事实上股票X只有1秒钟的数据，股票X将设定 seconds_multipliers = [1]，那么Charting Library将自行构建5S和15S的周期。

has_daily

布尔值显示商品是否具有以日为单位的历史数据。如果它为false，则Charting Library将自行构建日单位的周期。如果没有，则会向datafeed请求这些数据。

has_weekly_and_monthly

布尔值显示商品是否具有以W和M为单位的历史数据。如果它为false，则Charting Library将通过日单位的周期自行构建。如果没有，则会向datafeed请求这些数据。

has_empty_bars

布尔值显示在交易过程中，当datafeed没有数据返回时,library是否会生成空的K柱。

即，如果您的交易时间为0900-1600，而您的实际数据在11:00和12:00之间没有交易，而您的has_empty_bars为true，那么Library会在此段时间贴上退化的K柱。

force_session_rebuild

布尔值显示library是否会随着当前交易而过滤K柱。如果为false，则当library从其他周期构建数据或将has_empty_bars设置为true时，K柱将被过滤。 如果为true，Library将会删除那些不是交易K柱的数据。

has_no_volume

布尔表示商品是否拥有成交量数据。

has_fractional_volume | 已过时(1.1 - 1.5), 用法与volume_precision相反

如果has_fractional_volume = true，成交量指标值将不会舍入为整数值。

volume_precision <0>

整数显示此商品的成交量数字的小数位。0表示只显示整数。1表示保留小数位的1个数字字符，等等。

data_status

数据状态码。状态显示在图表的右上角。

支持的值:

- `streaming` (实时)
- `endofday` (已收盘)
- `pulsed` (脉冲)
- `delayed_streaming` (延迟流动中)

expired

期满，布尔值显示此商品是否为到期的期货合约。

expiration_date

到期日(Unix时间戳)。 如果 `expired` = `true` ，则必须设置此值。 图表库将从该时间点而不是实际时刻请求该商品的数据。

sector

板块，将在股票信息中显示。

industry

行业，将在股票信息中显示。

currency_code

货币代码，将在商品信息中显示。

3-5、交易时段

图表库期望在商品信息中获取交易时段。 交易时段是可交易的时间范围。每个交易时段都应该有左右边界。在图表库中，交易时段的格式为“HHMM-HHMM”。例如，交易时段从上午9:30到下午16:00应该表示为 `0930-1600` 。
有一个特殊情况的商品交易7*24小时(例如：比特币或其它数字货币)。交易时段的字符串应该为 `24x7` 。交易时段将会发生在交易时区。

如果交易时段左边界大于右边边界（例 `1700-0900` ），则此交易时段被视为隔夜。隔夜交易始终在前一天开始：例如，如果商品在星期一至星期五 `1700-0900` 交易，则这周（#i）第一交易时段 = 前一周（＃i-1）星期日17:00开始,到本周（#i）星期一09:00结束。

每个交易日可能会有多个交易时段。 如果有多个交易时段，您应该将整个交易时段分为以逗号分隔的多个交易时段。 例如，假设当天的交易时间为9:30至14:00，然后为14:30至17:00，交易时段应为 `0930-1400,1430-1700` .

此外，交易时间可能会有所差异。 这是您可以使用 `:` 特殊的说明符。例如，如果商品全部时间都为0900-1630，但星期一比较特殊（交易时段为0900-1400），这交易时段应为 `0900-1630|0900-1400:2` . 让我们看看这个字符串的细节。

片段	含义
0900-1630	交易时段为0900-1630。默认情况下，此会话将分配给所有非周末日，因为它后面没有 `:` 说明符。
\	交易时段分隔符。负责分隔不同的交易时段。
0900-1400	交易时段为0900-1400。这是一天的交易时段（见下文）。
:	日期说明符。该字符在少时说明符后，后跟日期号码。
2	上述交易时段的日期号码 (0900-1400)

日期号码：**星期日**为1，**星期六**为7（2–星期一，3–星期二，等等）。

可以覆盖一个或多个日期。 例如，在 `0900-1630 | 0900-1400：23` 中，0900-1400交易时段将被分配到第2天和第3天（星期一，星期二）。

version: 1.1:

可以使用分号指定一周的第一个交易日。例：

- `1;0900-1630|0900-1400:2` ：每周的第一天是星期日
- `0900-1630|0900-1400:2;6` ：每周的第一天是星期五
- `0900-1630|0900-1400:2` ：每周的第一天是星期一（默认值）

Remark: 默认情况下，所有非24x7商品在星期六和星期日被视为不可交易。所以如果您的商品在周末交易，您应该明确指定交易日。 例如：某个商品 `1000-1600` 星期日到星期五交易，则交易时段应写为 `1000-1600:123456`

使用此解析器检查交易时段字符串：http://tradingview.github.io/checksession.html

3-6、报价

报价

报价是简要描述交易的数据集。图表库支持观察列表（在 **交易终端**配置中），并使用报价来显示相关的商品信息。

图表库中不管是JS API还是UDF都使用相同的数据结构来进行报价。

以下是响应返回对象的描述：

商品报价数据

- `s`：商品的状态码。预期的值为: `ok` | `error`
- `n`：商品名称。 此值必须与请求中 **完全相同**
- `v`：object, 商品报价对象
  - `ch`：价格变动（通常从当天的开盘价计算）
  - `chp`：价格变动百分比
  - `short_name`：商品略称
  - `exchange`：交易所名称
  - `description`：商品的简短描述
  - `lp`：最后的成交价格
  - `ask`：买盘价
  - `bid`：卖盘价
  - `spread`：费率
  - `open_price`：当天开盘价
  - `high_price`：当天最高价
  - `low_price`：当天最低价
  - `prev_close_price`：昨天收盘价
  - `volume`：当天成交量

4、图表定制

4-1、定制概述

定制概述

定制是一个广泛的概念，这就是为什么我们有几篇关于它的文章。

通过数据流定制

这些主要是与数据相关的自定义。它们使用datafeed配置响应。

以下是配置响应的示例。

```
{
    supports_search: true,
    supports_group_request: false,
    supports_marks: true,
    exchanges: [
        {value: "", name: "All Exchanges", desc: ""},
        {value: "XETRA", name: "XETRA", desc: "XETRA"},
        {value: "NSE", name: "NSE", desc: "NSE"}
    ],
    symbolsTypes: [
        {name: "All types", value: ""},
        {name: "Stock", value: "stock"},
        {name: "Index", value: "index"}
    ],
    supportedResolutions: [ "1", "15", "30", "60", "D", "2D", "3D", "W", "
};
```

在JS API可以找到更详细的说明。

在客户端进行定制

允许您最大化的定制UI/UX。这些定制通过定义图表控件中的构造函数的参数完成。

图表控件构造函数调用的示例：

```
```

```
var widget = new TradingView.widget({
    fullscreen: true,
    symbol: 'AA',
    interval: 'D',
    toolbar_bg: '#f4f7f9',
    allow_symbol_change: true,
    container_id: "tv_chart_container",
    datafeed: new Datafeeds.UDFCompatibleDatafeed("http://demo_feed.tradin
    library_path: "charting_library/",
    locale: "en",
    drawings_access: { type: 'black', tools: [ { name: "Regression Trend"
    disabled_features: ["use_localstorage_for_settings", "volume_force_ove
    enabled_features: ["move_logo_to_main_pane"],
    overrides: {
        "mainSeriesProperties.style": 0,
        "symbolWatermarkProperties.color" : "#944",
        "volumePaneSize": "tiny"
    },
    studies_overrides: {
        "bollinger bands.median.color": "#33FF88",
        "bollinger bands.upper.linewidth": 7
    },
    debug: true,
    time_frames: [
        { text: "50y", resolution: "6M" },
        { text: "1d", resolution: "5" },
    ],
    charts_storage_url: 'http://saveload.tradingview.com',
    client_id: 'tradingview.com',
    user_id: 'public_user',
    favorites: {
        intervals: ["1D", "3D", "3W", "W", "M"],
        chartTypes: ["Area", "Line"]
    }
});
```

详情参考：Widget构造器

也可以看看

- Widget方法
- 定制的使用案例

4-2、Widget 构造器

Widget构造器

当调用构造函数时，您可以定义图表库widget的参数。例：

```
new TradingView.widget({
    symbol: 'A',
    interval: 'D',
    timezone: "America/New_York",
    container_id: "tv_chart_container",
    locale: "ru",
    datafeed: new Datafeeds.UDFCompatibleDatafeed("https://demo_feed.tradi
});
```

查看下列完整支持的参数列表。请记住，在图表初始化后在更改这些参数是不起作用的。如果要在初始化图表之后更改图表的状态，请使用widget方法。

属性标记为 💹 的只在交易终端可用。

symbol, interval

您的图表的初始商品和周期。 `interval` 的格式在另一篇文章中说明。 *必须项*

timeframe

设置图表的初始时间范围。时间范围是加载并显示在屏幕上的K线范围。有效的时间范围是一个数字加一个字母，D为数天，M为数月。

container_id

`id` 属性为指定要包含widget的DOM元素id。 *必须项*

datafeed

JavaScript对象的实现接口 JS API 以反馈数据显示在图表上。 *必须项*

timezone

图表的初始时区。时间刻度上的数字取决于这个时区。请参阅支持的时区列表。设置为交易所时区。覆盖默认值，您应该使用覆盖章节。

debug

将此属性设置为 `true` 时，可使图表将详细的API日志写入控制台。与功能集的 `charting_library_debug_mode` 用法相同。

library_path

`static` 文件夹的路径

width, height

widget的尺寸，请确保widget拥有足够的空间。

Remark: 如果您想让图表占据所有可用的空间，请不要使用 `100%` 这样的字段。使用 `fullscreen` 参数来代替（见下文）。这是因为DOM节点在不同浏览器中有调整大小的问题。

fullscreen

默认值: `false`

布尔值显示图表是否占用窗口中所有可用的空间。

autosize

默认值: `false`

布尔值，显示图表是否应使用窗格中的所有可用空间，并在调整窗格本身大小时自动调整大小。

symbol_search_request_delay

延迟阈值（以毫秒为单位），用于在用户在搜索框中键入商品名称时减少商品搜索请求的数量。

auto_save_delay

延迟秒数等待 `onAutoSaveNeeded` 可以被再次调用。 该参数介绍在1.5版本中。

toolbar_bg

工具栏背景颜色

study_count_limit

自1.5版本起。

多图布局图表的最大指标数量。 最小值为2。

studies_access

版本：1.1具有以下结构的对象：

```
{
    type: "black" | "white",
    tools: [
        {
            name: "<study name>",
            grayed: true
        },
        < ... >
    ]
}
```

- `type` 是列表类型。 支持的值: `black` (所有列出的项目会被禁用), `white` (只有列出的项目会被启用)。
- `tools` 对象数组。每个对象可以具有以下属性：
  - `name` (强制的) 指标的名称。使用相同的名称，你可以看到他们在指标控件。
  - `grayed` 布尔值，表明这项指标将可见，但看起来像是被禁用的。 如果指标为 `grayed` ，当用户点击它时，会调用 `onGrayedObjectClicked` 回调方法。

drawings_access

版本：1.1 该属性与上述的 `studies_access` 具有相同的结构。 使用与您在UI中看到的名称相同的名称。

Remark: 基于字体的绘图有一个特殊情况。 使用 `Font Icons` 的名字时， 这个组是一个特例，它的绘图不能被启用或禁用 - 可以启用或禁用整个组。

saved_data

JS对象包含保存的图表内容（JSON，请参阅下面的保存/加载调用）。如果在创建图表时已经有图表的JSON，请使用此参数。如果要将图表内容加载到已初始化的图表中，请使用 `loadData()` 控件方法。

locale

图表库的本地化处理 。详情：本地化

numeric_formatting

该对象包含数字的格式化选项。目前唯一可能的选择是 `decimal_sign` 。例: `numeric_formatting: { decimal_sign: "," }`

customFormatters

它是一个包含以下字段的对象：

1. timeFormatter
2. dateFormatter

您可以使用这些格式化方法自定义显示日期和时间的值。这两个值都是具有方法 `format` 和 `formatLocal` 的对象:

```
function format(date)
function formatLocal(date)
```

这些函数返回表示date或time的文本。 `formatLocal` 将日期和时间转换为本地时区。

例:

```
customFormatters: {
  timeFormatter: {
    format: function(date) { var _format_str = '%h:%m'; return _format_str
  },
  dateFormatter: {
    format: function(date) { return date.getUTCFullYear() + '/' + date.get
  }
}
```

overrides

对Widget对象的默认属性进行覆盖。 覆盖属性意味着为其分配默认值。 您可以覆盖大部分图表的属性（也可以由用户通过UI编辑）使用 `overrides` 参数构造控件 。 `overrides` 应该是一个具有范围的对象。每个字段名是重写属性的名称，字段值是这些属性的期望值。例子:

```
overrides: {
    "mainSeriesProperties.style": 0
}
```

这个 `override` 将使水印100％不透明（不可见）。 所有可定制的属性都列在单独的文章中。 从1.5开始，您可以使用绘图覆盖。绘图覆盖.

disabled_features, enabled_features

包含功能在默认情况下启用/禁用名称的数组。功能表示图表功能的一部分（更是UI/UX的一部分）。 这里. 此处列出了支持的功能。 例：

```
TradingView.onready(function()
{
    var widget = new TradingView.widget({
        /* .... */
        disabled_features: ["header_widget", "left_toolbar"],
```

```
        enabled_features: ["move_logo_to_main_pane"]
    });
});
```

snapshot_url

当用户按快照按钮时,使用base64编码将当前图表快照保存并返回URL。该服务返回完整的保存图像URL。

custom_indicators_getter

返回带有自定义指标数组的Promise对象的函数。 `PineJS` 变量将作为此函数的第一个参数传递，并可在指标内用于访问内部帮助函数。 查看更多细节。

preset

`preset` 是一组预定义widget设置的名称。现在预设中只支持 `mobile` 。此预设的示例可在线获取。

studies_overrides

使用此选项自定义默认指标的样式及输入值。 您还可以使用此参数自定义 `Compare` 数据列的样式和输入值。 查看更多

time_frames

在图表底部的时间范围选择器中可以看见这个时间范围列表。 例:

```
time_frames: [
    { text: "50y", resolution: "6M", description: "50 Years" },
    { text: "3y", resolution: "W", description: "3 Years", title: "3yr" },
    { text: "8m", resolution: "D", description: "8 Month" },
    { text: "3d", resolution: "5", description: "3 Days" },
    { text: "1000y", resolution: "W", description: "All", title: "All" },
]
```

time_frames是一个包含 `text` 和 `resolution` 属性的对象。 `text` 属性应具有以下格式： `<integer> <y | m | d>` (\d+(y|m|d) 为正则表达式)。 resolution是一个字符串,格式说明。

属性 `description` 在v1.7开始支持，显示在弹出菜单中。 此参数是可选的。如果未指定，则使用 `title` 或 `text` 属性作为描述。

属性 `title` 在v1.9开始支持，其值将覆盖基于 `text` 属性生成的默认标题。 此参数是可选的。

charts_storage_url, client_id, user_id

这些参数与用于保存/加载的高级API相关。 查看更多细节.

charts_storage_api_version

您的后台版本。支持的值: `"1.0"` | `"1.1"` 。 指标模板从 `1.1` 开始得到支持。

load_last_chart

如果您希望图表库为用户加载上次保存的图表，请将此参数设置为 `true` (您应首先实现save/load以使其工作)。

theme

从1.13版开始支持。

为图表添加自定义主题颜色。 支持的值是: `"Light"` | `"Dark"` .

custom_css_url

从1.4版开始支持。

将您的自定义CSS添加到图表中。url应该是到 `static` 文件夹的绝对或相对路径。

loading_screen

从1.12版开始支持。

定制加载进度条。值是具有以下可能 `key` 的对象。

- `backgroundColor`
- `foregroundColor`

例如:

```
loading_screen: { backgroundColor: "#000000" }
```

favorites

默认标记为收藏的项目。此选项使用时要求禁用localstorage（请参阅功能集以了解更多)。 `favorites` 属性为一个对象，拥有以下属性：

- **intervals(周期)**: 收藏的周期数组。 例： `["D", "2D"]`
- **chartTypes(图表类型)**: 收藏的图表类型数组 。图表类型名称与图表的UI中的英文版本相同。 例: `["Area", "Candles"]`

save_load_adapter

从1.12版开始支持。

包含保存/加载功能的对象。 如果设置了，应有以下方法：

Chart layouts

1. `getAllCharts(): Promise<ChartMetaInfo[]>`
   获取所有保存的图表。
   `ChartMetaInfo` 具有以下字段的对象:
   - `id` - 图表id
   - `name` - 图表名
   - `symbol` - 图表的商品
   - `resolution` - 周期
   - `timestamp` - 最后修改日期（从 `01/01/1970` 0时开始的毫秒单位UTC时间）。
2. `removeChart(chartId): Promise<void>`

删除图表。 `chartId` 是图表的唯一ID（参见上面的 `getAllCharts` ）。

3. `saveChart(chartData: ChartData): Promise<ChartId>`
   存储图表。
   `ChartData` 具有以下字段的对象:
   - `id` - 图表的唯一标识（如果未保存则可能是 `undefined` ）。
   - `name` - 图表名
   - `symbol` - 图表的商品
   - `resolution` - 周期
   - `content` - 图表的内容

   `ChartId` - 图表唯一id (string)
4. `getChartContent(chartId): Promise<ChartContent>`
   通过服务器加载图表
   `ChartContent` 带有图表内容的字符串（参见 `saveChart` 函数中的 `ChartData::content` 字段）。

Study Templates

1. `getAllStudyTemplates(): Promise<StudyTemplateMetaInfo[]>`
   获取所有保存的指标模板。
   `StudyTemplateMetaInfo` 具有以下字段的对象:
   - `name` - 指标模板名称
2. `removeStudyTemplate(studyTemplateInfo: StudyTemplateMetaInfo): Promise<void>`
   删除指标模板
3. `saveStudyTemplate(studyTemplateData: StudyTemplateData): Promise<void>`
   存储指标模板
   `StudyTemplateData` 具有以下字段的对象:
   - `name` - 指标模板名称
   - `content` - 指标模板的内容
4. `getStudyTemplateContent(studyTemplateInfo: StudyTemplateMetaInfo): Promise<StudyTemplateContent>`
   通过服务器加载指标模板
   `StudyTemplateContent` - 指标模板的内容 (string)
   如果同时设置了 `charts_storage_url` 和 `save_load_adapter` , 将使用 `save_load_adapter`

重要： 所有函数都会返回 `Promise` （或 `Promise` 类对象）。

settings_adapter

从1.11版开始支持。

包含设置/删除方法的对象。 使用它将图表设置保存到您想要存储的地方，包括服务器端。 如果设置了，应该有以下方法：

1. `initialSettings: Object` 初始化设置
2. `setValue(key: string, value: string): void` 存储键/值对
3. `removeValue(key: string): void` 删除键

交易终端专属

widgetbar

💹 *仅适用于交易终端*

含图表右侧窗口小部件面板设置的对象。可以使用Widget构造函数中的“widgetbar”字段启用图表右侧的监视列表，新闻和详细信息窗口小部件：

```
widgetbar: {
    details: true,
    watchlist: true,
    watchlist_settings: {
        default_symbols: ["NYSE:AA", "NYSE:AAL", "NASDAQ:AAPL"],
        readonly: false
    }
}
```

- `details` (*default:* `false`): 在右侧的小部件面板中启用详细信息小部件。
- `watchlist` (*default:* `false`): 在右侧的小部件面板中启用观察列表小部件。
- `watchlist_settings.default_symbols` (*default:* `[]`): 设置监视列表的默认商品列表。
- `watchlist_settings.readonly` (*default:* `false`): 为监视列表启用只读模式。

rss_news_feed

仅适用于交易终端

使用此属性更改新闻的RSS源。 您可以为每个商品类型设置不同的RSS。或为所以商品使用同一个RSS。该对象将拥有 `default` 属性，其他属性是可选的; 属性的名称与商品类型匹配。每个属性都是具有以下属性的对象(或对象数组):

1. `url` 请求的URL。 它可以包含以下花括号中的标签(将会被终端所更改): `{SYMBOL}`, `{TYPE}`, `{EXCHANGE}`。
2. `name` 要在新闻下面显示的Feed名称。

例:

```
{
    "default": [ {
        url: "https://articlefeeds.nasdaq.com/nasdaq/symbols?symbol={SYMBO
        name: "NASDAQ"
      }, {
        url: "http://feeds.finance.yahoo.com/rss/2.0/headline?s={SYMBOL}&r
        name: "Yahoo Finance"
      } ]
}
```

另一个例子:

```
{
    "default": {
        url: "https://articlefeeds.nasdaq.com/nasdaq/symbols?symbol={SYMBO
        name: "NASDAQ"
    }
}
```

更多例子:

```
{
    "default": {
        url: "https://articlefeeds.nasdaq.com/nasdaq/symbols?symbol={SYMBO
        name: "NASDAQ"
     },
    "stock": {
        url: "http://feeds.finance.yahoo.com/rss/2.0/headline?s={SYMBOL}&r
        name: "Yahoo Finance"
    }
}
```

news_provider

✅ *仅适用于交易终端*

指定新闻提供者的对象。 它可能包含以下属性：

1. `is_news_generic` - 如果为 `true` ，则新闻小部件的标题将不包含商品名称（仅包含“标题”）。 否则将添加 `for SYMBOL_NAME` 。
2. `get_news` - 使用此属性设置自己的新闻getter方法。 `symbol` 和 `callback` 都将传递给函数。
   回调函数被调用时，会传递以下结构的新闻对象:
   1. `title` (必须) - 新闻标题。
   2. `published` (必须) - 新闻时间（以毫秒为单位的UTC时间）
   3. `source` (可选) - 标题的新闻来源。
   4. `shortDescription` (可选) - 将在标题下显示的新闻项目的简短描述。
   5. `link` (可选) - 新闻报道的URL
   6. `fullDescription` (可选) - 新闻项目的完整描述（正文）

   注意:当用户点击新闻项目时，将打开带有 `link` URL的新标签页。 如果没有指定 `link` ，将显示带有 `fullDescription` 的对话框弹出窗口。
   注意2:如果 `news_provider` 和 `rss_news_feed` 都可用，那么 `rss_news_feed` 将被忽略。

例:

```
news_provider: {
```

```
    is_news_generic: true,
    get_news: function(symbol, callback) {
        callback([
            {
                title: 'News for symbol ' + symbol,
                shortDescription: 'Short description of the news item',
                fullDescription: 'Full description of the news item',
                published: new Date().valueOf(),
                source: 'My own source of news',
                link: 'https://www.tradingview.com/'
            },
            {
                title: 'Another news for symbol ' + symbol,
                shortDescription: 'Short description of the news item',
                fullDescription: 'Full description of the news item. Long
                published: new Date().valueOf(),
                source: 'My own source of news',
            }
        ]);
    }
}
```

brokerFactory

仅适用于交易终端

使用这个字段来传递构造经纪商API的实现类。 这是一个接收交易主机并返回经纪商API的方法。

brokerConfig

仅适用于交易终端

`brokerConfig: { configFlags: {...} }` 使用此字段设置交易终端的配置标志。 了解更多。

也可以看看

- 定制概述

- Widget方法
- 功能集
- 存储于加载图表
- 覆盖默认指标参数
- 覆盖默认图表参数

4-3、Widget 方法

Widget方法

以下是widget支持的方法列表。您可以使用widget构造函数返回给您的widget对象来调用它们。

Remark: 请注意，只有在onChartReady回调触发后才可以调用这些方法。

```
widget.onChartReady(function() {
    // 现在可以调用其他widget的方法了
});
```

Methods

- 订阅图表事件
  - onChartReady(callback)
  - headerReady()
  - onGrayedObjectClicked(callback)
  - onShortcut(shortcut, callback)
  - subscribe(event, callback)
  - unsubscribe(event, callback)
- 图表动作
  - chart()
  - setLanguage(locale)
  - setSymbol(symbol, interval, callback)
  - remove()
  - closePopupsAndDialogs()
  - selectLineTool(drawingId)
  - selectedLineTool()
  - takeScreenshot()
  - lockAllDrawingTools()
  - hideAllDrawingTools()

    - magnetEnabled
    - magnetMode
- 保存/加载图表
    - save(callback)
    - load(state)
    - getSavedCharts(callback)
    - loadChartFromServer(chartRecord)
    - saveChartToServer(onCompleteCallback, onFailCallback, saveAsSnapshot, options)
    - removeChartFromServer(chartId, onCompleteCallback)
- 自定义UI控件
    - onContextMenu(callback)
    - createButton(options)
- 对话框
    - showNoticeDialog(params)
    - showConfirmDialog(params)
    - showLoadChartDialog()
    - showSaveAsChartDialog()
- Getters
    - symbolInterval(callback)
    - mainSeriesPriceFormatter()
    - getIntervals()
    - getStudiesList()
    - undoRedoState()
- 定制
    - changeTheme(themeName)
    - addCustomCSSFile(url)
    - applyOverrides(overrides)
    - applyStudiesOverrides(overrides)
- 💹 交易终端专属
    - watchList()
- 💹 多图表布局
    - chart(index)
    - activeChart()
    - chartsCount()
    - layout()
    - setLayout(layout)

- layoutName()

订阅图表事件

onChartReady(callback)

1. `callback` : function()

当图表初始化并准备就绪时，图表库将调用提供的回调。 你可以从这一刻安全地调用所有其他方法。

headerReady()

返回一个 `Promise` 对象，该对象应该在图表库头部widget API准备就绪时用于处理其他事件（例如: createButton）。

onGrayedObjectClicked(callback)

1. `callback` : function(subject)
    1. `subject` : object `{type, name}`
        1. `type` : `drawing` | `study`
        2. `name` : string, 被点击的主题名称

每次用户点击灰色的对象时，图表库都会调用此回调函数。例：

```
new TradingView.widget({
    drawings_access: {
        type: "black",
        tools: [
            { name: "Regression Trend" },
            { name: "Trend Angle", grayed: true },
        ]
    },
    studies_access: {
        type: "black",
        tools: [
            { name: "Aroon" },
```

```
            { name: "Balance of Power", grayed: true },
        ]
    },
    <...> // 其他widget设置
});

widget.onChartReady(function() {
    widget.onGrayedObjectClicked(function(data) {
        // 当您尝试创建能量均衡指标或趋势形状时
        // 此方法将被调用

        alert(data.name + " is grayed out!");
    })
});
```

onShortcut(shortcut, callback)

1. `shortcut`
2. `callback` : function(data)

每当按下快捷键时，图表库将会调用此回调。

例:

```
widget.onShortcut("alt+s", function() {
  widget.executeActionById("symbolSearch");
});
```

subscribe(event, callback)

1. `event` :

Event name	Library Version	Description
`toggle_sidebar`		显示/隐藏 绘图工具栏
`indicators_dialog`		显示 指标对话框

`toggle_header`		显示/隐藏 图表头
`edit_object_dialog`		显示 图表/指标属性对话框
`chart_load_requested`		即将加载新图表
`chart_loaded`		
`mouse_down`		
`mouse_up`		
`drawing`	1.7	绘图将添加到图表中。参数包含一个带有 `value` 字段的对象，该字段与绘图名称相对应。
`study`	1.7	指标将添加到图表中。参数包含一个带有 `value` 字段的对象，该字段与指标名称相对应。
`undo`	1.7	
`redo`	1.7	
`undo_redo_state_changed`	1.14	Undo/Redo状态已更改。参数包含一个具有Undo/Redo堆栈状态的对象。该对象与UndoRedoState方法的返回值具有相同的结构。
`reset_scales`	1.7	重置比例按钮被点击
`compare_add`	1.7	显示比较对话框
`add_compare`	1.7	添加了一个商品的比较
`load_study` template	1.7	一个指标模板被加载
`onTick`		最新k线被更新

`onAutoSaveNeeded`		用户修改了图表, `Chart change` 表示可以撤销用户的任何操作。回调函数不可以在5秒内多次调用, 详见: auto_save_delay
`onScreenshotReady`		服务器返回屏幕截图URL
`onMarkClick`		用户点击K线标记。标记ID将作为参数传递
`onTimescaleMarkClick`		用户点击时间刻度标记。标记ID将作为参数传递
`onSelectedLineToolChanged`		选择的线条工具已更改
`study_event`	1.15	指标从图表中删除。回调函数接收2个参数: 指标id和event类型 (当前这个参数唯一可能的值是 `remove`)
`drawing_event`	1.15	隐藏，显示，移动，移除或单击绘图。 回调函数接收2个参数: 指标id和event类型。event类型的可能值是 `hide` ， `show` 、 `move` 、 `remove` 、 `click`
`study_properties_changed`	1.14	指标属性已更改。实体ID将作为参数传递。
`series_properties_changed`	1.15	主数据列属性发生变化
`panes_height_changed`	1.15	窗格大小已更改。
`panes_order_changed`	1.15	窗格订单发生变化。
`layout_about_to_be_changed`		图表的数量或位置即将改变
`layout_changed`		图表的数量或位置已更改
`activeChartChanged`		活动的图表已变更

2. `callback` : function(arguments)

当GUI `event` 发生时，图表库将调用 `callback` 。 每个事件都可以有不同的参数。

unsubscribe(event, callback)

取消订阅特定事件 (即上表中的事件之一) 。

图表动作

chart()

返回图表对象，可用于调用图表方法

setLanguage(locale)

1. `locale` : 语言代码

设置Widget的语言。目前此调用将重新加载图表。 **请避免使用**

setSymbol(symbol, interval, callback)

1. `symbol` : string
2. `interval` : string
3. `callback` : function()

使图表更改其商品和周期。 新商品的数据到达后调用回调。

remove()

从您的页面中删除widget。

closePopupsAndDialogs()

调用此方法会关闭上下文菜单或对话框（如果已显示）。

selectLineTool(drawingId)

1. `drawingId`：可以为一个标识符 或
    1. `cursor`
    2. `dot`
    3. `arrow_cursor`
    4. `eraser`
    5. `measure`
    6. `zoom`
    7. `brush`

选择与绘图按钮上的单击相同的形状或光标。

selectedLineTool()

返回所选形状或光标的标识符（见上文）。

takeScreenshot()

此方法创建图表的快照并将其上传到服务器。

完成后, 调用 onScreenshotReady 回调函数。

快照的 URL 将作为参数传递给回调函数。

lockAllDrawingTools()

此方法返回一个 WatchedValue 对象, 可用于读取/设置/监视 "锁定所有绘图工具" 按钮的状态。

hideAllDrawingTools()

此方法返回 WatchedValue 对象, 该对象可用于读取/设置/监视 "隐藏所有绘图工具" 按钮的状态。

magnetEnabled()

此方法返回 WatchedValue 对象, 该对象可用于读取/设置/监视 "磁铁" 按钮的状态。

magnetMode()

此方法返回 WatchedValue 对象, 该对象可用于读取/设置/监视 "磁铁" 的模式

可用模式:

- `0` - 弱磁模式
- `1` - 强磁模式

保存/加载图表

save(callback)

1. `callback` : function(object)

将图表状态保存到JS对象。图表库将调用您的回调函数并将状态对象作为参数传递。

此调用是低级保存/加载API的一部分。

load(state)

1. `state` : object

从 `state` 对象加载图表。 此调用是低级保存/加载API的一部分。

getSavedCharts(callback)

1. `callback` : function(objects)

`objects` is an array of:

1. `id`
2. `name`
3. `image_url`
4. `modified_iso`

5. `short_symbol`
6. `interval`

返回当前用户在服务器上保存的图表描述列表。

loadChartFromServer(chartRecord)

1. `chartRecord` 是您使用getSavedCharts(callback) 返回的对象

从服务器加载并显示图表。

saveChartToServer(onCompleteCallback, onFailCallback, saveAsSnapshot, options)

1. `onCompleteCallback` : function()
2. `onFailCallback` : function()
3. `saveAsSnapshot` : should be always `false`
4. `options` : object `{ chartName }`
    1. `chartName` : 图表名称。应指定新图表并重命名。
    2. `defaultChartName` : 图表的默认名称。 如果当前图表没有名称，它将被使用。

将当前图表保存到服务器。

removeChartFromServer(chartId, onCompleteCallback)

1. `chartId` : 调用 getSavedCharts(callback)后获得的 `id`
2. `onCompleteCallback` : function()

从服务器移除图表。

自定义UI控件

onContextMenu(callback)

1. `callback` : function(unixtime, price). 此回调将返回一个值（见下文）。

每当用户在图表上打开文菜单时，图表库就会调用回调函数。 传递给回调函数的参数包含图表上单击点的unix时间和价格。

您必须返回具有以下格式的对象数组，才能在菜单中添加或删除项目。

```
{
    position: 'top' | 'bottom',
    text: 'Menu item text',
    click: <onItemClicked callback>
}
```

- `position`：项目在菜单中的位置
- `text`：菜单项文本
- `click`：当用户选择您的菜单项时将被调用

添加分隔符使用减号。 例: `{ text: "-", position: "top" }`.

要从菜单中删除现有项目，请在项目文本前面使用减号。 例:
`{ text: "-Objects Tree..." }`

例:

```
widget.onChartReady(function() {
    widget.onContextMenu(function(unixtime, price) {
        return [{
            position: "top",
            text: "First top menu item, time: " + unixtime + ", price: " +
            click: function() { alert("First clicked."); }
        },
        { text: "-", position: "top" },
        { text: "-Objects Tree..." },
        {
            position: "top",
            text: "Second top menu item 2",
            click: function() { alert("Second clicked."); }
        }, {
            position: "bottom",
            text: "Bottom menu item",
            click: function() { alert("Third clicked."); }
        }];
```

```
19      });
```

createButton(options)

1. `options` : object `{ align: "left" }`
    - `align` : `right` | `left` . default: `left`

在图表的顶部工具栏中创建一个新的DOM元素，并为此按钮返回HTMLElement。 您可以使用它在图表上添加自定义控件。

注意：必须在headerReady返回的 `Promise` 为resolved之后使用。

例：

```
widget.headerReady().then(function() {
    var button = widget.createButton();
    button.setAttribute('title', 'My custom button tooltip');
    button.addEventListener('click', function() { alert("My custom button
    button.textContent = 'My custom button caption';
});
```

对话框

从 1.6 版本开始

showNoticeDialog(params)

1. `params` : 对象:
    1. `title` : 标题
    2. `body` : 正文
    3. `callback` : 当按下ok按钮时调用的函数。

此方法显示一个对话框，其中包含自定义标题和文本以及“确定”按钮。

showConfirmDialog(params)

1. `params`: 对象:
    1. `title`: 标题
    2. `body`: 正文
    3. `callback(result)`: 当按下ok按钮时调用的函数。
       `result` 点击ok时为 `true`, 否则为 `false` 。

此方法显示一个带有自定义标题和文本以及"确定"、"取消"按钮的对话框。

showLoadChartDialog()

显示加载图表对话框。

showSaveAsChartDialog()

显示另存为...图表对话框。

Getters

symbolInterval()

图表库返回一个包含symbol和interval的对象。

mainSeriesPriceFormatter()

返回一个带有 `format` 方法的对象，用来批量格式化价格。 被引入在1.5.

getIntervals()

返回支持的周期数组。被引入在1.7.

getStudiesList()

返回全部技术指标数组，您可以通过它们创建技术指标指示器。

undoRedoState()

返回具有Undo/Redo堆栈状态的对象。 该对象具有以下属性:

- `enableUndo` : boolean类型, 表示撤消操作是否可用。
- `undoText` : 下一个撤消操作的名称。如果撤消堆栈为空，则它的值为 `undefined` 。
- `enableRedo` : boolean类型, 表示恢复操作是否可用。
- `redoText` : 下一个恢复操作的名称。如果恢复堆栈为空，则它的值为 `undefined` 。

定制

changeTheme(themeName)

该方法在版本 `1.13` 中引入

1. `themeName` 可以为 `"Light"` | `"Dark"`

此方法可更改图表主题而不重新加载图表。

您还可以使用Widget构造函数中的theme来创建具有自定义主题的图表。

addCustomCSSFile(url)

1. `url` 绝对或相对路径的 `static` 文件夹

该方法在版本 `1.3` 中引入。从1.4开始，使用custom_css_url替代。

applyOverrides(overrides)

该方法在版本 `1.5` 中引入

1. `overrides` 为一个对象，和overrides相同。

此方法将 `覆盖` 图表属性，而无需重新加载图表。

applyStudiesOverrides(overrides)

该方法在版本 `1.9` *中引入*

1. `overrides` 为一个对象，和studies_overrides相同。

此方法将 `覆盖` 指标的指标样式或输入参数，而无需重新加载图表。

交易终端专属

以下方法只在交易终端可用.

watchList()

该方法在版本 `1.9` *中引入*

返回一个对象来操作观察列表。 该对象具有以下方法：

1. `defaultList()` - 允许您获取默认的商品列表。
2. `getList(id?: string)` - 允许您获取商品列表。 如果未传递 `id` 则返回当前列表。 如果没有监视列表则返回 `null` 。
3. `getActiveListId()` - 允许您获取当前列表的ID。如果没有监视列表则返回 `null` 。
4. `getAllLists()` - 允许您获取所有列表。如果没有监视列表则返回 `null` 。
5. `setList(symbols: string[])` - 允许您将商品列表设置到观察列表中。 它将替换整个列表。**已过时。将在 `1.13` 版本中删除。请改用 `updateList` 。**
6. `updateList(listId: string, symbols: string[])` - 允许您编辑商品列表。
7. `renameList(listId: string, newName: string)` - 允许您将列表重命名为 `newName` .
8. `createList(listName?: string, symbols?: string[])` - 允许您创建具有 `listName` 名称的符号列表。如果未传递 `listName` 参数或者没有监视列表，则返回 `null` 。
9. `saveList(list: SymbolList)` - 允许您保存一个商品列表， `list` 是具有以下key的集合对象:

```
id: string;
```

```
    title: string;
    symbols: string[];
```

如果没有监视列表或者已有一个等价列表，则返回 `false` 否则返回 `true` 。

10. `deleteList(listId: string)` - 允许您删除商品列表。
11. `onListChanged()` - 当在监视列表中的商品更改时, 可以使用此方法进行通知。您可以使用此方法返回的 Subscription对象进行订阅和取消订阅。
12. `onActiveListChanged()` - 当选择了不同的监视列表时, 可以使用此方法进行通知。您可以使用此方法返回的 Subscription对象进行订阅和取消订阅。
13. `onListAdded()` -- 当新的列表添加到监视列表中时, 可以使用此方法进行通知。您可以使用此方法返回的 Subscription对象进行订阅和取消订阅。
14. `onListRemoved()` - 当监视列表中删除商品列表时, 可以使用此方法进行通知。您可以使用此方法返回的 Subscription对象进行订阅和取消订阅。
15. `onListRenamed()` -- 当监视列表中重命名商品列表时, 可以使用此方法进行通知。您可以使用此方法返回的 Subscription对象进行订阅和取消订阅。

📈 多图表布局

📈 chart(index)

1. `index` : 从0开始的图表索引，默认为0。

返回chart对象，用于调用Chart-Methods

📈 activeChart()

返回当前chart对象，用于调用Chart-Methods

📈 chartsCount()

返回当前布局的图表数目。

📈 layout()

返回当前布局模式。 可能的值： `4` , `6` , `8` , `s` , `2h` , `2-1` , `2v` , `3h` , `3v` , `3s` , `1-2` , `3r` , `4h` , `4v` , `4s` , `1-3` , `2-2` , `1-4` , `5s` , `6c` , `8c` .

💹 setLayout(layout)

1. `layout` : 可能的值: `4` , `6` , `8` , `s` , `2h` , `2-1` , `2v` , `3h` , `3v` , `3s` , `1-2` , `3r` , `4h` , `4v` , `4s` , `1-3` , `2-2` , `1-4` , `5s` , `6c` , `8c` .

变更当前图表布局。

💹 layoutName()

返回当前布局名称。 如果尚未保存当前布局，则返回undefined。

也可以看看

- 图表方法
- 定制概述
- Widgetg构造函数
- 存储与加载图表
- 指标覆盖
- 覆盖

4-4、图表方法

图表方法

以下为图表的方法列表。

在 1.4 版本之前 您可以使用 Widget 的构造函数返回给您的 widget 对象来调用下列方法。

从 1.5 版本之后 您可以使用 Widget 的方法chart(index) 或 activeChart()返回给您的图表对象来调用下列方法。

方法

- 图表订阅事件
  - onDataLoaded()
  - onSymbolChanged()
  - onIntervalChanged()
  - dataReady(callback)
  - crossHairMoved(callback)
  - onVisibleRangeChanged()
- 图表动作
  - setVisibleRange(range, options)
  - setSymbol(symbol, callback)
  - setResolution(resolution, callback)
  - resetData()
  - executeActionById(action)
  - getCheckableActionState(action)
  - refreshMarks()
  - clearMarks()
  - setChartType(type)
  - closePopupsAndDialogs()
  - setTimezone(timezone)
  - getTimezone()
  - canZoomOut()
  - zoomOut()
- 指标与形状

    - getAllShapes()
    - getAllStudies()
    - setEntityVisibility(id, isVisible)[过时]
    - createStudy(name, forceOverlay, lock, inputs, overrides, options)
    - getStudyById(entityId)
    - getSeries()
    - showPropertiesDialog(entityId)
    - createShape(point, options)
    - createMultipointShape(points, options)
    - getShapeById(entityId)
    - removeEntity(entityId)
    - removeAllShapes()
    - removeAllStudies()
    - getPanes()
- 指标模板
    - createStudyTemplate(options)
    - applyStudyTemplate(template)
- Trading Primitives
    - createOrderLine()
    - createPositionLine()
    - createExecutionShape()
- Getters
    - symbol()
    - symbolExt()
    - resolution()
    - getVisibleRange()
    - getVisiblePriceRange()
    - scrollPosition()
    - defaultScrollPosition()
    - priceFormatter()
    - chartType()
- 其他
    - exportData(options)
    - selection()
    - setZoomEnabled(enabled)
    - setScrollEnabled(enabled)

图表订阅事件

onDataLoaded()

您可以使用此方法返回的订阅对象进行订阅，以便在加载新历史 K 线时收到通知，您还可以使用此订阅对象取消此订阅事件。

onSymbolChanged()

您可以使用此方法返回的订阅对象进行订阅，以便在更改商品时收到通知，您还可以使用此订阅对象取消此订阅事件。

onIntervalChanged()

您可以使用此方法返回的订阅对象进行订阅，以便在更改时间周期时收到通知，您还可以使用此订阅对象取消此订阅事件。 当事件被触发时，它将提供以下参数：

1. `interval` : 新周期
2. `timeframeParameters` : 此对象只有一个字段 `timeframe`
   如果在用户单击时间周期面板时更改时间周期，则它包含 `timeframe` 。
   否则 `timeframe` 为 `undefined` ，你可以改变它来显示某一范围的 K 线。 有效的 `timeframe` 是一个数字，字母 `D` 代表天数， `M` 代表月数。

例如:

```
widget
  .chart()
  .onIntervalChanged()
  .subscribe(null, function(interval, obj) {
    obj.timeframe = "12M";
  });
```

dataReady(callback)

1. `callback` : function(interval)

如果 K 线数据已被加载或被接收时，图表库将立即调用此回调。
返回 `true` 为已经加载，否则为 `false` 。

crossHairMoved(callback)

1.5 版本开始

1. `callback` : function({time, price})

每当十字线位置改变时，图表库将会调用回调函数。

onVisibleRangeChanged()

1.13 版本开始

您可以使用此功能返回的Subscription对象进行订阅，以便在可见时间范围更改时得到通知。 您还可以使用同一对象取消订阅该事件。

图表动作

setVisibleRange(range, options)

1. `range` : 对象, `{from to}`
   - `from` , `to` : unix时间戳, UTC
2. `options` : `{applyDefaultRightMargin, percentRightMargin}`
   - `applyDefaultRightMargin` : 布尔值，表示如果指向最后一个线，是否应将默认的右边距应用于右边框。
   - `percentRightMargin` : 布尔值，表示如果指向最后一个线，是否应将默认的右边距百分比应用于右边框。

强制图表调整其参数 (scroll, scale) 以使选定的时间段适合视口。

返回一个Promise对象，将在应用可见范围后，返回resolved。

此方法是在版本1.2中引入

setSymbol(symbol, callback)

1. `symbol` : string
2. `callback` : function()

更改图表商品。 新商品的数据到达后调用回调。

setResolution(resolution, callback)

1. `resolution` : string. 格式化详细参照:周期。
2. `callback` : function()

更改图表周期。 新周期的数据到达后调用回调。

resetData()

使图表重新请求 datafeed 中的数据。 通常你需要在图表数据发生变化时调用它。 在调用这个之前，你应该调用onResetCacheNeededCallback。

executeActionById(actionId)

1.3 版本开始

1. `actionId` : string

通过它的 id 执行一个动作。

显示对话框

- `chartProperties`
- `compareOrAdd`
- `scalesProperties`
- `tmzProperties`
- `paneObjectTree`
- `insertIndicator`
- `symbolSearch`
- `changeInterval`

- `gotoDate`

其他动作

- `timeScaleReset`
- `chartReset`
- `seriesHide`
- `studyHide`
- `lineToggleLock`
- `lineHide`
- `showLeftAxis`
- `showRightAxis`
- `scaleSeriesOnly`
- `drawingToolbarAction`
- `stayInDrawingModeAction`
- `hideAllMarks`
- `showCountdown`
- `showSeriesLastValue`
- `showSymbolLabelsAction`
- `showStudyLastValue`
- `showStudyPlotNamesAction`
- `undo`
- `redo`
- `paneRemoveAllStudiesDrawingTools`

例如:

```
// < ... >
widget.chart().executeActionById("undo");
// < ... >
widget.chart().executeActionById("drawingToolbarAction"); // 隐藏或显示绘图工
// < ... >
```

getCheckableActionState(actionId)

1.7 版本开始

1. `actionId` : string

根据动作 ID 获取是否可以勾选的状态（例如: `stayInDrawingModeAction` 、`showSymbolLabelsAction` ）（请参阅上面的动作ID）

refreshMarks()

再次请求可见标记。

clearMarks()

清除所有可见标记。

setChartType(type)

1. `type` : number

设置主数据列的样式。

```
// 美国线
STYLE_BARS = 0;
// K线图
STYLE_CANDLES = 1;
// 线形图
STYLE_LINE = 2;
// 面积图
STYLE_AREA = 3;
// 平均K线图
STYLE_HEIKEN_ASHI = 8;
// 空心K线图
STYLE_HOLLOW_CANDLES = 9;
// 基准线
STYLE_BASELINE = 10;
// HiLo线
STYLE_HILO = 12;

// 砖形图
STYLE_RENKO* = 4;
// 卡吉图
STYLE_KAGI* = 5;
```

```
// 点数图
STYLE_PNF* = 6;
// 新价图
STYLE_PB* = 7;
```

💹 交易终端专属

closePopupsAndDialogs()

调用此方法关闭上下文菜单或对话框,假设其已经显示。

setTimezone(timezone)

1. `timezone` : string

查看timezone更多信息

例:

```
widget.activeChart().setTimezone("Asia/Singapore");
```

更改图表时区。

getTimezone()

Since version 1.15.

返回图表的当前timezone。

canZoomOut()

该方法在版本 `1.14` *中引入*

当您调用此方法时，图表库会检查是否有任何缩放事件要撤消。

zoomOut()

该方法在版本 `1.14` 中引入

当您调用此方法时，它会模拟点击“缩小”按钮。 仅在图表缩放时才有效。 使用 `canZoomOut` 检查是否可以调用此方法。

指标与形状

getAllShapes()

返回所有已创建的形状对象数组。 每个对象都有以下字段：

- `id`：形状 id
- `name`：形状名称

getAllStudies()

返回所有已创建的指标对象的数组。 每个对象都有以下字段：

- `id`：指标 id
- `name`：指标名称

setEntityVisibility(id, isVisible)

设置具有 `id` 的实体的可见性。

不推荐使用：使用形状/指标 API（ `getShapeById` / `getStudyById` ）来代替此方法。 将在未来的版本中删除。

createStudy(name, forceOverlay, lock, inputs, overrides, options)

1. `name`：string, 指标名称，您可以在 `技术指标` 工具栏中看到。
2. `forceOverlay`：强制图表库将创建的指标放在主窗格中
3. `lock`：boolean, 是否锁定指标
4. `inputs`：(在 `1.2` 版本开始) 指标参数数组, 该数组应包含与指标属性对话框中相同顺序的输入值。

5. `overrides`：(在 `1.2` 版本开始) 一个对象 包含属性,覆盖你的新指标 。注意：您不应指定指标名称：应以具有绘图名称的属性路径为起始。
6. `options`：这个对象只支持关键字 `checkLimit`。如果为 `true` 时，超出限制，将显示指标限制对话框。
    - `checkLimit` - 如果是 `true` ，则超出限制时将显示指标限制对话框。
    - `priceScale` - 指标的首选价格刻度。 可能的值是：
        - `left` - 依附在左侧价格刻度
        - `right` - 依附在右侧价格刻度
        - `no-scale` - 不要将指标依附在任何价格刻度上。 该指标将以 `界面(无缩放)` 模式添加
        - `as-series` - 将指标依附在主数据列所依附的价格刻度（仅适用于将指标添加到主数据列的窗格中）

请参阅此处有关与指标相关的窗格和刻度特性的更多信息。

返回一个 Promise 类型的 `entityId` 。

从 1.12 版本开始，函数立即返回结果。 回调为保持兼容性

创建一个关于主商品的指标。 例子:

- `createStudy('MACD', false, false, [14, 30, "close", 9])`
- `createStudy('Moving Average Exponential', false, false, [26])`
- `createStudy('Stochastic', false, false, [26], {"%d.color" : "#FF0000"})`
- `createStudy('Moving Average', false, false, [26], {'Plot.linewidth': 10})`

Remark: `Compare` 指标有 2 个参数: `[dataSource, symbol]`. `dataSource` 支持的值: `["close", "high", "low", "open"]`.

Remark 2: 当您选择在图表上添加数据列时，您实际使用了 `Overlay` 指标，这个指标只有一个参数 – `symbol`. 以下是添加商品的示例：

```
widget.chart().createStudy("Overlay", false, false, ["AAPL"]);
```

Remark 3: 当您选择比较数据列时，您实际上使用了 `Compare` 指标。 它有 2 个参数 – `source` 和 `symbol`. 下面是一个添加比较数据列的例子:

```
widget.chart().createStudy("Compare", false, false, ["open", "AAPL"]);
```

getStudyById(entityId)

1. `entityId` ：对象。 通过 API 创建指标时返回的值。

返回指标 API的一个实例，它允许您与指标进行交互。

getSeries()

返回允许您与主数据列进行交互的SeriesApi的实例。

showPropertiesDialog(entityId)

1. `entityId` : 实体id。通过API创建指标或形状时返回的值。

用于显示指定的指标或形状的属性对话框。

createShape(point, options)

1. `point` : 对象 `{time, [price], [channel]}`
   - `time` : unix 时间戳. 唯一的强制性参数。
   - `price` : 如果指定 `price` ，则形状将以此价格位置放置。
     如果未指定，则根据 `channel` 值将形状放置在 K 线的相关位置。
   - `channel` : 如果未设置 `price` ，则通过 `channel` 设置的k线位置放置形状，可能的值：(`open` , `high` , `low` , `close`)。
     如果未指定 `channel` ，则以 `open` 为默认值。
2. `options` : object `{shape, [text], [lock], [overrides]}`
   - `shape` 可能的值为 `arrow_up` 、 `arrow_down` 、 `flag` 、 `vertical_line` 、 `horizontal_line` 。
     `flag` 为默认值。
   - `text` 是一个可选参数。 如果支持，为包含在形状中的文本。

- `lock` 是否锁定形状
- `disableSelection` (开始于 `1.3`) 禁用选中
- `disableSave` (开始于 `1.3`) 禁用保存
- `disableUndo` (开始于 `1.4`) 禁用撤销
- `overrides` (开始于 `1.2`). 它是一个对象，包含为新形状设置的属性。
- `zOrder` (开始于 `1.3`) 可能的值为 `top` 、 `bottom` 。
  `top` 将线条工具放在所有其他图表对象的顶部, 而 `bottom` 将线条工具放在所有其他图表对象底部, `top` 为默认值。
- `showInObjectsTree` : `true` 为默认值。在 `工具树状图` 对话框中显示形状。

该函数返回 `entityId` - 如果创建成功则返回形状的唯一 ID，如果不成功则返回 `null` 。

此调用会在图表上的指定地点创建一个形状，前提是它位于主数据列区域内。

createMultipointShape(points, options)

1. `points` : 具有以下字段的数组 `[{time, [price], [channel]},...]`
   - `time` : unix 时间戳. 唯一的强制性参数。
   - `price` : 如果指定 `price` ，则形状将以此价格位置放置。
     如果未指定，则根据 `channel` 值将形状放置在 K 线的相关位置。
   - `channel` : 如果未设置 `price` ，则通过 `channel` 设置的k线位置放置形状，可能的值：(`open` , `high` , `low` , `close`)。
     如果未指定 `channel` ，则以 `open` 为默认值。
2. `options` : object `{shape, [text], [lock], [overrides]}`
   - `shape` 可能的值为 `arrow_up` 、 `arrow_down` 、 `flag` 、 `vertical_line` 、 `horizontal_line` 。
     `flag` 为默认值。
   - `text` 是一个可选参数。 如果支持，为包含在形状中的文本。
   - `lock` 是否锁定形状
   - `disableSelection` (开始于 `1.3`) 禁用选中
   - `disableSave` (开始于 `1.3`) 禁用保存
   - `disableUndo` (开始于 `1.4`) 禁用撤销
   - `overrides` (开始于 `1.2`). 它是一个对象，包含为新形状设置的属性。
   - `zOrder` (开始于 `1.3`) 可能的值为 `top` 、 `bottom` 。

`top` 将线条工具放在所有其他图表对象的顶部, 而 `bottom` 将线条工具放在所有其他图表对象底部, `top` 为默认值。

- `showInObjectsTree`： `true` 为默认值。在 `工具树状图` 对话框中显示形状。

该函数返回 `entityId` - 如果创建成功则返回形状的唯一 ID，如果不成功则返回 `null` 。

查看形状与覆盖以获取更多信息。

此调用会在图表上的指定地点创建一个多点形状，前提是它位于主数据列区域内。

getShapeById(entityId)

1. `entityId` ：对象。 通过 API 创建形状时返回的值。

返回允许您与形状交互的形状 API实例。

removeEntity(entityId)

1. `entityId` ：对象。 为创建实体 (形状或指标) 后返回的值。

删除指定实体。

removeAllShapes()

删除全部形状。

removeAllStudies()

删除全部指标。

getPanes()

返回窗格Api 的实例数组，允许您与窗格进行交互。

指标模板

createStudyTemplate(options)

1. `options` : 对象 `{saveInterval}`
    1. `saveInterval` : boolean

将指标模板保存到 JS 对象。 图表库将调用您的回调函数并将状态对象作为参数传递。

此调用是低级存储与加载图表的一部分。

applyStudyTemplate(template)

1. `template` : object

从状态对象加载指标模板。

此调用是低级存储与加载图表的一部分。

交易元语(Trading Primitives)

createOrderLine(options)

`options` 是一个具有字段： `disableUndo` 的对象, 可以是 `true` 或 `false` . 出于兼容性原因，默认值为 `false` 。

在图表上创建新的交易订单并返回可用于调整其属性和特性的 API 对象。

强烈建议在使用此调用之前阅读交易元语。

API 对象方法：

- `remove()` : 从图表中移除仓位。 调用方法后不能再使用此 API 对象。
- `onModify(callback)` / `onModify(data, callback)`
- `onMove(callback)` / `onMove(data, callback)`
- `onCancel(callback)` / `onCancel(data, callback)`

API 对象具有下面列出的一组属性。 每个属性应通过各自的访问器调用。 例如，如果你想使用 `Extend Left` 属性，那么请使用 `setExtendLeft()` 和 `getExtendLeft()` 方法。

一般属性:

属性名称	类型	支持的值	默认值
Price	Double	Double	0.0
Text	String	String	""
Tooltip	String	String	""
Modify Tooltip	String	String	""
Cancel Tooltip	String	String	""
Quantity	String	String	""
Editable	Boolean	Boolean	true

趋势线属性:

属性名称	类型	支持的值	默认值
Extend Left	Boolean	"inherit" or Boolean	True
Line Length	Integer	"inherit" or 0 .. 100	0
Line Style	Integer	"inherit" or 0 .. 2	2
Line Width	Integer	"inherit" or 1 .. 4	1

字体:

属性名称	类型	默认值
Body Font	String	"bold 7pt Verdana"

Quantity Font	String	"bold 7pt Verdana"

颜色:

属性名称	类型	默认值
Line Color	String	"rgb(255, 0, 0)"
Body Border Color	String	"rgb(255, 0, 0)"
Body Background Color	String	"rgba(255, 255, 255, 0.75)"
Body Text Color	String	"rgb(255, 0, 0)"
Quantity Border Color	String	"rgb(255, 0, 0)"
Quantity Background Color	String	"rgba(255, 0, 0, 0.75)"
Quantity Text Color	String	"rgb(255, 255, 255)"
Cancel Button Border Color	String	"rgb(255, 0, 0)"
Cancel Button Background Color	String	"rgba(255, 255, 255, 0.75)"
Cancel Button Icon Color	String	"rgb(255, 0, 0)"

例子:

```
widget.chart().createOrderLine()
    .setTooltip("Additional order information")
    .setModifyTooltip("Modify order")
    .setCancelTooltip("Cancel order")
    .onMove(function() {
        this.setText("onMove called");
    })
    .onModify("onModify called", function(text) {
        this.setText(text);
    })
    .onCancel("onCancel called", function(text) {
        this.setText(text);
    })
```

```
14      .setText("STOP: 73.5 (5,64%)")
15      .setQuantity("2");
```

createPositionLine(options)

`options` 是一个具有字段： `disableUndo` 的对象, 可以是 `true` 或 `false` . 出于兼容性原因，默认值为 `false` 。

在图表上创建新的交易头寸并返回一个可用于调整其属性和特性的 API 对象。

强烈建议在使用此调用之前阅读交易元语。

API 对象方法：

- `remove()` : 从图表中移除位置。 调用此方法后不能再使用 API 对象。
- `onClose(callback)` / `onClose(data, callback)`
- `onModify(callback)` / `onModify(data, callback)`
- `onReverse(callback)` / `onReverse(data, callback)`

API 对象具有下面列出的一组属性。 每个属性应通过各自的访问器调用。 例如，如果你想使用 `Extend Left` 属性，那么请使用 `setExtendLeft()` 和 `getExtendLeft()` 方法。

一般属性:

属性名称	类型	支持的值	默认值
Price	Double	Double	0.0
Text	String	String	""
Tooltip	String	String	""
Protect Tooltip	String	String	""
Reverse Tooltip	String	String	""
Close Tooltip	String	String	""

Quantity	String	String	""

趋势线属性:

属性名称	类型	支持的值	默认值
Extend Left	Boolean	"inherit" or Boolean	True
Line Length	Integer	"inherit" or 0 .. 100	0
Line Style	Integer	"inherit" or 0 .. 2	2
Line Width	Integer	"inherit" or 1 .. 4	1

字体:

属性名称	类型	默认值
Body Font	String	"bold 7pt Verdana"
Quantity Font	String	"bold 7pt Verdana"

颜色:

属性名称	类型	默认值
Line Color	String	"rgb(0, 113, 224)"
Body Border Color	String	"rgb(0, 113, 224)"
Body Background Color	String	"rgba(255, 255, 255, 0.75)"
Body Text Color	String	"rgb(0, 113, 224)"
Quantity Border Color	String	"rgb(0, 113, 224)"
Quantity Background Color	String	"rgba(0, 113, 224, 0.75)"

Quantity Text Color	String	"rgb(255, 255, 255)"
Reverse Button Border Color	String	"rgb(0, 113, 224)"
Reverse Button Background Color	String	"rgba(255, 255, 255, 0.75)"
Reverse Button Icon Color	String	"rgb(0, 113, 224)"
Close Button Border Color	String	"rgb(0, 113, 224)"
Close Button Background Color	String	"rgba(255, 255, 255, 0.75)"
Close Button Icon Color	String	"rgb(0, 113, 224)"

例子:

```
widget.chart().createPositionLine()
    .onModify(function() {
        this.setText("onModify called");
    })
    .onReverse("onReverse called", function(text) {
        this.setText(text);
    })
    .onClose("onClose called", function(text) {
        this.setText(text);
    })
    .setText("PROFIT: 71.1 (3.31%)")
    .setTooltip("附加仓位信息")
    .setProtectTooltip("保护仓位")
    .setCloseTooltip("关闭仓位")
    .setReverseTooltip("翻转仓位")
    .setQuantity("8.235")
    .setPrice(15.5)
    .setExtendLeft(false)
    .setLineStyle(0)
    .setLineLength(25);
```

createExecutionShape(options)

1. `options` 是一个具有字段： `disableUndo` 的对象, 这可以是 `true` 或 `false` . 出于兼容性原因，默认值为 `false` 。

在图表上创建新的交易执行并返回可用于控制执行属性的 API 对象。

强烈建议在使用此调用之前阅读交易元语。

API 对象具有下面列出的一组属性。 每个属性应通过各自的访问器调用。 例如，如果你想使用 `Extend Left` 属性，那么请使用 `setExtendLeft()` 和 `getExtendLeft()` 方法。

API 对象方法：

- `remove()` ：从图表中删除执行信号形状。调用后，您无法再次使用此 API 对象。

一般属性:

属性名称	类型	支持的值	默认值
Price	Double	Double	0.0
Time	Integer	Unix time	0
Direction	String	"buy" or "sell"	"buy"
Text	String	String	"execution"
Tooltip	String	String	""
Arrow Height	Integer	Integer	8
Arrow Spacing	Integer	Integer	1

字体:

属性名称	类型	默认值
Font	String	"8pt Verdana"

颜色:

属性名称	类型	默认值

Text Color	String	"rgb(0, 0, 0)""
Arrow Color	String	"rgba(0, 0, 255)"

例子:

```
widget.chart().createExecutionShape()
    .setText("@1,320.75 Limit Buy 1")
    .setTooltip("@1,320.75 Limit Buy 1")
    .setTextColor("rgba(0,255,0,0.5)")
    .setArrowColor("#0F0")
    .setDirection("buy")
    .setTime(1413559061758)
    .setPrice(15.5);
```

Getters

symbol()

返回图表商品。

symbolExt()

返回图表的商品信息对象。 该对象具有以下字段：

- `symbol` : 与方法symbol()结果相同
- `full_name` : 商品全称
- `exchange` : 商品交易所
- `description` : 商品描述
- `type` : 商品类型

resolution()

返回图表的周期。格式在这个周期中描述。

getVisibleRange()

返回对象 `{from, to}`. `from` 和 `to` 是 **图表时区**的单位时间戳

getVisiblePriceRange()

在 1.7 版本开始

不建议使用，请改用Price Scale API。

返回对象 `{from, to}`. `from` 和 `to` 是主数据列的可见范围边界。

scrollPosition()

在 1.15 版本开始

返回从图表右边缘到最后一根K线的距离，以K线为单位。 这实际上是图表的当前滚动位置，包含右边距。

defaultScrollPosition()

在 1.15 版本开始

返回从图表右边缘到最后一根K线的默认距离，以K线为单位。

priceFormatter()

返回带有 `format` 函数的对象，可用于格式化价格。

chartType()

返回图表类型。

其他

exportData(options)

从 1.14 版本开始

1. `options` (可选)是一个对象，它可以包含以下属性：
   - `from` (`number`) - 第一个导出 k□ 线的时间(UNIX 时间戳，以秒为单位)。
     默认情况下，使用最左侧加载的 k□ 线的时间。
   - `to` (`number`) - 最后一个导 k□ 线的时间(UNIX 时间戳，以秒为单位)。
     默认情况下，使用最右侧(实时)k 线的时间。
   - `includeTime` (`boolean` ,默认 `true`) - 定义导出数据的每个项目是否应包含时间。
   - `includeSeries` (`boolean` ,默认 `true`) - 定义导出的数据是否应包含数据列
     (open，high，low，close)。
   - `includedStudies` - 哪些指标应包括在导出的数据中
     (默认情况下，值为 `'all'` 表示包含所有指标, 但如果您只想导出其中一些,则可以在数组中指定指标 ID)。

从图表中导出数据，返回 Promise 对象。 此方法不加载数据。 结果具有以下结构:

- `schema` 是一个字段描述符数组，每个描述符可能是以下类型之一：
  - `TimeFieldDescriptor` - 时间字段的描述。 它只包含一个属性 - 带有 `'time'` 值的 `type` 。
  - `SeriesFieldDescriptor` - 数据列字段的描述。 它包含以下属性：
    - `type` (`'value'`)
    - `sourceType` (`'series'`)
    - `plotTitle` (`string`) - 绘图的名称（open，high，low，close)。
  - `StudyFieldDescriptor` - 指标字段的描述。 它包含以下属性：
    - `type` (`'value'`)
    - `sourceType` (`'study'`)
    - `sourceId` (`string`) - 指标 ID
    - `sourceTitle` (`string`) - 指标的标题
    - `plotTitle` (`string`) - 绘图的标题
- `data` 为Float64Array类型数组。 每个 `Float64Array` 数组与 `schema` 数组的长度相同，表示相关字段的项目。

例如:

1. ```
   chart.exportData({ includeTime: false, includeSeries: true,
   includedStudies: [] })
   ```
   - 仅导出数据列。
   ```

2. `chart.exportData({ includeTime: true, includeSeries: true, includedStudies: [] })`
   - 随时间导出数据列。
3. `chart.exportData({ includeTime: false, includeSeries: false, includedStudies: ['STUDY_ID'] })`
   - 导出 ID 为 `STUDY_ID` 的指标数据。
4. `chart.exportData({ includeTime: true, includeSeries: true, includedStudies: 'all' })`
   - 从图表中导出所有可用数据。
5. `chart.exportData({ includeTime: false, includeSeries: true, to: Date.UTC(2018, 0, 1) / 1000 })`
   - 导出 `2018-01-01` 以前的数据。
6. `chart.exportData({ includeTime: false, includeSeries: true, from: Date.UTC(2018, 0, 1) / 1000 })`
   - 导出 `2018-01-01` 之后的数据。
7. `chart.exportData({ includeTime: false, includeSeries: true, from: Date.UTC(2018, 0, 1) / 1000, to: Date.UTC(2018, 1, 1) / 1000 })`
   - 导出在 `2018-01-01` 和 `2018-02-01` 之间的数据。

selection()

在 1.15 版本开始

返回SelectionApi，可用于更改图表选择和订阅图表选择的更改。

setZoomEnabled(enabled)

在 1.15 版本开始

启用 (true) 或 禁用 (false) 缩放图表。

setScrollEnabled(enabled)

在 1.15 版本开始

启用 (true) 或 禁用 (false) 滚动图表。

也可以看看

- Widget 方法
- 定制概述
- Widgetg 构造函数
- 存储于加载图表
- 指标覆盖默认参数
- 覆盖默认参数

4-5、功能集

`功能` 或 `功能集` 是一个字符串，可用于更改图表的功能。有简单（原子）和复杂（复合）功能。

复杂功能由简单功能组成。

禁用复杂功能会使得其所有简单功能都被禁用。 支持的功能如下所示。

请注意，下表中的 `-` 字符不是功能集名称的一部分。

控件和其他视觉元素的可见性

功能集互动地图

ID	默认状态	版本	描述
header_widget	on		
- header_widget_dom_node	on		隐藏头部小部件 DOM 元素
- header_symbol_search	on		
- symbol_search_hot_key	on	1.9	按任意键进行商品搜索
- header_resolutions	on		
- - header_interval_dialog_button	on		
- - - show_interval_dialog_on_key_press	on		
- header_chart_type	on		
- header_settings	on		与图表属性按钮相关
- header_indicators	on		

- header_compare	on		
- header_undo_redo	on		
- header_screenshot	on		
- header_fullscreen_button	on		
compare_symbol	on	1.5	您可以使用此功能集从上下文菜单中删除 `比较/叠加` 对话框
border_around_the_chart	on		
header_saveload	on		隐藏保存/加载按钮（该功能不是 `header_widget` 功能集的一部分）
left_toolbar	on		
control_bar	on		与图表底部的导航按钮相关联
timeframes_toolbar	on		
legend_widget	on	1.15	禁用此功能会隐藏图例窗口小部件
edit_buttons_in_legend	on		
- show_hide_button_in_legend	on	1.7	
- format_button_in_legend	on	1.7	
- study_buttons_in_legend	on	1.7	
- delete_button_in_legend	on	1.7	
context_menus	on		
- pane_context_menu	on		
- scales_context_menu	on		
- legend_context_menu	on		

main_series_scale_menu	on	1.7	显示图表右下角的设置按钮
display_market_status	on		
remove_library_container_border	on		
chart_property_page_style	on		
property_pages	on	1.11	禁用所有属性页
show_chart_property_page	on	1.6	关闭此功能会禁用 `属性`
chart_property_page_scales	on		
chart_property_page_background	on		
chart_property_page_timezone_sessions	on		
chart_property_page_trading	on		此功能仅适用于交易终端
chart_property_page_right_margin_editor	on	1.15	在设置对话框中显示右边距编辑器
countdown	on	1.4	在价格标尺上显示倒计时标签
caption_buttons_text_if_possible	off	1.4	尽可能在标题和比较按钮上显示文本而不是图标
dont_show_boolean_study_arguments	off	1.4	隐藏 true/false 指标参数
hide_last_na_study_output	off	1.4	隐藏最后的 n/a 指标输出数据
symbol_info	on	1.5	启用商品信息对话框
timezone_menu	on	1.5	禁用时区上下文菜单
snapshot_trading_drawings	off	1.6	在截图中包括订单/头寸/成交
source_selection_markers	on	1.11	禁用数据列和指标的选择标记
go_to_date	on	1.11	允许您使用 `转到` 对话框跳转到特定栏

adaptive_logo	on	1.11	允许您在小屏幕设备上隐藏 logo 的 `TradingView` 文字
show_dom_first_time	off	1.12	当用户第一次打开图表时显示 DOM 面板
hide_left_toolbar_by_default	off	1.12	当用户第一次打开图表时，隐藏左侧工具栏
chart_style_hilo	off	1.15	在图表样式控件中添加 Hi-Lo 选项

元素放置

ID	默认状态	版本	描述
move_logo_to_main_pane	off		将 logo 放在主数据列窗格上而不是底部窗格

特性

ID	默认状态	版本	描述
use_localstorage_for_settings	on		允许将所有属性（包括收藏夹）存储到 localstorage
- items_favoriting	on		禁用此功能会隐藏所有“收藏此项目”按钮

- save_chart_properties_to_local_storage	on		可以禁用以禁止将图表属性存储到 localstorage，同时允许保存其他属性。 其他属性是收藏在图表库和 Watchlist 的商品，以及交易终端中的一些面板状态。
create_volume_indicator_by_default	on		
create_volume_indicator_by_default_once	on		
volume_force_overlay	on		在主数据量列的窗格上放置成交量指标
right_bar_stays_on_scroll	on		确定缩放功能的特性：如果禁用此功能，鼠标光标下的 K 线将保持在同一位置
constraint_dialogs_movement	on		将对话框保留在图表中
charting_library_debug_mode	off		启用日志
show_dialog_on_snapshot_ready	on		禁用此功能允许您以静默方式创建快照
study_market_minimized	on		与“指标”对话框相关，并确定它是否紧凑或包含搜索栏以及类别
study_dialog_search_control	on	1.6	在指标对话框中显示搜索控件
side_toolbar_in_fullscreen_mode	off		可以在全屏模式下启用绘图工具栏
same_data_requery	off		允许您使用相同的商品调用 `setSymbol` 来刷新数据
disable_resolution_rebuild	off		显示完全由 datafeed 提供的 K 线时间而不进行任何调整。
chart_scroll	on	1.10	允许图表滚动
chart_zoom	on	1.10	允许图表缩放

high_density_bars	off	1.11	允许缩小以在单个屏幕上显示超过 60000 根 K 线
cl_feed_return_all_data	off	1.11	允许您从 datafeed 返回多于请求的 K 线，并同时在图表上显示
uppercase_instrument_names	on	1.12	禁用此功能允许用户输入区分大小写的商品
low_density_bars	off	1.15	允许放大以在视口中最多显示一个K线
no_min_chart_width	off	1.14	禁用最小图表宽度限制
fix_left_edge	off	1.14	阻止滚动到第一个历史 K 线的左侧
lock_visible_time_range_on_resize	off	1.14	防止在图表调整大小时更改可见时区
shift_visible_range_on_new_bar	on	1.15	如果禁用，则添加新的K线会缩小图表并保留第一个可见点。 否则，当出现新K线时，图表将向左滚动一点。
custom_resolutions	off	1.15	如果启用，则可以添加自定义分辨率

重要功能

ID	默认状态	库版本	描述
study_templates	off		
datasource_copypaste	on		允许复制绘图和指标
seconds_resolution	off	1.4	支持秒的周期

交易终端

ID	默认状态	终端版本	描述
support_multicharts	on		启用与多图表布局相关的上下文菜单操作（克隆，同步）
header_layouttoggle	on		显示标题中的“选择布局”按钮
show_logo_on_all_charts	off		在多功能布局的每个图表上显示 logo
chart_crosshair_menu	on	1.7	在价格范围内启用“加号”按钮以进行快速交易
add_to_watchlist	on	1.9	在菜单中启用“添加商品到观察列表”项
footer_screenshot	on	1.11	显示页脚中的截图按钮（账户管理器）
open_account_manager	on	1.11	默认打开账户管理器
trading_notifications	on	1.11	在图表上显示交易通知
multiple_watchlists	on	1.12	启用创建多个监视列表
show_trading_notifications_history	on	1.13	启用底部面板中的 "通知日志" 选项卡

4-6、服务端定制

定制概述

定制是一个广泛的概念，这就是为什么我们有几篇关于它的文章。

通过数据流定制

这些主要是与数据相关的自定义。它们使用datafeed配置响应。

以下是配置响应的示例。

```
{
    supports_search: true,
    supports_group_request: false,
    supports_marks: true,
    exchanges: [
        {value: "", name: "All Exchanges", desc: ""},
        {value: "XETRA", name: "XETRA", desc: "XETRA"},
        {value: "NSE", name: "NSE", desc: "NSE"}
    ],
    symbolsTypes: [
        {name: "All types", value: ""},
        {name: "Stock", value: "stock"},
        {name: "Index", value: "index"}
    ],
    supportedResolutions: [ "1", "15", "30", "60", "D", "2D", "3D", "W", "
};
```

在JS API可以找到更详细的说明。

在客户端进行定制

允许您最大化的定制UI/UX。这些定制通过定义图表控件中的构造函数的参数完成。

图表控件构造函数调用的示例：

```
```

```
var widget = new TradingView.widget({
    fullscreen: true,
    symbol: 'AA',
    interval: 'D',
    toolbar_bg: '#f4f7f9',
    allow_symbol_change: true,
    container_id: "tv_chart_container",
    datafeed: new Datafeeds.UDFCompatibleDatafeed("http://demo_feed.tradin
    library_path: "charting_library/",
    locale: "en",
    drawings_access: { type: 'black', tools: [ { name: "Regression Trend"
    disabled_features: ["use_localstorage_for_settings", "volume_force_ove
    enabled_features: ["move_logo_to_main_pane"],
    overrides: {
        "mainSeriesProperties.style": 0,
        "symbolWatermarkProperties.color" : "#944",
        "volumePaneSize": "tiny"
    },
    studies_overrides: {
        "bollinger bands.median.color": "#33FF88",
        "bollinger bands.upper.linewidth": 7
    },
    debug: true,
    time_frames: [
        { text: "50y", resolution: "6M" },
        { text: "1d", resolution: "5" },
    ],
    charts_storage_url: 'http://saveload.tradingview.com',
    client_id: 'tradingview.com',
    user_id: 'public_user',
    favorites: {
        intervals: ["1D", "3D", "3W", "W", "M"],
        chartTypes: ["Area", "Line"]
    }
});
```

详情参考：Widget构造器

也可以看看

- Widget方法
- 定制的使用案例

4-7、定制的使用案例

图表库允许您自定义外观、数据显示的方式、默认属性等等。

客户端和服务端的定制， 其中一些是通过构造函数，其他的可以使用widget或图表方法实现。

这里是唯一可以找到最常用的定制链接和描述的地方。

默认仪表和周期

更改默认商品和周期。

最小支持的周期为1秒。

文档说明

默认可见范围 (时间范围)

更改默认周期K线的时间范围

文档说明

周期的默认可见范围

当用户更改周期时，更改K线的时间范围。 看这里的样本：

文档说明

初始时区

您可以设置默认使用的时区。 用户也可以在菜单中更改。

文档说明

图表大小

您可以将图表作为元素放置在网页上或使用全屏模式。

宽度和高度

全屏模式

自动尺寸

图表颜色

自定义图表的颜色，使其完美适合您的网站。

1. 工具栏颜色 - 文档说明
2. 图表颜色 - 文档说明

指标

1. 限制1个图表布局的指标量 - 文档说明
2. 限制显示和可以添加的指标 - 文档说明
3. 在服务器上添加您自己的指标 - 文档说明
4. 更改指标的默认属性 - 文档说明
5. 更改默认属性（立即生效） - 文档说明

绘图

1. 限制哪些绘图可以被显示或被添加 - 文档说明
2. 更改绘图的默认属性 - 文档说明
3. 更改默认属性（立即生效） - 文档说明

语言

选择图表库20多个翻译中的一个。文档说明

注意：语言是在创建图表时设置的。如果没有重新创建图表，就无法更改。

数字和日期的格式化

1. 更改十进制数字 - 文档说明
2. 为数据和时间设置自定义格式化方法 - 文档说明
3. 价格根据商品信息进行格式化 - 文档说明

图表的默认属性

您可以更改属性对话框中显示的任何属性。

1. 初始化 - 文档说明
2. 立即生效 - 文档说明

服务器的快照

TradingView允许您在其服务器上保存快照，但如果您希望您也可以更改它。

文档说明

显示/隐藏图表的元素

如果您不需要图表的某些元素 (工具栏、按钮或其他控件), 您可以隐藏它们。

1. 大多数图表元素可以通过功能集使用shown/hidden
2. 您可以添加自己的CSS - 文档说明

图表底部的时间范围

时间范围是K线的时间段和优先显示时段的周期。 您可以自定义列表。
文档说明

收藏的周期/图表样式的初始化列表

默认情况下，您可以选择在顶部工具栏上显示什么周期和图表样式。如果在 功能集中设置 `items_favoriting` 为 enabled，则允许用户改变他们。

文档说明

菜单中显示周期

1. 在datafeed的配置对象中提供了完整的周期列表 -文档说明
2. 根据商品信息在列表中启用或禁用周期 - 文档说明
3. 可以设置喜欢的周期的初始列表 - 文档说明

成交量指标

尽管有其他指标，如果仪表支持，则默认添加成交量指标 文档说明。
您可以禁用此特性 功能集

上下文菜单

您可以向上下文菜单添加新元素或隐藏现有项目。

文档说明

定制工具栏上的按钮

您可以将自己的按钮添加到图表的顶部工具栏上。

文档说明

观察列表

可以为观察列表选择默认商品，并根据需要设置只读状态。

文档说明

新闻资讯

您可以附加任何RSS订阅，甚至可以根据金融工具类型选择feed。

文档说明

5、交易终端

5-1、交易终端简介

交易终端

💹 此页面上的所有内容仅与交易终端相关。

交易终端是一个即用型产品，为那些想要有一个全功能图表解决方案，以及从图表交易的能力。该产品基于图表库，并包含其所有功能，而且还包含一大堆新功能。交易终端资源链接。

该产品正在开发中，因此一些功能还没有。 它们被标记为：🕓

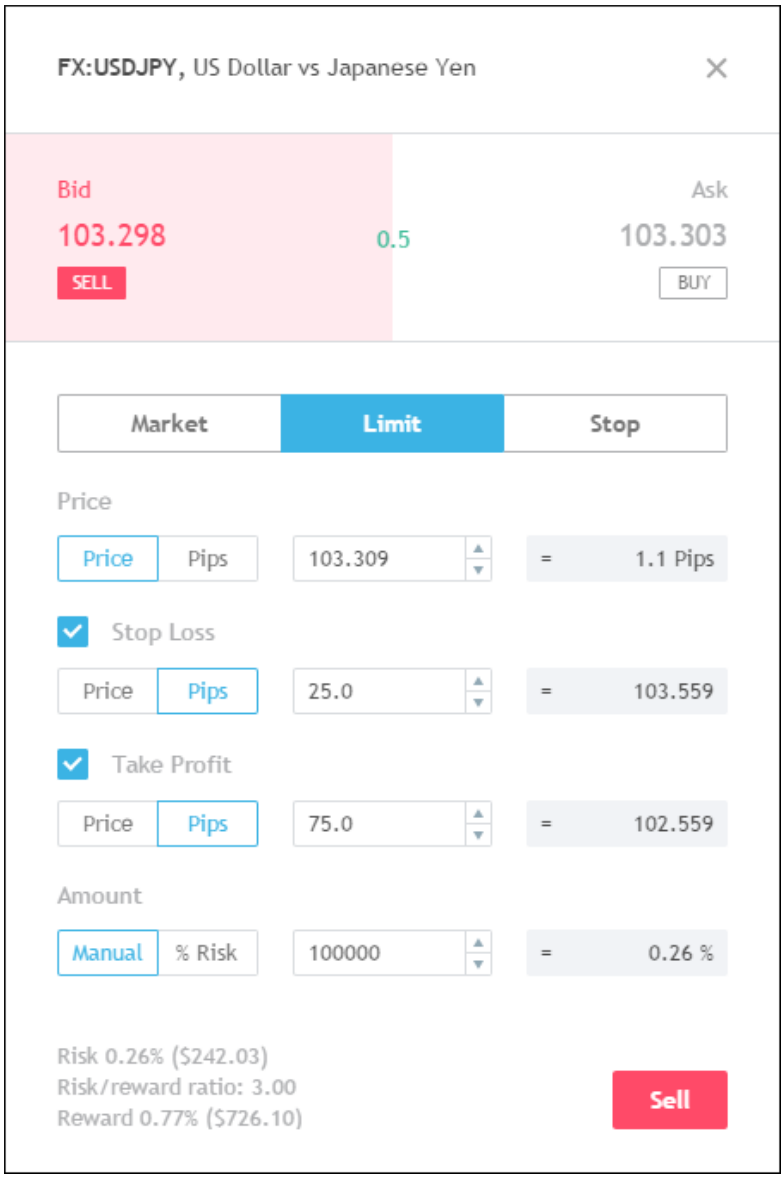
交易终端功能

交易功能

您可以通过图表进行交易，您需要做的所有工作就是实现您的经纪商API并将其接入到图表widget。

高级订单对话框

完全可定制的订单对话框允许下达市价/限价/止损/限价止损订单，输入止损和止盈价格，选择到期日期并计算风险。



账户管理器

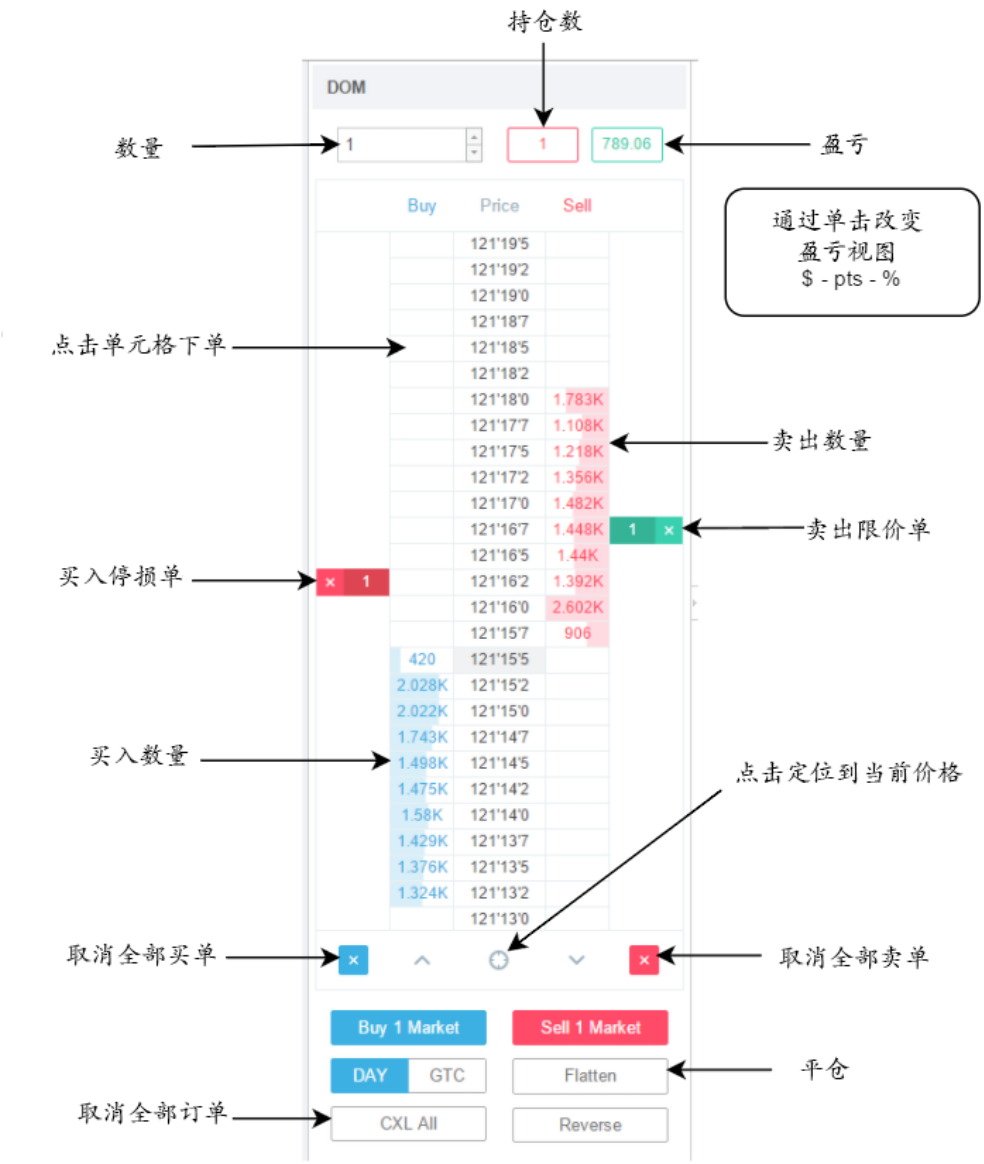
您可以在底部的交互式表格中显示订单/仓位和帐户信息。

阅读有关此功能的更多信息:

- 如何启用账户管理器

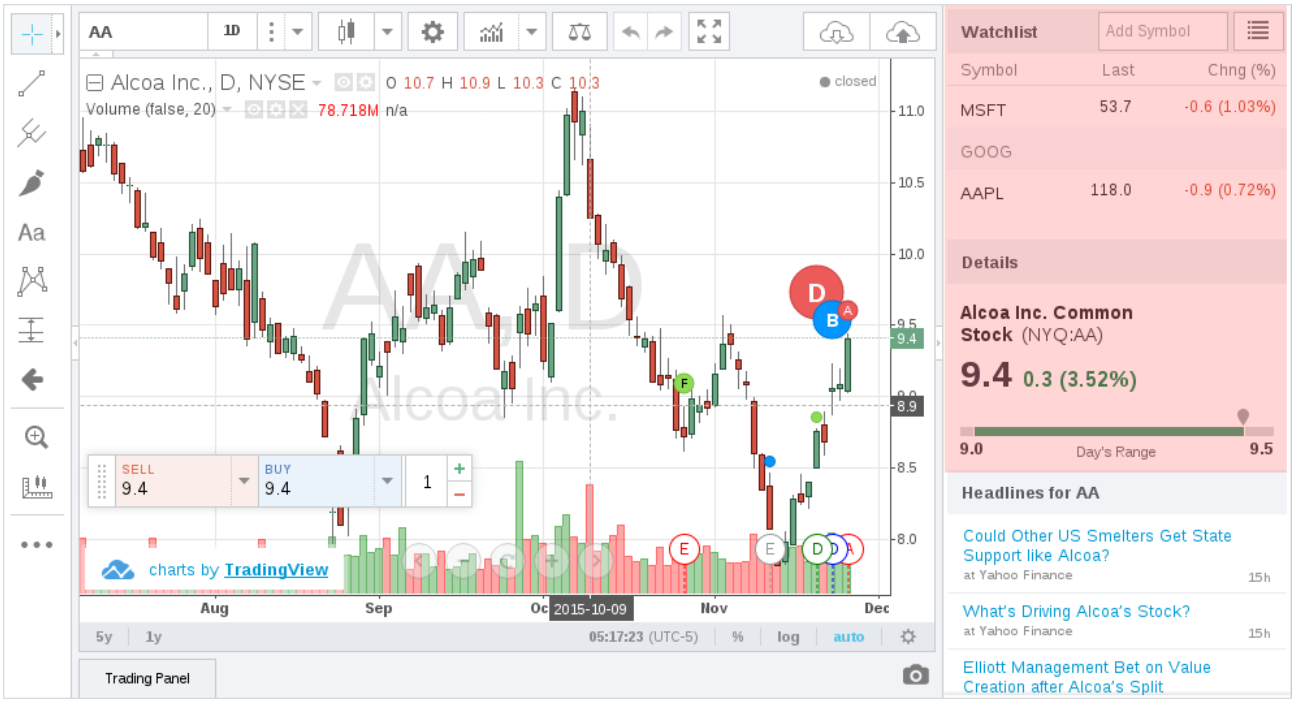
DOM小部件

您可以在交互式DOM中显示订单/持仓以及Level-2数据。



侧边栏报价（商品详情和观察列表）

在交易终端中，您可以拥有观察列表和商品详情窗口小部件（请参阅下面的快照）功能。



阅读有关此功能的更多信息:

- 如何启用侧边栏报价
- 如何提供报价数据：取决于您使用什么样的数据集成 –JS API 或 UDF

侧边栏市场新闻Feed

您可以在图表的侧栏中直接显示新闻提要。我们对新闻Feed的支持是灵活的：例如，您可以为不同类型的商品提供不同的Feed，等等。

阅读有关此功能的更多信息:

- 如何启用侧边栏新闻
- 如何设置要使用的Feed

多图表布局

在同一个小部件中，您可以同时显示多个图表。 这使您的用户能够使用更广泛的策略，以及更广泛的市场观点的能力。您不必做任何事情来启用或调整它：它可以开箱即用。

日本图表类型：卡吉图、砖形图、OX（点数）图、新价线

这些类型的图表将可以开箱即用，就像Heikin Ashi（平均K线图）在图表库中可用一样。

量能分布图：

这项指标将需要一些服务器端的支持。准备好后，我们会提供更多的细节。

绘图工具模板：

此功能将需要您的后端支持。 我们将更新我们的开源数据后端以支持此功能，因此请考虑使用它来最大限度地减少您的工作。

如何使用文档

由于交易终端基于图表库，我们决定将文档合并到单个维基中。 所以所有的文档都在一个地方。 您唯一应该记住的是，特定于交易终端的功能标有这个可爱的绿色标记： 。

也可以看看

- 如何将您的交易接口连接到图表
- 交易终端专用的Widget方法
- 用于交易终端的Widget构造函数参数

5-2、经纪商 API

经纪商 API

💹 此页面上的所有内容仅与交易终端相关。

经纪商 API 是支持实时交易的关键组件。其主要目的是将图表库与您的交易逻辑联系起来。对 `JS` 而言，它是一个预期暴露特定接口的 `对象` 。下面是经纪商 API 的**方法**列表。

必要的方法

constructor(host)

经纪商 API 的构造函数通常需要交易主机。

positions: `Promise<Position[]>`

交易终端调用此方法来请求仓位。

orders: `Promise<Order[]>`

交易终端调用此方法来请求订单。

executions(symbol): `Promise<Execution[]>`

交易终端调用此方法来请求执行。

trades: `Promise<Trade[]>`

交易终端调用此方法来请求交易（个别仓位）。

chartContextMenuActions(e)

`e` 是浏览器传递的上下文对象

图表可以在菜单中拥有 `交易` 子菜单。此方法返回子菜单的项目列表。格式与 `buttonDropdownItems` 相同。

connectionStatus()

通常你不需要返回 `1` 以外的值，因为当你创建 widget 时经纪商已经连接。如果要在加载数据时在底部面板中显示一个微调控制器(spinner)，则可以使用它。 可能的返回值是：

```
ConnectionStatus.Connected = 1;
ConnectionStatus.Connecting = 2;
ConnectionStatus.Disconnected = 3;
ConnectionStatus.Error = 4;
```

isTradable(symbol): `Promise<boolean | IsTradableResult>`

该方法是浮动交易面板所必需的。通过面板进行交易的能力取决于这个方法的返回值是：`true` 还是 `false` 。 如果所有商品都可以交易，则不需要实现这个方法。

如果要显示自定义消息显示无法交易商品的原因，则可以返回对象 `IsTradableResult` 。 它只有两个键：tradable（ `true` 或 `false` ）和 reason（ `string` ）。

accountManagerInfo()

此方法用于返回账户管理器的信息。请参阅账户管理器了解更多信息。

placeOrder(order)

方法在用户想要下订单时调用。订单预先填写了部分或全部信息。

modifyOrder(order)

1. `order` 是要修改的订单对象

方法在用户想要修改现有订单时被调用。

cancelOrder(orderId, silently)

调用这个方法来取消 `id` 的订单。

如果 `silently` 是 `true` ，则不显示任何对话框。

cancelOrders(symbol, side, ordersIds, silently)

1. `symbol` - symbol string
2. `side` : Side 或 `undefined`
3. `ordersIds` - `symbol` 和 `side` 收集的 ids。如果 `silently` 是 `true` ，则不显示任何对话框。

这个方法被调用来取消 `symbol` 和 `side` 的多个订单。

如果 `silently` 是 `true` ，则不显示任何对话框。

editPositionBrackets(positionId, brackets)

1. `positionId` 要修改的现有仓位 ID
2. `brackets` - 新的包围单 (可选).

如果启用了 `supportPositionBrackets` 配置标志，则请求此方法，可显示一个用于编辑止盈和止损的对话框。

closePosition(positionId)

如果启用了 `supportClosePosition` 配置标志，则请求此方法，可通过仓位 id 平仓。

reversePosition(positionId)

如果启用了 `supportReversePosition` 配置标志，则请求此方法，可通过仓位 id 反转仓位。

editTradeBrackets(tradeId, brackets)

1. `tradeId` 是要修改的交易 id
2. `brackets` - 新的包围单 (可选).

如果启用了 `supportTradeBrackets` 配置标志，则请求此方法，应显示一个用于编辑止盈和止损的对话框。

closeTrade(tradeId)

如果启用了 `supportCloseTrade` 配置标志，则请求此方法，可通过交易 id 关闭交易。

symbolInfo(symbol): Deferred (或 Promise)

1. `symbol` - symbol string

此方法通过内部订单对话框调用。DOM 面板和浮动交易面板请求此方法，以获取商品信息。

返回值是具有以下属性的对象:

- `qty` - 对象拥有这 3 个属性： `min` 、 `max` 、 `step` ，用于指定数量、 `step` 和边界。
- `pipSize` - 点的大小（例如，EURUSD 为 `0.0001` ）
- `pipValue` - 对于合约账户币种的每 `1pip` 的值
- `minTick` - 最小价格变动（例如，EURUSD 为 `0.00001` ）。用于价格字段。
- `description` - 要在对话框中显示的描述
- `type` - 合约类型, 只有 `forex` 比较特殊 - 它允许在订单对话框中实现负的点数。
- `domVolumePrecision` - DOM `卖出/买入` 数量的小数位数（可选，默认为 0）
- `marginRate` - 合约的保证金要求。 `3%` 的保证金率应表示为 `0.03`

accountInfo(): Deferred (或 Promise)

此方法通过内部订单对话框调用，以获取账户信息。

只返回一个字段：

1. currencySign：字符串 - 这是账户货币的标志

一旦调用此方法，经纪商应停止提供盈利/亏损。

subscribeEquity()

如果您使用标准订单对话框并支持止损，则应实现这个方法。 `Equity` 用于计算风险百分比。

一旦调用此方法，经纪商应通过equityUpdate方法，提供 `equity(余额+P/L)` 的更新订阅。

unsubscribeEquity()

如果您使用标准订单对话框并支持止损，则应实现这个方法。

一旦调用此方法，经纪商应停止提供 `Equity` 的更新。

subscribeMarginAvailable()

如果您使用标准订单对话框并想要显示可用保证金，则应实现这个方法。

一旦调用此方法，经纪商应通过marginAvailableUpdate方法，提供可用保证金的更新。

unsubscribeMarginAvailable()

如果您使用标准订单对话框并想要显示可用保证金，则应实现这个方法。

一旦调用此方法，经纪商应停止提供可用保证金的更新。

subscribePipValue()

如果您使用标准订单对话框，则应实现这个方法。 `pipValues` 显示在订单信息中，用于计算交易价值和风险。 如果没有实现这个方法，则在订单面板/对话框中使用 `symbolInfo` 中的 `pipValue` 。

一旦调用此方法，经纪商应通过pipValueUpdate方法，提供 `pipValue` 的更新。

unsubscribePipValue()

如果使用标准订单对话框并实现 `subscribePipValue` ，则应实现该方法。

调用此方法后，经纪商应停止提供 `pipValue` 的更新。

也可以看看

- 如何连接你的交易控制器到图表
- 交易主机

5-3、交易主机

✅ 此页面上的所有内容仅适用于交易终端。

交易主机是经纪商API与图表交易子系统之间交互的API。其主要目的是用您的交易适配器与图表之间交换信息。 就 `JS` 而言，它是一个具有一组函数的 `object` 。 以下是交易主机的 **方法** 列表。

指令

showOrderDialog(order, focus): Promise

1. `order` 下单或修改
2. `focus` - 焦点常量.

显示标准订单对话框以创建或修改订单，并在按下买/卖/修改时执行处理程序。

showCancelOrderDialog(orderId, handler): Promise

1. `orderId` 要取消订单id
2. `handler` 处理取消的方法。它应该返回 `Promise`

显示一个确认对话框，并在按下 `YES/OK` 时，执行处理程序。

showCancelMultipleOrdersDialog(symbol, side, qty, handler): Promise

1. `symbol` 取消订单的商品
2. `side` -取消订单的方向
3. `qty` -取消订单的数量
4. `handler` 处理取消的方法。它应该返回 `Promise` 。

显示一个确认对话框，并在按下 `YES/OK` 时，执行处理程序。

showClosePositionDialog(positionId, handler): Promise

1. `positionId` 要平仓的仓位id
2. `handler` 处理平仓的方法。它应该返回 `Promise` 。

显示一个确认对话框，并在按下 `YES/OK` 时，执行处理程序。

showReversePositionDialog(position, handler): Promise

1. `position` 反转仓位
2. `handler` 处理反转仓位的方法。它应该返回 `Promise` 。

显示一个确认对话框，并在按下 `YES/OK` 时，执行处理程序。

showPositionBracketsDialog(position, brackets, focus): Promise

1. `position` 修改仓位
2. `brackets` (可选) 新的包围单
3. `focus` - 焦点常量

显示默认的编辑包围单对话框，并在按下MODIFY时执行处理程序。

activateBottomWidget: Promise

打开底部面板并切换到交易标签页上。

showTradingProperties()

显示属性对话框，切换当前标签页到交易标签页。

showNotification(title, text, type)

显示通知。类型可以是 `1` - 成功 或 `0` - 错误。

triggerShowActiveOrders()

触发显示活动订单。

numericFormatter(decimalPlaces)

返回具有指定小数位的Formatter。

defaultFormatter(symbol: string, alignToMinMove?: boolean = true)

返回指定合约的默认Formatter。此formatter基于SymbolInfo创建。

默认情况下，formatter将价格舍入到最低价格，但有时您可能希望禁用此舍入。例如，仓位的平均价格不应四舍五入到最低价格。让我们假设我们将一笔交易价格在 `100.25` ，另一笔交易价格在 `100.50` 。该持仓的平均价格将是 `100.375` 。 如果你使用 `defaultFormatter(symbol)` 获得formatter，那么这个formatter会将这个价格舍入为 `100.38` ，但是如果你将第二个参数设置为 `false` ，那么价格将四舍五入为 `100.50` 。

factory

`factory` 是一个对象属性。 其成员如下所述。

factory.createDelegate

创建一个Delegate对象。

factory.createWatchedValue

创建一个WatchedValue对象。

factory.createPriceFormatter(priceScale, minMove, fractional, minMove2)

创建价格Formatter。此函数的参数在另一个文章中描述。

symbolSnapshot(symbol): Promise

返回商品报价。

Getters and Setters

floatingTradingPanelVisibility: WatchedValue

返回浮动交易面板是否可见。

domVisibility: WatchedValue

返回DOM面板是否可见。

orderPanelVisibility: WatchedValue

返回订单面板是否可见。

showPricesWithZeroVolume: WatchedValue

返回是否折叠0成交量（最小和最大成交量级别）的级别。

silentOrdersPlacement: WatchedValue

返回是否可以在不显示订单窗口的情况下将订单发送给经纪商。

suggestedQty(): Object

返回的对象属性： 1. value - 获取当前值。 它返回Promise. 2. setValue - 设置新值 3. changed : Subscription

在浮动交易面板和对话框中同步数量。

setButtonDropdownActions(actions)

底部交易面板有一个带有下拉列表项目的按钮。此方法可用于替换现有项目。 1. `actions` ActionMetainfo的数组, 每个对象都代表一个下拉项。

defaultContextMenuActions()

提供默认的买/卖，显示属性操作作为默认值chartContextMenuItems返回。

defaultDropdownMenuActions(options)

提供操作的默认下拉列表。您可以在setButtonDropdownActions中使用默认操作。 您可以使用 `options` 从结果中添加/删除默认操作: 1. `showFloatingToolbar` : boolean; 1. `tradingProperties` : boolean; 1. `selectAnotherBroker` : boolean; 1. `disconnect` : boolean; 1. `showDOM` : boolean; 1. `showOrderPanel` : boolean;

数据更新

使用这些方法来通知图表它需要更新的信息。

orderUpdate(order)

在添加或更改订单时调用此方法。

orderPartialUpdate(order)

当订单未更改时调用此方法，但您添加到在账户管理器中显示的订单对象的字段已更改。 仅当您要在账户管理器中显示自定义字段时才应使用它。

positionUpdate (position)

在添加或更改持仓时调用此方法。

positionPartialUpdate (position)

当持仓未更改时调用此方法，但您添加到在账户管理器中显示的持仓对象的字段已更改。 仅当您要在账户管理器中显示自定义字段时才应使用它。

executionUpdate([execution)

添加执行时调用此方法。

fullUpdate()

所有数据都已更改时调用此方法。例如，用户帐户已更改。

plUpdate(positionId, pl)

当经纪商连接收到PL更新时调用此方法。当 `configFlags` 中设置 `supportPLUpdate` 标志时，应使用此方法。

equityUpdate(equity)

当经纪商连接收到equity时调用此方法。标准订单对话框需要此方法来计算风险。

marginAvailableUpdate(marginAvailable)

当经纪商连接收到可用保证金更新时，请调用此方法。标准订单对话框需要此方法来显示可用保证金。 当在 `configFlags` 中设置 `supportMargin` 标志时，应该使用此方法。交易终端使用subscribeMarginAvailable，订阅保证金可用更新。

tradeUpdate(trade)

在添加或更改交易时调用此方法。

tradePartialUpdate (trade)

交易未更改时调用此方法，但您添加到交易对象中以显示在客户经理中的字段将更改。

tradePLUpdate(tradeId, pl)

当经纪商连接收到交易PL更新时，请调用此方法。

pipValueUpdate(symbol, pipValues)

当经纪商连接具有 `pipValue` 更新时，请调用此方法。图表库使用subscribePipValue订阅 `pipValue` 更新。

`pipValues` 是一个包含以下字段的对象:

1. `buipPipValue` - 如果你买入 `symbol` ，值为1点
2. `sellPipValue` - 如果卖出 `symbol` ，值为1点

5-4、账户管理器

✅ 此页面上的所有内容仅适用于交易终端。

帐户管理器是一个显示交易信息的交互式表格。

它包含3个标签页：订单/持仓和帐户信息。

要创建帐户管理器，您需要提供每个标签页的列和数据。

备注1. Broker API应实现accountManagerInfo

帐户管理器Mete信息

accountManagerInfo应返回以下信息。

帐户管理器标题信息

帐户管理器的标题包括经纪商名称和帐户名称或帐户列表。

accountTitle: String

accountsList: AccountInfo数组

account: 账户信息的WatchedValue

`AccountInfo` 是以下属性的对象。

1. `id` - 帐户ID
2. `name` - 账户名称
3. `currency` - 账户货币

未设置 `currency` 时，默认值为 `USD` 。

订单页

orderColumns: 列数组

要在 `订单` 页面上显示的列描述。

您可以显示订单上的任何字段，也可以将自己的字段添加到订单对象中并显示它们。

orderColumnsSorting: SortingParameters

可选的表格排序。如未设置，则表格按第一列排序。

possibleOrderStatuses: array of OrderStatus

订单过滤器中使用的可选状态列表。如未设置，则使用默认列表。

hasHistoryColumns: array of Column

如果存在，将显示历史记录页面。之前会话的所有订单都将显示在历史记录中。

historyColumnsSorting: SortingParameters

可选的表格排序。如未设置，则表格按第一列排序。

持仓页

positionColumns: 列数组

您可以显示仓位的任何字段，或者将您自己的字段添加到仓位对象中并显示它们。

附加页面（例如: 帐户摘要）

pages: 页面数组

您可以使用 `pages` 在帐户管理器中添加新的tab页。每个tab都是一组表。

Page

`Page` 是对额外的账户管理器tab页的说明。它是包含以下字段的对象：

1. `id` : String 页面的唯一标识
2. `title` : String 页面标题。显示在tab选项卡上。
3. `tables` : Table数组.

可以在此tab选项卡中显示一个或多个表。

Table

您可以向Page添加一个或多个表。 帐户摘要表metainfo是一个包含以下字段的对象：

1. `id` : String 唯一标识
2. `title` : String 表的可选标题。
3. `columns` : 列数组
4. `getData` : Promise

此方法用于请求表的数据。它返回promise（或Deferred）并解析它返回的数据数组。

每一行都是一个对象。此对象的属性值是与列名称一一对应。

预定义字段 `isTotalRow` ，可用于在表底部显示行的总数。

1. `changeDelegate` : Delegate

此委托用于监视数据的变动并更新表。将新的用户管理器数据传递给委托的 `fire` 方法。

1. `initialSorting` : SortingParameters

可选的表格排序。如未设置，则表格按第一列排序。

注意：如果表中有多行，并且想使用 `changeDelegate` 更新一行，请确保每行中都有 `id` 字段来标识它。 如果表中只有一行，则不是必须的。

SortingParameters

具有以下属性的对象:

- `columnId` - 将用于排序的列的 `id` 或 `property` 。
- `asc` - (可选, 默认为 `false`) - 为 `true` 时,初始排序将按升序排列。

Formatters

customFormatters: 列格式的描述数组

用于定义自定义格式器的可选数组。每个描述都是一个包含以下字段的对象：

`name` : 唯一标识 `format(options)` : 用于格式化单元格值的方法。 `options` 是一个具有以下键的对象： 1. `value` - 要格式化的值 2. `priceFormatter` - 价格标准格式。 您可以使用 `format(price)` 方法来设置价格的值。 3. `prevValue` - 可选字段。 它是一个以前的值，所以你可以相应地进行比较和格式化。如果当前列具有 `highlightDiff: true` key. 4. `row` - 具有当前行中所有键/值对的对象

列描述

帐户管理器描述中最有价值的部分是其列的描述。

label

列标题。它将显示在表的标题行中。

className

可选的 `className` 被添加到每个值单元格的html标签。 您可以使用它来自定义表的样式。

以下是预定义类的列表：

class名	描述
`tv-data-table__cell--symbol-cell`	商品字段的特殊格式化方法

`tv-data-table__cell--right-align`	它将单元格值右对齐
`tv-data-table__cell--buttons-cell`	单元格按钮

formatter

用于格式化数据的格式化方法。如果没有设置 `formatter` ，则按照原样显示该值。formatter可以是默认格式或自定义格式

以下是默认格式化的列表：

名称	描述
`symbol`	用于商品字段。 它显示 `brokerSymbol` ，当单击 `symbol` 时，图表会更加 `symbol` 字段而改变。 `property` 被忽略。
`side`	用于显示交易方向：卖出或买入。
`type`	用于显示订单类型：限价/止损/限价止损/市价。
`formatPrice`	格式化价格
`formatPriceForexSup`	与 `formatPrice` 一样，但它使得价格的最后一个字符被上标。只有当合约的类型为 `forex` 时，它才起作用。
`status`	格式化状态
`date`	显示日期或时间
`localDate`	显示本地日期或时间
`fixed`	显示2位小数点的数字。
`pips`	显示1位小数点的数字。
`profit`	显示利润。它还添加了 `+` ，分隔成千位，并设置红色或绿色的单元文本颜色。

有一些特殊的格式化方法用于向表中添加按钮：

`orderSettings` 将修改/取消按钮添加到订单选项卡。始终为此格式化方法将 `modificationProperty` 值设置为 `status` 。

`posSettings` 将编辑/关闭按钮添加到仓位/净头寸选项卡

`tradeSettings` 将编辑/关闭按钮添加到个人位置选项卡。始终为此格式化方法将 `modificationProperty` 值设置为 `canBeClosed` 。

property

`property` 是用于获取显示数据对象的关键字。

sortProp

可选的 `sortProp` 是用于数据排序的数据对象的键。

modificationProperty

可选的 `modifyProperty` 是数据对象的一个关键字，它被用于修改。

notSortable

可选的 `notSortable` 可以设置为防止列的排序。

help

`help` 列的提示字符串。

highlightDiff

`highlightDiff` 可以使用 `formatPrice` 和 `formatPriceForexSup` 格式化方法来设置字段的更改。

fixedWidth

如果为 `true` ，则当数字减少时，列宽不会减小。

supportedStatusFilters

订单状态的可选数字数组，仅应用于订单列。如果可用，则该列将仅显示在状态过滤器的指定选项卡中。

以下是可能的订单状态列表：

- 0 - All
- 1 - Canceled
- 2 - Filled
- 3 - Inactive
- 5 - Rejected,
- 6 - Working

上下文菜单

contextMenuActions(e, activePageItems)

`e` : 浏览器传递的上下文对象

`activePageItems` : 当前页面的 `ActionMetainfo` 项目数组

可选方法以创建一个自定义上下文菜单。 它返回用 `ActionMetainfo` 数组解析的 `Promise` 。

5-5、交易对象和常量

交易对象和常量

💹 此页面上的所有内容仅适用于交易终端。

注意： 如果您使用TypeScript，您可以从 `broker-api.d.ts` 文件中导入本文的常量/接口/类型。

经纪商配置

configFlags: object

这是一个应该在交易终端的构造函数中传递给brokerConfig的对象。每个字段应该有一个布尔值（ `true` / `false` ）：

- `supportReversePosition`
  Default: `false`
  是否支持反转持仓。 如果经纪商不支持，图表将有反转按钮，但是它会发出反转订单。
- `supportClosePosition`
  Default: `false`
  是否支持平仓。 如果经纪商不支持，图表将有平仓按钮，但它将发出平仓订单。
- `supportReducePosition`
  Default: `false`
  是否支持在没有订单的情况下更改持仓。
- `supportPLUpdate`
  Default: `false`
  是否支持持仓损益(PL)。如果经纪商本身计算利润/损失，则应在PL更改后立即调用PLUpdate。否则，图表将通过当前交易与仓位平均价格之差计算PL。
- `supportMargin`
  Default: `false`
  是否支持保证金。如果支持保证金，当交易终端使用subscribeMarginAvailable订阅时，应该调用marginAvailableUpdate。
- `supportOrderBrackets`
  Default: `false`

是否支持订单的包围单。如果此标志为 `true` ，则图表将显示附加字段在图表和账户管理器中的订单窗口和修改按钮中。

- `supportPositionBrackets`
  Default: `false`
  是否支持仓位的包围单。如果此标志为 `true` ，则图表将显示仓位的编辑按钮，并将 `编辑持仓...` 添加到仓位的上下文菜单中。
- `supportTradeBrackets`
  Default: `false`
  是否支持单一交易的包围单。 如果此标志为 `true` ，则图表将显示用于交易（单个持仓）的编辑按钮，并将 `编辑持仓...` 添加到交易的上下文菜单中。
- `supportTrades`
  Default: `false`
  是否支持单个持仓（交易）。如果设置为 `true` ，帐户管理器中将有两个选项卡：单个持仓和净持仓。
- `requiresFIFOCloseTrades`
  Default: `false`
  是否交易账户需要以先进先出顺序结算交易。
- `supportCloseTrade`
  Default: `false`
  是否支持单个持仓（交易）的关闭。
- `supportMultiposition`
  Default: `false`
  是否支持多持仓,防止创建反转持仓的默认实现。
- `showQuantityInsteadOfAmount`
  Default: `false`
  是否将 `Amount` 更改为 `Quantity`
- `supportLevel2Data`
  Default: `false`
  是否支持Level2数据用于DOM小部件。应实现 `subscribeDepth` 和 `unsubscribeDepth` 方法。
- `supportMarketOrders`
  Default: `true`
  是否支持将市价订单添加到订单对话框。
- `supportLimitOrders`
  Default: `true`

是否支持将限价订单添加到订单对话框。

- `supportStopOrders`
  Default: `true`
  是否支持将止损订单添加到订单对话框。
- `upportStopLimitOrders`
  Default: `true`
  是否支持将止损限价订单添加到订单对话框。
- `upportStopLimitOrders`
  Default: `true`
  是否支持将止损限价订单添加到订单对话框。
- `supportMarketBrackets`
  Default: `true`
  是否支持将市价包围订单添加到订单对话框。
- `supportModifyDuration`
  Default: `false`
  是否支持修改现有订单的持续时间。默认情况下它被禁用。
- `supportModifyOrder`
  Default: `true`
  是否支持修改现有订单。 它默认启用。
- `cancellingBracketCancelsParentOrder`
  如果止损或止盈被取消，经纪商将取消主订单。
- `cancellingOnePositionBracketsCancelsOther`
  如果第一个保护订单被用户取消，经纪上也会取消第二个保护订单（止损或止盈）。

durations: 对象数组

订单的到期选项列表。这是可选的。如果您不希望在订单窗口单中显示持续时间，请不要设置它。 对象具有以下键：
`{ name, value, hasDatePicker?, hasTimePicker?, default? }`。

例子:

```
durations: [
    { name: 'DAY', value: 'DAY' },
    { name: 'WEEK', value: 'WEEK', default: true },
    { name: 'GTC', value: 'GTC' }
```

```
];
```

customNotificationFields: 对象数组

可选字段。如果您在显示通知时考虑到订单或仓位中的自定义字段，则可以使用它。

例如，如果在订单中有字段 `additionalType` ，并且希望图表在更改时显示通知，则应该设置：

```
customNotificationFields: ['additionalType']
```

orderDialogOptions

可选字段。包含订单窗口选项的对象。使用这些选项，您可以自定义订单窗口。

- `showTotal` : boolean
  使用此标志，您可以在订单窗口中的 `订单信息` 部分中将 `Trade Value` 更改为 `Total` 。
- `customFields` : (TextWithCheckboxFieldMetaInfo | CustomComboBoxMetaInfo)[];
  使用 `customFields` ，您可以向订单窗口添加其他输入字段。

例:

```
customFields: [
    {
        inputType: 'TextWithCheckBox',
        id: '2410',
        title: 'Digital Signature',
        placeHolder: 'Enter your personal digital signature',
        value: {
            text: '',
            checked: false,
        },
        customInfo: {
            asterix: true,
            checkboxTitle: 'Save',
        },
    }
```

```
]
```

customUI

此可选字段可用于您自己的标准订单窗口和添加保护对话框。 以下两个字段的值是交易终端调用以显示对话框的函数。每个函数都显示一个对话框并返回一个 `Promise` 对象，该对象应在操作完成或取消时解析。

NOTE: The returned `Promise` object should be resolved with either `true` or `false` value.
注意： 返回的 `promise` 对象应使用 `true` 或 `false` 进行解析。

```
customUI: {
    createOrderDialog?: (order: Order, focus?: OrderTicketFocusControl) =>
    createPositionDialog?: (position: Position | Trade, brackets: Brackets
}
```

Order

描述单个订单。

- `id` : String
- `symbol` : String
- `brokerSymbol` : String. 如果经纪商商品代码与TV商品代码相同，可以为空。
- `type` : OrderType
- `side` : Side
- `qty` : double
- `status` : OrderStatus
- `limitPrice` : double
- `stopPrice` : double
- `avgPrice` : double
- `filledQty` : double
- `parentId` : String. 如果订单是包围单，则parentOrderId应该包含基本订单/仓位的ID。
- `parentType` : ParentType
- `duration` : OrderDuration

- `customFields` : CustomInputFieldsValues

Position

描述单个仓位。

- `id` : String. 通常id应等于brokerSymbol
- `symbol` : String
- `brokerSymbol` : String. 如果经纪商商品代码与TV商品代码相同，可以为空。
- `qty` : 正数
- `side` : Side
- `avgPrice` : number

Trade

描述单笔交易（个别仓位）。

- `id` : String. 通常id应等于brokerSymbol
- `symbol` : String
- `date` : number (UNIX时间戳，毫秒单位)
- `brokerSymbol` : String. 如果经纪商商品代码与TV商品代码相同，可以为空。
- `qty` : 正数
- `side` : Side
- `price` : number

Execution

描述单个执行。执行是图表上显示交易信息的标记。

- `symbol` : String
- `brokerSymbol` : String. 如果经纪商商品代码与TV商品代码相同，则可以为空。
- `price` : number
- `time` : Date
- `side` : Side
- `qty` : number

ActionMetainfo

描述一个操作将其放入下拉菜单或菜单中。它有如下结构:

- `text` : String
- `checkable` : Boolean. 如果需要复选框，将其设置为true。
- `checked` : Boolean
- `enabled` : Boolean
- `action` : function. 当用户单击该项目时执行的方法。它有一个参数 - 复选框的值（如果存在）。

OrderType

用于描述订单状态的字符串常量。

```
OrderType.Limit = 1
OrderType.Market = 2
OrderType.Stop = 3
OrderType.StopLimit = 4
```

Side

用于描述订单/交易执行方向的字符串常量。

```
Side.Buy = 1
Side.Sell = -1
```

ParentType

用于描述包围单类型的字符串常量。

```
ParentType.Order = 1
```

```
ParentType.Position = 2
```

OrderStatus

用于描述订单状态的字符串常量。

```
// 订单被取消
OrderStatus.Canceled = 1
// 订单已完全执行
OrderStatus.Filled = 2
// 创建了包围单，但等待基本订单全部成交
OrderStatus.Inactive = 3
// 提交订单中
OrderStatus.Placing = 4
// 由于某种原因拒绝订单
OrderStatus.Rejected = 5
// 已创建订单但尚未执行
OrderStatus.Working = 6
```

OrderDuration

订单的持续时间或到期时间。

- `type` ：传递给durations列表中的字符串标识符
- `datetime` number

DOMEObject

描述单个DOM响应的对象。

- `snapshot` : Boolean 正值意味着以前的数据应该被清理
- `asks` : DOMELevel数组按价格按升序排序
- `bids` : DOMELevel数组按价格按升序排序

DOMELevel

单个DOM价格level对象。

- `price` : number
- `volume` : number

OrderTicketFocusControl

打开标准 `订单` 对话框或 `仓位` 对话框时用于设置焦点的常量。

```
// 设置焦点到止损控制
OrderTicketFocusControl.StopLoss = 1
// 设置焦点到止损价格
OrderTicketFocusControl.StopPrice = 2
// 设置焦点到获利控制
OrderTicketFocusControl.TakeProfit = 3
```

Brackets

`stopLoss` : number

`takeProfit` : number

Formatter

具有 `format` 方法的对象可用于将数字格式化为字符串。

CustomInputFieldsValues

包含特定于代理的用户输入结果的对象（例如数字签名）。 自定义字段有两种可能的值：带有复选框的输入字段和自定义组合框。

```
{
    [fieldId: string]: TextWithCheckboxValue | string
}
```

`TextWithCheckboxValue` 是一个用于带有复选框的输入字段的对象，它有两个属性:

- `text` : string
- `checked` : boolean

自定义组合框的结果始终是用户输入的 `字符串` 。

TextWithCheckboxFieldMetaInfo

描述带有复选框的自定义输入字段的对象。

- `inputType` : 'TextWithCheckBox'
- `id` : string
- `title` : string
- `placeHolder?` : string
- `value` : TextWithCheckboxValue
- `validator?` : (value: string) => PositiveBaseInputFieldValidatorResult | NegativeBaseInputFieldValidatorResult
- `customInfo` : TextWithCheckboxFieldCustomInfo

TextWithCheckboxValue

包含复选框的自定义输入字段初始值的对象。

- `text` : string
- `checked` : boolean

TextWithCheckboxFieldCustomInfo

使用复选框描述自定义输入字段的其他设置的对象。 使用 `asterix` 属性可以管理输入类型。如果 `asterix` 设置为 `true` ，则将呈现密码输入。

- `checkboxTitle` : string
- `asterix` : boolean

CustomComboBoxMetaInfo

描述自定义组合框的对象。

- `inputType` : 'ComboBox'
- `id` : string
- `title` : string
- `items` : CustomComboBoxItem[]

CustomComboBoxItem

描述自定义组合框的项目的对象。

- `text` : string
- `value` : string

PositiveBaseInputFieldValidatorResult

描述正数验证结果的对象。

- `valid` : true

NegativeBaseInputFieldValidatorResult

描述负数验证结果的对象。

- `valid` : false
- `errorMessage` : string

6、储存和载入图表

储存和载入图表

图表库支持使用2个级别的API，用以保存/加载图表和指标模板：

1. **低级**：保存/加载功能由 `save()` / `load()` 方法和 `createStudyTemplate()` / `applyStudyTemplate()` 方法提供实现。 应该考虑服务器上的存储数据。 您可以将JSON保存在您希望的位置。 例如，您可以将它们嵌入到已保存的页面或用
2. **高级**：图表库能够从您指向的存储中保存/加载图表和指标模板。 我们使用Python和PostgreSQL创建了一个小型存储示例，可以在 我们的GitHub中找到。 您可以使用它并在自己的服务器上运行，这样您就可以控制所有用户的已保存数据。

使用高级别保存/加载

下面是想拥有自己图表存储功能所需的步骤：

1. 克隆我们的资源到您的主机上
2. 运行数据服务或使用我们的演示服务。对于不熟悉Django的人来说，这是一个简短的待办事项列表。
    1. Install Python 3.x and Pip.
    2. Install PostgreSQL or some other Django-friendly database engine.
    3. 转到图表存储文件夹并运行 `pip install -r requirements.txt`
    4. 转到charting_library_charts文件夹，并在settings.py中设置数据库连接（参见 `DATABASES` @ line #12）。请记住在PostgreSQL中创建适当的数据库。
    5. 运行 `python manage.py migrate` 。 这将创建没有任何数据的数据库schema。
    6. 运行 `python manage.py runserver` 运行您的数据库的TEST实例。 不要在生产环境中使用上面的命令。 使用一些其他的东西（例如Gunicorn）
3. 设置您的图表库页面: 设置 `charts_storage_url = url-of-your-charts-storage` ，并在widget的.ctor中设置 `client_id` 和 `user_id` （见下文）。
4. 请享用！

备注：手动填充/编辑数据库并不是理想做饭。 请避免这个，因为你可能会伤害Django。

开发自己的后端

图表库将HTTP/HTTPS命令发送到
`charts_storage_url / charts_storage_api_version / charts?client = client_id& user = user_id`
。 `charts_storage_url` , `chart_storage_api_version` , `client_id` *和* `user_id` *是widget构造函数的参数。* 您应该执行4个请求的处理：保存图表/加载图表/删除图表/列出图表。

列出图表

GET REQUEST:
```
charts_storage_url/charts_storage_api_version/charts?
client=client_id&user=user_id
```

响应：JSON对象

1. `status` : `ok` 或 `error`
2. `data` : 对象数组
    1. `timestamp` : 保存图表时的UNIX时间（例如，1449084321）
    2. `symbol` : 图表的商品（例如， `AA` ）
    3. `resolution` : 周期（例如， `D` ）
    4. `id` : 图表的唯一整数标识符（例如， `9163` ）
    5. `name` : 图表名称（例如， `Test` ）

存储图表

POST REQUEST:
```
charts_storage_url/charts_storage_api_version/charts?
client=client_id&user=user_id
```

1. `name` : 图表名称
2. `content` : 图表内容
3. `symbol` : 图表商品(例如, `AA`)
4. `resolution` : 图表周期 (例如, `D`)

响应：JSON对象

1. `status`：`ok` 或 `error`
2. `id`：图表的唯一整数标识符（例如，`9163`）

存储为图表

POST REQUEST:

```
charts_storage_url/charts_storage_api_version/charts?
client=client_id&user=user_id&chart=chart_id
```

1. `name`：图表名称
2. `content`：图表内容
3. `symbol`：图表商品(例如，`AA`)
4. `resolution`：图表周期 (例如，`D`)

响应：JSON对象

1. `status`：`ok` 或 `error`

加载图表

GET REQUEST:

```
charts_storage_url/charts_storage_api_version/charts?
client=client_id&user=user_id&chart=chart_id
```

响应：JSON对象

1. `status`：`ok` 或 `error`
2. `data`：对象数组
    1. `timestamp`：保存图表时的UNIX时间（例如，1449084321）
    2. `symbol`：图表的商品（例如，`AA`）
    3. `resolution`：周期（例如，`D`）
    4. `id`：图表的唯一整数标识符（例如，`9163`）
    5. `name`：图表名称（例如，`Test`）

删除图表

DELETE REQUEST:

```
charts_storage_url/charts_storage_api_version/charts?
client=client_id&user=user_id&chart=chart_id
```

响应：JSON对象

1. `status`：`ok` 或 `error`

使用演示图表和指标模板存储

我们正在运行演示图存储服务，以便在构建图表库时立即保存/加载图表。 这是链接 http://saveload.tradingview.com。 请注意，它是按原样提供的，因为它是一个演示。

我们不保证其稳定性。 另请注意，我们会定期删除存储中的数据。

管理对已保存图表的访问

您负责用户能够查看和加载的图表。 用户可以查看/加载和用户相同的 `client_id` 和 `user_id` 的图表。 `client_id` 是用户组的标识符。 预期用途是当您有几组用户或有少数几个使用相同图表存储的网站时。 所以通常的做法是设置 `client_id = your-site's-URL` 。这由你来决定。

`user_id` 是用户的唯一标识符。 用户ID属于特定的 `client_id` 组。 您可以单独为每个用户设置它（每个用户的私有存储），也可以为所有用户或用户组（公共存储）设置它。

这里有一些例子：

client_id	user_id	作用
您的站点URL或其他链接	唯一用户ID	每个用户都有一个其他用户看不到的私有图表存储。

您的站点URL或其他链接	所有用户都为相同值	每个用户都可以查看和加载任何已保存的图表。
您的站点URL或其他链接	注册用户的唯一用户ID以及匿名用户的单独设置	每个注册用户都有一个其他用户看不到的私有图表存储。 所有匿名用户共享一个存储。

7、创建自定义指标

如何显示您的数据作为一个指标

如果您想要在图表上显示一些数据，例如指标，则此处为食用说明。

请遵循以下几个步骤:

1. 为您的数据创建一个新的ticker，并设置您的服务器返回此ticker有效的SymbolInfo。
2. 设置服务器以返回此ticker的有效历史数据。
3. 使用以下指标模板并填写所有占位符(placeholder)的值：名称，说明和代码。 如果需要，还可以修改绘图的默认样式。

```
{
    // 将<study name>替换为您的指标名称
    // 它将由图表库内部使用
    name: "<study name>",
    metainfo: {
        "_metainfoVersion": 40,
        "id": "<study name>@tv-basicstudies-1",
        "scriptIdPart": "",
        "name": "<study name>",
        // 此说明将显示在指标窗口中
        // 当调用createStudy方法时，它也被用作“name”参数
        "description": "<study description>",
        // 该描述将显示在图表上
        "shortDescription": "<short study description>",

        "is_hidden_study": true,
        "is_price_study": true,
        "isCustomIndicator": true,

        "plots": [{"id": "plot_0", "type": "line"}],
        "defaults": {
            "styles": {
                "plot_0": {
                    "linestyle": 0,
                    "visible": true,

                    // 绘图线宽度
                    "linewidth": 2,

                    // 绘制类型:
```

```
                    //    1 - 直方图
                    //    2 - 线形图
                    //    3 - 十字指针
                    //    4 - 山形图
                    //    5 - 柱状图
                    //    6 - 圆圈图
                    //    7 - 中断线
                    //    8 - 中断区块
                    "plottype": 2,

                    // 显示价格线?
                    "trackPrice": false,

                    // 绘制透明度，百分比。
                    "transparency": 40,

                    // 以#RRGGBB格式绘制颜色
                    "color": "#0000FF"
                }
            },

            // 指标输出值的精度
            // (小数点后的位数)。
            "precision": 2,

            "inputs": {}
        },
        "styles": {
            "plot_0": {
                // 输出的名字将在样式窗口显示
                "title": "-- output name --",
                "histogramBase": 0,
            }
        },
        "inputs": [],
    },

    constructor: function() {
        this.init = function(context, inputCallback) {
            this._context = context;
            this._input = inputCallback;

            // 定义要绘制的商品。
            // 商品应该是一个字符串。
            // 您可以使用PineJS.Std.ticker（this._context）获取所选商品的代码。
            // 例，
            //    var symbol = "AAPL";
            //    var symbol = "#EQUITY";
            //    var symbol = PineJS.Std.ticker(this._context) + "#TEST";
            var symbol = "<TICKER>";
            this._context.new_sym(symbol, PineJS.Std.period(this._context)
```

```
        };

        this.main = function(context, inputCallback) {
            this._context = context;
            this._input = inputCallback;

            this._context.select_sym(1);

            // 您可以在PineJS.Std对象中使用以下内置函数：
            //     open, high, low, close
            //    hl2, hlc3, ohlc4
            var v = PineJS.Std.close(this._context);
            return [v];
        }
    }
}
```

1. 将custom_indicators_getter添加到widget构造函数中。它的值是一个方法，返回带有自定义指标列表的Promise对象。

```
{
    custom_indicators_getter: function(PineJS) {
        return Promise.resolve([
            // *** 您的指标对象，通过模板创建 ***
        ]);
    },
}
```

1.更新widget的初始化代码，以便在图表准备就绪时创建此指标。

例

1. 使用custom_indicators_getter选项，将指标添加到图表库。
2. 更改widget的初始化代码。下面是一个例子。

```
widget = new TradingView.widget(/* ... */);

```

```
widget.onChartReady(function() {
    widget.chart().createStudy('<indicator-name>', false, true);
});
```

请求另一个商品代码的数据

假设您希望在图表上显示用户的收益曲线。你必须做以下事情：

- 为新的代码创建一个名称。 假设它为 `#EQUITY` 代码。 您可以使用您想像到的任何名字。
- 更改服务器的代码以将此代码作为有效商品。 为此返回最小的有效SymbolInfo。
- 使服务器返回有效的历史记录。 即，服务器可以询问您的数据库的股权历史记录，并返回此数据，就像返回普通商品的历史记录一样（例如 `AAPL`）。
- 采用上述指标模板，创建指标文件（或向现有指标文件添加新指标）。

例如：

```
{
    name: "Equity",
    metainfo: {
        "_metainfoVersion": 40,
        "id": "Equity@tv-basicstudies-1",
        "scriptIdPart": "",
        "name": "Equity",
        "description": "Equity",
        "shortDescription": "Equity",

        "is_hidden_study": true,
        "is_price_study": true,
        "isCustomIndicator": true,

        "plots": [{"id": "plot_0", "type": "line"}],
        "defaults": {
            "styles": {
                "plot_0": {
                    "linestyle": 0,
                    "visible": true,

                    // 使线条变细
                    "linewidth": 1,

                    // 绘制类型为线性图
```

```
                    "plottype": 2,

                    // 显示价格线
                    "trackPrice": true,

                    "transparency": 40,

                    // 为图线设置深红色。
                    "color": "#880000"
                }
            },

            // 精度为小数点后一位数，如777.7
            "precision": 1,

            "inputs": {}
        },
        "styles": {
            "plot_0": {
                // 输出名字在样式窗口显示
                "title": "Equity value",
                "histogramBase": 0,
            }
        },
        "inputs": [],
    },

    constructor: function() {
        this.init = function(context, inputCallback) {
            this._context = context;
            this._input = inputCallback;

            var symbol = "#EQUITY";
            this._context.new_sym(symbol, PineJS.Std.period(this._context)
        };

        this.main = function(context, inputCallback) {
            this._context = context;
            this._input = inputCallback;

            this._context.select_sym(1);

            var v = PineJS.Std.close(this._context);
            return [v];
        }
    }
}
```

K线变色

```
__customIndicators = [
    {
        name: "Bar Colorer Demo",
        metainfo: {
            _metainfoVersion: 42,

            id: "BarColoring@tv-basicstudies-1",

            name: "BarColoring",
            description: "Bar Colorer Demo",
            shortDescription: "BarColoring",
            scriptIdPart: "",
            is_price_study: true,
            is_hidden_study: false,
            isCustomIndicator: true,

            isTVScript: false,
            isTVScriptStub: false,
            defaults: {
                precision: 4,
                palettes: {
                    palette_0: {
                        // 调色板颜色
                        // 将其更改为您喜欢的默认颜色，
                        // 但请注意，用户可以在“样式”选项卡中更改它们
                        // 指标属性
                        colors: [
                            { color: "#FFFF00" },
                            { color: "#0000FF" }
                        ]
                    }
                }
            },
            inputs: [],
            plots: [{
                id: "plot_0",

                // plot类型应设置为 'bar_colorer'
                type: "bar_colorer",

                // 这是定义的调色板的名称
                // 在 'palettes' 和 'defaults.palettes' 部分
                palette: "palette_0"
            }],
            palettes: {
                palette_0: {
```

```
                    colors: [
                        { name: "Color 0" },
                        { name: "Color 1" }
                    ],

                    // 值之间的映射
                    // 返回脚本和调色板颜色
                    valToIndex: {
                        100: 0,
                        200: 1
                    }
                }
            }
        },
        constructor: function() {
            this.main = function(context, input) {
                this._context = context;
                this._input = input;

                var valueForColor0 = 100;
                var valueForColor1 = 200;

                // 在这里执行计算并返回其中一个常量
                // 在'valToIndex'映射中指定为键
                var result =
                    Math.random() * 100 % 2 > 1 ? // 我们随机选择一个颜色值
                        valueForColor0 : valueForColor1;

                return [result];
            }
        }
    }
];
```

7、最佳做法

创造最好的用户体验

我们喜欢我们的图表。 我们希望它们成为整个HTML5世界中最好，最美丽，响应最快，功能最强大的图表。 我们正在努力实现这些目标。

我们了解所有关于我们的图表以及使用它们创建最佳用户体验，我们很乐意与您分享我们的知识。 本文档介绍了将图表库集成到您的网站/应用程序中的几种最佳实践。 重点是始终考虑您的用户和他们的体验。

1. 了解图表库是什么，不是什么

图表库是一个能够显示价格，形状和技术分析工具的图表组件。 我们的图表库使图表变得神奇，仅此而已。 如果您想要一些额外的功能（如聊天，特殊的商品列表，最热门交易栏，广告等），最好的方法是在图表之外实现它们。 但是，如果要将外部功能与图表库链接，可以使用图表库的API链接它们。

2. 返回与库请求一样多的K线

图表库将向您的后端询问数据，并为提供每个请求所需的数据范围界限。 请遵守这些边界并尽可能完整地返回填充此范围的数据。 不要返回范围外的数据。 如果要扩展库请求的默认数据范围，请使用我们的JS API（请参阅calculateHistoryDepth）。

3. 返回与库请求一样多的标记

与上述K线相同。只发送符合要求范围的标记。

4. 不要覆盖calculateHistoryDepth()以获取超过2个屏幕的数据

图表库避免加载用户没有要求的内容。在图表中加载更多的K线，意味着需要更多的CPU和内存。这意味着的响应效率会被降低。

5. 不要让你的图表看起来一团糟

用户喜欢漂亮的图表。我们也是。请务必在自定义尺寸或样式时保持图表看起来不错。避免嵌入看起来与整个图表样式不同的自定义控件。

6. 避免制作非常小的图表

图表库支持的最小尺寸是 `600x600px` 。避免使图表变的更小，因为它会看起来一团糟。请使用 `mobile` 预设，或者隐藏一些工具栏。

7. 使用适当的语言

图表库已翻译成数十种语言。使用符合用户需求的语言。

8. 如果您遇到问题

我们总是渴望帮助你。 但是，不幸的是，我们真的很忙，所以我们没有太多时间。 请帮助我们有效地度过时间，并始终将您的图表库的版本更新为最新的 `unstable` 版本，以检查问题是否仍然发生。 如果有，请与我们联系。

另外，检查您传递给图表库的数据，并确保它符合我们的要求，如文档中所述。 要特别注意SymbolInfo的内容，因为它是最常见的发生错误的地方（根据我们的统计）。

您可以看我们的输出demo data service 并将其与您的对比，以确保您的后端特性是正确的。

在开发过程中始终在Widget构造函数选项中使用 `debug：true` ，并在生产环境中将其删除，以使代码更快地工作。

9. 阅读文档

我们花了很多时间为您创建这些文档，使您的生活更轻松。 请试一试。

10. 为您的解决方案选择适当的数据传输

注意JS API和UDF之间的差异，并选择最符合您需求的API。 如果您需要真正快速的数据更新或数据流传输，请勿使用UDF。 如果您的后端有十几个商品，请勿使用UDF用于数据分组

（请参阅 `supports_group_request` ）。

11. 不要尝试嗅探我们的代码并使用未记录的功能

我们的文档中未提及的所有功能都是可以更改的主题，没有任何警告和向后兼容性。此外，您签署的法律协议严格禁止更改源代码。

12. 不要在您的生产网站上使用我们的演示数据源

这个数据源只是一个演示，不适合实际使用。它可能不稳定，不能承受过大的负载。

13. 使用API进行自定义。 避免编辑CSS。

我们不保证CSS选择器的向后兼容性。

14. 发送到客户端时，将服务器设置为gzip文件

这是静态HTML内容的常见最佳做法。 加载图标库的HTML文件会减少用户的等待时间。

15.设置charting_library.min.js的最短到期时间

除了添加到HTML文件的 `charting_library.min.js` 之外，图表库中的所有文件名中都包含哈希值。 将图表库更新为较新版本时，所有文件的名称也会更改。 如果浏览器从缓存中加载 `charting_library.min.js` ，则该文件中的所有链接都将被破坏。 应将此文件的到期时间设置为最小值，以确保浏览器不缓存该文件。

9、经常被问到的问题

数据问题

1. 如何连接我的数据？ 如何添加新的商品代码?

图表库应由技术专家使用。 它需要JavaScript的高级技能和对WEB协议的深入了解。 您应该了解自己，或者雇用知道这个的人。另外，如果您没有WEB API，则至少需要服务器语言程序员和系统管理员才能在服务器端实现WEB API。

我们做了很多工作，使连接数据的过程简单明了。

首先，您需要阅读并理解本文: [[How to connect my data|https://github.com/tradingview/charting_library/wiki/How-To-Connect-My-Data]]

如果还有问题，请打开 [[Demo Chart|https://demo_chart.tradingview.com]], 然后打开Debugger-Network，并通过`demo_feed'过滤请求。您将在[[UDF]]格式中看到所有请求和相应的响应。

2. 是否有JS API实现的例子?

如果您看下图，您将看到UDF适配器作为JS API实现的示例。 它的代码没有被缩小，它的写法使我们的客户能够理解它的工作原理。

[[Scheme|https://github.com/tradingview/charting_library/wiki/How-To-Connect-My-Data#udf-scheme]]

3. 是否有WebSocket数据传输的例子?

我们没有这样的一体化的例子，但我们仍然希望在将来做出这个例子。**4. 是否有ASP.NET，Python，PHP等后端数据源的例子。?**

我们所用的后端Feed的唯一示例是用于NodeJS的JavaScript。 你可以在这里找到它：[[yahoo_datafeed|https://github.com/tradingview/yahoo_datafeed]]**5. 如何显示存储在TXT/CSV/Excel文件中的数据?**

首先，图表库并不用于显示文件中的数据。它用于显示来自服务器的K线数据。其次，您应该记住，根据协议，您只能在公共网站上使用图表库。如果您仍然想使用文件作为数据源，则需要执行以下步骤： 1. 使用任何服务器语言编写应用程序（.NET，PHP，NodeJS，Python等）。该应用程序应读取该文件，并通过HTTP(S)以[[UDF]]格式提供数据。 注意：您可以以另一种格式提供数据，或使用websocket来传输数据，但在这种情况下，您将需要在客户端上实现[[JS-Api]]适配器。 2. 您应该具有静态IP或注册域，以便浏览器可以向您的服务器发送请求。 3. 打开`index.html`，将`demo_feed.tradingview.com`替换成你的服务器的URL。**6. 为什么我的数据没有显示/显示不正确/从服务器提取错误?**

您应该做的第一件事是打开`index.html`或你创建库widget的脚本，并在widget的初始化选项中加入：`debug: true,`。完成之后，您将在浏览器控制台中看到很多有用的信息。图表库中发生的大部分重要操作都在控制台中进行了说明。

请仔细阅读[[Symbology]]。 大部分数据错误发生在商品设置不正确。**7. 图表库不断要求数据。 如何判断数据是否完成?**

具体而言，有一个标志可以添加到服务器的响应中，它告诉库服务器上没有更多的数据。它被称为 `no_data` 为[[UDF|https://github.com/tradingview/charting_library/wiki/UDF#bars]]和`noData`为[[JS API|https://github.com/tradingview/charting_library/wiki/JS-Api#getbarssymbolinfo-resolution-from-to-onhistorycallback-onerrorcallback-firstdatarequest]]**8. 如何在图表上更改小数位数?**

请仔细阅读[[Symbology]]。小数位数是根据 `minmov` 和 `pricescale` 值计算的。**9. 如果每个时间戳都有一个单一的价格怎么办？?**

如果每个时间戳只有一个价格，您仍然可以显示数据，但显然您将无法将数据显示为K线/蜡烛线。由于图表库旨在显示不同风格的数据：蜡烛线，K线，直方图，您应该为每个时间戳提供Open, High, Low, Close和可选的Volume(成交量)。如果您只有一个价格，你可以通过`Open = High = Low = Close = price`。为了更好地查看此数据，您可以将默认图表样式更改为“Line”（请参阅GUI问题）。

GUI问题

1. 如何订阅图表事件？?

我们有几种方式来订阅这些事件：

1. 订阅与整个图表布局相关的一般事件，而不是特定图表。 [[Open article|https://github.com/tradingview/charting_library/wiki/Widget-Methods#subscribing-to-chart-events]]

2. 订阅与单个图表相关的事件 [[Open article|https://github.com/tradingview/charting_library/wiki/Chart-Methods#subscribing-to-chart-events]]

检查订阅方法的结果值。 其中一些返回 [[Subscription|https://github.com/tradingview/charting_library/wiki/Subscription]] 对象拥有 `subscribe`/`unsubscribe`方法。其他接受一个回调函数。**2. 如何将默认K线风格从蜡烛更改为另一种?**

您可以使用[[overrides|https://github.com/tradingview/charting_library/wiki/Widget-Constructor#overrides]] 的小部件构造函数。 添加`mainSeriesProperties.style`键。 您可以找到允许的值 [[this article|https://github.com/tradingview/charting_library/wiki/Overrides]]**3. 如何更改图表上的周期列表（时间周期），使其变灰?** * 在图表弹出窗口中显示的周期列表由data feed配置中的[[supported_resolutions|https://github.com/tradingview/charting_library/wiki/JS-Api#supported_resolutions]] 定义。 * 某些仪器的周期由仪器/商品信息中的 [[supported_resolutions|https://github.com/tradingview/charting_library/wiki/Symbology#supported_resolutions]] 定义。 * 如果您支持日内周期，则需要设置 [[has_intraday|https://github.com/tradingview/charting_library/wiki/Symbology#has_intraday-]] * 另外，如果您支持秒，你需要设置 [[has_seconds|https://github.com/tradingview/charting_library/wiki/Symbology#has_seconds-]] * 如果您支持日周期，你应该设置 [[has_daily|https://github.com/tradingview/charting_library/wiki/Symbology#has_daily-]] * 如果您支持周和月，你应该设置 [[has_weekly_and_monthly|https://github.com/tradingview/charting_library/wiki/Symbology#has_weekly_and_monthly-]] * 此外，您应该设置的周期，这是由您的服务器[[[intraday resolutions|https://github.com/tradingview/charting_library/wiki/Symbology#intraday_multipliers-]]和 [[seconds|https://github.com/tradingview/charting_library/wiki/Symbology#seconds_multipliers-]]。 * 如果仪器支持（`supported_resolutions`）更多的周期，可以由服务器提供（`intraday_multipliers`），其他周期由图表构建。**4. 如何隐藏GUI元素(商品、周期、按钮**

等)? * 大多数GUI元素可以使用[[Featureets]]隐藏。 请查看[Interactive map of featuresets](http://tradingview.github.io/featuresets.html)，找到您需要的内容。 * 有不可隐藏的基本元素，但如果您仍然想要摆脱这些元素，您可以使用[CSS自定义](https://github.com/tradingview/charting_library/wiki/Widget-Constructor#custom_css_url-since-14)。 请注意，DOM产品的名称，类别和标识符可能会在将来版本的产品中更改，而不会有任何通知。

其他问题

1. [[Widget|http://tradingview.com/widget/]], [[Charting Library|https://www.tradingview.com/HTML5-stock-forex-bitcoin-charting-library/]] 和 [[Trading Terminal|https://www.tradingview.com/trading-terminal/]] 都有什么区别?

[[Widget|http://tradingview.com/widget/]] 连接到tradingview数据。完美的网站，博客和论坛，你需要一个快速和免费的解决方案。集成只是简单地剪切和粘贴预制的iframe代码。它有很多显示模式。

[[Charting Library|https://www.tradingview.com/HTML5-stock-forex-bitcoin-charting-library/]] 是使用您自己的数据的图表。这是一个独立的解决方案，您可以下载，托管在您的服务器上，连接自己的数据，并在您的网站/应用程序中免费使用。

[[Trading Terminal|https://www.tradingview.com/trading-terminal/]] 是一个独立的产品，授权给经纪商。它包括所有功能的图表库可用，但它也有交易功能，多种图表布局名单，详情，新闻插件和其他先进的工具。它有它自己的许可费与它相关联的费用。

2. 如何添加自定义指标?

目前只有一种方法来添加自定义指标。 它在[[dedicated article|https://github.com/tradingview/charting_library/wiki/Creating-Custom-Studies]]中有描述。

10、版本变更点

我们将尽最大努力使每个新版本与前一版本完全兼容，但有时重大的更改需要您在升级到新版本时对代码进行微小的更改。

注意：您可以通过在浏览器控制台中执行 `TradingView.version()` *来检查图表库版本。*

以下是重大变更列表：

Version 1.15

- 功能集 `show_logo_on_all_charts` 被删除。
- 功能集 `cl_feed_return_all_data` 被删除。
- 动作 `magnetAction` 从 executeActionById 和 getCheckableActionState 中被删除。 使用 magnetEnabled 代替。
- createStudy 的 `callback` 参数被删除。
- createStudy 返回值使用 Promise 代替 `entityId` 。
- Pane-Api 的方法 `getLeftPriceScale` 被替换为 `getLeftPriceScales` 返回 scale 数组。
- Pane-Api 的方法 `getRightPriceScale` 被替换为 `getRightPriceScales` 返回 scale 数组。
- setVisibleRange 现在返回Promise对象，且删除之前的参数: `callback` 回调函数
- 已使 symbol选项**优先级**高于saved_data选项。如果不想覆盖saved_data中的**symbol**,则不要为 symbol选择分配值。

交易终端

我们改变了经纪商的互动流程。请仔细阅读以下内容，以了解应在代码中进行哪些更改来切换到新版本。

到目前为止，当用户点击 `买入/卖出/修改` 按钮时，交易终端会调用代理适配器的方法 (例如: `placeOrder` , `modifyOrder`)。在调用这些方法时，交易终端会传递 `silently` 参数。 当 `silently` 设置为 `true` 时, 代理适配器可以在不显示对话框的情况下发送订单。当它被设

置为 `false` 时, 代理适配器必须从 `host` 调用一个方法来显示订单窗口 (或仓位变更对话框)。

从 1.15 开始，交易终端单独显示所有对话框，并调用代理适配器的方法向代理的服务器发送订单或头寸。这种变化的原因是我们添加了一个订单面板，可以随时用于下订单。 如果您使用自己的订单对话框，那么您仍然需要在代理适配器的方法中进行更改, 另外您需要使用 `metainfo` 参数将对话框构造函数传递给交易终端。

- 参数 `silently` 已从 Broker API 的这些方法: `placeOrder`, `modifyOrder`, `reversePosition`, `closePosition`, `closeTrade`, `cancelOrder`, `cancelOrders` 中删除。
- 参数 `handler` 和 `options` 从 Trading Host 的 `showOrderDailog` 方法中被删除。
- 参数 `handler` 从 Trading Host 的 `showPositionBracketsDailog` 方法中被删除。
- 标记 `supportCustomPlaceOrderTradableCheck` 不再被支持。
- 覆盖 `symbolWatermarkProperties` 不再被支持。 您可以使用 settings_adapter 的 `symbolWatermark` 。
- `indicators_file_name` 构造函数项目已被删除。 请改用 custom_indicators_getter 。
  我们进行了此更改以加快图表库的加载速度，并消除在加载文件时可能发生的漏洞。
  您只需要将自定义指标的代码从JS文件移动到widget构造函数，将它们包装在函数和Promise对象中。

Version 1.14

- createButton 返回 `HTMLElement` 以代替 `JQuery`.
- createButton 必须在 headerReady()返回的 `Promise` 为 resolved 之后使用.
- getVisibleRange 返回 UTC 时区的 `{from, to}` (之前返回的是图表选择的时区)。
- 方法 `onready` 被删除。 您可以使用 `window.addEventListener('DOMContentLoaded', callback, false)` 代替。
- saveChartToServer中 `saveAsSnapshot` 参数被删除。
- `indicators_file_name` 构造选项被删除。 请改用 custom_indicators_getter。
  我们进行了此更改以加快图表库的加载并消除加载文件时可能发生的漏洞。
  您只需将自定义指标的代码从 JS 文件移动到 widget 构造函数，将它们包装在函数和 Promise 中。

TypeScript 类型定义

- `StudyInputValueType` 改名为 `StudyInputValue` .

功能集

- 从此版本开始，您将无法再使用 `keep_left_toolbar_visible_on_small_screens` 功能集。 此功能集已删除，左侧工具栏可见性不再取决于屏幕大小。

Version 1.13

- 动作 `takeScreenshot` 从 executeActionById 方法中被删除。 改用 takeScreenshot 方法。
- 动作 `lockDrawingsAction` 从executeActionById 和 getCheckableActionState 方法中被删除。 改用lockAllDrawingTools 代替。
- 动作 `hideAllDrawingsAction` 从 executeActionById 和 getCheckableActionState 方法中被删除。 改用hideAllDrawingTools 代替。
- 功能集 `caption_buttons_text_if_possible` 默认开启。
- 修复一个问题 导致 K 线发生偏移。当日 K 线有一个负交易所时区偏移至 24x7 交易时段时，会出现 K 线的偏移。 如果您已通过变通方法解决此问题，请在更新此版本之前将其删除。

Version 1.12

图表库

- `charting_library/charting_library.min.js` 现在为 UMD 通用模块规范。 如果您只是将这个脚本嵌入到 HTML 中 - 没有任何改变。 但是，如果将它作为模块导入，则应该直接导入它的 `widget` ， `version` 和 `onready` 函数。
- `searchSymbolsByName` 从 `JS-API` 中移除, 使用 `searchSymbols` 代替。

指标覆盖:

- `Overlay` 的覆盖只能通过 `studies_overrides` (或运行时的 `applyStudiesOverrides`)。 在以前版本中您可以使用 `overrides` 和 `applyOverrides`). 参见 指标覆盖 页.
- 从这个版本开始，您将不能再使用 `options` 关键字以覆盖 `showStudyArguments` 和 `showLastValue` 。

交易终端

- `hasHistory` flag 被删除。 使用 `historyColumns` 来显示账户管理器中的历史记录。

交易终端中仍然支持以下内容，但在未来的版本中将不再使用：:

- `subscribePL` 和 `unsubscribePL` 。 每当损益变化时经纪商应该调用 `plUpdate` 方法。
- `supportDOM` 被删除。 DOM widget 的可见性可以使用 `dome_widget` 功能集。

交易控制器被替换为经纪商 API.

下列变更将让您的交易控制器实现迁移到新的经纪商 API:

- 方法 `setHost` 被删除。 主机将传递给经纪商 API 的构造函数。
- 方法 `buttonDropdownItems` 被删除。 经纪商 API 将使用 `setButtonDropdownActions` 更新交易主机。
- 方法 `configFlags` 和 `durations` 被删除。 使用 Widget 构造器 的字段代替。
- 全部方法返回 `$.Deferred` 的变为返回 Promise。
- 方法 `chartContextMenuItems` 重命名为 `chartContextMenuActions` 。
- 方法 `isTradable` 变更为返回一个 Promise 的布尔值。
- 全部字符串常量 ("working", "buy" etc.) 被替换为数字常量
- 仓位的 `avg_price` 重命名为 `avgPrice` 。
- `tradingController` 字段在 Widget 构造器 被删除。 改用 `brokerFactory` 。

Version 1.11

交易终端

交易终端中仍然支持以下内容，但在未来的版本中将不再使用：

- `supportDOME` 重命名为 `supportDOM`
- 更改了 `showClosePositionDialog` 的签名
- `showEditBracketsDialog` 重命名为 `showPositionBracketsDialog`，更改了签名。

Version 1.10

- 更改成交量指标的默认特性。

先前的特性：在仪表或周期切换时，根据成交量支持选项来确定成交量指标的添加/删除。
您可以通过禁用 `create_volume_indicator_by_default_once` 功能集来恢复到此特性。

新的特性：如果当前仪表支持成交量，则在空白图表的第一次加载时会添加成交量指标。

Version 1.9

- 我们不再编译更多 Pine 脚本。您仍然可以使用以前编译过的脚本。

Version 1.8 的交易终端

- 图表不可以只显示当前订单。适当的方法已被删除。
- `showOrderDialog` 输入参数时一个对象而不是列表
- `showSampleOrderDialog` 已被移除。 请使用 showOrderDialog 代替。
- 在 交易控制器 中删除 `showOrderDialog`，使用 `placeOrder` 和 `modifyOrder` 接收 `silently` 参数。
- `reversePosition`，`closePosition`，`cancelOrder` 有一个额外的参数 `silently`. 从现在起他们有了自己的对话框。

Version 1.7

- 从这个版本开始不能够用相同的商品代码调用 `setSymbol` 。 您应该先从 `subscribeBars` 调用 `onResetCacheNeededCallback` 开始。 然后您才可以使用图表的 `setSymbol` 或 新的 `resetData` 方法。
- JSAPI 协议版本 1 不在被支持。必须提供 `nextTime` 和 `noData` 。

Version 1.5

- 添加 `source` 参数给 MACD. 您也可以通过创建代码传递 `source` 参数来改变 MACD 。
  `chartWidget.chart().createStudy('MACD', false, false, [12, 26, "close", 9])`

Version 1.4

- 覆盖 `transparency` 不在被支持。 我们为每个颜色数据添加了透明度支持。 使用 `rgba` 来定义颜色的透明度。 例如:
  `"symbolWatermarkProperties.color" : "rgba(60, 70, 80, 0.05)"`

Version 1.3

- 覆盖 `paneProperties.gridProperties.*` 不在被支持。 请使用 `paneProperties.vertGridProperties.*` 和 `paneProperties.horzGridProperties.*`
- 覆盖 `mainSeriesProperties.candleStyle.wickColor` 不在被支持。 请使用 `mainSeriesProperties.candleStyle.wickUpColor` 和 `mainSeriesProperties.candleStyle.wickDownColor`

附录

周期

周期或时间周期是K线的时间段。 图表库支持日内周期(seconds, minutes, hours) and DWM周期 (daily, weekly, monthly)。图表库API有很多方法用以接收和返回周期。

日内

秒

格式: `xS`, 条件 `x` 为数字类型的秒数。 例如: 1S - 1秒, 2S - 2秒, 100S - 100秒。

分钟

格式: `x`, 条件 `x` 为数字类型的分钟数。 例如: 1 - 1分钟, 2 - 2分钟, 100 - 100分钟。

小时

重要: 用户界面允许用户输入几个小时，格式为 `xh` 或 `xH`, 它永远不会传递给API。小时必须使用图表库API中的分钟数来设置。

例如: 60 - 1小时, 120 - 2小时, 240 - 4小时.

DWM

天（Days）

格式: `xD`, 条件 `x` 为数字类型的天数。 例如: 1D - 1天, 2D - 2天, 100D - 100天。

周（Weeks）

格式: `xW` , 条件 `x` 为数字类型的周数。 例如: 1W - 1周, 2W - 2周, 100W - 100周。

月（Months）

格式: `xM` , 条件 `x` 为数字类型的月数。 例如: 1M - 1个月, 2M - 2个月, 100M - 100个月。

年

年是使用月数设置的。 例如: 12M - 1年, 24M - 2年, 48 - 4年。

也可以看看

如何设置图表上可用周期的列表

如何设置产品支持的周期列表

在图表上设置初始周期

获取当前图表周期

更改图表的周期

时间范围

可以看到图表底部的工具栏。 左侧的每个按钮都会切换图表的时间范围。 切换时间范围意味着：

1. 切换图表周期
2. 强制图表K线水平缩放以使整个请求范围可见

即，单击 `1Y` 将使图表将周期切换到 `1D` ，并按比例缩放以显示1年前的所有K线。 每个时间范围都有自己的图表周期。 这里是默认时间范围的列表：

Time Frame	Chart Resolution
5Y	W
1Y	D
6M	120
3M	60
1M	30
5D	5
1D	1

可以使用相应的widget自定义默认时间范围列表。

备注：要求周期不适用于当前图表商品的时间范围将被隐藏。

本地化

图表库支持本地化，并已被翻译成多种语言。您所要做的仅仅是在创建图表时指定 `locale` 参数。

我们的翻译为众包的，所以每个人都可以参与。访问我们的Webtranslateit页面来参与。

支持的语言列表

语言	`locale` 参数值
中国语（台湾）	zh_TW
中国语	zh
荷兰语（荷兰）	nl_NL
英语	en
法语	fr
德语 (德国)	de_DE
希腊语	el
意大利语	it
日语	ja
韩语	ko
波斯语(伊朗)	fa_IR
葡萄牙语	pt
俄语	ru
西班牙语	es
泰语(泰国)	th_TH

土耳其语	tr
越南语	vi

覆盖

本主题包含图表属性说明。 这些属性被处理作为可定制的。 其他属性的自定义则不被支持。

文件的格式:

`<属性路径>: <图表库的默认值>`

```
// 支持的值: large, medium, small, tiny
volumePaneSize: "large"

// 在文本编辑器中可用的字体（即，在`文本`绘图工具属性对话框中）
editorFontsList: ['Verdana', 'Courier New', 'Times New Roman', 'Arial']

paneProperties.background: "#ffffff"
paneProperties.vertGridProperties.color: "#E6E6E6"
paneProperties.vertGridProperties.style: LINESTYLE_SOLID
paneProperties.horzGridProperties.color: "#E6E6E6"
paneProperties.horzGridProperties.style: LINESTYLE_SOLID
paneProperties.crossHairProperties.color: "#989898"
paneProperties.crossHairProperties.width: 1
paneProperties.crossHairProperties.style: LINESTYLE_DASHED

// 边际（百分比）。 用于自动缩放。
paneProperties.topMargin: 5
paneProperties.bottomMargin: 5

paneProperties.axisProperties.autoScale: true
paneProperties.axisProperties.lockScale: false
paneProperties.axisProperties.percentage: false
paneProperties.axisProperties.indexedTo100: false
paneProperties.axisProperties.log: false
paneProperties.axisProperties.alignLabels: true
paneProperties.axisProperties.isInverted: false

paneProperties.legendProperties.showStudyArguments: true
paneProperties.legendProperties.showStudyTitles: true
paneProperties.legendProperties.showStudyValues: true
paneProperties.legendProperties.showSeriesTitle: true
paneProperties.legendProperties.showSeriesOHLC: true
paneProperties.legendProperties.showLegend: true
paneProperties.legendProperties.showBarChange: true
paneProperties.legendProperties.showOnlyPriceSource: true

```

```
scalesProperties.backgroundColor : "#ffffff"
scalesProperties.fontSize: 11
scalesProperties.lineColor : "#555"
scalesProperties.textColor : "#555"
scalesProperties.scaleSeriesOnly : false
scalesProperties.showSeriesLastValue: true
scalesProperties.showSeriesPrevCloseValue: false
scalesProperties.showStudyLastValue: false
scalesProperties.showStudyPlotLabels: false
scalesProperties.showSymbolLabels: false

timeScale.rightOffset: 5

timezone: "Etc/UTC" #查看支持的时区列表（在Symbology页面中的timezone）的可用值

// 数据列风格。 请参阅下面的支持的值
//  Bars = 0              #美国线
//  Candles = 1           #K线图
//  Line = 2              #线形图
//  Area = 3              #面积图
//  Heiken Ashi = 8       #平均K线图
//  Hollow Candles = 9    #空心K线图
//  Renko = 4             #转形图
//  Kagi = 5              #卡吉图
//  Point&Figure = 6      #点数图
//  Line Break = 7        #新价图
mainSeriesProperties.style: 1

mainSeriesProperties.showCountdown: true
mainSeriesProperties.visible:true
mainSeriesProperties.showPriceLine: true
mainSeriesProperties.priceLineWidth: 1
mainSeriesProperties.priceLineColor: ''
mainSeriesProperties.showPrevClosePriceLine: false
mainSeriesProperties.prevClosePriceLineWidth: 1
mainSeriesProperties.prevClosePriceLineColor: 'rgba( 85, 85, 85, 1)'
mainSeriesProperties.lockScale: false
mainSeriesProperties.minTick: "default"

mainSeriesProperties.priceAxisProperties.autoScale:true              (see #
mainSeriesProperties.priceAxisProperties.autoScaleDisabled:false     (see #
mainSeriesProperties.priceAxisProperties.percentage:false
mainSeriesProperties.priceAxisProperties.percentageDisabled:false
mainSeriesProperties.priceAxisProperties.log:false
mainSeriesProperties.priceAxisProperties.logDisabled:false

// 可能的值包括: description, ticker.
mainSeriesProperties.statusViewStyle.symbolTextSource: 'description'

symbolWatermarkProperties.color : "rgba(0, 0, 0, 0.00)"

```

```
// 不同的图表类型默认值

// K线图样式
mainSeriesProperties.candleStyle.upColor: "#6ba583"
mainSeriesProperties.candleStyle.downColor: "#d75442"
mainSeriesProperties.candleStyle.drawWick: true
mainSeriesProperties.candleStyle.drawBorder: true
mainSeriesProperties.candleStyle.borderColor: "#378658"
mainSeriesProperties.candleStyle.borderUpColor: "#225437"
mainSeriesProperties.candleStyle.borderDownColor: "#5b1a13"
mainSeriesProperties.candleStyle.wickUpColor: 'rgba( 115, 115, 117, 1)'
mainSeriesProperties.candleStyle.wickDownColor: 'rgba( 115, 115, 117, 1)'
mainSeriesProperties.candleStyle.barColorsOnPrevClose: false

// 空心K线图样式
mainSeriesProperties.hollowCandleStyle.upColor: "#6ba583"
mainSeriesProperties.hollowCandleStyle.downColor: "#d75442"
mainSeriesProperties.hollowCandleStyle.drawWick: true
mainSeriesProperties.hollowCandleStyle.drawBorder: true
mainSeriesProperties.hollowCandleStyle.borderColor: "#378658"
mainSeriesProperties.hollowCandleStyle.borderUpColor: "#225437"
mainSeriesProperties.hollowCandleStyle.borderDownColor: "#5b1a13"
mainSeriesProperties.hollowCandleStyle.wickColor: "#737375"

// 平均K线图样式
mainSeriesProperties.haStyle.upColor: "#6ba583"
mainSeriesProperties.haStyle.downColor: "#d75442"
mainSeriesProperties.haStyle.drawWick: true
mainSeriesProperties.haStyle.drawBorder: true
mainSeriesProperties.haStyle.borderColor: "#378658"
mainSeriesProperties.haStyle.borderUpColor: "#225437"
mainSeriesProperties.haStyle.borderDownColor: "#5b1a13"
mainSeriesProperties.haStyle.wickColor: "#737375"
mainSeriesProperties.haStyle.barColorsOnPrevClose: false

// 美国线样式
mainSeriesProperties.barStyle.upColor: "#6ba583"
mainSeriesProperties.barStyle.downColor: "#d75442"
mainSeriesProperties.barStyle.barColorsOnPrevClose: false
mainSeriesProperties.barStyle.dontDrawOpen: false

// 线形图样式
mainSeriesProperties.lineStyle.color: "#0303F7"
mainSeriesProperties.lineStyle.linestyle: LINESTYLE_SOLID
mainSeriesProperties.lineStyle.linewidth: 1
mainSeriesProperties.lineStyle.priceSource: "close"

// 面积图样式
mainSeriesProperties.areaStyle.color1: "#606090"
mainSeriesProperties.areaStyle.color2: "#01F6F5"
mainSeriesProperties.areaStyle.linecolor: "#0094FF"
```

```
mainSeriesProperties.areaStyle.linestyle: LINESTYLE_SOLID
mainSeriesProperties.areaStyle.linewidth: 1
mainSeriesProperties.areaStyle.priceSource: "close"

// 基准线样式
mainSeriesProperties.baselineStyle.baselineColor: "rgba( 117, 134, 150, 1)
mainSeriesProperties.baselineStyle.topFillColor1: "rgba( 83, 185, 135, 0.1
mainSeriesProperties.baselineStyle.topFillColor2: "rgba( 83, 185, 135, 0.1
mainSeriesProperties.baselineStyle.bottomFillColor1: "rgba( 235, 77, 92, 0
mainSeriesProperties.baselineStyle.bottomFillColor2: "rgba( 235, 77, 92, 0
mainSeriesProperties.baselineStyle.topLineColor: "rgba( 83, 185, 135, 1)"
mainSeriesProperties.baselineStyle.bottomLineColor: "rgba( 235, 77, 92, 1)
mainSeriesProperties.baselineStyle.topLineWidth: 1
mainSeriesProperties.baselineStyle.bottomLineWidth: 1
mainSeriesProperties.baselineStyle.priceSource: "close"
mainSeriesProperties.baselineStyle.transparency: 50
mainSeriesProperties.baselineStyle.baseLevelPercentage: 50

// Hi-Lo style
mainSeriesProperties.hiloStyle.color: "#2196f3"
mainSeriesProperties.hiloStyle.showBorders: true
mainSeriesProperties.hiloStyle.borderColor: "#2196f3"
mainSeriesProperties.hiloStyle.showLabels: true
mainSeriesProperties.hiloStyle.labelColor: "#2196f3"
mainSeriesProperties.hiloStyle.fontFamily: 'Trebuchet MS'
mainSeriesProperties.hiloStyle.fontSize: 7
```

LineStyles

```
LINESTYLE_SOLID = 0;
LINESTYLE_DOTTED = 1;
LINESTYLE_DASHED = 2;
LINESTYLE_LARGE_DASHED = 3;
```

绘图覆盖

以下是绘图覆盖的全部列表及默认值。 您可以使用Widget构造函数的 `overrides` 参数来更改默认值。在列表的底部，您可以找到值中使用的常量和缩写的列表。

```
linetoolicon: {
    singleChartOnly: true,
    color: 'rgba( 61, 133, 198, 1)',
    snapTo45Degrees:true,
    size: 40,
    icon: 0x263A,
    angle: Math.PI * 0.5,
    scale: 1.0
},
linetoolbezierquadro: {
    linecolor: 'rgba( 21, 153, 128, 1)',
    linewidth: 1.0,
    fillBackground: false,
    backgroundColor: 'rgba( 21, 56, 153, 0.5)',
    transparency: 50,
    linestyle: LINESTYLE_SOLID,
    extendLeft: false,
    extendRight: false,
    leftEnd: LINEEND_NORMAL,
    rightEnd: LINEEND_NORMAL
},
linetoolbeziercubic: {
    linecolor: 'rgba( 21, 153, 128, 1)',
    linewidth: 1.0,
    fillBackground: false,
    backgroundColor: 'rgba( 21, 56, 153, 0.5)',
    transparency: 50,
    linestyle: LINESTYLE_SOLID,
    extendLeft: false,
    extendRight: false,
    leftEnd: LINEEND_NORMAL,
    rightEnd: LINEEND_NORMAL
},
linetooltrendline: {
    linecolor: 'rgba( 21, 153, 128, 1)',
    linewidth: 1.0,
    linestyle: LINESTYLE_SOLID,
    extendLeft: false,
    extendRight: false,
    leftEnd: LINEEND_NORMAL,
    rightEnd: LINEEND_NORMAL,
```

```
        font: 'Verdana',
        textcolor: 'rgba( 21, 119, 96, 1)',
        fontsize: 12,
        bold:false,
        italic:false,
        snapTo45Degrees:true,
        alwaysShowStats: false,
        showPriceRange: false,
        showBarsRange: false,
        showDateTimeRange: false,
        showDistance:false,
        showAngle: false
    },
    linetooltimecycles: {
        linecolor: 'rgba(21, 153, 128, 1)',
        linewidth: 1.0,
        fillBackground: true,
        backgroundColor: 'rgba(106, 168, 79, 0.5)',
        transparency: 50,
        linestyle: LINESTYLE_SOLID
    },
    linetoolsineline: {
        linecolor: 'rgba( 21, 153, 128, 1)',
        linewidth: 1.0,
        linestyle: LINESTYLE_SOLID
    },
    linetooltrendangle: {
        singleChartOnly: true,
        linecolor: 'rgba( 21, 153, 128, 1)',
        linewidth: 1.0,
        linestyle: LINESTYLE_SOLID,
        snapTo45Degrees:true,
        font: 'Verdana',
        textcolor: 'rgba( 21, 119, 96, 1)',
        fontsize: 12,
        bold:true,
        italic:false,
        alwaysShowStats: false,
        showPriceRange: false,
        showBarsRange: false,
        extendRight: false,
        extendLeft: false
    },
    linetooldisjointangle: {
        linecolor: 'rgba( 18, 159, 92, 1)',
        linewidth: 2.0,
        linestyle: LINESTYLE_SOLID,
        fillBackground: true,
        backgroundColor: 'rgba( 106, 168, 79, 0.5)',
        transparency: 50,
        extendLeft: false,
```

```
        extendRight: false,
        leftEnd: LINEEND_NORMAL,
        rightEnd: LINEEND_NORMAL,
        font: 'Verdana',
        textcolor: 'rgba( 18, 159, 92, 1)',
        fontsize: 12,
        bold:false,
        italic:false,
        showPrices: false,
        showPriceRange: false,
        showDateTimeRange: false,
        showBarsRange: false
    },
    linetoolflatbottom: {
        linecolor: 'rgba( 73, 133, 231, 1)',
        linewidth: 2.0,
        linestyle: LINESTYLE_SOLID,
        fillBackground: true,
        backgroundColor: 'rgba( 21, 56, 153, 0.5)',
        transparency: 50,
        extendLeft: false,
        extendRight: false,
        leftEnd: LINEEND_NORMAL,
        rightEnd: LINEEND_NORMAL,
        font: 'Verdana',
        textcolor: 'rgba( 73, 133, 231, 1)',
        fontsize: 12,
        bold:false,
        italic:false,
        showPrices: false,
        showPriceRange: false,
        showDateTimeRange: false,
        showBarsRange: false
    },
    linetoolfibspiral: {
        linecolor: 'rgba( 21, 153, 128, 1)',
        linewidth: 1.0,
        linestyle: LINESTYLE_SOLID
    },
    linetooldaterange: {
        linecolor: 'rgba( 88, 88, 88, 1)',
        linewidth: 1.0,
        font: 'Verdana',
        textcolor: 'rgba( 255, 255, 255, 1)',
        fontsize: 12,
        fillLabelBackground: true,
        labelBackgroundColor: 'rgba( 91, 133, 191, 0.9)',
        labelBackgroundTransparency: 30,
        fillBackground: true,
        backgroundColor: 'rgba( 186, 218, 255, 0.4)',
        backgroundTransparency: 60,
```

```
    drawBorder: false,
    borderColor: 'rgba( 102, 123, 139, 1)'
},
linetoolpricerange: {
    linecolor: 'rgba( 88, 88, 88, 1)',
        linewidth: 1.0,
    font: 'Verdana',
    textcolor: 'rgba( 255, 255, 255, 1)',
    fontsize: 12,
    fillLabelBackground: true,
    labelBackgroundColor: 'rgba( 91, 133, 191, 0.9)',
    labelBackgroundTransparency: 30,
    fillBackground: true,
    backgroundColor: 'rgba( 186, 218, 255, 0.4)',
    backgroundTransparency: 60,
    drawBorder: false,
    borderColor: 'rgba( 102, 123, 139, 1)'
},
linetooldateandpricerange: {
    linecolor: 'rgba( 88, 88, 88, 1)',
    linewidth: 1.0,
    font: 'Verdana',
    textcolor: 'rgba( 255, 255, 255, 1)',
    fontsize: 12,
    fillLabelBackground: true,
    labelBackgroundColor: 'rgba( 91, 133, 191, 0.9)',
    labelBackgroundTransparency: 30,
    fillBackground: true,
    backgroundColor: 'rgba( 186, 218, 255, 0.4)',
    backgroundTransparency: 60,
    drawBorder: false,
    borderColor: 'rgba( 102, 123, 139, 1)'
},
linetoolriskrewardshort: {
    linecolor: 'rgba( 88, 88, 88, 1)',
    linewidth: 1.0,
    font: 'Verdana',
    textcolor: 'rgba(255, 255, 255, 1)',
    fontsize: 12,
    fillLabelBackground: true,
    labelBackgroundColor: 'rgba( 88, 88, 88, 1)',
    labelBackgroundTransparency: 0,
    fillBackground: true,
    stopBackground: 'rgba( 255, 0, 0, 0.2)',
    profitBackground: 'rgba( 0, 160, 0, 0.2)',
    stopBackgroundTransparency: 80,
    profitBackgroundTransparency: 80,
    drawBorder: false,
    borderColor: 'rgba( 102, 123, 139, 1)'
},
linetoolriskrewardlong: {
```

```
        linecolor: 'rgba( 88, 88, 88, 1)',
        linewidth: 1.0,
        font: 'Verdana',
        textcolor: 'rgba(255, 255, 255, 1)',
        fontsize: 12,
        fillLabelBackground: true,
        labelBackgroundColor: 'rgba( 88, 88, 88, 1)',
        labelBackgroundTransparency: 0,
        fillBackground: true,
        stopBackground: 'rgba( 255, 0, 0, 0.2)',
        profitBackground: 'rgba( 0, 160, 0, 0.2)',
        stopBackgroundTransparency: 80,
        profitBackgroundTransparency: 80,
        drawBorder: false,
        borderColor: 'rgba( 102, 123, 139, 1)'
    },
    linetoolarrow: {
        linecolor: 'rgba( 111, 136, 198, 1)',
        linewidth: 2.0,
        linestyle: LINESTYLE_SOLID,
        extendLeft: false,
        extendRight: false,
        leftEnd: LINEEND_NORMAL,
        rightEnd: LINEEND_ARROW,
        font: 'Verdana',
        textcolor: 'rgba( 21, 119, 96, 1)',
        fontsize: 12,
        bold:false,
        italic:false,
        alwaysShowStats: false,
        showPriceRange: false,
        showBarsRange: false,
        showDateTimeRange: false,
        showDistance:false,
        showAngle: false
    },
    linetoolray: {
        linecolor: 'rgba( 21, 153, 128, 1)',
        linewidth: 1.0,
        linestyle: LINESTYLE_SOLID,
        extendLeft: false,
        extendRight: true,
        leftEnd: LINEEND_NORMAL,
        rightEnd: LINEEND_NORMAL,
        font: 'Verdana',
        textcolor: 'rgba( 21, 119, 96, 1)',
        fontsize: 12,
        bold:false,
        italic:false,
        alwaysShowStats: false,
        showPriceRange: false,
```

```
        showBarsRange: false,
        showDateTimeRange: false,
        showDistance:false,
        showAngle: false
    },
    linetoolextended: {
        linecolor: 'rgba( 21, 153, 128, 1)',
        linewidth: 1.0,
        linestyle: LINESTYLE_SOLID,
        extendLeft: true,
        extendRight: true,
        leftEnd: LINEEND_NORMAL,
        rightEnd: LINEEND_NORMAL,
        font: 'Verdana',
        textcolor: 'rgba( 21, 119, 96, 1)',
        fontsize: 12,
        bold:false,
        italic:false,
        alwaysShowStats: false,
        showPriceRange: false,
        showBarsRange: false,
        showDateTimeRange: false,
        showDistance:false,
        showAngle: false
    },
    linetoolhorzline: {
        linecolor: 'rgba( 128, 204, 219, 1)',
        linewidth: 1.0,
        linestyle: LINESTYLE_SOLID,
        showPrice: true,
        showLabel: false,
        text: '',
        font: 'Verdana',
        textcolor: 'rgba( 21, 119, 96, 1)',
        fontsize: 12,
        bold:false,
        italic:false,
        horzLabelsAlign: 'center',
        vertLabelsAlign: 'top'
    },
    linetoolhorzray: {
        linecolor: 'rgba( 128, 204, 219, 1)',
        linewidth: 1.0,
        linestyle: LINESTYLE_SOLID,
        showPrice: true,
        showLabel: false,
        text: '',
        font: 'Verdana',
        textcolor: 'rgba( 21, 119, 96, 1)',
        fontsize: 12,
        bold:false,
```

```
        italic:false,
        horzLabelsAlign: 'center',
        vertLabelsAlign: 'top'
    },
    linetoolvertline: {
        linecolor: 'rgba( 128, 204, 219, 1)',
        linewidth: 1.0,
        linestyle: LINESTYLE_SOLID,
        showTime: true
    },
    linetoolcirclelines: {
        trendline: {
            visible: true,
            color: 'rgba( 128, 128, 128, 1)',
            linewidth: 1.0,
            linestyle: LINESTYLE_DASHED
        },
        linecolor: 'rgba( 128, 204, 219, 1)',
        linewidth: 1.0,
        linestyle: LINESTYLE_SOLID
    },
    linetoolfibtimezone: {
        horzLabelsAlign: 'right',
        vertLabelsAlign: 'bottom',
        baselinecolor: 'rgba( 128, 128, 128, 1)',
        linecolor: 'rgba( 0, 85, 219, 1)',
        linewidth: 1.0,
        linestyle: LINESTYLE_SOLID,
        showLabels: true,
        font: 'Verdana',
        fillBackground:false,
        transparency:80,
        trendline: {
            visible: true,
            color: 'rgba( 128, 128, 128, 1)',
            linewidth: 1.0,
            linestyle: LINESTYLE_DASHED
        },
        level1-11: LEVELS_TYPE_C
    },
    linetooltext: {
        color: 'rgba( 102, 123, 139, 1)',
        text: $.t('Text'),
        font: 'Verdana',
        fontsize: 20,
        fillBackground: false,
        backgroundColor: 'rgba( 91, 133, 191, 0.9)',
        backgroundTransparency: 70,
        drawBorder: false,
        borderColor: 'rgba( 102, 123, 139, 1)',
        bold:false,
```

```
        italic:false,
        locked: false,
        fixedSize: true,
        wordWrap: false,
        wordWrapWidth: 400
    },
    linetooltextabsolute: {
        singleChartOnly: true,
        color: 'rgba( 102, 123, 139, 1)',
        text: $.t('Text'),
        font: 'Verdana',
        fontsize: 20,
        fillBackground: false,
        backgroundColor: 'rgba( 155, 190, 213, 0.3)',
        backgroundTransparency: 70,
        drawBorder: false,
        borderColor: 'rgba( 102, 123, 139, 1)',
        bold: false,
        italic: false,
        locked: true,
        wordWrap: false,
        wordWrapWidth: 400
    },
    linetoolballoon: {
        color: 'rgba( 102, 123, 139, 1)',
        backgroundColor: 'rgba( 255, 254, 206, 0.7)',
        borderColor: 'rgba( 140, 140, 140, 1)',
        fontWeight: 'bold',
        fontsize: 12,
        font: 'Arial',
        transparency: 30,
        text: $.t('Comment')
    },
    linetoolbrush: {
        linecolor: 'rgba( 53, 53, 53, 1)',
        linewidth: 2.0,
        smooth:5,
        fillBackground: false,
        backgroundColor: 'rgba( 21, 56, 153, 0.5)',
        transparency: 50,
        leftEnd: LINEEND_NORMAL,
        rightEnd: LINEEND_NORMAL
    },
    linetoolpolyline: {
        linecolor: 'rgba( 53, 53, 53, 1)',
        linewidth: 2.0,
        linestyle: LINESTYLE_SOLID,
        fillBackground: true,
        backgroundColor: 'rgba( 21, 56, 153, 0.5)',
        transparency: 50,
        filled: false
```

```
},
linetoolarrowmark: {
    color: 'rgba( 120, 120, 120, 1)',
    text: '',
    fontsize: 20,
    font: 'Verdana'
},
linetoolarrowmarkleft: {
    color: 'rgba( 120, 120, 120, 1)',
    text: '',
    fontsize: 20,
    font: 'Verdana'
},
linetoolarrowmarkup: {
    color: 'rgba( 120, 120, 120, 1)',
    text: '',
    fontsize: 20,
    font: 'Verdana'
},
linetoolarrowmarkright: {
    color: 'rgba( 120, 120, 120, 1)',
    text: '',
    fontsize: 20,
    font: 'Verdana'
},
linetoolarrowmarkdown: {
    color: 'rgba( 120, 120, 120, 1)',
    text: '',
    fontsize: 20,
    font: 'Verdana'
},
linetoolflagmark: {
    color: 'rgba( 255, 0, 0, 1)'
},

linetoolnote: {
    markerColor: 'rgba( 46, 102, 255, 1)',
    textColor: 'rgba( 0, 0, 0, 1)',
    backgroundColor: 'rgba( 255, 255, 255, 1)',
    backgroundTransparency: 0,
    text: 'Text',
    font: 'Arial',
    fontSize: 12,
    bold: false,
    italic: false,
    locked: false,
    fixedSize: true
},

linetoolnoteabsolute: {
    singleChartOnly: true,
```

```
        markerColor: 'rgba( 46, 102, 255, 1)',
        textColor: 'rgba( 0, 0, 0, 1)',
        backgroundColor: 'rgba( 255, 255, 255, 1)',
        backgroundTransparency: 0,
        text: 'Text',
        font: 'Arial',
        fontSize: 12,
        bold: false,
        italic: false,
        locked: true,
        fixedSize: true
    },

    linetoolthumbup: {
        color: 'rgba( 0, 128, 0, 1)'
    },
    linetoolthumbdown: {
        color: 'rgba( 255, 0, 0, 1)'
    },
    linetoolpricelabel: {
        color: 'rgba( 102, 123, 139, 1)',
        backgroundColor: 'rgba( 255, 255, 255, 0.7)',
        borderColor: 'rgba( 140, 140, 140, 1)',
        fontWeight: 'bold',
        fontsize: 11,
        font: 'Arial',
        transparency: 30
    },
    linetoolrectangle: {
        color: 'rgba( 21, 56, 153, 1)',
        fillBackground: true,
        backgroundColor: 'rgba( 21, 56, 153, 0.5)',
        linewidth: 1.0,
        snapTo45Degrees:true,
        transparency: 50
    },
    linetoolrotatedrectangle: {
        color: 'rgba( 152, 0, 255, 1)',
        fillBackground: true,
        backgroundColor: 'rgba( 142, 124, 195, 0.5)',
        transparency: 50,
        linewidth: 1.0,
        snapTo45Degrees:true
    },
    linetoolellipse: {
        color: 'rgba( 153, 153, 21, 1)',
        fillBackground: true,
        backgroundColor: 'rgba( 153, 153, 21, 0.5)',
        transparency: 50,
        linewidth: 1.0
    },
```

```
linetoolarc: {
    color: 'rgba( 153, 153, 21, 1)',
    fillBackground: true,
    backgroundColor: 'rgba( 153, 153, 21, 0.5)',
    transparency: 50,
    linewidth: 1.0
},
linetoolprediction: {
    singleChartOnly: true,
    linecolor: 'rgba( 28, 115, 219, 1)',
    linewidth: 2.0,

    sourceBackColor: 'rgba( 241, 241, 241, 1)',
    sourceTextColor: 'rgba( 110, 110, 110, 1)',
    sourceStrokeColor: 'rgba( 110, 110, 110, 1)',

    targetStrokeColor:'rgba( 47, 168, 255, 1)',
    targetBackColor:'rgba( 11, 111, 222, 1)',
    targetTextColor: 'rgba( 255, 255, 255, 1)',

    successBackground: 'rgba( 54, 160, 42, 0.9)',
    successTextColor: 'rgba( 255, 255, 255, 1)',

    failureBackground: 'rgba( 231, 69, 69, 0.5)',
    failureTextColor: 'rgba( 255, 255, 255, 1)',

    intermediateBackColor: 'rgba( 234, 210, 137, 1)',
    intermediateTextColor: 'rgba( 109, 77, 34, 1)',

    transparency: 10,
    centersColor: 'rgba( 32, 32, 32, 1)'
},
linetooltriangle: {
    color: 'rgba( 153, 21, 21, 1)',
    fillBackground: true,
    backgroundColor: 'rgba( 153, 21, 21, 0.5)',
    transparency: 50,
    linewidth: 1.0
},
linetoolcallout: {
    color: 'rgba( 255, 255, 255, 1)',
    backgroundColor: 'rgba( 153, 21, 21, 0.5)',
    transparency: 50,
    linewidth: 2.0,
    fontsize: 12,
    font:'Verdana',
    text: 'Text',
    bordercolor: 'rgba( 153, 21, 21, 1)',
    bold: false,
    italic: false,
    wordWrap: false,
```

```
    wordWrapWidth: 400
},
linetoolparallelchannel: {
    linecolor: 'rgba( 119, 52, 153, 1)',
    linewidth: 1.0,
    linestyle: LINESTYLE_SOLID,
    extendLeft: false,
    extendRight: false,
    fillBackground: true,
    backgroundColor: 'rgba( 180, 167, 214, 0.5)',
    transparency: 50,
    showMidline: false,
    midlinecolor: 'rgba( 119, 52, 153, 1)',
    midlinewidth: 1.0,
    midlinestyle: LINESTYLE_DASHED
},
linetoolelliottimpulse: {
    degree: 7,
    showWave:true,
    color: 'rgba( 61, 133, 198, 1)',
    linewidth: 1
},
linetoolelliotttriangle: {
    degree: 7,
    showWave:true,
    color: 'rgba( 255, 152, 0, 1)',
    linewidth: 1
},
linetoolelliotttriplecombo: {
    degree: 7,
    showWave:true,
    color: 'rgba( 106, 168, 79, 1)',
    linewidth: 1
},
linetoolelliottcorrection: {
    degree: 7,
    showWave:true,
    color: 'rgba( 61, 133, 198, 1)',
    linewidth: 1
},
linetoolelliottdoublecombo: {
    degree: 7,
    showWave:true,
    color: 'rgba( 106, 168, 79, 1)',
    linewidth: 1
},
linetoolbarspattern: {
    singleChartOnly: true,
    color:'rgba( 80, 145, 204, 1)',
    mode:BARS_MODE,
    mirrored:false,
```

```
603         flipped:false
604     },
605     linetoolghostfeed: {
606         singleChartOnly: true,
607         averageHL: 20,
608         variance: 50,
609         candleStyle: {
610             upColor: '#6ba583',
611             downColor: '#d75442',
612             drawWick: true,
613             drawBorder: true,
614             borderColor: '#378658',
615             borderUpColor: '#225437',
616             borderDownColor: '#5b1a13',
617             wickColor: '#737375'
618         },
619         transparency: 50
620     },
621     linetoolpitchfork: {
622         fillBackground:true,
623         transparency:80,
624         style:PITCHFORK_STYLE_ORIGINAL,
625         median: {
626             visible: true,
627             color: 'rgba( 165, 0, 0, 1)',
628             linewidth: 1.0,
629             linestyle: LINESTYLE_SOLID
630         },
631         level0-8: LEVELS_TYPE_C
632     },
633     linetoolpitchfan: {
634         fillBackground:true,
635         transparency:80,
636         median: {
637             visible: true,
638             color: 'rgba( 165, 0, 0, 1)',
639             linewidth: 1.0,
640             linestyle: LINESTYLE_SOLID
641         },
642         level0-8: LEVELS_TYPE_C
643     },
644     linetoolgannfan: {
645         showLabels: true,
646         font: 'Verdana',
647         fillBackground:true,
648         transparency:80,
649         level1-9: LEVELS_TYPE_F
650     },
651     linetoolganncomplex: {
652         fillBackground:false,
653         arcsBackground: {
```

```
            fillBackground: true,
            transparency: 80
        },
        levels: [/* 6 LEVELS_TYPE_D */],
        fanlines: [/* 11 LEVELS_TYPE_E */],
        arcs: [/* 11 LEVELS_TYPE_E */]
    },
    linetoolgannsquare: {
        color: 'rgba( 21, 56, 153, 0.8)',
        linewidth: 1.0,
        linestyle: LINESTYLE_SOLID,
        font: 'Verdana',
        showTopLabels:true,
        showBottomLabels:true,
        showLeftLabels:true,
        showRightLabels:true,
        fillHorzBackground:true,
        horzTransparency:80,
        fillVertBackground:true,
        vertTransparency:80,
        hlevel1-7: LEVELS_TYPE_B,
        vlevel1-7: LEVELS_TYPE_B
    },
    linetoolfibspeedresistancefan: {
        fillBackground:true,
        transparency:80,
        grid: {
            color: 'rgba( 128, 128, 128, 1)',
            linewidth: 1.0,
            linestyle: LINESTYLE_SOLID,
            visible:true
        },
        linewidth: 1.0,
        linestyle: LINESTYLE_SOLID,
        font: 'Verdana',
        showTopLabels:true,
        showBottomLabels:true,
        showLeftLabels:true,
        showRightLabels:true,
        snapTo45Degrees:true,
        hlevel1-7: LEVELS_TYPE_B,
        vlevel1-7: LEVELS_TYPE_B
    },
    linetoolfibretracement: {
        showCoeffs: true,
        showPrices: true,
        font: 'Verdana',
        fillBackground:true,
        transparency:80,
        extendLines:false,
        horzLabelsAlign: 'left',
```

```
        vertLabelsAlign: 'middle',
        reverse:false,
        coeffsAsPercents:false,
        trendline: {
            visible: true,
            color: 'rgba( 128, 128, 128, 1)',
            linewidth: 1.0,
            linestyle: LINESTYLE_DASHED
        },
        levelsStyle: {
            linewidth: 1.0,
            linestyle: LINESTYLE_SOLID
        },
        level1-24: LEVELS_TYPE_B
    },
    linetoolfibchannel: {
        showCoeffs: true,
        showPrices: true,
        font: 'Verdana',
        fillBackground:true,
        transparency:80,
        extendLeft:false,
        extendRight:false,
        horzLabelsAlign: 'left',
        vertLabelsAlign: 'middle',
        coeffsAsPercents:false,
        levelsStyle: {
            linewidth: 1.0,
            linestyle: LINESTYLE_SOLID
        },
        level1-24:  LEVELS_TYPE_B
    },
    linetoolprojection: {
        showCoeffs: true,
        font: 'Verdana',
        fillBackground:true,
        transparency:80,
        color1: 'rgba( 0, 128, 0, 0.2)',
        color2: 'rgba( 255, 0, 0, 0.2)',
        linewidth: 1.0,
        trendline: {
            visible: true,
            color: 'rgba( 128, 128, 128, 1)',
            linestyle: LINESTYLE_SOLID
        },
        level1: LEVELS_TYPE_C
    },
    linetool5pointspattern: {
        color: 'rgba( 204, 40, 149, 1)',
        textcolor: 'rgba( 255, 255, 255, 1)',
        fillBackground: true,
```

```
        backgroundColor: 'rgba( 204, 40, 149, 0.5)',
        font: 'Verdana',
        fontsize:12,
        bold:false,
        italic:false,
        transparency: 50,
        linewidth: 1.0
    },
    linetoolcypherpattern: {
        color: '#CC2895',
        textcolor: '#FFFFFF',
        fillBackground: true,
        backgroundColor: '#CC2895',
        font: 'Verdana',
        fontsize:12,
        bold:false,
        italic:false,
        transparency: 50,
        linewidth: 1.0
    },
    linetooltrianglepattern: {
        color: 'rgba( 149, 40, 255, 1)',
        textcolor: 'rgba( 255, 255, 255, 1)',
        fillBackground: true,
        backgroundColor: 'rgba( 149, 40, 204, 0.5)',
        font: 'Verdana',
        fontsize:12,
        bold:false,
        italic:false,
        transparency: 50,
        linewidth: 1.0
    },
    linetoolabcd: {
        color: 'rgba( 0, 155, 0, 1)',
        textcolor: 'rgba( 255, 255, 255, 1)',
        font: 'Verdana',
        fontsize:12,
        bold:false,
        italic:false,
        linewidth: 2.0
    },
    linetoolthreedrivers: {
        color: 'rgba( 149, 40, 255, 1)',
        textcolor: 'rgba( 255, 255, 255, 1)',
        fillBackground: true,
        backgroundColor: 'rgba( 149, 40, 204, 0.5)',
        font: 'Verdana',
        fontsize:12,
        bold:false,
        italic:false,
        transparency: 50,
```

```
        linewidth: 2.0
    },
    linetoolheadandshoulders: {
        color: 'rgba( 69, 104, 47, 1)',
        textcolor: 'rgba( 255, 255, 255, 1)',
        fillBackground: true,
        backgroundColor: 'rgba( 69, 168, 47, 0.5)',
        font: 'Verdana',
        fontsize:12,
        bold:false,
        italic:false,
        transparency: 50,
        linewidth: 2.0
    },
    linetoolfibwedge: {
        singleChartOnly: true,
        showCoeffs: true,
        font: 'Verdana',
        fillBackground:true,
        transparency:80,
        trendline: {
            visible: true,
            color: 'rgba( 128, 128, 128, 1)',
            linewidth: 1.0,
            linestyle: LINESTYLE_SOLID
        },
        level1-11: LEVELS_TYPE_C
    },
    linetoolfibcircles: {
        showCoeffs: true,
        font: 'Verdana',
        fillBackground:true,
        transparency:80,
        snapTo45Degrees:true,
        coeffsAsPercents:false,
        trendline: {
            visible: true,
            color: 'rgba( 128, 128, 128, 1)',
            linewidth: 1.0,
            linestyle: LINESTYLE_DASHED
        },
        level1-11: LEVELS_TYPE_C
    },
    linetoolfibspeedresistancearcs: {
        showCoeffs: true,
        font: 'Verdana',
        fillBackground:true,
        transparency:80,
        fullCircles: false,
        trendline: {
            visible: true,
```

```
            color: 'rgba( 128, 128, 128, 1)',
            linewidth: 1.0,
            linestyle: LINESTYLE_DASHED
        },
        level1-11: LEVELS_TYPE_C
    },
    linetooltrendbasedfibextension: {
        showCoeffs: true,
        showPrices: true,
        font: 'Verdana',
        fillBackground:true,
        transparency:80,
        extendLines:false,
        horzLabelsAlign: 'left',
        vertLabelsAlign: 'middle',
        reverse:false,
        coeffsAsPercents:false,
        trendline: {
            visible: true,
            color: 'rgba( 128, 128, 128, 1)',
            linewidth: 1.0,
            linestyle: LINESTYLE_DASHED
        },
        levelsStyle: {
            linewidth: 1.0,
            linestyle: LINESTYLE_SOLID
        },
        level1-24: LEVELS_TYPE_B
    },
    linetooltrendbasedfibtime: {
        showCoeffs: true,
        font: 'Verdana',
        fillBackground:true,
        transparency:80,
        horzLabelsAlign: 'right',
        vertLabelsAlign: 'bottom',
        trendline: {
            visible: true,
            color: 'rgba( 128, 128, 128, 1)',
            linewidth: 1.0,
            linestyle: LINESTYLE_DASHED
        },
        level1-11: LEVELS_TYPE_C
    },
    linetoolschiffpitchfork: {
        fillBackground:true,
        transparency:80,
        style:PITCHFORK_STYLE_SCHIFF,
        median: {
            visible: true,
            color: 'rgba( 165, 0, 0, 1)',
```

```
            linewidth: 1.0,
            linestyle: LINESTYLE_SOLID
        },
        level0-8: LEVELS_TYPE_C
    },
    linetoolschiffpitchfork2: {
        fillBackground:true,
        transparency:80,
        style:PITCHFORK_STYLE_SCHIFF2,
        median: {
            visible: true,
            color: 'rgba( 165, 0, 0, 1)',
            linewidth: 1.0,
            linestyle: LINESTYLE_SOLID
        },
        level0-8: LEVELS_TYPE_C
    },
    linetoolinsidepitchfork: {
        fillBackground:true,
        transparency:80,
        style:PITCHFORK_STYLE_INSIDE,
        median: {
            visible: true,
            color: 'rgba( 165, 0, 0, 1)',
            linewidth: 1.0,
            linestyle: LINESTYLE_SOLID
        },
        level0-8: LEVELS_TYPE_C
    },
    linetoolvisibilities: {
        intervalsVisibilities: {
            seconds:true,
            secondsFrom:1,
            secondsTo:59,
            minutes:true,
            minutesFrom:1,
            minutesTo:59,
            hours:true,
            hoursFrom:1,
            hoursTo:24,
            days:true,
            daysFrom:1,
            daysTo:366,
            weeks:true,
            months:true
        }
    }
```

常量

这些常量用于绘图的默认值。 你不能直接使用他们的名字，而是使用这些值。

```
LINESTYLE_SOLID = 0;
LINESTYLE_DOTTED = 1;
LINESTYLE_DASHED = 2;
LINESTYLE_LARGE_DASHED = 3;

LINEEND_NORMAL = 0;
LINEEND_ARROW  = 1;
LINEEND_CIRCLE = 2;

BARS_MODE = 0;
LINE_MODE = 1;
OPENCLOSE_MODE = 2;
LINEOPEN_MODE = 3;
LINEHIGH_MODE = 4;
LINELOW_MODE = 5;
LINEHL2_MODE = 6;

PITCHFORK_STYLE_ORIGINAL = 0;
PITCHFORK_STYLE_SCHIFF = 1;
PITCHFORK_STYLE_SCHIFF2 = 2;
PITCHFORK_STYLE_INSIDE = 3;

LEVELS_TYPE_A = {
    color: color,
    visible: visible
};

LEVELS_TYPE_B = {
    coeff: coeff,
    color: color,
    visible: visible
};

LEVELS_TYPE_C = {
    coeff: coeff,
    color: color,
    visible: visible,
    linestyle: linestyle,
    linewidth: linewidth
};

LEVELS_TYPE_D = {
    color: color,
```

```
    width: width,
    visible: visible
};

LEVELS_TYPE_E = {
    color: color,
    visible: visible,
    width: width,
    x: x,
    y: y
};

LEVELS_TYPE_F = {
    coeff1: coeff1,
    coeff2: coeff2,
    color: color,
    visible: visible,
    linestyle: linestyle,
    linewidth: linewidth
};
```

指标覆盖

您可以使用 `studies_overrides` 参数设置默认样式以及新创建的指标的输入值。 它的值应该是一个对象，其中key是要更改的属性的路径，而value是它的新值。

例：

```
studies_overrides: {
    "volume.volume.color.0": "#00FFFF",
    "volume.volume.color.1": "#0000FF",
    "volume.volume.transparency": 70,
    "volume.volume ma.color": "#FF0000",
    "volume.volume ma.transparency": 30,
    "volume.volume ma.linewidth": 5,
    "volume.show ma": true,
    "bollinger bands.median.color": "#33FF88",
    "bollinger bands.upper.linewidth": 7
}
```

在上面的示例中，所有创建的布林带都将上规宽设置为7（除非您通过API创建它并指定不同的值）。

如何设置指标名称

您应该在新建指标对话框中使用指标名称，但要采用小写形式。 如果您想覆盖默认的EMA长度，请尝试使用 `moving average exponential.length` 。

同样的逻辑适用于输入名称：使用在 `指标属性` 对话框中看到的名称（使用小写字母）。

示例： `stochastic.smooth d` 。

比较

您可以通过 `Compare` 自定义添加新的数据。

使用 `compare.plot` 来自定义画线，使用 `compare.source` 来更改价格来源:

```
"compare.plot.color": "#000000",
"compare.source": "high"
```

覆盖

从V1.12开始，您可以使用以下属性来自定义 `Overlay` ：

```
Overlay.style: (bars = 0, candles = 1, line = 2, area = 3, heiken ashi = 8
Overlay.showPriceLine: boolean

Overlay.candleStyle.upColor: color
Overlay.candleStyle.downColor: color
Overlay.candleStyle.drawWick: boolean
Overlay.candleStyle.drawBorder: boolean
Overlay.candleStyle.borderColor: color
Overlay.candleStyle.borderUpColor: color
Overlay.candleStyle.borderDownColor: color
Overlay.candleStyle.wickColor: color
Overlay.candleStyle.barColorsOnPrevClose: boolean

Overlay.hollowCandleStyle.upColor: color
Overlay.hollowCandleStyle.downColor: color
Overlay.hollowCandleStyle.drawWick: boolean
Overlay.hollowCandleStyle.drawBorder: boolean
Overlay.hollowCandleStyle.borderColor: color
Overlay.hollowCandleStyle.borderUpColor: color
Overlay.hollowCandleStyle.borderDownColor: color
Overlay.hollowCandleStyle.wickColor: color
Overlay.hollowCandleStyle.barColorsOnPrevClose: boolean

Overlay.barStyle.upColor: color
Overlay.barStyle.downColor: color
Overlay.barStyle.barColorsOnPrevClose: boolean
Overlay.barStyle.dontDrawOpen: boolean

Overlay.lineStyle.color: color
Overlay.lineStyle.linewidth: integer
Overlay.lineStyle.priceSource: open/high/low/close
Overlay.lineStyle.styleType: (bars = 0, candles = 1, line = 2, area = 3, h

Overlay.areaStyle.color1: color
Overlay.areaStyle.color2: color
Overlay.areaStyle.linecolor: color
Overlay.areaStyle.linestyle: (solid = 0; dotted = 1; dashed = 2; large das
```

```
Overlay.areaStyle.linewidth: integer
Overlay.areaStyle.priceSource: open/high/low/close
```

语法

属性路径是一组用点（`.`）分割的小写标识符。路径格式如下所述。

备注：如果一个plot/band/area/input名称是相同的，则您会得到一个错误。 在这种情况下，您可以通过在路径中添加 `:plot` ， `:band` ， `:area` 或 `:input` 来指定y要更改的确切目标。（例如 `short:plot.color` ）

Study input

格式: `indicator_name.input_name`

- **indicator_name**: 使用在指标对话框中看到的名称。
- **input_name**: 使用在指标的属性对话框中看到的名称（例如: `show ma` ）

例如: `volume.show ma` , `bollinger bands.length`

绘图属性

格式: `indicator_name.plot_name.property_name`

- **indicator_name**: < ... >
- **plot_name**: 你可以在指标的属性对话框中看到它（例如 `Volume` 或 `Plot` ）
- **property_name**: 下列之一:
  - **transparency**
  - **linewidth**
  - **plottype**. 支持的绘图类型有:
    - `line` （线形图）
    - `histogram` （直方图）
    - `cross` （十字指针）

    - `area`（山形图）
    - `columns`（柱状图）
    - `circles`（圆圈图）
    - `line_with_breaks`（中断线）
    - `area_with_breaks`（中断区块）

例子: `volume.volume.transparency`, `bollinger bands.median.linewidth`

绘图颜色

格式: `indicator_name.plot_name.color<.color_index>`

- **indicator_name**: < ... >
- **plot_name**: < ... >
- **color** 这只是一个关键字。
- **color_index**（可选）: 颜色索引（如果有的话）。这只是一个颜色索引。也就是说，要取代成交量默认为绿色的颜色，应该使用 `color_index = 1` 。

备注1: `color.0` 是 `color` 的同义词。因此路径 `volume.volume.color.0` 和 `volume.volume.color` 被视为相同。

备注2: 现在，不支持自定义区域填充颜色和透明度。

限制:

- 颜色只支持 `#RRGGBB` 格式。不要使用短格式的 `#RGB` 。
- 透明度在[0..100]范围内变化。 100意味着完全不透明的。
- 厚度是一个整数。

指标选项

格式: `indicator_name.options.option_name`

- **indicator_name**: < ... >
- **options**: 关键字
- **option_name**: 你想分配的选项名称。支持的值是：

- **showStudyArguments**: boolean, 控制标题中的参数可见性
- **showLastValue**: boolean, 控制价格标签的可见性

例子: `volume.options.showStudyArguments` , `volume.options.showLastValue`

默认精度

1.6版本开始，您可以使用 `name.precision` 格式更改指标的默认精度。 例：

```
"average true range.precision": 8
```

形状与覆盖

您可以使用createMultipointShape(points, options)创建50多个不同的形状。它们在下表中列出，可以使用 `overrides` 更改默认属性。

shape	backgroundColor	backgroundTransparency	bold	borderColor	cc
text	#9BBED5	70	FALSE	#667B8B	#(
anchored_text	#9BBED5	70	FALSE	#667B8B	#(
note	#FFFFFF	0	FALSE		
anchored_note	#FFFFFF	0	FALSE		
callout	#991515		FALSE	#991515	#I
balloon	#fffece			#8c8c8c	#(

shape	backgroundColor	borderColor	color	font	fontsize	text	tra
arrow_up			#787878	Verdana	20	text	
arrow_down			#787878	Verdana	20	text	
arrow_left			#787878	Verdana	20	text	
arrow_right			#787878	Verdana	20	text	
price_label	#ffffff	#8c8c8c	#667b8b	Arial	11		30
flag							

shape	backgroundColor	bold	color	fillBackground	font
xabcd_pattern	#CC2895	FALSE	#CC2895	TRUE	Verdana
abcd_pattern		FALSE	#009B00		Verdana

triangle_pattern	#9528CC	FALSE	#9528FF	TRUE	Verdana
3divers_pattern	#9528CC	FALSE	#9528FF	TRUE	Verdana
head_and_shoulders	#45A82F	FALSE	#45682F	TRUE	Verdana
cypher_pattern	#CC2895	FALSE	#CC2895	TRUE	Verdana

shape	backgroundColor	color	degree	extendLeft	extendRight
elliott_impulse_wave		#3d85c6	7		
elliott_triangle_wave		#ff9800	7		
elliott_triple_combo		#6aa84f	7		
elliott_correction		#3d85c6	7		
elliott_double_combo		#6aa84f	7		
cyclic_lines					
time_cycles	#6AA84F				
sine_line					
rectangle	#153899	#153899			
rotated_rectangle	#8e7cc3	#9800ff			
ellipse	#999915	#999915			
triangle	#991515	#991515			
polyline	#153899				
curve	#153899			FALSE	FALSE
double_curve	#153899			FALSE	FALSE
arc	#999915	#999915			

shape	backgroundColor	bold	extendLeft	extendRight	fillBackground
vertical_line					
horizontal_line		TRUE			
cross_line					
horizontal_ray		TRUE			
trend_line		FALSE	FALSE	FALSE	
trend_infoline		FALSE	FALSE	FALSE	
trend_angle		TRUE	FALSE	FALSE	
arrow		FALSE	FALSE	FALSE	
ray		FALSE	FALSE	TRUE	
extended		FALSE	TRUE	TRUE	
parallel_channel	#b4a7d6		FALSE	FALSE	TRUE
disjoint_angle	#6AA84F	FALSE	FALSE	FALSE	TRUE
flat_bottom	#153899	FALSE	FALSE	FALSE	TRUE
fib_spiral					

shape	fillBackground	transparency	style	median.visible	media
pitchfork	TRUE	80	0	TRUE	#A50
schiff_pitchfork_modified	TRUE	80	1	TRUE	#A50
schiff_pitchfork	TRUE	80	3	TRUE	#A50
inside_pitchfork	TRUE	80	2	TRUE	#A50
pitchfan	TRUE	80		TRUE	#A50

shape	fillBackground	arcsBackground.fillBackground	arcsBackground.tra
gannbox_square	FALSE	TRUE	50

shape	showLabels	font	fillBackground	transparency	level1.visible	le
gannbox_fan	TRUE	Verdana	TRUE	80	TRUE	#

shape	color	linewidth	linestyle	font	showTopLabels	show
gannbox	#153899	1	0	Verdana	TRUE	TRUE
fib_speed_resist_fan		1	0	Verdana	TRUE	TRUE

shape	showCoeffs	showPrices	font	fillBackground	transparen
fib_retracement	TRUE	TRUE	Verdana	TRUE	80
fib_trend_ext	TRUE	TRUE	Verdana	TRUE	80
fib_timezone			Verdana	FALSE	80
fib_trend_time	TRUE		Verdana	TRUE	80
fib_circles	TRUE		Verdana	TRUE	80
fib_speed_resist_arcs	TRUE		Verdana	TRUE	80
fib_wedge	TRUE		Verdana	TRUE	80
fib_channel	TRUE	TRUE	Verdana	TRUE	80

shape	borderColor	drawBorder	fillBackground	fillLabelBackground
date_range	#667B8B	FALSE	TRUE	TRUE
price_range	#667B8B	FALSE	TRUE	TRUE

date_and_price_range	#667B8B	FALSE	TRUE	TRUE

shape	borderColor	drawBorder	fillBackground	fillLabelBackground	font
long_position	#667B8B	FALSE	TRUE	TRUE	Verda
short_position	#667B8B	FALSE	TRUE	TRUE	Verda

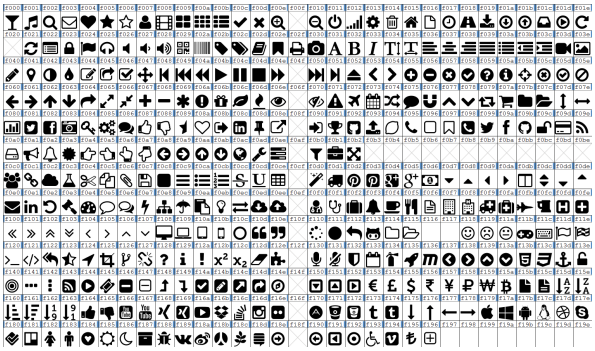
shape	showCoeffs	font	fillBackground	transparency	color1	color2
projection	TRUE	Verdana	TRUE	80	#008000	#FF0000

shape	linecolor	linewidth	centersColor	failureBackground	failureTextColor	in
forecast	#1c73db	2	#202020	#e74545	#ffffff	#e

shape	averageHL	variance	transparency	candleStyle.upColor	candleStyle.d
ghost_feed	20	50	50	#6BA583	#D75442

shape	color	size	angle (rad)	scale	icon*

icon	#3d85c6	40	1,571	1	0x263A

* 图标可以是以下值之一: 

shape	color	flipped	mirrored	mode
bars_pattern	#5091CC	FALSE	FALSE	0

一些属性的可能值：

- `linestyle`：`[0 (solid), 1 (dotted), 2 (dashed), 3 (large dashed)]`
- `linewidth`：`[1, 2, 3, 4]`
- `horzLabelsAlign`：`["center", "left", "right"]`
- `vertLabelsAlign`：`["top", "middle", "bottom"]`
- `leftEnd`, `rightEnd`：`[0 (Normal), 1 (Arrow)]`
- `bars_pattern` - `mode`：
  `[0 (HL Bars, 1 (Line-Close), 2 (OC Bars), 3 (Line-Open), 4 (Line-High), 5 (Line-Low), 6 (Line-HL/2)]`

订阅

订阅对象由图表方法返回。此对象允许您订阅和取消订阅图表事件。它有两种方法：

---

subscribe(object, method, singleshot)

1. `object` 是一个用作 `method` 函数的 `this` 指针的上下文对象。如果您不需要上下文，请使用 `null` 。
2. `method` 是事件发生时要调用的方法
3. `singleshot` 是一个可选的参数。如果您希望在事件第一次发生时自动取消订阅，请将其设置为“true”。

---

unsubscribe(object, method)

使用在 `subscribe` 函数中相同的 `object` 和 `method` 取消订阅事件。

---

unsubscribeAll(object)

使用在 `subscribe` 函数中相同的 `object` 来取消全部订阅。

交易元语

交易原语是一种低级机制，旨在为您提供对交易原语特性的最详细控制。

通用数据

属性

您可以为交易行对象的某些属性使用特殊值 `inherit` 这将使图表库使用该属性的出厂默认值。 您可以开启 `trading_options` 功能来显示交易界面。

您不能控制特定对象的可见性 - 在图表属性窗口的交易选项卡中可以为用户提供持仓、订单和交易指令的通用属性。

颜色和字体

您可以使用以下字符串格式设置颜色：

1. "#RGB"
2. "#RRGGBB"
3. "rgb(red, green, blue)"
4. "rgba(red, green, blue, alpha)"

您可以使用以下字符串格式设置字体： `<bold> <italic> <size>pt <family>` 。颜色和字体字符串将自动解析，以确定GUI控件的值。

线条样式

支持以下线条样式：

Style	Value
Solid	0

Dotted	1
Dashed	2

回调

1. 您可以通过 `this` 关键字在回调中访问API对象
2. 您可以将两个参数传递给回调注册函数 - 在这种情况下，第一个参数是一个对象，它将作为第一个参数传递给回调函数（请参阅示例）。
3. 如果未设置回调，则相应的按钮将被隐藏（影响 `onReverse`，`onClose` 和 `onCancel` 的回调).

在 K 线上做标记

图表库支持在K线上显示标记。 这里是一个例子：

每个标记可以有一个颜色，大小，标签和弹出消息。 如果它声明支持它们，则从后端请求标记。 标记旨在让您能够显示附加到K线的一些事件。 这里有一些例子：

- 新闻
- 一些比较特殊的K线配置
- 拆分/股息
- 等等

委托

Delegate是账户管理器中使用的对象，用于通知表中显示的订单，仓位和其他信息所发生的事件。

---

subscribe(object, member)

订阅事件. 1. `object` 是 `member` 的所有者，对于一个函数可以为null 2. `member` 是对象的方法

订阅事件: `object::member` 。

---

unsubscribe(object, member)

取消订阅事件: `object::member` 。

使用在 `subscribe` 函数中的相同对象和 `member` 取消订阅事件。

WatchedValue

WatchedValue对象由一些Trading Terminal方法返回。 使用这个对象，你可以获得/设置值，并在值改变时得到通知。

---

value()

返回当前值。

---

setValue(value)

设置新价值。

---

subscribe(callback, options)

1. `callback` 值被改变时被调用的函数
2. `options` 具有以下属性的对象：
    1. `once` - 如果是true，回调将只执行一次
    2. `callWithLast` - 如果它是true，回调将被执行以前的值（如果可用）

---

unsubscribe(callback)

使用您在 `subscribe` 函数中使用的相同函数来取消订阅更新。

指标 API

指标API

isUserEditEnabled()

如果用户能够 `删除/更改/隐藏` 指标，则返回 `true` 。

setUserEditEnabled(enabled)

1. `enabled` - `true` or `false`

启用或禁用用户的 `删除/更改/隐藏` 指标。

getInputsInfo()

返回所有输入的信息 - StudyInputInfo对象的数组。

getInputValues()

返回指标输入的值 - StudyInputValueItem对象。

setInputValues(inputs)

1. `inputs` 应该是StudyInputValueItem对象数组。

设置指标的输入值，它可以只包含您希望更改的输入。

mergeUp()

指标向上合并（如果可能）

mergeDown()

指标向下合并（如果可能）

unmergeUp()

取消指标向上合并（如果可能）

unmergeDown()

取消指标向下合并（如果可能）

changePriceScale(priceScale)

1. `priceScale` 应该是一个具有下值之一的字符串:
   - `left` - 将指标附加到左边的价格刻度
   - `right` - 将指标附加到右边的价格刻度
   - `no-scale` - 不要将指标纳入任何价格刻度。该指标将以'No Scale'模式添加
   - `as-series` - 将指标附加到住系列的价格刻度（仅当指标和主系列位于同一窗格时才适用）

改变指标的价格刻度

isVisible()

如果指标可见，将返回 `true`

setVisible(value)

1. `value` - `true` 或 `false`

显示/隐藏指标

bringToFront()

将指标置于所有其他图表对象之上。

sendToBack()

将指标放在所有其他图表对象后面。

applyOverrides(overrides)

1. `overrides` - 指标的新overrides

将 `overrides` 应用于指标。

注意: `overrides` 对象key不需要以指标名称开头。key应用于特定的指标。 例如，您应该使用 `style` 而不是 `Overlay.style` 来覆盖Overlay指标的当前样式。

Primitive types

StudyInputInfo

具有以下key的对象：

- `id` - 指标的输入id
- `name` - 输入的名称
- `type` - 输入的类型
- `localizedName` - 翻译为当前语言的输入名称

StudyInputValueItem

具有以下key的对象：

- `id` - 指标的输入id
- `value` - 输入的值

形状 API

形状API

isSelectionEnabled()

如果用户无法选择形状，则返回 `true` 。

setSelectionEnabled(enable)

1. `enable` - `true` 或 `false`

启用或禁用形状选择（请参阅 `createMultipointShape` 的 `disableSelection` 选项）。

isSavingEnabled()

如果形状未保存在图表上，则返回 `true` 。

setSavingEnabled(enable)

1. `enable` - `true` or `false`

启用或禁用在图表布局中保存形状（请参阅 `createMultipointShape` 的 `disableSave` 选项）。

isShowInObjectsTreeEnabled()

如果形状显示在 `对象树` 对话框中，则返回 `true` 。

setShowInObjectsTreeEnabled(enabled)

1. `enabled` - `true` 或 `false`

启用或禁用“对象树”对话框中的形状显示。

isUserEditEnabled()

如果用户可以删除/更改/隐藏形状，则返回 `true` 。

setUserEditEnabled(enabled)

1. `enabled` - `true` 或 `false`

启用或禁用用户删除/更改/隐藏形状

bringToFront()

将线条工具放在所有其他图表对象的顶部。

sendToBack()

将线条工具放在所有其他图表对象后面。

getProperties()

获取形状的所有属性。

setProperties(properties)

1. `properties` - 具有新属性的对象。 它应该与getProperties中的对象具有相同的结构。 它只能包含您要覆盖的属性。

设置形状的属性。

getPoints()

返回形状的点 - PricedPoint 对象的数组。

setPoints(points)

1. `points` - 一个带有形状新点的数组。 每个形状的格式与createMultipointShape方法的 `points` 参数相同。

设置形状的新点。

原始类型

PricedPoint

具有以下属性的对象：

- `price` - 点的价格
- `time` - 点的时间

选择 API

clear()

清除选择。

---

add(id)

1. `id` : `entityId`

通过 `id` 将指定的对象添加到选择中。 `id` 是绘图或指标的ID。 如果该对象不存在，则抛出错误。

---

add(ids)

1. `ids` : `entityId` 数组

通过 `ids` 将指定的对象数组添加到选择中。 `ids` 是绘图或指标ID的数组。如果其中某个id不存在，则抛出错误。

---

set(id)

1. `id` : `entityId`

清除当前的选择并通过 `id` 将指定的对象添加到选择中。 `set(id)` 与 `clear();add(id)` 相同。如果该对象不存在，则抛出错误。

---

set(ids)

1. `ids` : `entityId` 数组

清除当前选择并选择由 `ids` 指定的对象。 `ids` 是绘图或指标ID的数组。 `set(ids)` 与 `clear();add(ids)` 相同。如果其中任何一个对象不存在，则抛出错误。

---

remove(id)

1. `id` : `entityId`

从选择中删除指定的对象。 `id` 是绘图或指标ID。如果该对象未被选择，则此方法不执行任何操作。如果该对象不存在，则抛出错误。

---

remove(ids)

1. `ids` : `entityId` 数组

通过 `ids` 将指定的对象数组清除选择。 `ids` 是绘图或指标ID的数组。如果某个id不存在，则抛出错误。

---

contains(id)

1. `id` : `entityId` 数组

检查指定的对象是否被选择。 `id` 是绘图或指标ID。如果该对象未被选择，则返回 `false` 。如果该对象不存在，则抛出错误。

---

allSources()

返回所有已被选择对象的id数组。

---

isEmpty()

检查选择是否为空。如果图表上没有选择的对象，则返回 `true` 。

---

onChanged()

返回可用于订阅选择更改的订阅对象。

多项选择:

多个选择仅适用于使用以下规则的形状:

- 如果您在选择中添加指标，则清空选择并选择该指标。
- 如果在当前所选对象为指标时，向选区添加形状，则会清空选择并选择该形状。
- 如果向选择中添加对象数组，则其工作方式就像逐个添加这些对象一样。

例:

```
var chart = tvWidget.activeChart();
// 将所有选择更改打印到控制台
chart.selection().onChanged().subscribe(
  null,
  s => console.log(chart.selection().allSources())
);
// 创建指标
var studyId = chart.createStudy("Moving Average", false, false, [10]);
// 将指标添加到选择中（[<id>]打印到控制台）
chart.selection().add(studyId);
// 清空选择（[]打印到控制台）
chart.selection().clear();
```

窗格 API

hasMainSeries()

如果窗格包含主系列，则返回 `true` 。

---

getLeftPriceScales()

返回价格刻度 Api的实例数组，它允许您与左侧价格刻度进行交互。 如果窗格左侧没有任何价格刻度，则改数组可能为空。

---

getRightPriceScales()

返回价格刻度 Api的实例数组，它允许您与右侧价格刻度进行交互。 如果窗格右侧没有任何价格刻度，则改数组可能为空。

---

getMainSourcePriceScale()

返回:

- 价格刻度 Api的一个实例，它允许您与主窗口的价格刻度进行交互
- 如果主窗口未附加到任何价格格刻度，则为 `null` (处于“无比例”模式)

---

getHeight()

从1.15版开始

返回窗格的高度。

---

setHeight()

从1.15版开始

Sets the pane's height.

---

paneIndex()

从1.15版开始

返回窗格的索引，它是0到所有窗格计数-1之间的数字。

---

moveTo(paneIndex)

从1.15版开始

将窗格移动到新位置， `paneIndex` 应该是介于0和所有窗格计数-1之间的数字。

价格刻度 Api

价格刻度 Api

getMode()

返回价格刻度的当前模式 。

setMode(newMode)

1. `newMode` - 为价格刻度创建新模式

改变价格刻度的当前模式。

isInverted()

从1.15版开始

返回价格刻度是否反转。

setInverted(isInverted)

从1.15版开始

1. `isInverted` - 设置价格刻度的新反转状态。

改变价格刻度为当前的倒置状态。

getVisiblePriceRange()

从1.15版开始

返回价格刻度的当前可见价格范围。返回是带有 `from` 和 `to` 的对象，这是价格刻度的可见范围的边界。

setVisiblePriceRange(range)

从1.15版开始

设置当前价格刻度的可见价格范围， `range` 是带有 `from` 和 `to` 的对象，这是价格刻度可见范围的边界。

原始类型

PriceScaleMode

以下为价格刻度的可用模式:

- `0` - 价格刻度的正常模式
- `1` - 价格刻度的对数模式
- `2` - 价格刻度的百分比模式
- `3` - 索引到价格刻度的 100 模式

窗格和刻度特性

以下是创建指标时窗格和刻度如何工作的详细说明。

首先，所有指标分为两种：**价格指标**和**非价格指标**。价格指标的例子是**移动平均线**和**布林线**。 这些指标的结果与其输入源具有相同的维度。 价格指标默认情况下和**放在与其来源相同的窗格上**，并固定在**相同的价格刻度内和**。 相比之下，非价格指标的结果和**具有不同的维度**。例如，随机指标始终返回 0 到 100 之间的值。 默认情况下，非价格指标会**放置在单独的窗格中**。

- 您可以使用 `forceOverlay` 更改指标的默认窗格。如果设置为 `true` ，则将指标放在主窗格（包含主系列数据的窗格）上。此选项不会影响默认的价格刻度。此标志仅适用于 `非价格指标` 。目前默认情况下无法在单独的窗格中放置价格指标。
- 您可以使用 `priceScale` 更改默认价格刻度。如果未设置，则图表的特性如上所述。 `no-scale` 为指标设置**无比例（全屏）模式**。
  即使放置价格指标时， `left` 和 `right` 也会将指标放在新的价格刻度内。 `as-series` 选项将指标放在与主系列相关的相同价格刻度内。

下表说明了这一点

	价格指标, force_overlay=true	价格指标, force_overlay=false	非价格指标, force_overlay=true	非价格指标, force_overlay=fa
left	放置在主窗格的新左侧刻度上	放置在主窗格的新左侧刻度上	放置在主窗格的新左侧刻度上	放置在单独窗格的新左侧刻度上
right	放置在主窗格的新右侧刻度上	放置在主窗格的新右侧刻度上	放置在主窗格的新右侧刻度上	放置在单独窗格的新右侧刻度上
no-scale	在主窗格上放置“无比例”	在主窗格上放置“无比例”	在主窗格上放置“无比例”	在单独窗格上放置“无比例”
as-series	在主窗格上设置相同的价格刻度	在主窗格上设置相同的价格刻度	在主窗格上设置相同的价格刻度	在单独窗格上设置相同的价格刻度