View.js - API参考手册

目录

[1 View静态方法 1](#_Toc489948647)

[1) ofId(id) 1](#_Toc489948648)

[2) ifExists() 1](#_Toc489948649)

[3) listAll() 1](#_Toc489948650)

[4) setAsDefault(id) 1](#_Toc489948651)

[5) isDirectlyAccessible() 1](#_Toc489948652)

[6) setIsDirectlyAccessible(accessible) 1](#_Toc489948653)

[7) getActiveView() 1](#_Toc489948654)

[8) getDefaultView() 1](#_Toc489948655)

[9) setSwitchAnimation(animationFunction) 1](#_Toc489948656)

[10) getSwitchAnimation() 2](#_Toc489948657)

[11) getActiveViewOptions() 2](#_Toc489948658)

[12) hasActiveViewOption(name) 2](#_Toc489948659)

[13) getActiveViewOption(name) 2](#_Toc489948660)

[14) implIsPortrait(impl) 2](#_Toc489948661)

[15) navTo(targetViewId[, ops]) 3](#_Toc489948662)

[16) changeTo(targetViewId[, ops]) 3](#_Toc489948663)

[17) back(ops) 3](#_Toc489948664)

[18) forward(ops) 3](#_Toc489948665)

[19) setDocumentTitle(title) 4](#_Toc489948666)

[20) beforeInit(callback) 4](#_Toc489948667)

[21) ready(callback) 4](#_Toc489948668)

[22) setInitializer(initializer[, execTime]) 4](#_Toc489948669)

[23) on(type, handler) 4](#_Toc489948670)

[24) off(type, handler) 4](#_Toc489948671)

[25) fire(type, data[, async=true]) 5](#_Toc489948672)

[2 View属性 5](#_Toc489948673)

[1) SWITCHTYPE\_HISTORYFORWARD 5](#_Toc489948674)

[2) SWITCHTYPE\_HISTORYBACK 5](#_Toc489948675)

[3) SWITCHTYPE\_VIEWNAV 5](#_Toc489948676)

[4) SWITCHTYPE\_VIEWCHANGE 5](#_Toc489948677)

[5) currentState 5](#_Toc489948678)

[6) Logger 5](#_Toc489948679)

[7) layout 5](#_Toc489948680)

[3 View实例方法 5](#_Toc489948681)

[1) on(type, handler) 5](#_Toc489948682)

[2) off(type, handler) 5](#_Toc489948683)

[3) fire(type, data[, async=true]) 6](#_Toc489948684)

[4) getLatestEventData(eventName) 6](#_Toc489948685)

[5) getContext() 6](#_Toc489948686)

[6) clearContext() 6](#_Toc489948687)

[7) getId() 6](#_Toc489948688)

[8) getDomElement() 6](#_Toc489948689)

[9) find(selector) 6](#_Toc489948690)

[10) findAll(selector) 6](#_Toc489948691)

[11) setLayoutAction(layoutAction[,layoutWhenLayoutChanges=true]) 7](#_Toc489948692)

[12) hasParameter(name) 7](#_Toc489948693)

[13) getParameter(name) 7](#_Toc489948694)

[14) seekParameter(name) 7](#_Toc489948695)

[15) isReady() 7](#_Toc489948696)

[16) isActive() 7](#_Toc489948697)

[17) isDefault() 7](#_Toc489948698)

[18) isDirectlyAccessible() 8](#_Toc489948699)

[19) setAsDirectlyAccessible(isDirectlyAccessible) 8](#_Toc489948700)

[20) setTitle(title) 8](#_Toc489948701)

[21) getTitle() 8](#_Toc489948702)

[22) setFallbackViewId(fallbackViewId) 8](#_Toc489948703)

[23) getFallbackView() 8](#_Toc489948704)

[4 View实例属性 9](#_Toc489948705)

[1) logger 9](#_Toc489948706)

[2) config 9](#_Toc489948707)

[3) context 9](#_Toc489948708)

[5 View实例 - 配置集合实例（view.config）方法 9](#_Toc489948709)

[1) has(key) 9](#_Toc489948710)

[2) get(name) 9](#_Toc489948711)

[3) applyAll() 9](#_Toc489948712)

[4) listAll() 9](#_Toc489948713)

[6 View实例 - 配置集合实例属性 9](#_Toc489948714)

[7 View实例 - 配置项实例方法 9](#_Toc489948715)

[1) getName() 9](#_Toc489948716)

[2) getValue (dftValue) 10](#_Toc489948717)

[3) setValue (value[,overrideExistingValue=false]) 10](#_Toc489948718)

[4) getApplication () 10](#_Toc489948719)

[5) setApplication (application) 10](#_Toc489948720)

[6) apply () 10](#_Toc489948721)

[7) reflectToDom () 10](#_Toc489948722)

[8 View实例 - 配置项实例属性 10](#_Toc489948723)

[9 View实例 - 上下文实例方法 11](#_Toc489948724)

[1) has(name) 11](#_Toc489948725)

[2) set(name, value) 11](#_Toc489948726)

[3) get(name) 11](#_Toc489948727)

[4) remove(name) 11](#_Toc489948728)

[5) clear() 11](#_Toc489948729)

[10 View实例 – 上下文实例属性 11](#_Toc489948730)

[11 View – 日志静态方法 11](#_Toc489948731)

[1) ofName(name) 11](#_Toc489948732)

[2) isGloballyEnabled() 12](#_Toc489948733)

[3) setIsGloballyEnabled(isEnabled) 12](#_Toc489948734)

[12 View实例 – 日志实例（view.logger）方法 12](#_Toc489948735)

[1) isEnabled() 12](#_Toc489948736)

[2) setIsEnabled(isEnabled) 12](#_Toc489948737)

[3) getName() 12](#_Toc489948738)

[4) debug(template[, placeHolderValues]) 12](#_Toc489948739)

[5) info(template[, placeHolderValues]) 13](#_Toc489948740)

[6) warn(template[, placeHolderValues]) 13](#_Toc489948741)

[7) error(template[, placeHolderValues]) 13](#_Toc489948742)

[8) log(template[, placeHolderValues]) 13](#_Toc489948743)

[13 View – 布局（View.layout）静态方法 14](#_Toc489948744)

[1) getLayoutWidth() 14](#_Toc489948745)

[2) getLayoutHeight() 14](#_Toc489948746)

[3) getBrowserWidth() 14](#_Toc489948747)

[4) getBrowserHeight() 14](#_Toc489948748)

[5) isLayoutPortrait() 14](#_Toc489948749)

[6) isLayoutLandscape() 14](#_Toc489948750)

[7) isBrowserPortrait() 14](#_Toc489948751)

[8) isBrowserLandscape() 15](#_Toc489948752)

[9) getLayoutWidthHeightRatio() 15](#_Toc489948753)

[10) getBrowserWidthHeightRatio() 15](#_Toc489948754)

[11) getExpectedWidthHeightRatio() 15](#_Toc489948755)

[12) setExpectedWidthHeightRatio(ratio) 15](#_Toc489948756)

[13) init(ops) 16](#_Toc489948757)

[14) doLayout() 16](#_Toc489948758)

[15) addLayoutChangeListener(listener) 16](#_Toc489948759)

[16) removeLayoutChangeListener(listener) 17](#_Toc489948760)

1. View静态方法
2. ofId(id)

[用途]：查找给定ID对应的视图实例，如果不存在则创建，否则返回已经存在的实例。

[入参]：

1. id - {String} 视图编号（或视图ID）

[返回]：{View} - View实例

1. ifExists()

[用途]：判断指定ID的视图是否存在。

[入参]：无

[返回]：{Boolean} - 视图是否存在。true：视图存在；false：视图不存在

1. listAll()

[用途]：列举所有视图

[入参]：无

[返回]：{ViewArray} – 视图实例数组

1. setAsDefault(id)

[用途]：设置指定ID的视图为默认视图

[入参]：

1. id – {String} 视图编号（或视图ID）

[返回]：{Function} – View静态对象

1. isDirectlyAccessible()

[用途]：判断视图默认是否可以直接访问

[入参]：无

[返回]：视图默认是否可以直接访问。true：可以直接访问；false：不能直接访问

1. setIsDirectlyAccessible(accessible)

[用途]：设置视图默认是否可以直接访问

[入参]：

1. accessible – {String} 是否可以直接访问

[返回]：{Function} – View静态对象

1. getActiveView()

[用途]：获取当前的活动视图

[入参]：无

[返回]：{View} - View实例。如果没有视图处于活动状态，则返回null

1. getDefaultView()

[用途]：获取默认视图

[入参]：无

[返回]：{View} - View实例。如果没有默认视图，则返回null

1. setSwitchAnimation(animationFunction)

[用途]：设置视图切换动画。注：该方法可以接受4个参数，如：

animationFunction(srcElement, tarElement, type, render)。其中srcElement是离开的视图对应的DOM元素，tarElement是进入的视图对应的DOM元素，type是视图的切换方式（浏览器后退：View.SWITCHTYPE\_HISTORYBACK, 浏览器前进：View.SWITCHTYPE\_HISTORYFORWARD, 压入历史堆栈：View.SWITCHTYPE\_VIEWNAV, 替换当前堆栈：View.SWITCHTYPE\_VIEWCHANGE），render则是动画播放完毕后，通知View.js重新渲染界面的方法（如果设置了视图切换动画，但没有在动画播放完毕后调用该方法，则界面将不会发生变化。）。

[入参]：

1. {Function} - animationFunction 视图切换动画

[返回]：{Function} – View静态对象

1. getSwitchAnimation()

[用途]：获取设置的视图切换动画

[入参]：无

[返回]：{Function} – 视图切换动画

1. getActiveViewOptions()

[用途]：获取当前的活动视图的视图选项集合

[入参]：无

[返回]：{JsonObject} – 视图选项集合。key：选项名称；value：选项取值。如果当前的活动视图没有视图选项，则返回null

1. hasActiveViewOption(name)

[用途]：判断当前的活动视图的视图选项中是否含有特定名称的选项

[入参]：

1. name – {String} 选项名称。区分大小写

[返回]：{Boolean} – 给定名称的视图选项是否存在。如果视图选项为空，或对应名称的选项不存在，则返回false

1. getActiveViewOption(name)

[用途]：获取当前的活动视图的视图选项中特定名称的选项

[入参]：

1. name – {String} 选项名称。区分大小写

[返回]：{String} – 视图选项取值。如果视图选项为空，或对应名称的选项不存在，则返回null

1. implIsPortrait(impl)

[用途]：提供自定义的“判断当前是否是竖屏（或需要以竖屏方式渲染）”方法。方法需要返回布尔值。true：竖屏；false：横屏。开发者在通过API：view.setLayoutAction(actionFunc)设置视图的布局动作时，可以通过第二个参数：layoutWhenLayoutChanges设定“外层布局改变时，是否执行布局动作”。如果设置为true，则View.js将监听渲染的布局尺寸的变化，当其发生变化时再次执行开发者提供的布局动作。而布局尺寸的变化，对于移动设备的横竖屏切换动作同样适用。

View.js默认使用浏览器的宽度和浏览器的高度判定当前是否处于竖屏模式。开发者可以根据自己的布局方式，调整这一标准，以满足“横竖屏切换时，重新布局”的需要。

[入参]：

1. impl – {Function} 实现方法

[返回]：{Function} – View静态对象

1. navTo(targetViewId[, ops])

[用途]：以“压入历史堆栈”的方式切换视图

[入参]：

1. targetViewId – {String} 目标视图ID
2. ops – {JsonObject} 可选。切换动作的控制选项
3. ops.withAnimation – {Boolean} 是否播放动画。默认为：true
4. ops.params – {JsonObject} 视图参数集合。key：参数名称；value：参数取值。默认为：null
5. ops.options – {JsonObject} 视图选项集合。key：选项名称；value：选项取值。默认为：null

[返回]：{Function} – View静态对象

1. changeTo(targetViewId[, ops])

[用途]：以“替换当前堆栈”的方式切换视图

[入参]：

1. targetViewId – {String} 目标视图ID
2. ops – {JsonObject} 可选。切换动作的控制选项
3. ops.withAnimation – {Boolean} 是否播放动画。默认为：true
4. ops.params – {JsonObject} 视图参数集合。key：参数名称；value：参数取值。默认为：null
5. ops.options – {JsonObject} 视图选项集合。key：选项名称；value：选项取值。默认为：null

[返回]：{Function} – View静态对象

1. back(ops)

[用途]：回退到上一个视图

[入参]：

1. ops – {JsonObject} 可选。切换动作的控制选项
2. ops.withAnimation – {Boolean} 是否播放动画。默认为：true
3. ops.params – {JsonObject} 视图参数集合。key：参数名称；value：参数取值。默认为：null

ops.options – {JsonObject} 视图选项集合。key：选项名称；value：选项取值。默认为：null

[返回]：{Function} – View静态对象

1. forward(ops)

[用途]：前进到下一个视图

[入参]：

1. ops – {JsonObject} 可选。切换动作的控制选项
2. ops.withAnimation – {Boolean} 是否播放动画。默认为：true
3. ops.params – {JsonObject} 视图参数集合。key：参数名称；value：参数取值。默认为：null

[返回]：{Function} – View静态对象

1. setDocumentTitle(title)

[用途]：开发者可以设定视图级别的标题，但如果特定视图没有自定义标题，将使用文档标题来呈现

[入参]：

1. title – {String} 文档标题

[返回]：{Function} – View静态对象

1. beforeInit(callback)

[用途]：添加“视图将要初始化”监听器

[入参]：

1. callback – {Function} 回调方法

[返回]：{Function} – View静态对象

1. ready(callback)

[用途]：添加“视图就绪”监听器

[入参]：

1. callback – {Function} 回调方法

[返回]：{Function} – View静态对象

1. setInitializer(initializer[, execTime])

[用途]：设置视图初始化器。默认情况下，视图在文档就绪后将自动执行初始化动作，但开发者可以通过该API控制期望的初始化时机。

[入参]：

1. initializer – {Function} 初始化器。View.js在调用该方法时，将传递给该方法一个操作句柄，以使得开发者在决定可以初始化操作时，调用该句柄。如：

View.setInitializer(function(init){

//…

setTimeout(init, 1000);

});

1. execTime – {StringEnum} 初始化器的自动执行时机。domready：DOM就绪后执行；rightnow：立即执行。默认为：domready。

[返回]：{Function} – View静态对象

1. on(type, handler)

[用途]：添加事件监听器

[入参]：

1. type – {String} 事件类型
2. handler – {Function} 事件处理器

[返回]：无

1. off(type, handler)

[用途]：移除事件监听器

[入参]：

1. type – {String} 事件类型
2. handler – {Function} 事件处理器

[返回]：无

1. fire(type, data[, async=true])

[用途]：触发事件

[入参]：

1. type – {String} 事件类型
2. data – {Any} 附加的数据。亦即，需要传递至监听器的数据
3. async – {Boolean} 可选。是否以异步的方式执行处理器。如果为false，则仅当所有处理器都执行完毕后，下一行代码才会被执行。

[返回]：无

1. View属性
2. SWITCHTYPE\_HISTORYFORWARD

常量：视图切换操作类型：由浏览器前进操作触发

1. SWITCHTYPE\_HISTORYBACK

常量：视图切换操作类型：由浏览器后退操作触发

1. SWITCHTYPE\_VIEWNAV

常量：视图切换操作类型：由视图切换：View.navTo操作触发

1. SWITCHTYPE\_VIEWCHANGE

常量：视图切换操作类型：由视图切换：View.changeTo操作触发

1. currentState

{JsonObject} 当前的历史状态（等同于history.state）

1. Logger

{Function} 日志输出组件构造器

1. layout

{JsonObject} 布局工具组件

1. View实例方法
2. on(type, handler)

[用途]：添加事件监听器

[入参]：

1. type – {String} 事件类型
2. handler – {Function} 事件处理器

[返回]：无

1. off(type, handler)

[用途]：移除事件监听器

[入参]：

1. type – {String} 事件类型
2. handler – {Function} 事件处理器

[返回]：无

1. fire(type, data[, async=true])

[用途]：触发事件

[入参]：

1. type – {String} 事件类型
2. data – {Any} 附加的数据。亦即，需要传递至监听器的数据
3. async – {Boolean} 可选。是否以异步的方式执行处理器。如果为false，则仅当所有处理器都执行完毕后，下一行代码才会被执行。

[返回]：无

1. getLatestEventData(eventName)

[用途]：获取指定名称对应的事件最后一次被触发时所附加的数据

[入参]：

1. eventName – {String} 事件名称。亦即，事件类型

[返回]：{Any} – 事件最后一次被触发时所附加的数据。如果该事件尚未被触发，或最后一次触发时没有附加数据，则返回null。

1. getContext()

[用途]：获取视图上下文

[入参]：无

[返回]：{JsonObject} – 视图上下文。开发者可以在视图上下文中存取数据。

1. clearContext()

[用途]：清空视图上下文

[入参]：无

[返回]：{View} – View实例

1. getId()

[用途]：获取视图编号。视图编号与对应的DOM元素的ID相同

[入参]：无

[返回]：{String} – 视图编号

1. getDomElement()

[用途]：获取视图对应的DOM元素

[入参]：无

[返回]：{HTMLElement} – 视图对应的DOM元素

1. find(selector)

[用途]：在视图内查找特定元素

[入参]：

1. selector – {String} CSS选择器。有关选择器的详细语法，请查阅document.querySelector

[返回]：{HTMLElement} - 找到的元素

1. findAll(selector)

[用途]：在视图内查找多个元素

[入参]：

1. selector – {String} CSS选择器。有关选择器的详细语法，请查阅document.querySelectorAll

[返回]：{NodeList} - 找到的元素列表

1. setLayoutAction(layoutAction[,layoutWhenLayoutChanges=true])

[用途]：设置视图布局方法

[入参]：

1. layoutAction – {Function} 布局方法
2. layoutWhenLayoutChanges – {Boolean} 可选。外层布局改变时，是否执行布局动作

[返回]：{View} – View实例

1. hasParameter(name)

[用途]：判断视图参数中是否含有指定名称的参数

[入参]：

1. name – {String} 参数名。区分大小写

[返回]：{Boolean} – 指定名称的参数是否存在。true：存在；false：不存在

1. getParameter(name)

[用途]：获取视图参数中指定名称的参数取值

[入参]：

1. name – {String} 参数名。区分大小写。

[返回]：{Any} - 指定名称对应的参数取值。如果指定名称的参数并不存在，则返回null。如果没有指定参数名，则返回整个参数集合

1. seekParameter(name)

[用途]：搜索指定名称的参数取值。区分大小写。搜索顺序：视图参数 > 视图选项 > 地址栏参数。

[入参]：

1. name – {String} 参数名。区分大小写

[返回]：{Any} - 指定名称对应的参数取值。如果指定名称的参数并不存在，则返回null

1. isReady()

[用途]：判断当前视图是否已经就绪。视图在第一次进入后变为“已就绪”。

[入参]：无

[返回]：{Boolean} - 视图是否已经就绪

1. isActive()

[用途]：判断当前视图是否为活动视图。亦即，当前视图是否是用户看到的视图。

[入参]：无

[返回]：{Boolean} – 视图是否是活动视图

1. isDefault()

[用途]：判断当前视图是否为默认视图。

[入参]：无

[返回]：{Boolean} - 视图是否为默认视图。声明了data-view-default的属性，且取值赋值为true的视图是默认视图。默认视图只能有一个。

1. isDirectlyAccessible()

[用途]：判断当前视图是否可以通过地址栏手动直接访问。如果视图级别上没有配置（包括API设置与DOM属性配置），则使用默认配置。如果默认配置也没有设置，则以“视图不允许直接访问”方式对待。

[入参]：无

[返回]：{Boolean} 视图是否可以通过地址栏手动直接访问。

1. setAsDirectlyAccessible(isDirectlyAccessible)

[用途]：设置当前视图是否允许通过地址栏手动直接访问。

[入参]：

1. isDirectlyAccessible – {Boolean} 是否可以手动直接访问。

[返回]：{View} – View实例

1. setTitle(title)

[用途]：设置视图标题

[入参]：

1. title – {String} 标题。如果为null，则移除对应的DOM属性，改用网页的默认标题

[返回]：{View} – View实例

1. getTitle()

[用途]：获取视图标题

[入参]：无

[返回]：视图标题。如果当前视图没有自定义标题，则返回null

1. setFallbackViewId(fallbackViewId)

[用途]：设置回退视图

[入参]：

1. fallbackViewId – {String} 回退视图ID

[返回]：{View} – View实例

1. getFallbackView()

[用途]：获取该视图的回退视图。注：该方法返回的视图，是最终可以呈现出来的视图，不一定等同于设定的回退视图（设定的回退视图也可以是不能直接访问的）。如果所有回退视图都不能直接访问，则呈现默认视图。

[入参]：无

[返回]：{View} – View实例

1. View实例属性
2. logger

{Logger} 与特定视图相关联的日志输出组件实例

1. config

{ViewConfigSet} 与特定视图相关联的配置集合实例

1. context

{ViewContext} 与特定视图相关联的上下文实例

1. View实例 - 配置集合实例（view.config）方法
2. has(key)

[用途]：判断特定名称的配置项是否存在

[入参]：

1. key – {String} 配置项名称

[返回]：配置项是否存在

1. get(name)

[用途]：获取指定名称对应的配置项，如果对应的配置项不存在，则自动创建一个

[入参]：

1. key – {String} 配置项名称

[返回]：{ViewConfig} – 配置项实例

1. applyAll()

[用途]：应用所有配置

[入参]：无

[返回]：无

1. listAll()

[用途]：列举所有的配置项名称

[入参]：无

[返回]：视图的配置项名称数组

1. View实例 - 配置集合实例属性

无

1. View实例 - 配置项实例方法
2. getName()

[用途]：获取配置项名称

[入参]：无

[返回]：{String} – 配置项名称

1. getValue (dftValue)

[用途]：获取配置项取值

[入参]：

1. dftValue – {Any} 配置值没有指定时的默认值

[返回]：{Any} 配置取值

1. setValue (value[,overrideExistingValue=false])

[用途]：设置配置项取值

[入参]：

1. value – {Any} 要设定的配置项取值
2. overrideExistingValue – {Boolean}如果已经设置过配置项取值，是否使用新取值覆盖既有取值。如果配置项取值尚未设置过，则无论是否覆盖，均执行赋值动作

[返回]：{ViewConfiguration} – 视图配置项

1. getApplication ()

[用途]：获取配置的应用方法。应用方法，是指配置项设定新的取值并应用时，需要执行的动作。

[入参]：无

[返回]：{Function} - 配置项的应用方法。如果没有提供应用方法，则返回null

1. setApplication (application)

[用途]：设置配置的应用方法

[入参]：

1. application– {Any} 配置的应用方法。应用方法，是指配置项设定新的取值并应用时，需要执行的动作。

[返回]：{ViewConfiguration} – 视图配置项

1. apply ()

[用途]：应用配置。其中this指向的上下文为当前的配置项。如果配置项没有提供应用方法，该方法在将什么也不做（但不会报错）

[入参]：无

[返回]：{ViewConfiguration} – 视图配置项

1. reflectToDom ()

[用途]：将配置以"data-viewconfig\_[name]=[value]"的方式附加至视图的DOM元素上。开发者可以使用该特性，以达到“通过CSS响应配置项”的目的。

[入参]：无

[返回]：{ViewConfiguration} – 视图配置项

1. View实例 - 配置项实例属性

无

1. View实例 - 上下文实例方法
2. has(name)

[用途]：判定上下文中是否含有指定名称的键。

[入参]：

1. name – {String} 名称

[返回]：指定名称的键是否存在。

1. set(name, value)

[用途]：设置属性。如果相同名称的属性已经存在，则覆盖。

[入参]：

1. name – {String} 属性名称
2. value – {Any} 属性取值

[返回]：{ ViewContext } – 视图上下文

1. get(name)

[用途]：获取指定名称的属性。如果属性不存在，则返回undefined

[入参]：

1. name – {String} 属性名称

[返回]：属性取值。如果属性不存在，则返回undefined

1. remove(name)

[用途]：移除指定名称的属性，并返回既有的属性值

[入参]：

1. name – {String} 属性名称

[返回]：{Any} – 属性的既有取值

1. clear()

[用途]：清空所有属性

[入参]：无

[返回]：{ ViewContext } – 视图上下文

1. View实例 – 上下文实例属性

无

1. View – 日志静态方法
2. ofName(name)

[用途]：获取指定名称的logger。如果相同名称的实例已经存在，则返回既有的实例。如果不存在，则自动创建一个

[入参]：

1. name – {String} 实例名称

[返回]：{Logger} – 日志输出组件实例

1. isGloballyEnabled()

[用途]：判断组件的全局状态：是否被启用。如果组件实例被禁用，或全局状态被警用，则任何日志输出都将不可见。

[入参]：无

[返回]：{Boolean} - 是否被启用

1. setIsGloballyEnabled(isEnabled)

[用途]：设置组件的全局状态：是否被启用

[入参]：

1. isEnabled – {Boolean}是否被启用

[返回]：无

1. View实例 – 日志实例（view.logger）方法
2. isEnabled()

[用途]：判断日志组件是否启用

[入参]：无

[返回]：{Boolean} - 日志组件是否启用

1. setIsEnabled(isEnabled)

[用途]：设置日志输出的启用状态

[入参]：

1. isEnabled – {Boolean} 是否启用

[返回]：{Logger} – 日志组件实例

1. getName()

[用途]：获取日志组件名称

[入参]：无

[返回]：{String} -日志组件名称

1. debug(template[, placeHolderValues])

[用途]：以DEBUG级别输出日志，如：

var logger = View.ofId(“detail”).logger;

// -> 0629 09:34:51 [View#detail]: hello, world

logger.debug("hello, {}", "world");

// -> 0629 09:34:51 [View#detail]: Dynamic contents: true, 12, text {}

logger.error("Dynamic contents: {}, {}, {} \\{}", true, 12, 'text', 33, 45);

logger.setIsEnabled(false);

logger.info("No output would be generated");

[入参]：

1. template– {String} 包含占位符的输出内容模板。占位符：{}，使用“\”进行转义。
2. placeHolderValues – {Any} 分别对应于内容模板中占位符的取值。参数个数不限。

[返回]：无

1. info(template[, placeHolderValues])

[用途]：以INFO级别输出日志

[入参]：

1. template– {String} 包含占位符的输出内容模板。占位符：{}，使用“\”进行转义。
2. placeHolderValues – {Any} 分别对应于内容模板中占位符的取值。参数个数不限。

[返回]：无

1. warn(template[, placeHolderValues])

[用途]：以WARN级别输出日志

[入参]：

1. template– {String} 包含占位符的输出内容模板。占位符：{}，使用“\”进行转义。
2. placeHolderValues – {Any} 分别对应于内容模板中占位符的取值。参数个数不限。

[返回]：无

1. error(template[, placeHolderValues])

[用途]：以ERROR级别输出日志

[入参]：

1. template– {String} 包含占位符的输出内容模板。占位符：{}，使用“\”进行转义。
2. placeHolderValues – {Any} 分别对应于内容模板中占位符的取值。参数个数不限。

[返回]：无

1. log(template[, placeHolderValues])

[用途]：输出日志

[入参]：

1. template– {String} 包含占位符的输出内容模板。占位符：{}，使用“\”进行转义。
2. placeHolderValues – {Any} 分别对应于内容模板中占位符的取值。参数个数不限。

[返回]：无

1. View – 布局（View.layout）静态方法
2. getLayoutWidth()

[用途]：获取布局宽度

[入参]：无

[返回]：{Integer} – 实际布局的宽度

1. getLayoutHeight()

[用途]：获取布局高度

[入参]：无

[返回]：{Integer} – 实际布局的高度

1. getBrowserWidth()

[用途]：获取浏览器宽度

[入参]：无

[返回]：{Integer} – 获取当前浏览器的宽度

1. getBrowserHeight()

[用途]：获取浏览器高度

[入参]：无

[返回]：{Integer} – 获取当前浏览器的高度

1. isLayoutPortrait()

[用途]：判断当前布局方向是否是竖屏方向（宽小于等于高）

[入参]：无

[返回]：{Boolean} - 当前布局方向是否是竖屏方向（宽小于等于高）

1. isLayoutLandscape()

[用途]：判断当前布局方向是否是横屏方向（宽大于高）

[入参]：无

[返回]：{Boolean} - 当前布局方向是否是横屏方向（宽大于高）

1. isBrowserPortrait()

[用途]：判断当前浏览器方向是否是竖屏方向

[入参]：无

[返回]：{Boolean} - 当前浏览器方向是否是竖屏方向

1. isBrowserLandscape()

[用途]：判断当前浏览器方向是否是横屏方向

[入参]：无

[返回]：{Boolean} - 当前浏览器方向是否是横屏方向

1. getLayoutWidthHeightRatio()

[用途]：获取当前布局尺寸的宽高比

[入参]：无

[返回]：{Float} - 当前布局尺寸的宽高比

1. getBrowserWidthHeightRatio()

[用途]：获取当前浏览器窗口尺寸的宽高比

[入参]：无

[返回]：{Float} - 当前浏览器窗口尺寸的宽高比

1. getExpectedWidthHeightRatio()

[用途]：获取蓝图（亦即，高保真效果图）的宽高比。开发者开发应用时，大多是根据特定效果图进行的。效果图的尺寸通常是固定的尺寸，如以iPhone 5的320 \* 568为参考标准的设计二倍图：640 \* 1136。而应用的实际运行环境是不固定的，此时开发者可以通过设置蓝图宽高比，以达到更好的表现效果。如：设置蓝图的宽高比为 320 / 568 ≈ 0.56，并设定PC上的布局表现方式，以实现“使用PC浏览手机网页时，使用 [高度等于浏览器高度，宽高比为iPhone5的宽高比] 的尺寸来呈现应用主题内容” 的效果。如下所示：

View.layout.setExpectedWidthHeightRatio(320 / 568).init({

autoReLayoutWhenResize: true,

layoutAsPcLandscape: function(width, height){

document.body.style.cssText = "width: " + width + "px; height: " + height + "px; margin: 0 auto;";

}

}).doLayout();

[入参]：无

[返回]：{Float} – 设置的蓝图宽高比。View.js默认将其设置为iPhone5的宽高比，亦即：320 / 568。如果开发者通过API：setExpectedWidthHeightRatio(ratio)设置了其它的宽高比，则返回开发者设置的数值。

1. setExpectedWidthHeightRatio(ratio)

[用途]：设置蓝图（亦即，高保真效果图）的宽高比。

[入参]：

1. ratio – {Float} 蓝图的宽高比

[返回]：{JsonObject} – View.layout对象

1. init(ops)

[用途]：初始化，用于设置View的布局方式等。

[入参]：

1. ops – {JsonObject} 初始化配置
2. [ops.autoReLayoutWhenResize=true] – {Boolean} 当视口尺寸发生变化时，是否自动重新布局
3. [ops.layoutAsMobilePortrait] – {Function} 手机以竖屏方式使用应用时的布局方式。方法可以接受两个参数：width, height，分别代表View.js计算后，可以用来布局的宽度和高度。如下所示：

View.layout.setExpectedWidthHeightRatio(320 / 568).init({

autoReLayoutWhenResize: true,

layoutAsPcLandscape: function(width, height){

document.body.style.cssText = "width: " + width + "px; height: " + height + "px; margin: 0 auto;";

}

}).doLayout();

默认的布局方式为：“宽等于浏览器的宽，高等于浏览器的高”

1. [ops.layoutAsMobileLandscape] – {Function} 手机以横屏方式使用应用时的布局方式。默认的布局方式为：“宽等于浏览器的宽，高等于浏览器的高”
2. [ops.layoutAsTabletPortrait] – {Function} 平板以竖屏方式使用应用时的布局方式。默认的布局方式为：“宽等于浏览器的宽，高等于浏览器的高”
3. [ops.layoutAsTabletLandscape] – {Function} 平板以横屏方式使用应用时的布局方式。默认的布局方式为：“宽等于浏览器的宽，高等于浏览器的高”
4. [ops.layoutAsPcPortrait] – {Function} PC桌面以竖屏方式使用应用时的布局方式。默认的布局方式为：“宽等于浏览器的宽，高等于浏览器的高”
5. [ops.layoutAsPcLandscape] – {Function} PC桌面以横屏方式使用应用时的布局方式。默认的布局方式为：“高等于浏览器的高，宽高比等于设定的蓝图宽高比”

[返回]：{JsonObject} – View.layout对象

1. doLayout()

[用途]：根据初始化时设置的各个模式下的浏览方式，结合设备当前的浏览方向和设备类型自动进行布局

[入参]：无

[返回]：{JsonObject} – View.layout对象

1. addLayoutChangeListener(listener)

[用途]：添加“布局发生改变”事件监听器

[入参]：

1. listener – {Function} 监听器。方法可以接受4个参数：layoutWidth, layoutHeight, browserWidth, browserHeight，分别代表布局宽度、布局高度、浏览器宽度，浏览器高度。如：

View.layout.addLayoutChangeListener(function(layoutWidth, layoutHeight, browserWidth, browserHeight){

//…

});

[返回]：{JsonObject} – View.layout对象

1. removeLayoutChangeListener(listener)

[用途]：移除“布局发生改变”事件监听器

[入参]：

1. listener – {Function} 监听器。方法可以接受4个参数：layoutWidth, layoutHeight, browserWidth, browserHeight，分别代表布局宽度、布局高度、浏览器宽度，浏览器高度。如：

View.layout.addLayoutChangeListener(function(layoutWidth, layoutHeight, browserWidth, browserHeight){

//…

});

[返回]：{JsonObject} – View.layout对象