Qunee for HTML5

Developer Guide

V 2.5

2016-4

上海酷利软件有限公司
Shanghai Kuli Software Co.,Ltd.
support@qunee.com

content

Qunee for HTML5 Developer Guide

Qunee for HTML5（hereinafter called Qunee）is a graphic component product based on HTML5 technology, for drawing clean and smooth network graph, which can be used for demands such as social network graph, topological graph, flow chart and map etc., JS component packaging, turn complicated to easy, easily building elegant Internet application and enterprise application, and make the on-line visualization of data easier.

Qunee - pronounced in “kju:ni”, meaning elegant and light

Qunee official website: http://qunee.com/index-en.html
Qunee techblog: http://blog.qunee.com
Online documents: http://doc.qunee.com/display/h5en
Online presentation: http://demo.qunee.com

The development and usage of Qunee will be introduced in this text in

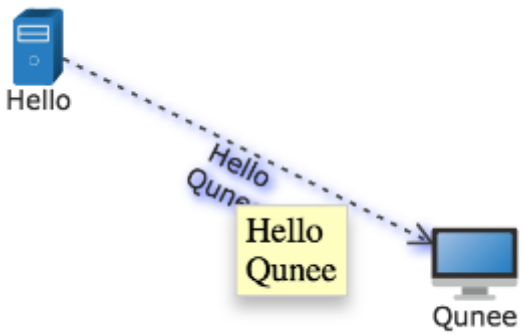
Hello Qunee

According to the tradition, the example of "Hello world" will be introduced firstly. The primary level of example of Qunee is to create two nodes and one edge, with the following code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Hello Qunee for HTML5</title>
   <meta charset="utf-8">
</head>
<body>
<div style="height: 500px;" id="canvas"/>
<script src="http://demo.qunee.com/lib/qunee-min.js"></script>
<script>
   var graph = new Q.Graph('canvas');
   var hello = graph.createNode("Hello", -100, -50);
   var qunee = graph.createNode("Qunee", 100, 50);
   var edge = graph.createEdge("Hello\nQunee", hello, qunee);
 edge.setStyle(Q.Styles.EDGE_LINE_DASH, [3,4]);
</script>
</body>
</html>
```

⚠ The HTML5 defines the declaration about type of simplified html file, which just needs to describe <!DOCTYPE HTML>

Operating interface



Overview

Qunee for HTML5 uses HTML5 technology. Know about relevant background knowledge before learning Qunee, such as Web, HTML5, Canvas, Javascript and CSS etc. Besides, as a kind of solution for Web images, Qunee has its own applicable scope and product features. Knowing about these features is helpful to better selection and usage by users.

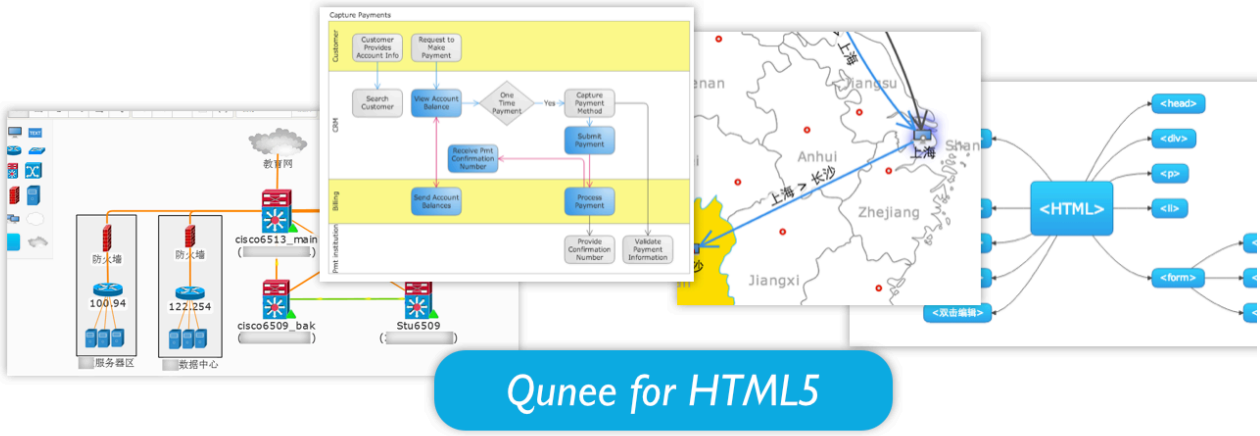
Qunee can be used for topological graph, flow chart, organization chart, floor plan of machine room and development of configuration software, with features of light weight, high efficiency and flexible expansion, supporting current main browsers (Safari, Firefox, Chrome, IE9+). It can be applied in different operation systems (Windows, Mac, Linux……) and mobile terminals(iOS, Android, Windows Phone……). By mobile development frame of PhoneGap, the mobile application programs can be developed.

Qunee provides solutions for Web：

- maps - metro map and statistical map
- Topological graph - social network graph, and network management chart
- Others - organization chart, mind map, and flow chart

Features of Qunee component:

- Light and high performance - supporting then thousand of elements, smooth operation
- Vector image - supporting vector image and infinitely-variable zooming
- Interaction experience - Roaming interaction, improving interaction events, supporting hand-held device
- Pay attention on details - GIF animation, rich gradient and level control etc.



Qunee for HTML5

Firefox Safari Chrome IE9+ Opera PhoneGap

Windows OS X iOS Android more...

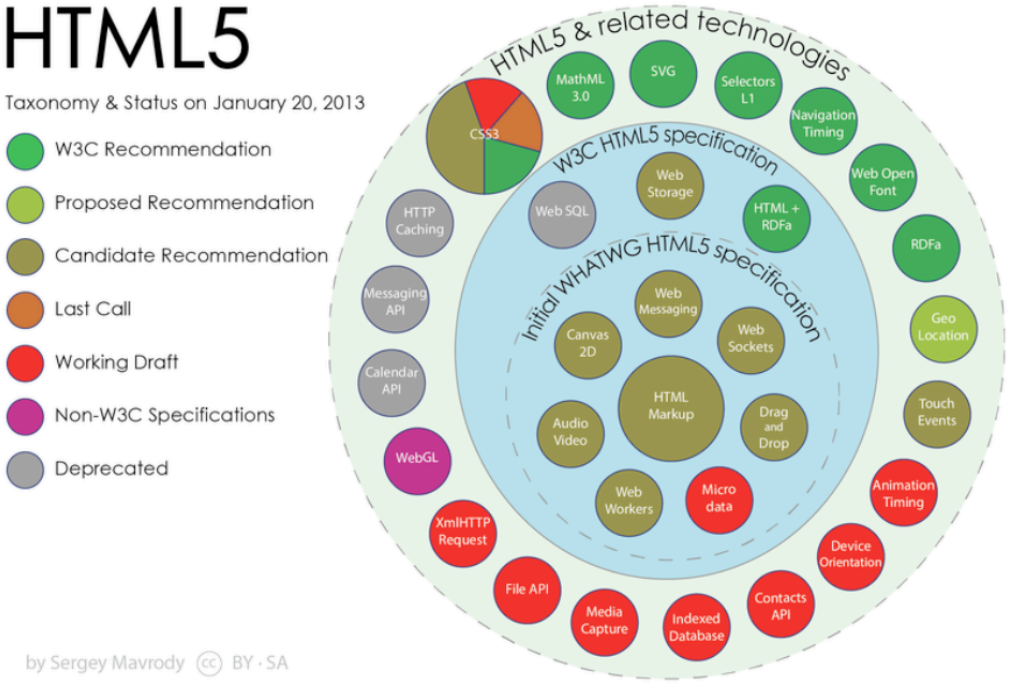
Background Knowledge

Relevant organizations of Web

- W3C - http://www.w3.org
- WHATWG - Work group of hypertext technology of webpage - http://www.whatwg.org

HTML5

HTML5 in narrow meaning means the next main revised version of HTML, the standard made by W3C. It is under development for the moment. On the basis of HTML 4.01 and XHTML1.0, HTML5 standard adds and revises some tag elements, among which there are medium-related <video>, <audio>, <canvas>, meanwhile integrating content of SVG. Elements of data include <section>, <article>, <header>, <nav>, and <menu>. Besides, new API is provided, such as 2D drawing, offline storage, drag and drop, messaging, management of browsing history, document API and position API etc.



For more introduction of HTML5, refer to:

- http://en.wikipedia.org/wiki/HTML5
- http://www.html5rocks.com
- http://dev.w3.org/html5/spec

The HTML5 in wider meaning includes HTML, CSS and JavaScript and any set of technical combination. Its target is to reduce the dependency the browser would have to the plug-in, providing rich RIA (rich client side) application.

- Why Selecting HTML5 Technology?

- HTML5 <Canvas>
- Canvas vs SVG
- Why Selecting Canvas Technology?

Why Selecting HTML5 Technology?

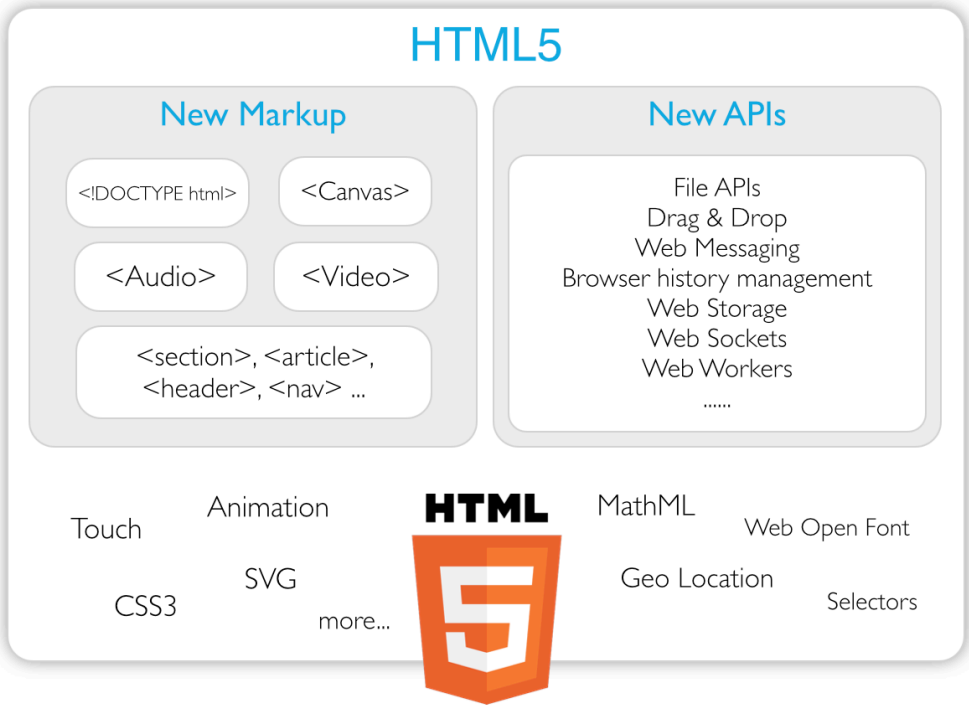
The application of previous rich client sides is mainly realized via plug-in technology, such as Adobe Flash, Microsoft Silverlight and Java Applet. All of them have some problems (to install plug-ins, not supporting mobile platform etc.). Compared with that, advantages of HTML5 include:

- Not necessary to install plug-in
- Support main browser and mobile devices
- International standards of w3c

With the development of technology, main browser gives more and more better support to HTML5. Webpage

doesn't show words and images any more, and uses technical combination of HTML5
Realize more functions: real-time messaging, local storage, image drawing; and realize rich image interface and animation via Canvas, SVG and CSS3 technologies.

With such technical support, professional image component is possible to be showed in the webpage. Qunee for HTML5 selects HTML5 technology, making the possibility realizable. Reducing technical threshold make high efficient, rich and dynamic image expression possible.



HTML5 <Canvas>

<Canvas> is the newly added label element in the HTML5 standard, which is showed before HTML5 standard,

firstly introduced in Safari browser of Apple. It is used to provide a group of pure 2D drawing API. At present main browsers have been supported. Qunee for HTML5 mainly uses Canvas technology to show image interface.

Canvas elements correspond to the type of HTML Canvas Element, inherit standard HTML Canvas Element t ype, and is the same with common webpage label element.

It exists in the tree of HTML DOM. Corresponding layout position and style property can be set via CSS. We call it canvas element. By the scripting language, we can draw 2D images on it .

With Canvas technology, the most-often used type is CanvasRenderingContext2D. It indicates the drawing of context, equal to java.aw t.Graphics2D in Java 2D. Provide drawn related functions, such as line drawing, graphic filling, word drawing, coordinate change and zooming etc.

There are a dozen of Canvas types, covering basic 2D drawing operation. Provide API comparison bottom layer. Internal core of Qunee for HTML5 will use these APIs, and make abstraction and packaging. Provide high level image elements. Using Qunee development application can use these advanced objects directly.
By this way, bottom layer of APIs will not be used, making the development easier.

If you want deeper customization, learn more bottom layer of API about Canvas. Refer to the following linkage:

- http://www.w3.org/TR/2dcontext/
- https://developer.mozilla.org/en/docs/HTML/Canvas
- http://www.w3schools.com/html/html5_canvas.asp

Canvas vs SVG

2D image drawing technology in HTML5

Canvas and SVG are main 2D graphic technologies in HTML5. The previous one provides the canvas label and drawing API. The post one provides one complete set of independent vector image language. There have been more than then years (from 2003.1 till now) since being W3C standard. Generally speaking, Canvas technology is much new. It develops from a small range to a wide scope of acceptance. Concern handling of grid images. SVG has long history. It has become the international standard very early, complicated, and developed slowly (Adobe SVG Viewer has no big update in recent ten years)

	Canvas	SVG
History	a new technology, by the private development of apple	has a long history, in 2003, has become the W3C standard
Functions	function is simple, 2D drawing API	feature rich, various graphics, filters, animation etc.
Features	pixels, only script driven	vector, XML, CSS, element operation
Support	mainstream browsers IE9+	mainstream browsers, IE9+, other SVG reader

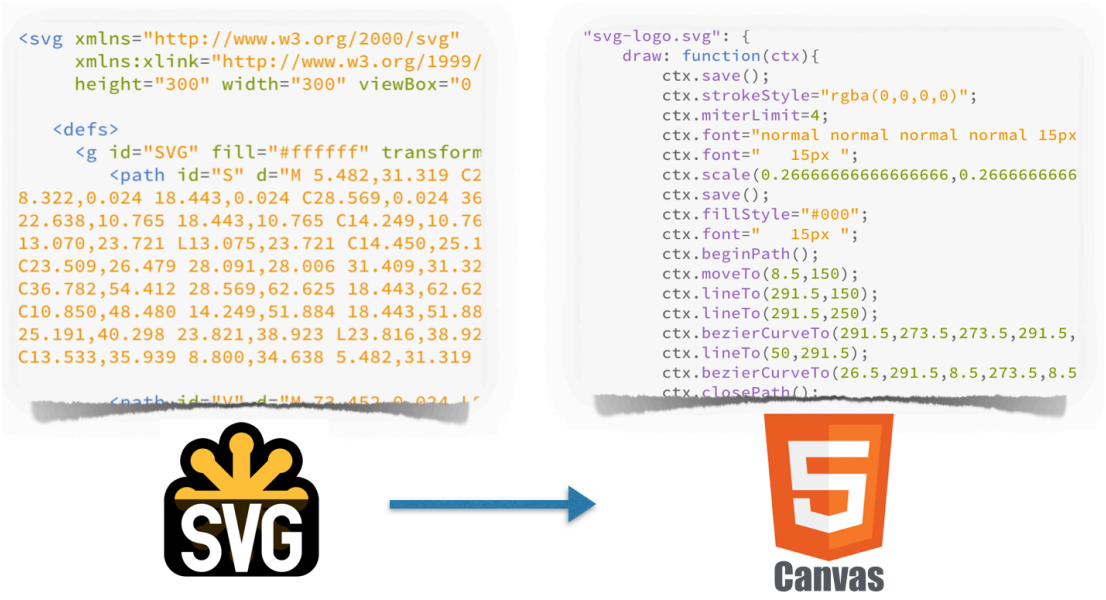
Canvas vs SVG

<canvas> and <svg> are image technologies recommended by HTML5. Canvas, based on elements, provides 2D drawing function, is a kind of HTML element type, and relies on HTML. It can draw images only via scripts; SVG is a vector and provides a series of image elements (Rect, Path, Circle, Line.... There are complete animation and event mechanism. It can be used independently, or built into HTML. SVG has become an international standard earlier. The current stable version is 1.1 –http://www.w3.org/TR/SVG/. Main features of two parts are showed in the following table:

	Canvas	SVG
Operation object	Based on the pixel (dynamic bitmap)	Based on the graphic elements
Element	A single HTML element	Graphic elements (Rect, Path, Line...)
Drive	Only script driven	Support script and CSS
Event interaction	User interaction to the pixel (x, y)	User interaction elements (rect, path)
Performance	Suitable for small area, big data applications	Suitable for large area, a small number of scenarios

SVG Converts to Canvas

SVG is an old technology. There are many existing SVG images and icons. These sources are also be used to Canvas, such as usage of CanvasRenderingContext2D#drawImage(), possible to draw SVG images in Canvas. In addition, there is a third-party base(canvg.js) that supports to analysis child elements of SVG, and draws via Canvas. If to convert SVG file to Canvas code. Use online tools of SVG2Canvas



Why Selecting Canvas Technology?

According to different features of two parts, Canvas and SVG have their own applicable scopes

Applicable scene of Canvas

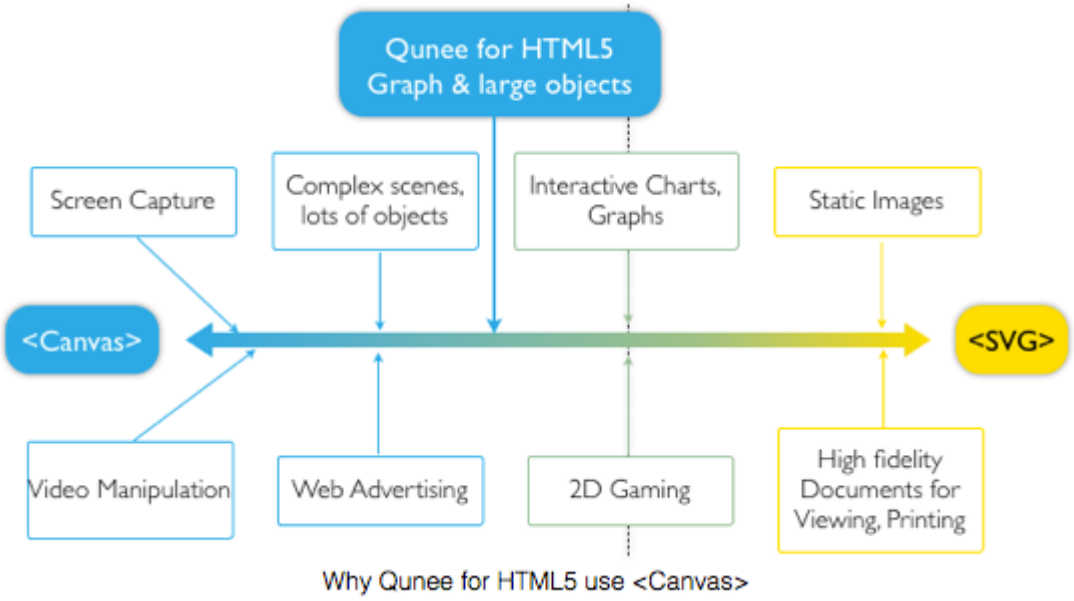
Functions provided by Canvas are more original, applicable for pixel handling, dynamic rendering and high volume of drawing.

Applicable scene of SVG

SVG function is more perfect, applicable to demonstration of static images, reviewing of high fidelity files and application scenes of printing

Qunee for HTML5 product positioning

Qunee for HTML product positioning covers: large data volume, dynamic rendering, and flexibly-expanded application scenes of graphic component. SVG can satisfy the demand of large data volume and pixel handling. So Qunee will finally select Canvas as main graphic technology.



Why Qunee for HTML5 use <Canvas>

Introduction of Graph Component

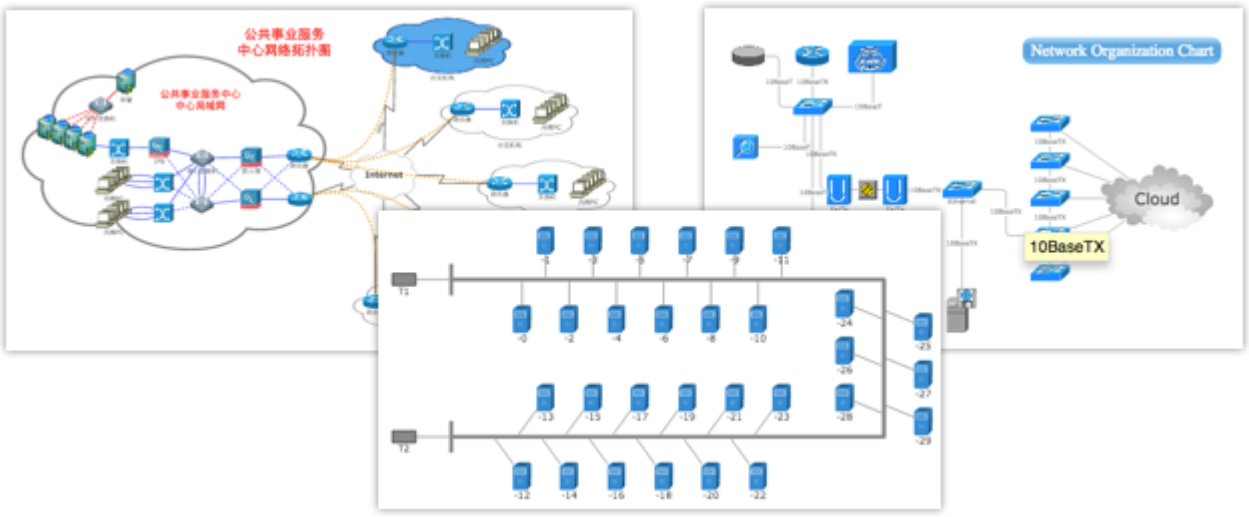
Qunee mainly provides Graph components. Different elements can be added to Graph (Node, Edge, Group,ShapeNode, Text, Bus...). The element can be configured with different presentation styles, or mounted or added with different UI elements (LabelUI, ImageUI....). Via the combination and layout, realize rich presentation effect and satisfy several application scenes:

Maps - metro map, statistic maps

Qunee provides rich vector images. For demonstration of point or line data, it is easily to solve. It can be applied to metro and pipelines.
Qunee supports roaming interaction and infinitely-variable zooming, not limiting coordinate scope. This is applicable to presentation of maps. With rich styles of polygon, it can be used to solve problems regarding map background and statistic maps.

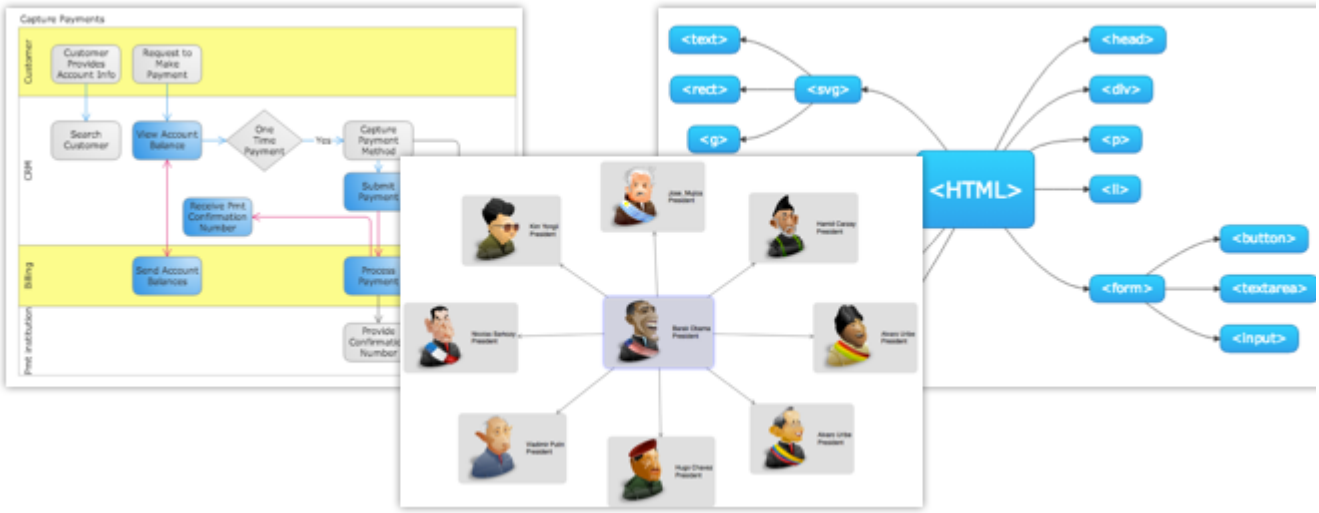
Topological graph - social network graph, and network management chart

Support node, edge and grouping and the like element types. With good level control, support then thousands of graphic elements, and smooth operation. Bring light, quick, high efficient visual experience, applicable to solve problems regarding presentation and interaction of web data.



Others - organization chart, mind map, and flow chart

With powerful tree-form of layouter, solve problems regarding automatic layout for tree-form of structural data. It can be applied to organization chart and mind map etc.
Provide rich basic images and arrow styles. Support edge types such as curve and orthogonality. Added with control of inclusion relation, problems regarding flow charts can be solved



Features of Qunee Component

Qunee is a set of elegant, high efficient and light solution for Web graphic component. We adopt several technologies to realize this target, and make continuous improvement. Qunee can answer to most of application scenes. For specific browsers, realize thousands of demonstration of magnitude elements via custom-made code.

Double cache drawing and partial update

Qunee realizes double cache drawing and partial update via Vanvas technology, so that the smoothness of interface can be ensured. Solve the performance performance when solving large data volume. Present thousands of element elements easily. Smooth roaming, zooming and interaction.

Graphic-theoretical algorithm

The relation of node and edge of Qunee adopts data structure of orthogonal list. This is good to convenience and analysis of topological image. In addition, lots of graphic-theoretical algorithm are used in the layout algorithm, helpful in case of large data

Design mode

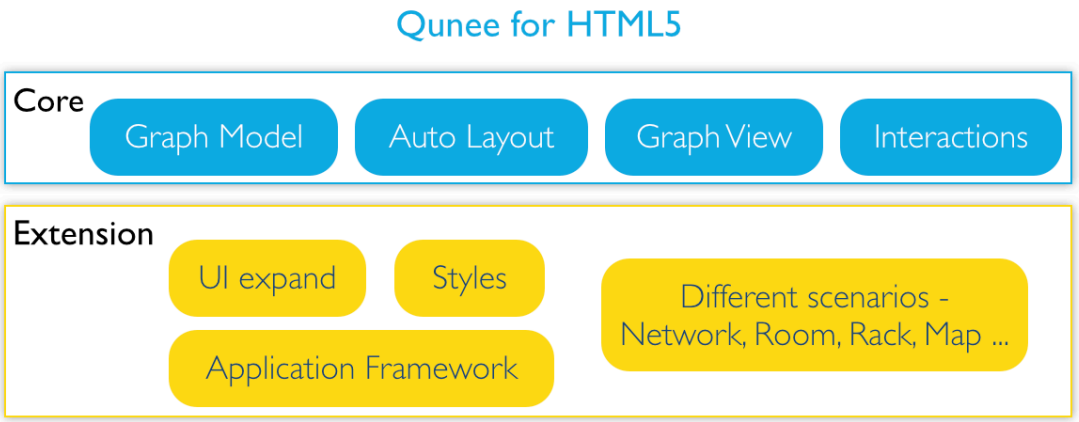
Qunee uses mature MVP design mode, making separation to data mode, UI presentation and interaction monitoring, optimizing graphic structure, realizing high efficiency graphic mode and graphic demonstration and flexible expansion mechanism

Cross-platform

Adopting HTML5 technology avoids limitation of operation system, applicable at all browsers supporting HTML5. Meanwhile, it supports mobile platform, conforms to the web-based demand, and avoids the installation of plug-ins. One development and one employment can cover all platforms.

Concern on graphic technology

Qunee focuses on graphic component technology. Our core components include: graphical-theoretical model, automatic layout, graphic presentation and user interaction, other auxiliary components (messaging, application frame, and styles), and the expansion of each special application scenes will be provided independently. This will ensure the light weight of core packages
and flexible expansion.



Quick Start

This chapter introduces the introduction on development of Qunee for HTML5. The development flow and attention matters are explained step by step via many examples.

Development Environments

Browser support

Google Chrome (recommended), Firefox, Safari, IE9+

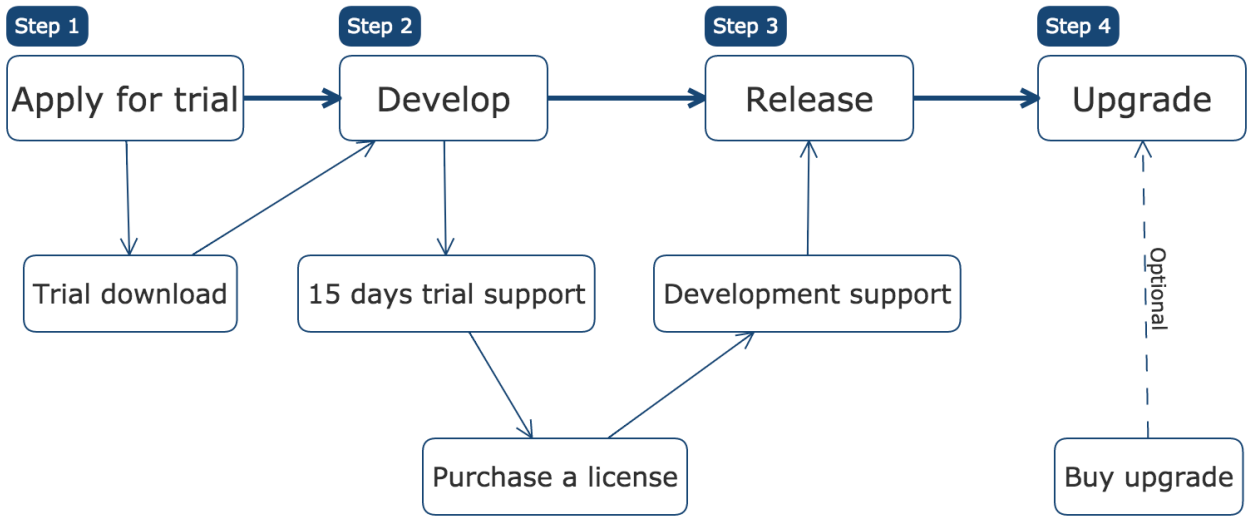
Recommended development tools

HTML is the text format. So there are many selection on development tools. It may be simple text editor, or complex visual edition tool. Recommend the following editors:

Light editors: Brackets, Sublime Text 2, TextMate 2
Integrated development environment: Jetbrains Webstorm (recommended), Netbeans 7, Eclipse WTP
Debugging tools: Google Chrome(recommended), Safari, Firefox

Apply Qunee development packages

Qunee for HTML5 development package can be applied online by visiting official website: http://qunee.com, Click for trial use, or send an E-mail to support@qunee.com directly. Note information of companies and people, convenient for better feedback and services.



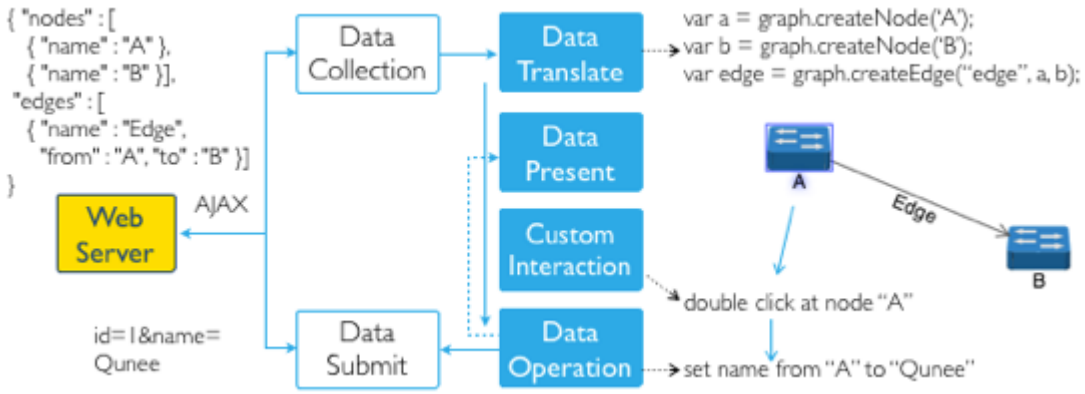
Start development

Start the development at the next step. Start from the example of Hello Qunee. If this example has been learned, continue to read the following documents

Development Flow

Qunee is a front-end technology, not related to backstage service and data messaging. Generally Qunee development application is used according to the following flows:

Data acquisition --> Data conversion --> Data presentation --> Interaction monitoring --> Data operation --> Data submission, where four steps in the middle are directly-related to Qunee.



Now we will simulate these flows to complete one example

- Data Acquisition
- Data Conversion
- Data Presentation
- Interaction and Operation

Download the complete code of example

work-process.zip

Data Acquisition

Build backstage data service to provide services via data base and technologies of web services (servlet, php, web socket, asp.net...). It is converted to XML,JSON,text or binary format to corresponding front ends, which acquires data via web messaging technology(ajax, web socket ...) and then submits to JavaScript engine for handling.

Backstage data

Generally the JSON data format is used, such as the following linkage of ./data-server

```
{
  "nodes": [
    {
      "name": "A",
      "x": -100,
      "y": -50,
      "id": 1
    },
    {
      "name": "B",
      "id": 2
    }
  ],
  "edges": [
    {
      "name": "Edge",
      "from": 1,
      "to": 2
    }
  ]
}
```

Request data

Request backstage data via AJAX or Web socket. Generally take the usage of AJAX

AJAX for acquiring backstage data for example

```
function request(url, params, callback, callbackError) {
    try {
        var req = new XMLHttpRequest();
        req.open('GET', encodeURI(url));
        req.onreadystatechange = function(e) {
            if (req.readyState != 4) {
                return;
            }
            if (200 == req.status) {
                var code = req.responseText;
                if (code && callback) {
                    callback(req.responseText);
                }
                return;
            }else{
                if (callbackError) {
                    callbackError();
                }
            }
        }
        req.send(params);
    } catch (error) {
        if (callbackError) {
            callbackError();
        }
    }
}
```

Usage

```
request("./data-server", "", onDataCollected);
```

⚠ Web socket technology has requirements for both Web server and browser, which has not gotten common support and application

Data Conversion

After the front end gets the data, firstly it has to be converted to data format supported by Javascript, so as to read data property, such as text data in JSON format. It can be analysed via JSON#parse(...). For the text in XML format, it can be converted to XML object via DOMParse#parseFromString

Conversion to JS object

Continue the above example. Analyze the text to JSON object

```
function onDataCollected(txt){
    var json = JSON.parse(txt);
    translateToQuneeElements(json);
}
```

Conversion to Qunee element

Images in Qunee graphic component have corresponding model objects, called Qunee element. User data has to firstly be converted to this element and then demonstrated in Qunee graphic component. Qunee elements have many types, support various presentation styles and expansion ways. Different data can be converted to different types of Qunee elements according to explicit intents. Set corresponding image styles. Special demastration effect can be realized by UI extension.

Convert JSON objects to Qunee elements, and add them to graphic vessel

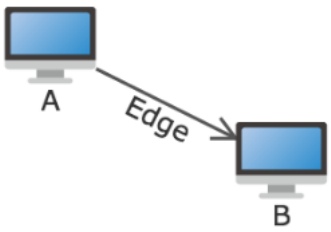
```
function translateToQuneeElements(json, graph){
    var map = {};
    if(json.nodes){
        Q.forEach(json.nodes, function(data){
            var node = graph.createNode(data.name, data.x || 0, data.y || 0);
            node.set("data", data);
            map[data.id] = node;
        });
    }
    if(json.edges){
        Q.forEach(json.edges, function(data){
            var from = map[data.from];
            var to = map[data.to];
            if(!from || !to){
                return;
            }
            var edge = graph.createEdge(data.name, from, to);
            edge.set("data", data);
        }, graph);
    }
}
```


Data Presentation

Data can be converted to Qunee element objects. After graphic components are added, view presented effect:

```
var graph = new Q.Graph("canvas");
function onDataCollected(txt){
    var json = JSON.parse(txt);
    translateToQuneeElements(json);
    graph.moveToCenter();
}
function translateToQuneeElements(json){
    if(json.nodes){
        Q.forEach(json.nodes, toQuneeNode);
    }
    if(json.edges){
        Q.forEach(json.edges, toQuneeEdge);
    }
}
request("./data-server", "", onDataCollected);
```

Operation effect



Automatic layout

In addition, for topological data without position information, the automatic layout of nodes can be completed by using automatic layout. Such as usage of spring layout

```
var layouter = new Q.SpringLayouter(graph);
layouter.start();
```

Interaction and Operation

Actual application has to response to user interaction. Qunee defaults to provide a group of interaction mode, allowing roaming, zooming, frame selection, moving and double response etc. Users may also ask for custom-made interaction according to service conditions. There are several ways to add monitoring for mouse keyboard. The most simple one is Graph#on***, such as monitoring double click event: graph.ondblclick = function(evt){ ... }

Data operation

During user interaction, data will be modified. Set the property or the style of Qunee element directly. The interface will make real-time update automatically.

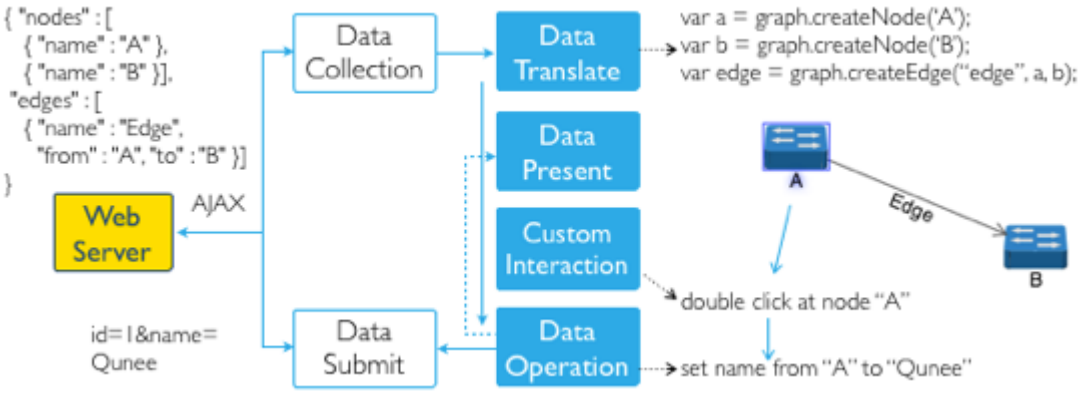
And then the effect after modification will be presented, such as double click to revise the element name. You may refer to the following codes:

Example - double click a graphic element, to revise the element name

```
graph.ondblclick = function (evt) {
    var node = graph.getElementByMouseEvent(evt);
    if (node) {
        var newName = prompt("New Name:");
        if (newName) {
            node.name = newName;
        }
    }
}
```

Data submission

Data modification at the front end of interface will generally be submitted to the backstage. By various messaging technologies at frontstage and backstage (such as AJAX, Web Socket), submit the information to be modified to the backstage server. Update corresponding data elements at front end after the results are received (see data operation)
, or repeat the flow of data acquisition.

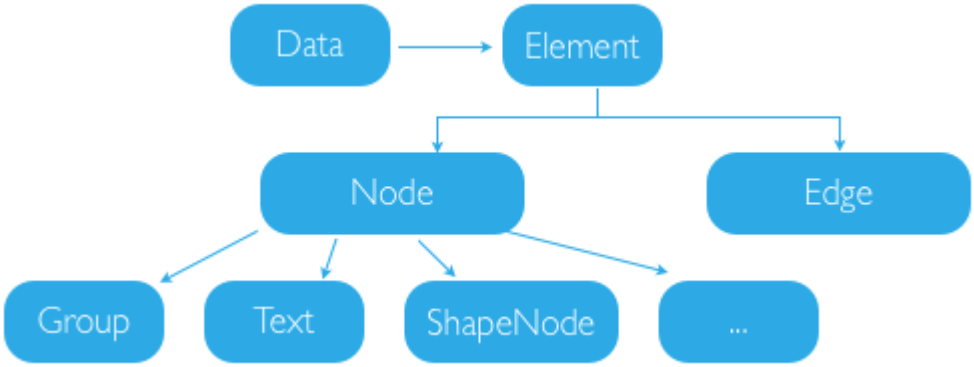


Download the complete code of example

work-process.zip

Graphic Element

Series of element types are provided in Qunee, including element base class, nodes, edges and extended type, with the only one ID, supporting event monitoring, style property and property of set membership. Different types have different purposes and features



Various element types are introduced hereinafter in details

- Element Base Class
- Node Element
- Edge Element
- Group Element
- Subnet Type

Element Base Class

Base class for graphic elements: Q.Element, simply called element, which provides basic support to graphic element, and basic functions such as event monitoring, property support and set membership etc. Two basic elements are provided on this basis: Q.Node and Q.Edge, separately for describing node and edge. In one graphic-theory, the Node represents a point, and Edge represents a side. The nodes are connected via edges. More edges exist among two nodes. In addition, there are some extension types: Q.Group, Q.Text, Q.ShapeNode etc.

Basic features of elements are introduced hereinafter

- Support to Element Event
- Support to Element Property
- Graphic Set Membership
- Adds UI Component

Support to Element Event

Every element object can be set with one listener, by which related events of elements can be handled, such as change event of element property, child change events and so on. In addition, the element event is generally divided into two steps. The "beforeEvent" is called before event handling. To go back to false, stop event execution. "onEvent" will be called after event execution.

Related properties and methods of event

- #listener - in GraphModel, the listener will be handled by GraphModel in unification, not directly on graph #beforeEvent(evt) - before event handling
- #onEvent(evt) - after event handling

Example

The following is the default code for setting child order. The handling process of internal event can be showed

```
setChildIndex: function(child, index) {
        if(!this._children || !this._children.length){
            return false;
        }
        var oldIndex = this._children.indexOf(child);
        if(oldIndex < 0 || oldIndex == index){
            return false;
        }
        var event = new ChildIndexChangeEvent(this, child, oldIndex, index);
        if(this.beforeEvent(event) === false){
            return false;
        }
        if(this._children.remove(child)){
            this._children.add(child, index);
        }
        this.onEvent(event);
        return true;
    }
```

Support to Element Property

Except direct setting of properties supported by JavaScript (by clicking operator or bracket operator, such as node.name = ‘qunee’;), add the setting methods for properties to graphic object:

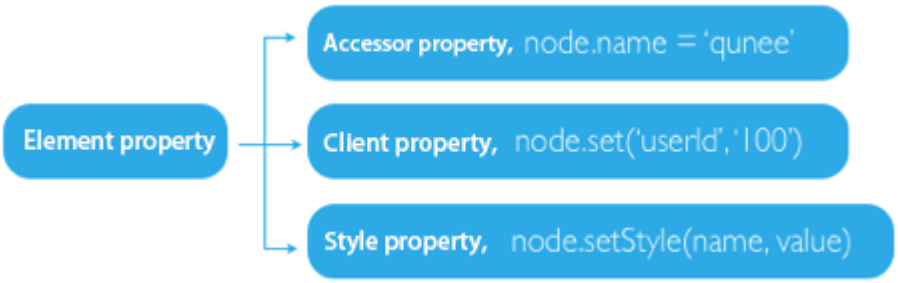
Client property: acquire and set by get/set function

#get(key) / #set(key, value)

Style property: acquire and set by getStyle/setStyle

#getStyle(name) / #setStyle(name, style)

Three properties of graphics



Example

Set the text label position of edge, and set user property ID to "100":

```
edge.setStyle(Q.Styles.LABEL_POSITION, Q.Position.CENTER_TOP);
edge.set(‘userId’, ‘100’);
```


Graphic Set Membership

Graphic object supports the set membership. Child may be added or deleted. Or parent node can be modified. The graphics can form a relation of tree-form graph in GraphModel.

- #children - child collection
- #parent - parent node
- #addChild - add child
- #clearChildren - clear child node
- #forEachChild - traverse child node
- #getChildAt - get child node
- #getChildIndex - get child order number

Example

Set up the set membership of nodes

```
var node = graph.createNode();
var child = graph.createNode("child");
child.parent = node;
//or use#addChild(...)
//node.addChild(child);
```


Adds UI Component

Looking from appearance, each graph is comprised of one main part and several children components. The node is defaulted to demonstrate one icon and text. The icon here is the main part of node, and one path for the main part of edge.

Mount child component

Except default graphic element, more child components may be mounted by the method of node.name = ‘qunee’. Each component can indicate specific information. Connect the graphic property and the child component, to realize the interactive effect of data

#addUI - add child components

Example

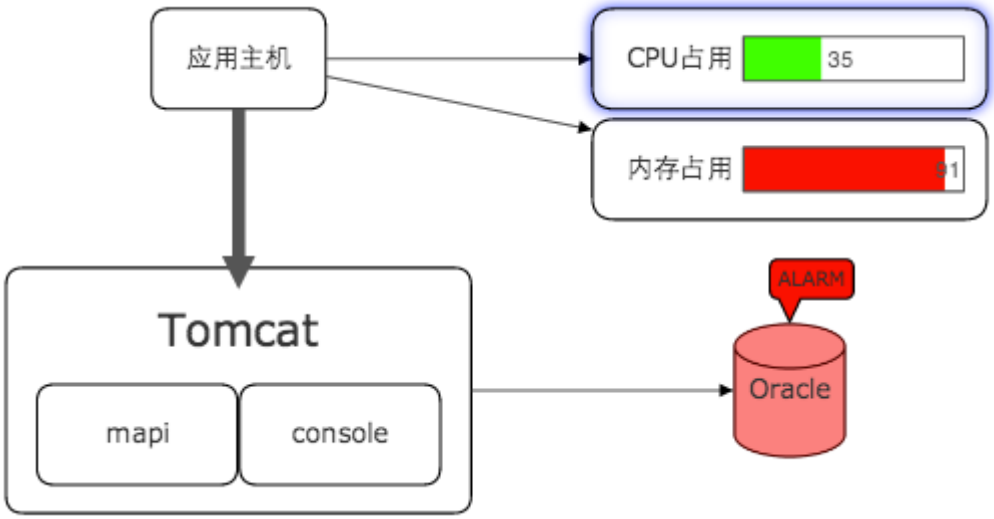
Adds the text label to the node. And bundle contents and font color of label in this text with node user property. Change the content and color by changing node properties

```
var nodeWithLabels = graph.createNode("Node with Labels", 0, -110);
var label2 = new Q.LabelUI();
label2.position = Q.Position.CENTER_TOP;
label2.anchorPosition = Q.Position.CENTER_BOTTOM;
label2.border = 1;
label2.padding = new Q.Insets(2, 5);
label2.showPointer = true;
label2.offsetY = -10;
label2.backgroundColor = Colors.yellow;
label2.fontSize = 16;
label2.fontStyle = "italic 100";
nodeWithLabels.addUI(label2, [{
    property : "label2",
    propertyType : Q.Consts.PROPERTY_TYPE_CLIENT,
    bindingProperty : "data"
   }, {
    property : "label2.color",
    propertyType : Q.Consts.PROPERTY_TYPE_CLIENT,
    bindingProperty : "color"
   }]);
nodeWithLabels.set("label2", "another label");
nodeWithLabels.set("label2.color", Colors.blue);
```


Effect

Extend UI component

Realize rich effect by extension of child component, such as customizing one bar graph, mounted to a node, indicating CPU occupation ratio, or mounted to the status of warning and bubbling marked node.



Node Element

The point object in the graphical-theory of node element (Q.Node) can represent the device in the topological graph, step in the flow chart, people in the social network chart and block and point of the map in the actual application

- Create Node
- Node Position
- Topological Property of Nodes
- Node Follow
- Node Image
- Node Label
- Node Style Property

Create Node

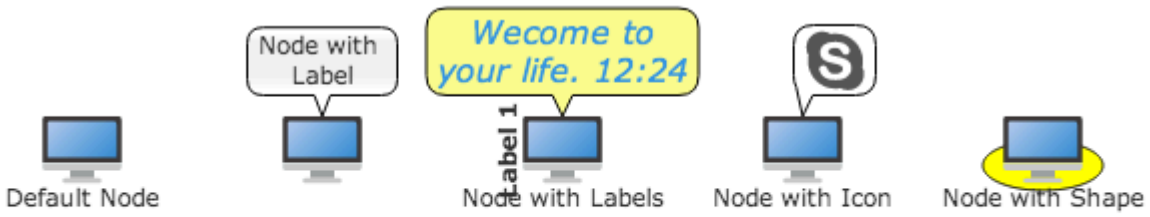
Generally nodes are created via the following codes:

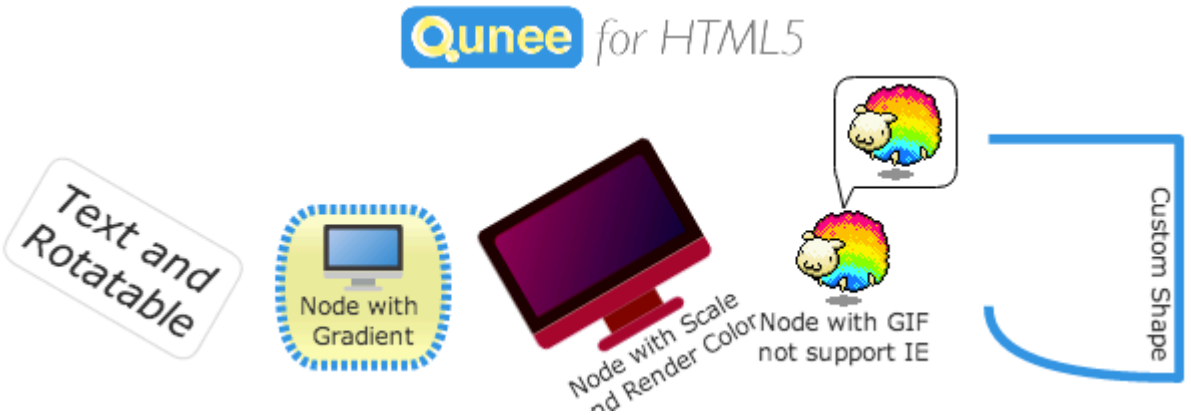
```
var node = graph.createNode(“node name”, 100, 100);
```

Nodes may also be created via the constructor

```
var node = new Q.Node();
node.name = “node name”;
node.x = 100;
node.y = 100;
graph.graphModel.add(node);
```

Set node property and style, support various vector or grid image and animation, support path and shape, support different positions and mounting points and support node rotation. Display layer can be set. Several child component can be added. Rich representation effect can be expressed from nodes.





Node Position

Property of node position

- #location - Coordinate position
- #rotate - Rotate angle
- #zIndex - Layer property for controlling the display layer of graph. The high value one is displayed at the upper layer
- #anchorPosition - Anchor position, which can control the align method of nodes. The same position coordinate and different anchors will present different effects. It is defaulted to be the center of node graph

Anchor position

Example

The node at same position and different anchors

```
var node = graph.createNode("Right");
node.anchorPosition = Q.Position.LEFT_TOP;
node = graph.createNode("Left");
node.anchorPosition = Q.Position.RIGHT_TOP;
node = graph.createNode("TOP");
node.anchorPosition = Q.Position.CENTER_BOTTOM;
```

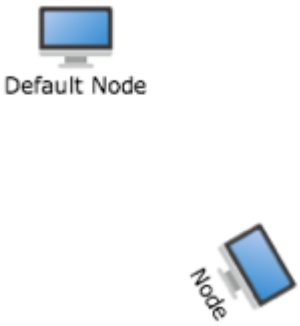
Operation effect:

Example

Create nodes, and set appointed position, rotate angle and anchor position


```
var defaultNode = graph.createNode("Default Node");
var node = graph.createNode("Node");
node.location = new Q.Point(100, 100);
node.rotate = Math.PI / 3;
node.anchorPosition = Q.Position.LEFT_TOP;
```

Operation effect:



Topological Property of Nodes

Nodes connected to the graph connect with edges to form a topological relation. Nodes in the topological graph contains information about edges connected to it, and all edge conditions among two points.

- #edgeCount - Quantity of edges connected to this node
- #forEachEdge() - Traverse all edges connected to this node
- #forEachInEdge() - Traverse all edges connected into this node
- #forEachOutEdge() - Traverse all edges connected from this node
- #getEdgeBundle(node) - Acquire edge collection between appointed node and this node

Example

Create two nodes and three edges to acquire the edge count between two nodes

```
var a = graph.createNode('A');
var b = graph.createNode('B');
graph.createEdge(a, b);
graph.createEdge(a, b);
graph.createEdge(a, b);
alert(a.getEdgeBundle(b).edges.length);
```

The operation result indicates "3"

Node Follow

Node(A) can follow another node (B) to move. In this case, node B is called host node. Node A is called follower

- host - host node
- addFollower - add following node
- clearFollowers - clear following node

Example

Set node A to follow Host, and node B to follow A

```
var host = graph.createNode("Host");
var a = graph.createNode("A follow Host", -50, 50);
a.host = host;
var b = graph.createNode("B follow A", 50, 50);
b.host = a;
```

Operation effect:

Node Image

Node Image

In the graphical interface, the node is generally expressed to an icon and text label, where the icon is the main part of node, and the text label is the auxiliary component. The main part not only supports the icon (static image) but also supports Canvas code, GIF animation, SVG and vector graph (Q.Path) etc.

Set node graph:

Q.Node#image - main part of node

Example

Set GIF icon as the main part of node

```
var nodeWithGIF = graph.createNode("Node with GIF\nnot support IE", 120, 110);
nodeWithGIF.image = "./images/sheep.gif";
```

Operation effect:

Built-in icon

All built-in icons of Qunee are showed hereinafter. The Canvas code is used, with 'Q-' as prefix of name

```
var graph = new Q.Graph('canvas');
graph.moveToCenter();

var images = Q.getAllImages();
var x = 0, y = 0;
images.forEach(function(image){
    graph.createNode(image, x, y).image = image;
    x += 100;
})
```

Operation effect is as follows

Canvas code icon - recommended

Qunee recommends to use Canvas code as the icon, with two advantages: support vector; synchronized load.

Qunee built-in icons are applied with this method.

Let's see how to make this kind of graph from following examples

Make Canvas code icon

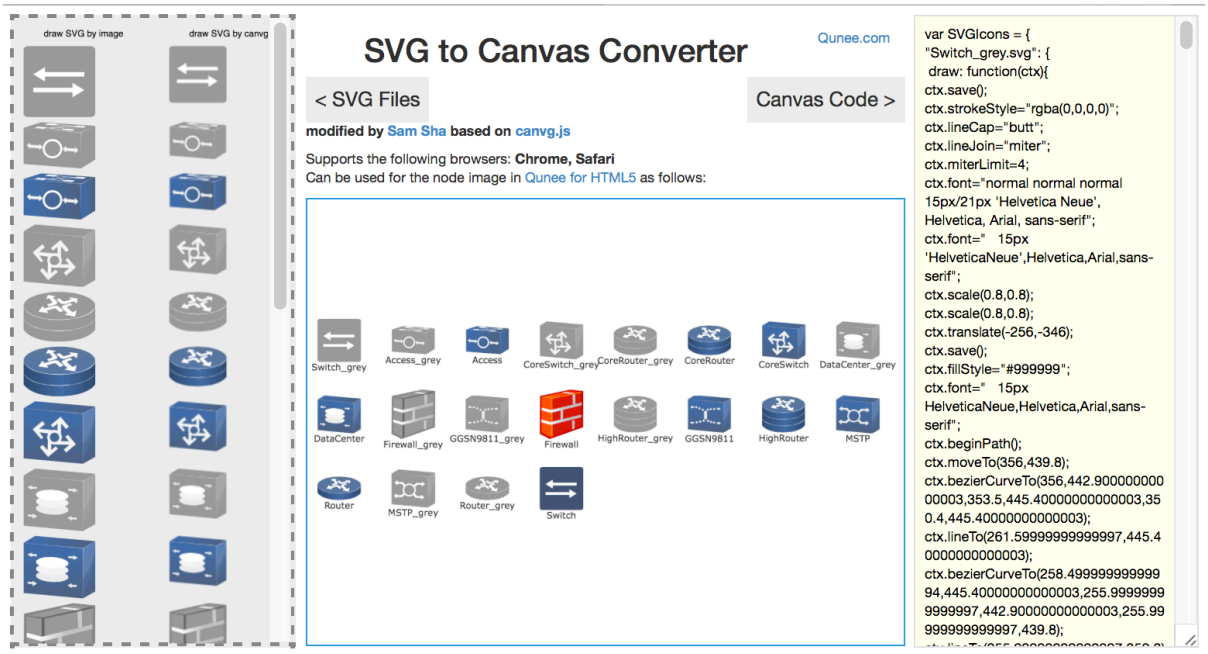
Make Canvas code icon by SVG2Canvas

1. Make SVG icon

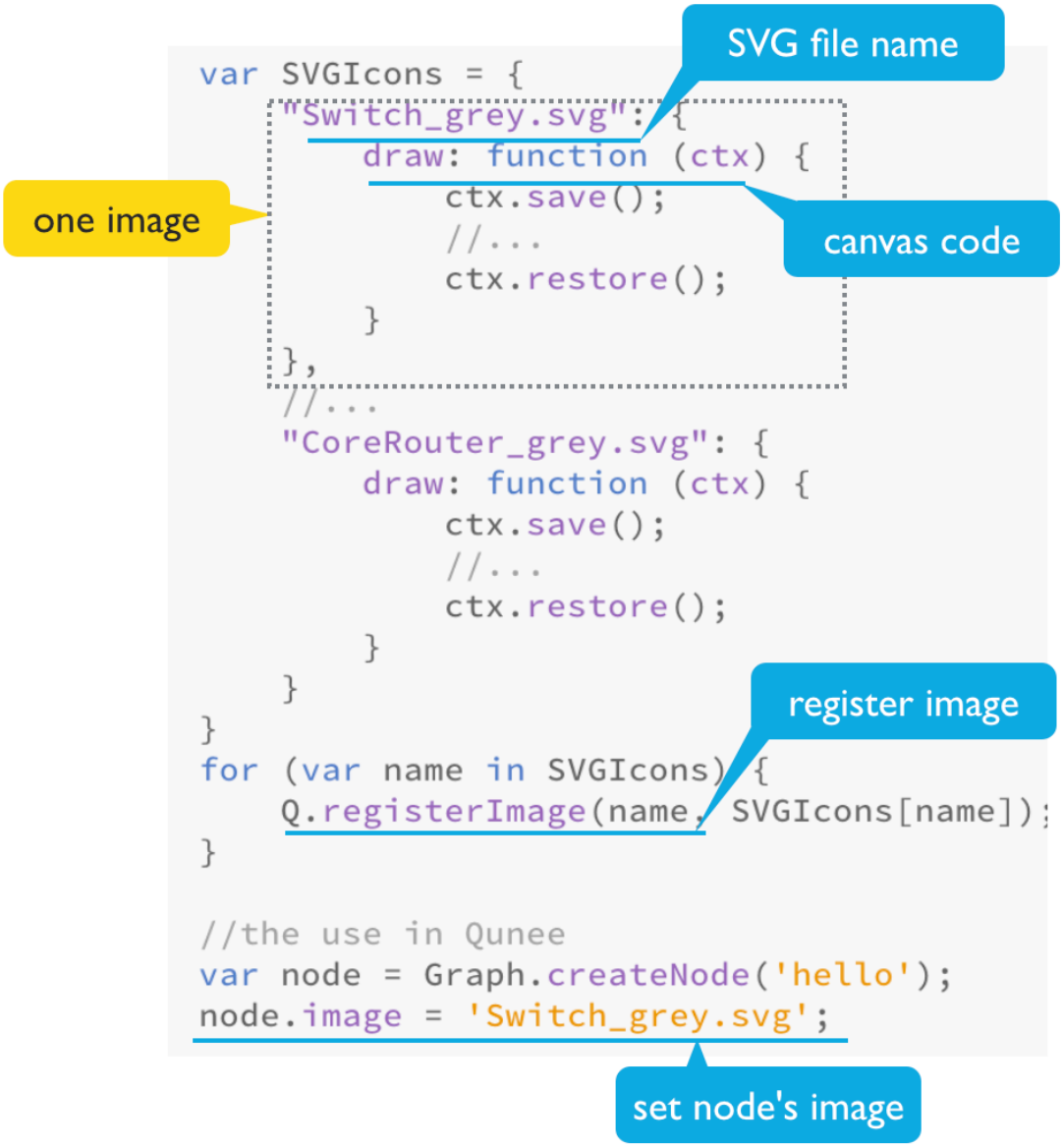
Firstly the art designer uses Adobe Illustrator (AI) or similar tools to make vector icons, and export files in SVG format

2. SVG converted to Canvas code

Use Chrome or Safari browser. Visit SVG2Canvas - http://demo.qunee.com/svg2canvas/. Drag SVG file to left dotted line frame. After a while, you may see generated Canvas code at the right text box. The central bottom shows the effect that these codes are used as node icons



Generated code structure and usage

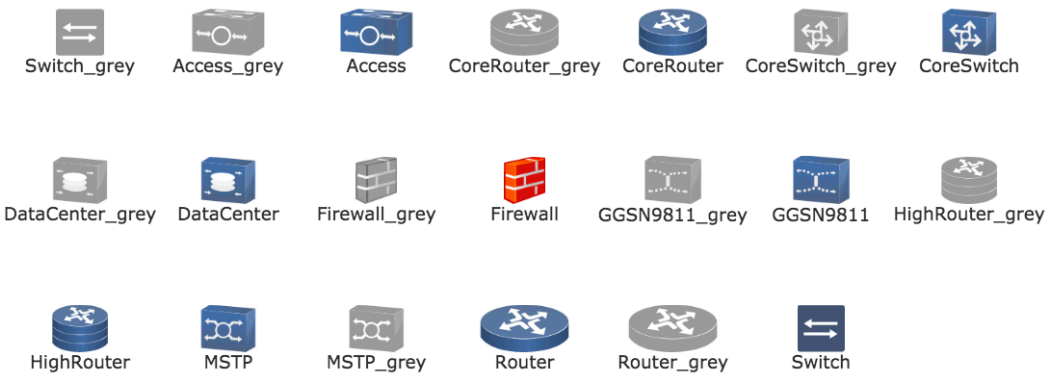


3. Usage in Qunee

Save generated code in the text box to js file, such as SVGIcons.js, and input from html file. For example:


```
<body style="margin: 0px">
<div style="position: absolute; width: 100%; top: 0px; bottom: 0px;" id="canvas"></div>
<script src="http://demo.qunee.com/lib/qunee-min.js"></script>
<script src="SVGIcons.js"></script>
<script>
    var graph = new Q.Graph('canvas');
    graph.moveToCenter();

    var x = 0, y = 0;
    for(var name in SVGIcons){
        var node = graph.createNode(name.substring(0, name.indexOf('.svg')), x, y);
        node.image = name;
//        node.setStyle(Q.Styles.RENDER_COLOR, Q.randomColor())
        x += 100;
        if(x > 600){
            x = 0;
            y += 100;
        }
    }
</script>
</body>
```

Operation effect



Use SVG Vector Image

To make icons by SVG will realize perfect vector effect, without distortion to zooming

> Pay attention that in IE explorer, you can draw SVG images in Canvas, but unable to acquire pixel information of SVG, and thus it is unable to make image dyeing, also unable to seek node edge. So Qunee recommends to use SVG file only after converting the SVG file to Canvas

Example

```
var graph = new Q.Graph("canvas");
var svg = graph.createNode("SVG Logo");
svg.image = 'SVG_logo.svg';
```

Operation effect

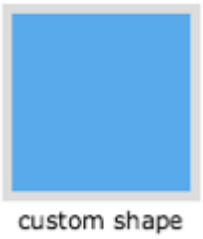
Draw custom-made icon

Support full custom-made icons. 2D function may be applied for drawing desired effect

Example, set the custom-made icon as the main part of node

```
var customDraw = {
    width: 100,
    height: 100,
    draw: function (g, scale) {
        g.beginPath();
        g.rect(0, 0, 100, 100);
        g.fillStyle = Q.toColor(0xCC2898E0);
        g.fill();
        g.lineWidth = 10;
        g.strokeStyle = '#DDD';
        g.stroke();
    }
}
var node = graph.createNode("custom shape");
node.image = customDraw;
```

Operating interface:



Shape Icon

The path may be used as the main part of node. By multi-control points, draw the shape and scope of node

Example

Custom-made path as the main part of node

```
var nodeShape = graph.createNode("Custom Shape", 240, 110);
var customShape = new Q.Path();
customShape.moveTo(20, -50);
customShape.lineTo(100, -50);
customShape.lineTo(100, 50);
customShape.quadTo(20, 50, 20, 20);
nodeShape.image = customShape;
nodeShape.setStyle(Q.Styles.SHAPE_STROKE, 4);
nodeShape.setStyle(Q.Styles.SHAPE_STROKE_STYLE, Colors.blue);
nodeShape.setStyle(Q.Styles.LAYOUT_BY_PATH, true);
nodeShape.size = {width: 100, height: -1};
```

Operating interface:

Image size

Set the image size. Support zooming as per certain ratio. For vector image, adjust the image size that will not lead to distortion of image. And for grid images, it can be clearly showed.

#size - size, used to set the size of node image. If only width is set, the height will be zoomed as per equal ratio

⚠ Select HD images for nodes. Then set smaller size. When interface zooming can be realized, the image will still be clear, such as selecting images at pixel width of 256. Set the node size to 64. When the interface is zoom in four times, the image is still clear

Example

Set the node name, vector icon, and assign width to 100, and zoom in as per equal ratio:

```
var hello = graph.createNode("Hello\nNode");
hello.image = Q.Graphs.server;
hello.size = {width: 100};
```

Presentation:

Image Registration

Set a name for the resource by registration of images in Qunee. Then set a favourable serialization and source management by this name.

Register image

Key is the source name. Data is the content of image, which can be image URL, Path object or custom-made draw function, referring to Node Image

```
Q.registerImage = function (key, data){};
```

Usage

After image registration, set image property of node by image name

Example

```
var graph = new Q.Graph("canvas");
Q.registerImage('logo', '../demos/images/logo.svg');

var logo = graph.createNode('Logo');
logo.image = 'logo';
```

Operation effect

Built-in icon

Q.Graphs is configured with some icons, which can be visited via the name of Q-XXX, such as

```
node.image = 'Q-server';
```

All built-in icons are as follows

```
var x = -300;
for(var n in Q.Graphs){
    if(Q.Graphs[n].draw){
        var node = graph.createNode('Q-' + n, x, 100);
        node.image = 'Q-' + n;
        x += 100;
    }
}
```

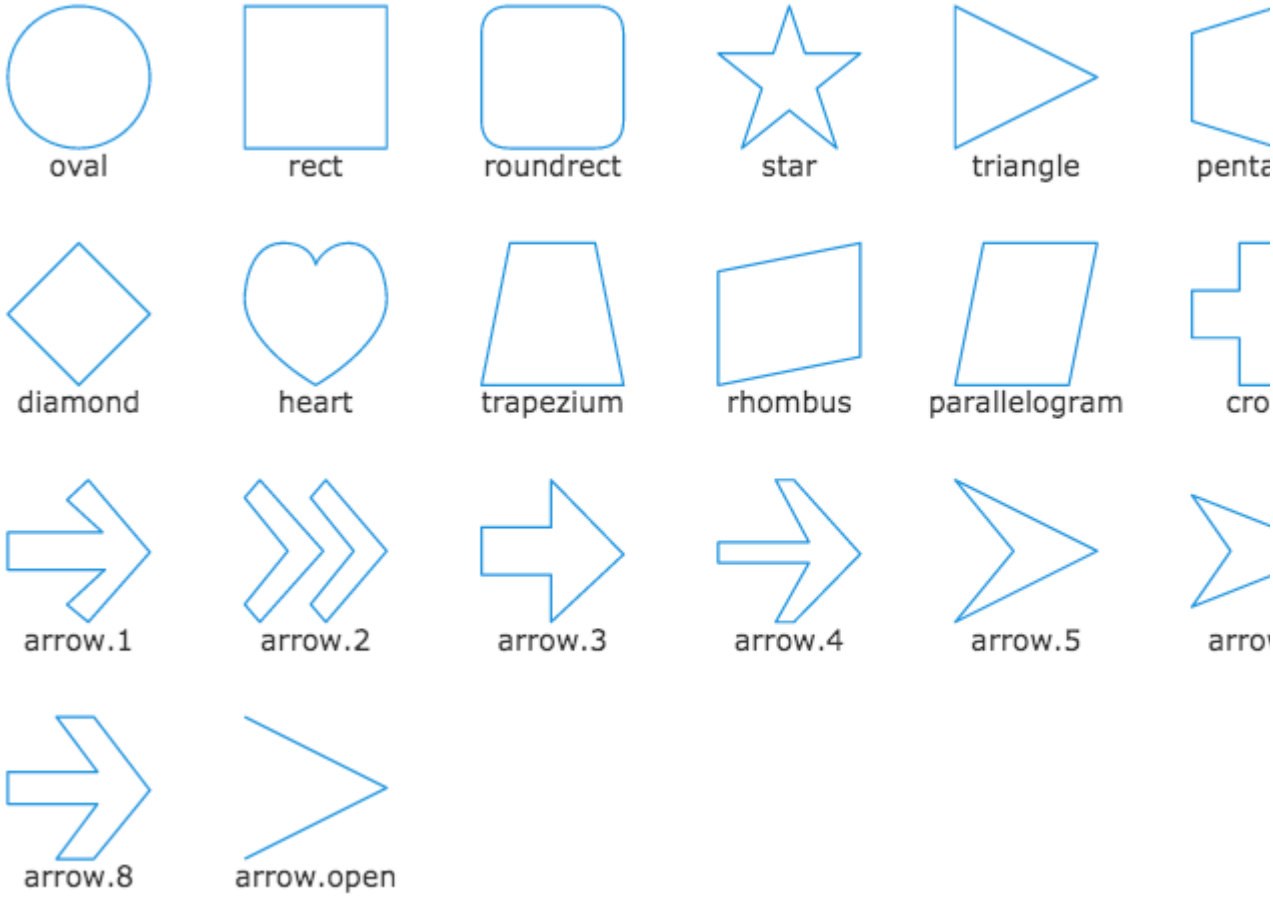
Operation effect

Built-in images

In Q.Shapes some built-in images are defined and can be used as icons

```
var x = -300;
var y = 200;
var shapes = Q.Shapes.getAllShapes(60, 60);
for(var n in shapes){
    var node = graph.createNode(n, x, y);
    node.image = shapes[n];
    x += 100;
    if(x > 300){
        x = -300;
        y += 100;
    }
}
```

Operation effect



Node Label

Default that the node is comprised of images and text labels. The text label is the name property of node. Line feed is allowed. Relevant styles of label can be set to change fond size, color and background effect of text label

Text style

Example, set the style of text label to realize gradient background and bubble effect

```
var nodeWithLabel = graph.createNode("Node with \nLabel", -120, -110);
nodeWithLabel.setStyle(Q.Styles.LABEL_OFFSET_Y, -10);
nodeWithLabel.setStyle(Q.Styles.LABEL_POSITION, Q.Position.CENTER_TOP);
nodeWithLabel.setStyle(Q.Styles.LABEL_ANCHOR_POSITION, Q.Position.CENTER_BOTTOM);
nodeWithLabel.setStyle(Q.Styles.LABEL_BORDER, 1);
nodeWithLabel.setStyle(Q.Styles.LABEL_POINTER, true);
nodeWithLabel.setStyle(Q.Styles.LABEL_PADDING, new Q.Insets(2, 5));
nodeWithLabel.setStyle(Q.Styles.LABEL_BACKGROUND_GRADIENT, Q.Gradient.LINEAR_GRADIENT_VERTICAL);
```

Operation effect:

Several text label

Realize several text label by adding child components. Each text label can be placed at different positions, and set with different styles and information

Example

Adds one descriptive text label for the node, and place it after the node


```
var graph = new Q.Graph('canvas');
function createTextWithDescription(text, description, x, y){
    var text = graph.createText(text, x, y);
    text.setStyle(Q.Styles.LABEL_BACKGROUND_COLOR, "#2898E0");
    text.setStyle(Q.Styles.LABEL_BACKGROUND_GRADIENT, new Q.Gradient(Q.Consts.GRADIENT_TYPE_LINEAR,
['#00d4f9', '#1ea6e6'], null, Math.PI/2));
    text.setStyle(Q.Styles.LABEL_COLOR, "#FFF");
    text.setStyle(Q.Styles.LABEL_BORDER, 0.5);
    text.setStyle(Q.Styles.LABEL_PADDING, 4);
    text.setStyle(Q.Styles.LABEL_BORDER_STYLE, "#1D4876");
    text.setStyle(Q.Styles.LABEL_RADIUS, 0);
    text.setStyle(Q.Styles.LABEL_SIZE, new Q.Size(60, 25));
    text.setStyle(Q.Styles.SELECTION_COLOR, "#0F0");
    var label1 = new Q.LabelUI();
    label1.border = 0.5;
    label1.borderColor = "#DDD";
    label1.borderRadius = 0;
    label1.size = new Q.Size(150, 25);
    label1.padding = 4;
    label1.alignPosition = Q.Position.LEFT_MIDDLE;
    label1.position = Q.Position.RIGHT_MIDDLE;
    label1.anchorPosition = Q.Position.LEFT_MIDDLE;
    text.addUI(label1, [{
        bindingProperty : "data",
        property : "description",
        propertyType : Q.Consts.PROPERTY_TYPE_CLIENT
    }]);
    text.set("description", description);
    return text;
}
var text = createTextWithDescription("TEXT", "描述");
var a = graph.createNode("HELLO", -100, -50);
graph.createEdge(a, text);
```

Operation effect

Node Style Property

Node supports the background color, gradient, border, and font color and the like style properties. It may be set via Node#setStyle().

For more information, please refer to style list

Example

Set the color and gap of node border

```
var style = graph.createNode("Style Node", 100, 0);
style.setStyle(Q.Styles.BORDER, 1);
style.setStyle(Q.Styles.BORDER_COLOR, '#AABBEE');
style.setStyle(Q.Styles.PADDING, 5);
```

Operation effect:

Edge Element

Edge element (Q.Edge) is the graph element connecting two nodes, showing the topological relation among points. It represents transmission pipe, routines or relation of things in actual application.

Create edge

Generally the following codes would be used for creating edges

```
var edge = graph.createEdge(from, to, “edge name”);
```

The edge constructor may also be adopted

```
var edge = new Q.Edge(from, to);
edge.name = “edge name”;
graph.graphModel.add(edge);
```

- Basic Property of Edge
- Topological Eelation of Edge
- Edge Appearance
- Bus Effect

Basic Property of Edge

Each edge shall contain two end points, start node and end node respectively. If any one node is null, this edge will be invalid. If two nodes are the same node, this edge is looped

- #from - start node
- #to - end node
- #otherNode() - another node
- #isInvalid() - whether it is invalid
- #isLooped() - whether it is looped

Looped edge

For example, creates a looped edge

```
var a = graph.createNode("A", -100, 0);
var edge = graph.createEdge(a, a, 'looped');
```

Looped effect of edge

Topological Eelation of Edge

When the edge is connected to the image, the topological structure of image will change. Now the system will call the method of Edge#connect() automatically. When the edge is removed from the image, the topological relation will be disconnected automatically.

- #connect() - connect to the image
- #disconnect() - disconnect topological relation
- #isConnected() - whether it is connected
- #getEdgeBundle() - acquire bundle edge collection

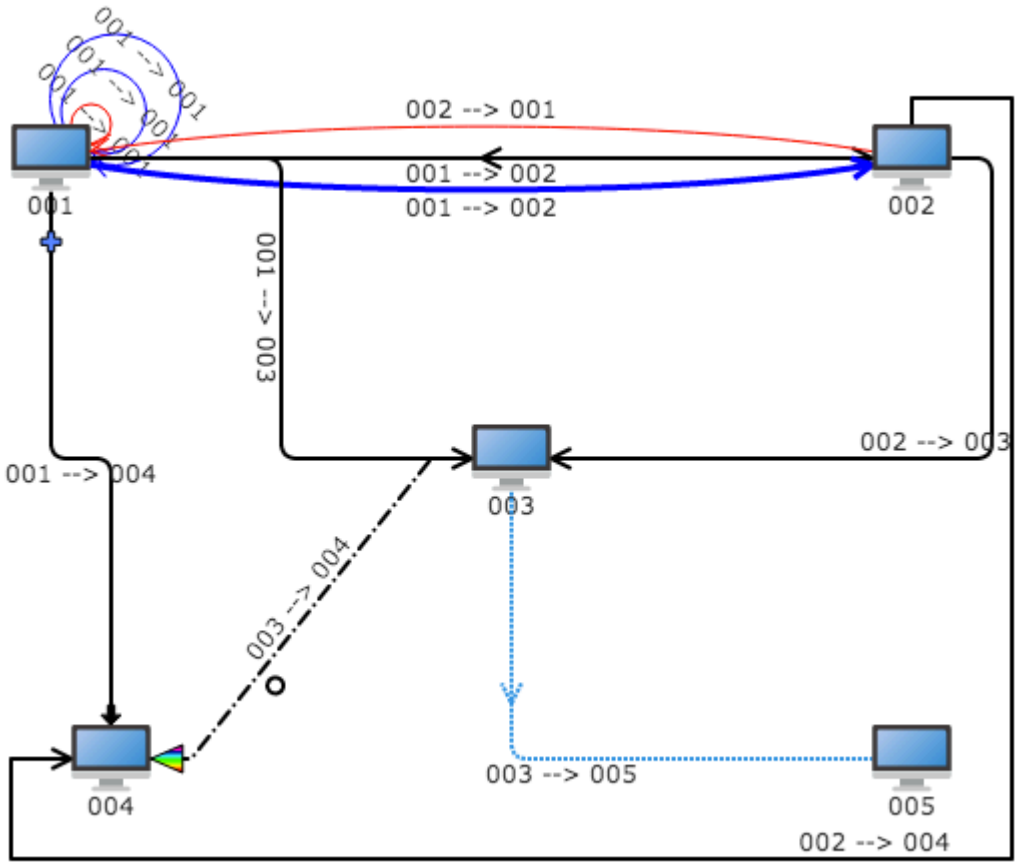
Example

```
var a = graph.createNode("A", -100, 0);
var b = graph.createNode("B", 100, 0);
graph.createEdge(a, b);
var edge = graph.createEdge(a, b);
alert(edge.getEdgeBundle().edges.length);
```

Operation result is: 2

Edge Appearance

Thickness, color and line style of edge can be set. Supports arrow. Various trend, and child components on edge (including text labe) can be distributed along the line. More edges can be displayed at the same time between two nodes. Supports looped effect of edge.



Example for setting edge appearance

Sets width, color and type of edge

```
var a = graph.createNode("A", -100, 50);
var b = graph.createNode("B", 100, -50);
var edge = graph.createEdge(a, b);
edge.setStyle(Q.Styles.EDGE_COLOR, '#88AAEE');
edge.setStyle(Q.Styles.EDGE_WIDTH, 2);
edge.edgeType = Q.Consts.EDGE_TYPE_VERTICAL_HORIZONTAL;
```

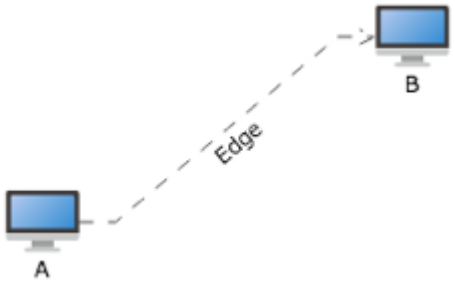
Operating interface:

Example of dotted line style of edge

Set dotted line style of edge

```
var a = graph.createNode("A", -100, 50);
var b = graph.createNode("B", 100, -50);
var edge1 = graph.createEdge(a, b, 'Edge');
edge1.edgeType = Q.Consts.EDGE_TYPE_ELBOW;
edge1.setStyle(Q.Styles.EDGE_LINE_DASH, [8, 4, 0.01, 4]);
edge1.setStyle(Q.Styles.LINE_CAP, "round");
```

Operating interface:



Multi text label of edge

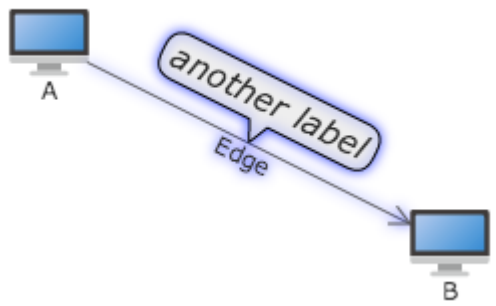
The edge can be set with several text label to show different information

Example

Adds one text label on the edge


```
var graph = new Q.Graph('canvas');
var A = graph.createNode("A", -100, -100);
var B = graph.createNode("B", 100, 0);
var edge = graph.createEdge("Edge", A, B);
var label2 = new Q.LabelUI();
label2.position = Q.Position.CENTER_TOP;
label2.anchorPosition = Q.Position.CENTER_BOTTOM;
label2.border = 1;
label2.padding = new Q.Insets(2, 5);
label2.showPointer = true;
label2.offsetY = -10;
label2.backgroundColor = "#EEE";
label2.fontSize = 16;
label2.fontStyle = "italic 100";
edge.addUI(label2, [{
    property : "label2",
    propertyType : Q.Consts.PROPERTY_TYPE_CLIENT,
    bindingProperty : "data"
}, {
    property : "label2.color",
    propertyType : Q.Consts.PROPERTY_TYPE_CLIENT,
    bindingProperty : "color"
}]);
edge.set("label2", "another label");
```

Operation effect



Bus Effect

Bus layout is a typical structure in the topological graph. Bus is a public messaging trunk for transmitting information among each functional part of computer. The bus and devices connected to it form a topological structure, which is called bus topological graph. In Qunee, the Bus element type is defined. But the edge has to be set for getting the bus effect.

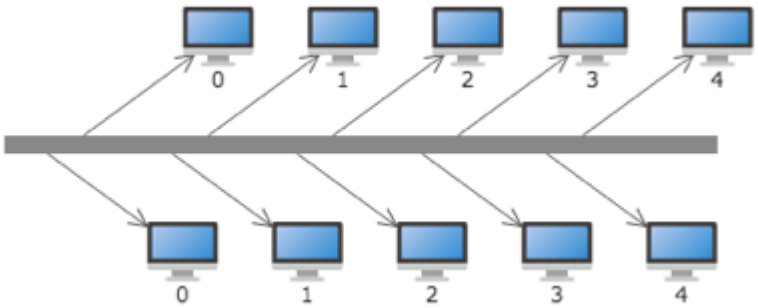
The edge angle property is defined in Qunee. In comination with bus element type, edge effect in certain direction can be realized

Edge#angle - strike angle of edge

Bus example

```
var graph = new Q.Graph("canvas");
var bus = new Q.Bus();
graph.addElement(bus);
bus.moveTo(-200, 0);
bus.lineTo(200, 0);
bus.setStyle(Q.Styles.SHAPE_STROKE, 10);
for(var i = 0; i < 5; i++){
    var node = graph.createNode("" + i, -100 + i * 70, 60);
    var edge = graph.createEdge(bus, node);
    edge.angle = Math.PI * 0.2;
    node = graph.createNode("" + i, -80 + i * 70, -60);
    edge = graph.createEdge(bus, node);
    edge.angle = -Math.PI * 0.2;
}
```

Operation effect



Group Element

Group graph element (Q.Group) contains many nodes which form a combination and can indicate equipment class, fragmentation and inclusion relation

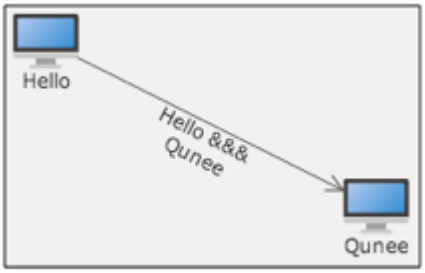
Creates a group

Creates a group via graph.createGroup(name, x, y). The parameters are separately group name and position

Example

```
var graph = new Q.Graph("canvas");
var hello = graph.createNode("Hello", -100, -50);
var qunee = graph.createNode("Qunee", 100, 50);
var edge = graph.createEdge("Hello &&&\nQunee", hello, qunee);

var group = graph.createGroup();
group.addChild(hello);
group.addChild(qunee);
```

Operation effect



- Group Background Image
- Group Style

Group Background Image

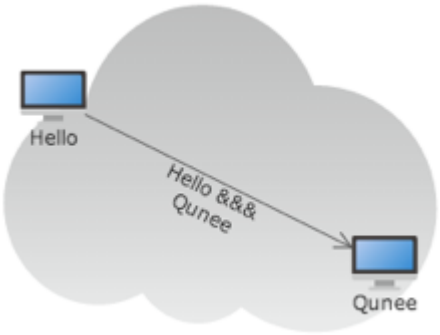
The group background image may also be set

Q.Group#groupImage - Background of group image

Example

```
group.padding = new Q.Insets(40, 10, 10, 10);
group.groupImage = Q.Graphs.cloud;
```

Operation effect



Group Style

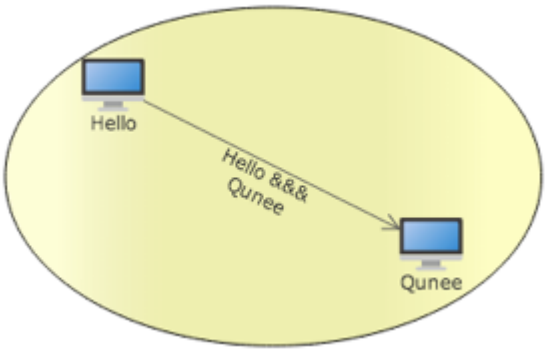
The group supports many shapes, internal gap, filling color and border style etc. In addition, the image can be set as the group background

- Q.Group#groupType - Group shape type
- Q.Group#padding - Internal gap
- Q.Styles.GROUP_*** - Related styles of group

Example

```
group.groupType = Q.Consts.GROUP_TYPE_ELLIPSE;
group.setStyle(Q.Styles.GROUP_BACKGROUND_COLOR, Q.toColor(0xCCfcfb9b));
group.setStyle(Q.Styles.GROUP_BACKGROUND_GRADIENT, Q.Gradient.LINEAR_GRADIENT_HORIZONTAL);
```

Operation effect



Subnet Type

Topological graph can describe the distribution of network structure. One image indicates a network. The network generally has hierarchical relationship. WAN has metropolitan area network. The metropolitan area network has local area network. This kind of affiliate network is the sub network we call (sub network)

In Qunee any element can be regarded as the subnet, such as word, node, edge and even the group. Just hast to set enableSubNetwork property as true. It can be expressed as subnet type.

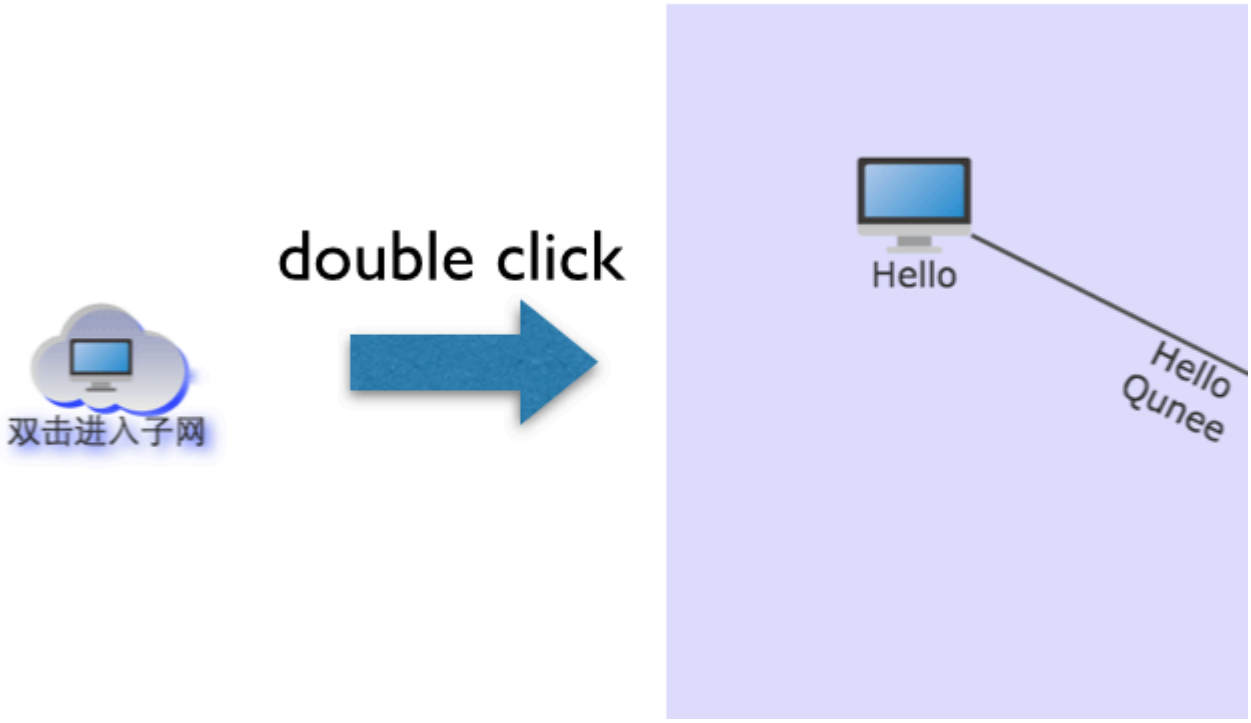
Related API of subnet

- Set as the subnet type - any element can be regarded as subnet
  element.enableSubNetwork = true;
- enter this subnet
  graph.currentSubNetwork = element;
- returns the upper layer of subnet
  graph.upSubNetwork();
- monitors the change event of subnet
  graph.onPropertyChange('currentSubNetwork', function(evt){ ... })

Example

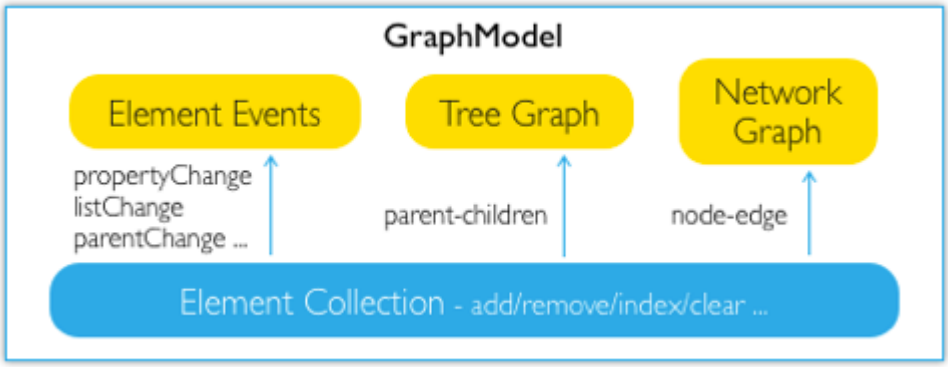
```
var graph;
$(function(){
    graph = new Q.Graph('canvas');

    var subnetwork = graph.createNode('双击进入子网');
    subnetwork.image = Q.Graphs.subnetwork;
    subnetwork.enableSubNetwork = true;

    var hello = graph.createNode("Hello", -100, -50);
    var qunee = graph.createNode("Qunee", 100, 50);
    var edge = graph.createEdge("Hello\nQunee", hello, qunee);

    hello.parent = subnetwork;
    qunee.parent = subnetwork;
    edge.parent = subnetwork;

    graph.onPropertyChange('currentSubNetwork', function(){
        graph.html.style.backgroundColor = graph.currentSubNetwork == subnetwork ? '#DDF' : '#FFF';
    });
})
```

Operation effect



Graph Model

Internal system, the element is saved at the graph model (Q.GraphModel). This model is used for management of element collection, for unified monitoring of element events, and providing support to traversal of tree graph and network graph



Build graph model

The type of graph model is Q.GraphModel, which is automatically created in Graph component

```
var graph = new Q.Graph(canvas);
var model = graph.graphModel;
model.add(new Q.Node());
```

It can also be created independently. And then it will be transferred to the graph component by formation parameter

```
var model = new Q.GraphModel();
model.add(new Q.Node());
var graph = new Q.Graph(canvas, model);
```

- Element Management
- Event Handling
- Selects Management Model
- Traverse by Tree Graph
- Traverse by Graph

Element Management

Element management is the basic function of graph model. The graph model has functions of Array and Map. The internal manages the elements via array and ID mapping table. Adding and removal of elements, sequence modification and acquire traversal operation

Adds or remove

- #add(data, index) - Adds elements
- #clear() - Clear elements
- #remove(data)
- #removeById(id)
- #set(data)

Other collection operation function

- #contains(data) - whether containing elements
- #containsById(id) - whether containing elements of appointed id
- #get(index) - acquire elements by array position
- #getById(id) - acquire elements by ID

#indexOf(data) - acquire array position of elements

#setIndex(data, index) - set the array position of elements

#size() - collection size

#length - collection size

#forEach(call, scope, params) - traverse according to array collection

#forEachReverse(call, scope, params) - traverse reversely by array collection

Example

Adds elements in graph model, and traverse the model reversely

```
var model = new Q.GraphModel();
model.add(new Q.Node('A'));
model.add(new Q.Node('B'));
model.add(new Q.Node('C'));
Q.log('model size: ' + model.length);
model.forEachReverse(function(node){
    Q.log(node.name);
});
Q.log('the first node is: ' + model.get(0).name);
```

Print results


```
model size: 3
C
B
A
```


Event Handling

The model will have unified handling on element property change events. By this way, users can response to all property change events just by monitoring graph model, instead of adding monitoring to every element alone

Element change events

When element property changes, the graph model will transmit corresponding events via #dataChangeDispatcher. When the set membership of elements change, the monitoring can be performed by #parentChangeDispatcher

- #dataChangeDispatcher :Q.Dispatcher - Dispatcher of element property change event
- #beforeDataPropertyChange(event) - before element property change event
- #onDataPropertyChanged(event) - after element property change event
- #parentChangeDispatcher :Q.Dispatcher - Dispatcher of change event about set membership of elements

Example 1

Adds the element property change listener

```
var model = graph.graphModel;
var a = new Q.Node();
model.add(a);
model.dataChangeDispatcher.addListener({
    onEvent: function(evt){
        Q.log(evt.kind + ' changed. from [' + evt.oldValue + '] to [' + evt.value + ']');
    }
});
a.name = 'A';
a.name = 'B';
```

Print results:

```
name changed. from [undefined] to [A]
name changed. from [A] to [B]
```

Example 2

Monitors the change of parent node. Pay attention that another writing method for listener is used here. The monitoring function is introduced directly

```
var model = graph.graphModel;

var a = new Q.Node('A');
model.add(a);
var b = new Q.Node('B');
model.add(b);

model.parentChangeDispatcher.addListener(function(evt){
    Q.log(evt.source.name + '\'s ' + evt.kind + ' changed. from [' + evt.oldValue + '] to [' + evt.value.name + ']');
});

a.parent = b;
```

Print results:

```
A's parent changed. from [undefined] to [B]
```

Change event of graph model

When any change happens to element collection (such as adding, deleting……), the system will dispatch the collection change event, and monitor via #listChangeDispatcher

#listChangeDispatcher :Q.Dispatcher - Dispatcher of collection change event

Example

Monitors change event of graph model collection

```
var model = graph.graphModel;

model.listChangeDispatcher.addListener(function(evt){
    Q.log(evt.kind);
});

var a = new Q.Node('A');
model.add(a);
var b = new Q.Node('B');
model.add(b);
model.remove(a);
model.clear();
```

Print results:

```
add
add
remove
clear
```

Selects Management Model

The graph model also provides the element selection management model, by which the elements will be selected or the selection may be cancelled.

#selectionModel :Q.SelectionModel - element selection model

- Selection element: model.selectionModel.select(...);
- Cancel element selection: model.selectionModel.unselect(...);
- Clear element selection status: model.selectionModel.clear();

Selection change events

Provides corresponding dispatcher for selection change event, applicable to selection change event of monitoring elements

#selectionChangeDispatcher :Q.Dispatcher - Dispatcher of element selection change event

Example

Monitor selection change events

```
var model = graph.graphModel;
var a = new Q.Node('A');
model.add(a);
var b = new Q.Node('B');
model.add(b);


model.selectionChangeDispatcher.addListener(function(evt){
    Q.log(evt.kind);
});
var selectionModel = model.selectionModel;
selectionModel.select(a);
selectionModel.unselect(a);
selectionModel.select(b);
selectionModel.clear();
```

Print results

```
add
remove
add
clear
```

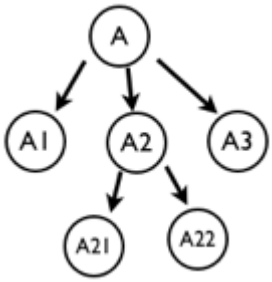
Traverse by Tree Graph

lements in graph model form a tree according to set membership. Traverses by tree map. Two methods are provided, separately depth first and breath first. The depth first traversal is divided into the preorder traversal and the postorder traversal. In addition, the reverse transversal is supported

- #forEachByBreadthFirst(call, scope) - breadth first traversal
- #forEachByBreadthFirstReverse(call, scope) - breadth first reverse traversal
- #forEachByDepthFirst(call, scope, postOrder) - The depth first traversal is divided into the preorder traversal and the postorder traversal
- #forEachByDepthFirstReverse(call, scope, postOrder) - The depth first reverse traversal is divided into the preorder traversal and the postorder traversal

Set membership

The set membership is showed in the following graph. The arrow indicates the set relation. Different call sequences will be realized by different traversal method



Example


```
var model = new Q.GraphModel();
var a = new Q.Node('A');
model.add(a);
var a1 = new Q.Node('A1');
model.add(a1);
var a2 = new Q.Node('A2');
model.add(a2);
var a3 = new Q.Node('A3');
model.add(a3);
var a21 = new Q.Node('A21');
model.add(a21);
var a22 = new Q.Node('A22');
model.add(a22);
a1.parent = a;
a2.parent = a;
a21.parent = a2;
a22.parent = a2;

Q.log('forEachByBreadthFirst');
model.forEachByBreadthFirst(function(node){
    Q.log(node.name);
}, null, true);

Q.log('forEachByDepthFirst by post-order');
model.forEachByDepthFirst(function(node){
    Q.log(node.name);
}, null, true);

Q.log('forEachByDepthFirst by pre-order');
model.forEachByDepthFirst(function(node){
    Q.log(node.name);
});
```

The traversal results are showed as follows:

forEachByBreadthFirst

```
A
    A3
    A1
    A2
    A21
    A22
```

forEachByDepthFirst by post-order

```
A1
    A21
```


A22

A2

A

A3

forEachByDepthFirst by pre-order

A

A1

A2

A21

A22

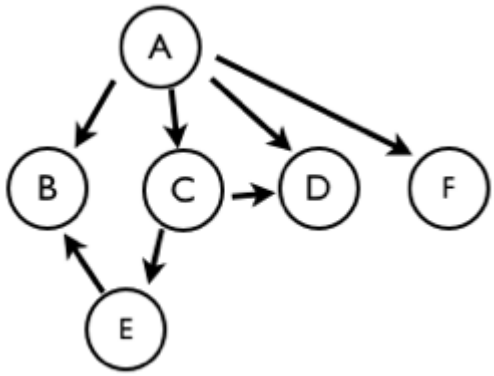
A3

Traverse by Graph

Elements in graph model will form another graph structure according to connection relation, also called graph. The node means the middle point in the graph. The edge represents the edge in the graph. The traversal that Qunee supports the nodes in the graph. It may also be applied to automatic layout method

- #forEachByTopoDepthFirstSearch(call, scope, postOrder) - Depth first traversal of graph, supports preorder and postorder traversal which is defaulted to preorder traversal
- #forEachByTopoBreadthFirstSearch(call, scope, postOrder) -Breadth first traversal of graph, supports preorder and postorder traversal which is defaulted to preorder traversal

The topological structure is showed in the following graph. The arrow indicates the edge direction. Different call sequences will be realized by different traversal method



Example


```
var model = graph.graphModel;
var a = model.add(new Q.Node('A'));
var b = model.add(new Q.Node('B'));
var c = model.add(new Q.Node('C'));
var d = model.add(new Q.Node('D'));
var e = model.add(new Q.Node('E'));
var f = model.add(new Q.Node('F'));
model.add(new Q.Edge(a, b));
model.add(new Q.Edge(a, c));
model.add(new Q.Edge(a, d));
model.add(new Q.Edge(c, d));
model.add(new Q.Edge(c, e));
model.add(new Q.Edge(e, b));
model.add(new Q.Edge(a, f));

Q.log('forEachByTopoDepthFirstSearch by pre-order');
model.forEachByTopoDepthFirstSearch(function(node){
    Q.log(node.name);
});
Q.log('forEachByTopoDepthFirstSearch by post-order');
model.forEachByTopoDepthFirstSearch(function(node){
    Q.log(node.name);
}, null, true);
Q.log('forEachByTopoBreadthFirstSearch by pre-order');
model.forEachByTopoBreadthFirstSearch(function(node){
    Q.log(node.name);
});
Q.log('forEachByTopoBreadthFirstSearch by post-order');
model.forEachByTopoBreadthFirstSearch(function(node){
    Q.log(node.name);
}, null, true);
```

Print results:

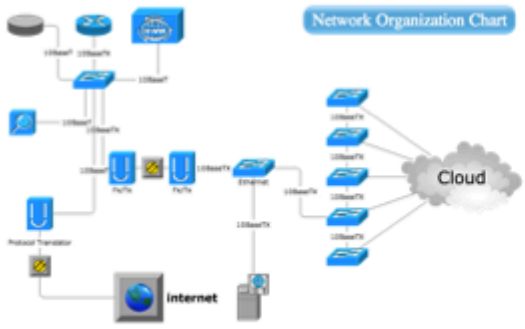
```
forEachByTopoDepthFirstSearch by pre-order
A
B
C
D
E
F
forEachByTopoDepthFirstSearch by post-order
B
D
E
```

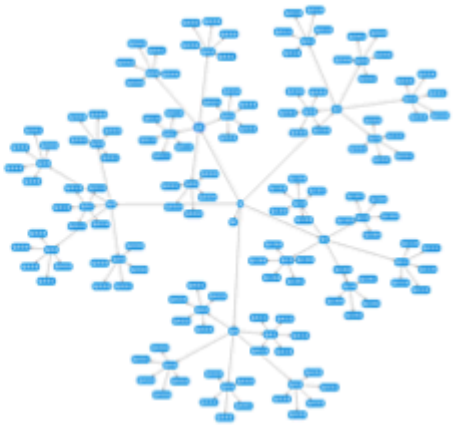


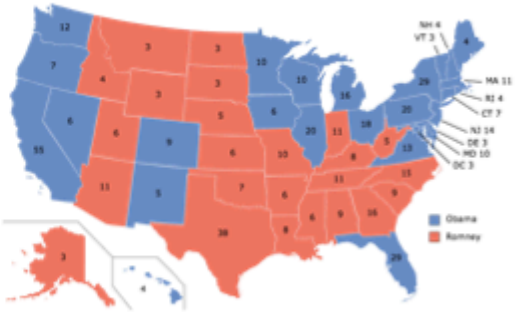
```
C
F
A
forEachByTopoBreadthFirstSearch by pre-order
A
B
C
D
F
E
forEachByTopoBreadthFirstSearch by post-order
B
E
C
D
F
A
```


Graph Component

The graph element is presented via graph component (Q.Graph). In Qunee, it is showed to be Q.Graph, hereinafter referred to Graph component. The graph elements in the graph component have various styles and expansion ways, support various mouse and keyboard interaction and also automatic layout and dynamic layout







- Graph Basic Functions
- Interaction
- Common Properties and Methods
- Selectable Filtering, Movable Filtering
- Other properties and methods
- Element Default Style Sheet
- Navigation Panel Type
- Delayed Rendering

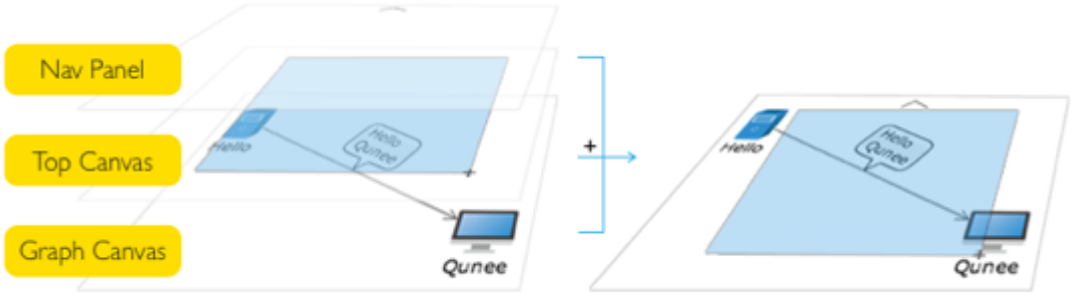
Graph Basic Functions

Graph components are used for imaging expression data, with the structure of many layers of canvas, taking the central point as the coordinate origin. Adopts roaming view mode. Supports various single click, double click, drag and drop, long press and touch and the like interaction modes. Thousands of photos can be showed easily. The export of large images is supported

- Graph Layer Structure
- Graph Coordinate System
- Element Operation of Graph
- Translation and Zooming

Graph Layer Structure

Generally the component is put in a DIV element, which contains many nodes and has the structure comprised of several layers. Each layer shows specific contents. The first layer is the canvas layer (Graph Canvas), used to draw graph element (node and edge and so on); the second layer is the interaction canvas (Top Canvas), used for realizing interface effect during the process of drawing and interaction, such as the semi-transparent rectangular frame during frame selection and interaction. The third layer is the navigation panel (Nav Panel), which contains four buttons, top, bottom, left and right. When the screen range exceeds the viewport, buttons will be displayed automatically, so as to represent the canvas scope. Click the button to translate the canvas.



DOM structure of Graph

 - parent container element, graph.html

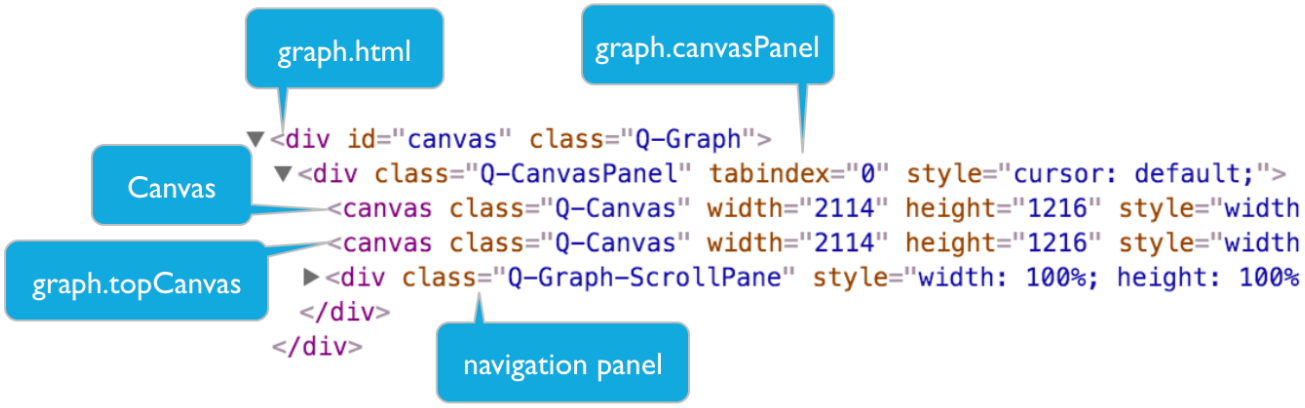
 - Local component elements

<canvas /> - Canvas layer

<canvas /> - Interaction layer

 - Navigation panel

The outer surface is corresponding HTML elements of the entire Graph component, where the first child node is the main canvas element, used to draw main parts of image(node, edge, shape and group etc.); the another one is the top layer canvas element, used for auxiliary drawing, such as interaction hint and frame selection rectangle etc.



Graph Coordinate System

The canvas takes the central point of component as the original coordinate. This means the node of coordinate (0,0) will be presented at the centre of component. In case of no limitation on coordinate at the top left corner. Supports negative coordinate and vector zooming. In addition, the canvas range adopts the way of navigation panel (indicating the element scope by four buttons, top, bottom, left and right). Avoids the interference to the interface from rolling bar, and makes the roaming interaction more smooth.

Coordinate origin

The initial default canvas coordinate origin at the center of the component, which means that the node (0, 0) coordinates will be presented in the central module, if you want the initial component origin is located in the upper left corner, you can set the following parameters

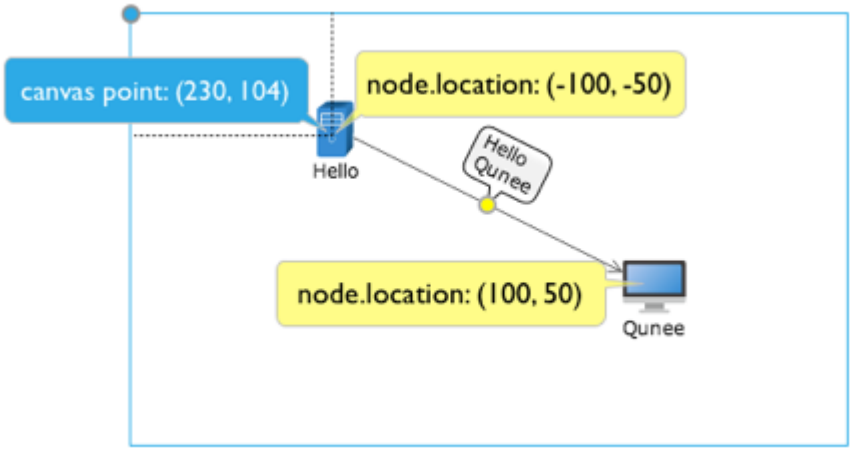
#originAtCenter

Set the upper left corner of the origin of coordinates

```
graph.originAtCenter = false;
```

Coordinate transformation

It includes canvas coordinate and logical coordinate. The previous one take the top left corner of the component interface as the original point. And the later one is the actual coordinate property of element



- #globalToLocal(evt) → {Point} - calculates the relative position between the mouse point and the component according to the mouse event object.
- #toCanvas(x, y) → {Q.Point} - logical position is converted to a canvas coordinate
- #toLogical(x, y) → {Q.Point} - Canvas coordinate is converted to logical position

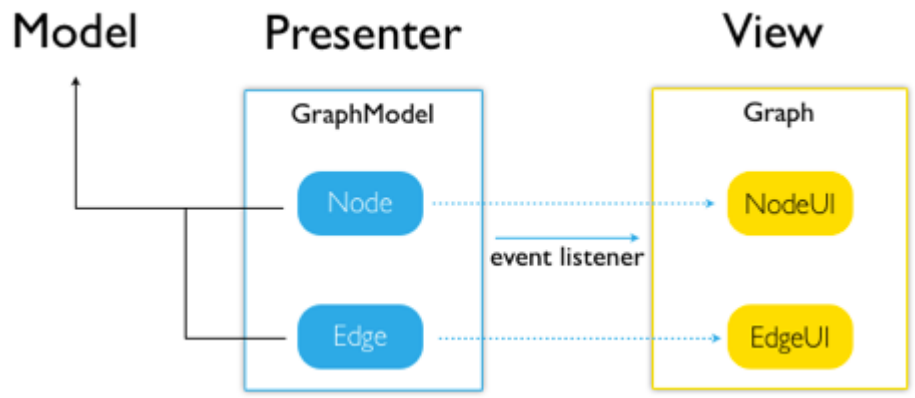
Example

Monitors click event. Acquires the information about mouse point position and converts to logical coordinate


```
graph.onclick = function(evt){
    var p = graph.globalToLocal(evt);
    var l = graph.toLogical(p.x, p.y);
    Q.log('canvas location: ' + p.x + ', ' + p.y);
    Q.log('logical location: ' + l.x + ', ' + l.y);
}
```


Element Operation of Graph

The graph component contains a graph management model. By the way the model manages the elements, each element system will generate a corresponding UI object automatically, for realizing the appearance drawing of element



Create elements

- #createEdge(name, from, to) → {Edge} - create edge
- #createNode(name, x, y) → {Node} - create node

Translation and Zooming

Graph component uses roaming translation interaction, supporting drag and drop and translation, rolling zooming. Touch translation and double finger zooming and the like functions are supported in mobile devices. Besides, inertial animation effect is also supported. Relevant port functions are showed hereinafter:

- #translate(tx, ty, byAnimate) - translation
- #translateTo(tx, ty, scale, byAnimate) - sets offset volume and zooming ratio
- #zoomAt(scale, px, py, byAnimate) - zooms by appointed point
- #zoomIn(px, py, byAnimate) - zooms in
- #zoomOut(px, py, byAnimate) - zooms out
- #zoomToOverview(byAnimate, maxScale) - zoom to the entire window
- #centerTo(cx, cy, scale, byAnimate) - move the appointed points to the component center
- #moveToCenter(scale, byAnimate) - move the entire canvas to the component center

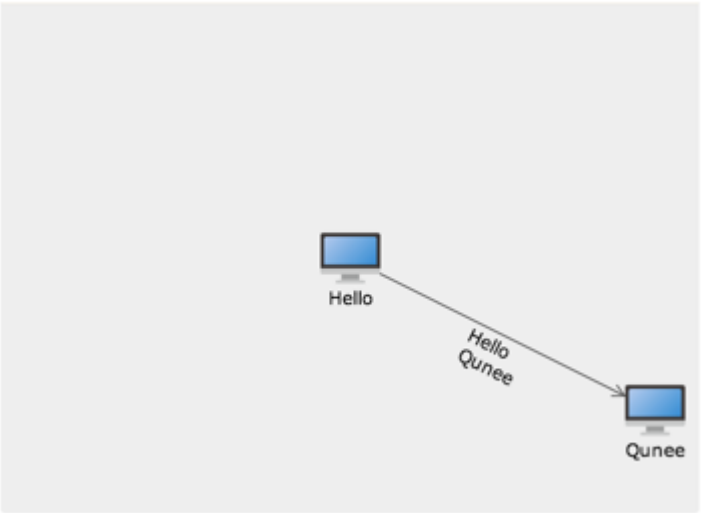
Example

Translate the canvas center to (-100, -50)

```
var graph = new Q.Graph(canvas);

var hello = graph.createNode("Hello", -100, -50);
var qunee = graph.createNode("Qunee", 100, 50);
var edge = graph.createEdge("Hello\nQunee", hello, qunee);

graph.callLater(function(){graph.centerTo(-100, -50, 1.5);});
```

Operation effect

The canvas will move to the right bottom corner. The node 'Hello' moves to the center of component



Interaction

Graph component supports the mouse, touch and keyboard event monitoring. The system enclosures the original mouse and touch events. Extends commonly used interaction events based on basic interaction events, such as distinguishing single click and double click, support drag and drop, long press by mouse, roller of mouse, and multi-point touch...all these functions can also be provided to users via the form of interaction mode

- Interaction Event Type
- Adds Interaction Listener
- Graph Interaction Mode

Interaction Event Type

Qunee makes some integration to Mouse, Key and TouchEvent. In order to facilitate the operation, it retains some original events and expands some other events.

KeyEvent

- onkeydown – keydown event

MouseEvent or TouchEvent

There are some integration made to the original MouseEvent or TouchEvent, some of which are unique to desktop browsers, like #onmousewheel; while some others of which are unique to mobile devices, like #onpinch.

- onclick – click event
- ondblclick – dblclick event
- onstart – mousedown event or touchdown event
- onrelease – mouseup event or touchup event
- startdrag – dragstart event
- ondrag – dragmove event
- enddrag – dragend event
- onlongpress – lonpress event, mousedown or touchdown for a certain time
- onmousewheel – mouse wheel, unique to desktop browsers
- startpinch – pinchstart event, unique to mobile devices
- onpinch – pinchmove event, unique to mobile devices
- endpinch – pinchend event, unique to mobile devices
- onmousemove – mousemove event

New Types of Events Added by V2.5

- onmousedown – mousedown event
- onmouseup – mouseup event
- onstart2 - rbuttondown
- onrelease2 – rbuttonup
- startdrag2 – rbutton dragstart event
- ondrag2 - rbutton dragmove event
- enddrag2 - rbutton dragend event
- onevent – all events
- accept – used for judging the response event

Interaction Events Expansion

The interaction events listed above are not all original MouseEvent or KeyEvent, but after encapsulation or improvement. For example, original mouseclick event makes no distinction between click and dblclick and may be triggered during the mousemove event. Graph components avoid all these problems.

Click and dblclick event - click, dblclick

Dblclick event will not trigger click event. System will cancel the event if mouseclick slide exceeds a certain distance, which is also applicable to mobile devices.

Longpree event - longpress

Mouseclick or touchdown lasts a certain time will both trigger the longpress event.

Example:

```
graph.onclick = function(evt){
    Q.log(‘click’);
}
graph.ondblclick = function(evt){
    Q.log(‘double click’);
}
graph.onlongpress = function(evt){
    Q.log(‘long press’);
}
```

Onstart and onrelease events

Qunee expands a set of start and release events to denote the start and end of lbutton click or touch interaction events, which are easily to be confused with mousedown and mouseup events. Usually, these two sets of events appear simultaneously. However, they are not exactly the same. We prefer to recommend the former one. Reasons are as follows:

1、start / release is applicable to both mouse and touch interaction

After blocking some acquiescent touch interaction events, mouseup and mousedown events will typically not be triggered during touch interaction. Only during touchstart, touchend or multi-touch event, touchstart and touchend event may be triggered several times. Under such circumstance, it is inconvenient to use the original touchevent directly. However, start / release event can avoid these problems effectively. Start event is triggered when touch starts and release event when all touch points are released.

2、start / release can distinguish left and right button

On desktop browser, start and release event are of significance. These two events can be triggered only when left button is clicked. If the right button is clicked, start2 and release2 event will be triggered. And the original mousedown event can be triggered whenever the left, middle or right button is clicked;

3、start / release event appear in a pair

The mousedown and mouseup event may not appear in a pair even monitored on the same . If the event is intercepted or stops spreading, it may occurs that only mousedown event appears while mouseup event not. At this time, the start and release event can be more secure.

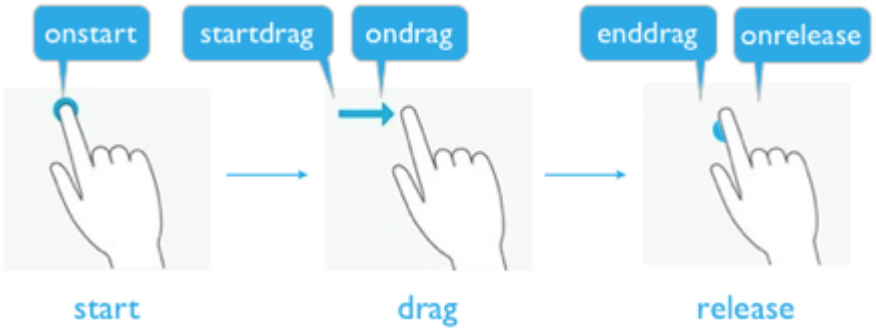
DragEvent

Three steps for drag event operation: start, drag, release, specific incorporations of different devices are as follows:

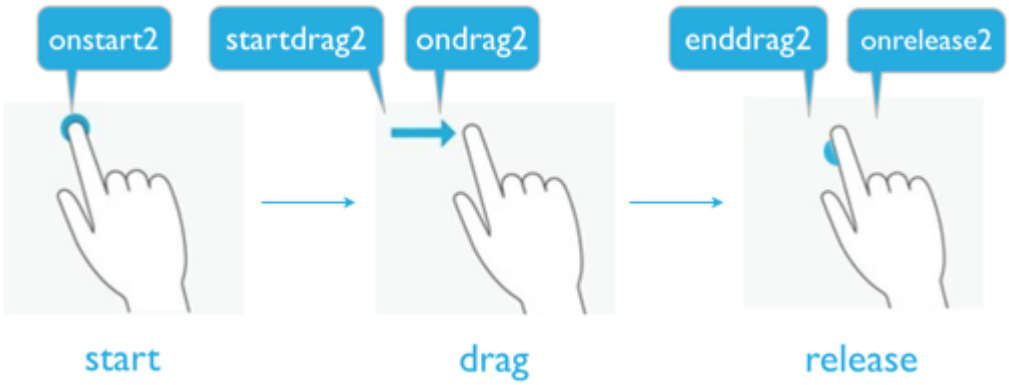
Desktop browser: mousedown-->mousemove-->mouseup

Mobile device: touchdown-->touchmove-->touchup or out of screen

Five events triggered during the whole drag event: onstart, startdrag, ondrag, enddrag, onrelease



V2.5 adds right button drag event to achieve the right button sselection function. There are five things will be triggered during the right button drag event: onstart2, startdrag2, ondrag2, enddrag2, onrelease2.



Interaction Response Interception

V2.5 adds two functions as follows:

- onevent – all events

onevent function can respond to all types of events and deal with them uniformly within itself.

```
graph.addCustomInteraction({
    onevent: function (type, evt, graph) {
        Q.log(type)
    }
})
```

- accept – used for judging the response event

This function is used for judging the response event. If we hope some specific interactive monitor to respond under a certain circumstance, we can control such logic to activate or cancel rbutton selection function through accept function, like rbutton selection interaction of Graph works acquiescently when graph.enableRectangleSelectionByRightButton !== false.

Example for rbutton selection event:


```
var RectangleSelectionInteractionByRightButton = {
    accept: function(type, evt, graph){
        return graph.enableRectangleSelectionByRightButton !== false;
    },
    startdrag2: function(evt, graph){
        //...
    },
    ondrag2: function(evt, graph){
        //...
    },
    enddrag2 : function(evt, graph) {
        //...
    },
}
```

Multi-touch event

As for mobile devices backing multi-touch operation, Graph components support multi-touch operation, like zooming the page through pinch event or translation through swipe event.

Parameters in the subjects of pinch event

```
graph.onpinch = function(evt, graph){
    evt.dScale;//change value of scaling
    evt.center;//scaling center
}
```

Reference code for pinch event to zoom pages

```
onpinch: function(evt, graph){
        this._start = true;
        var dScale = evt.dScale;


        if(dScale && dScale != 1){
            var p = graph.globalToLocal(evt.center);
            graph.zoomAt(dScale, p.x, p.y);
        }
    }
```

List of Interaction Events

Use the table below as a reference, know more about the purpose or application of the encapsulation of interaction events by Graph events, comprehend the peculiar characteristics of different desktop platforms and mobile devices relatively, and find out the unique attribute of different event subjects.

Action	DeskTop	Touch	Event Type	Properties
Click	click	tap	onclick	
Double Click	double click	double tap	ondblclick	
Long Press	longpress	longpress	onlongpress	
ToolTip	mouse move	-	onmousemove	
Wheel	mouse wheel	-	onmousewheel	delta
Start	mouse down	start	onstart / onstart2	
Release	mouse up	release	onrelease / onrelease2	
Move, Pan, Rectangle Select	drag right button drag	drag	startdrag / startdrag2 ondrag / ondrag2 enddrag / enddrag2	dx, dy, vx, vy
Pinch	-	pinch	onpinch	dScale, center
ContextMenu	contextmenu		oncontextmenu	
Basic Mouse Input	mousedown, mouse		onmousedown onmouseup	
Key Down	keydown		onkeydown	

Adds Interaction Listener

There are three ways for event monitoring: Graph#on***, Graph#addCustomInteraction(interaction), Graph#interactionMode, the previous two will not affect current interaction mode (such as default translation and zoom interaction). For the later one, you have to combine the interaction mode by yourself. Now the first method is introduced. And the later two kinds will be introduced in the chapter of interaction mode

Graph.on***
the usage of this method is simplest

Example:

```
graph.onclick = function(evt, graph){
    Q.log(evt);
}
```

Graph#addCustomInteraction(interaction)

Example

```
graph.addCustomInteraction({
    onevent: function (type, evt, graph) {
        Q.log(type)
    }
})
```

Graph#interactionMode

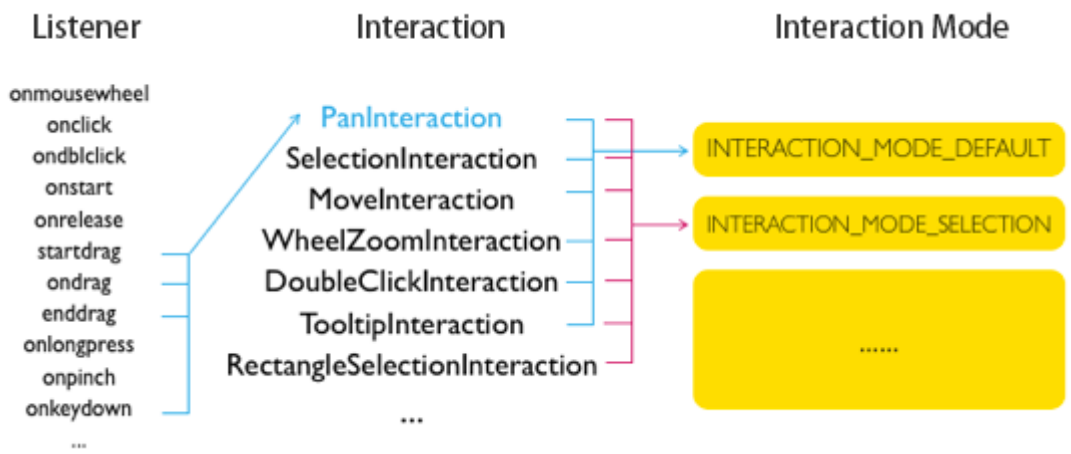
Example

```
graph.interactionMode = Q.Consts.INTERACTION_MODE_VIEW
```


Graph Interaction Mode

Generally it has to handle many interaction events for finishing one interaction action, such as node drag and drop. It requests to monitor #startdrag, #ondrag, #enddrag three events. At the beginning of drag and drop, records the element of current mouse position. During the drag and drop, change the element position, and release relevant resources in the end. The series of actions are coordinated for finishing one event, so called "interaction". To realize the object of this interaction or function, we called it "interaction device". So the node drag and drop corresponds to Q.MoveInteraction. Canvas translation corresponds to Q.PanInteraction. Wheel zoom canvas corresponds to Q.WheelZ oomInteraction. And a group of interaction device are used together, forming an "interaction mode"

The relation among listener, interaction device, and the interaction mode is showed in the following figure:



- Graph#addCustomInteraction(interaction)
- Graph#interactionMode

Graph#addCustomInteraction(interaction)

Graph#addCustomInteraction(interaction)

In actual application, generally the default interaction mode is kept. On such basis, adds custom interaction. To handle just one event, the method Graph#on*** mentioned above may be applied.

For complicated action, adds custom interaction, and sets the component with a custom interaction device

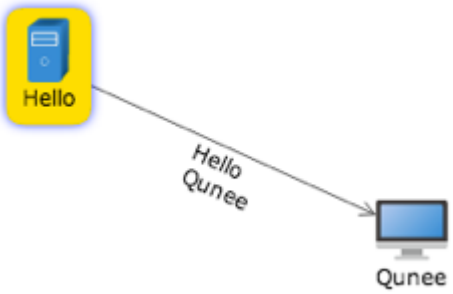
Example

Monitors mouse move event. Realizes the effect of highlight display of element

```
var graph = new Q.Graph(canvas);
var hello = graph.createNode("Hello", -100, -50);
hello.image = Q.Graphs.server;
var qunee = graph.createNode("Qunee", 100, 50);
var edge = graph.createEdge("Hello\nQunee", hello, qunee);

var currentElement;
var highlightColor = '#FFDB19';
function unhighlight(element){
    element.setStyle(Q.Styles.BACKGROUND_COLOR, null);
    currentElement.setStyle(Q.Styles.PADDING, null);
}
function highlight(element){
    if(currentElement == element){
        return;
    }
    if(currentElement){
        unhighlight(currentElement);
    }
    currentElement = element;
    if(!currentElement){
        return;
    }
    currentElement.setStyle(Q.Styles.BACKGROUND_COLOR, highlightColor);
    currentElement.setStyle(Q.Styles.PADDING, new Q.Insets(5));
}

graph.addCustomInteraction({
    onmousemove: function(evt, graph){
        var ui = graph.getUIByMouseEvent(evt);
        if(!ui){
            graph.cursor = null;
            highlight(null);
            return;
        }
        graph.cursor = "pointer";
        highlight(ui.data);
    }
});
```

Operation effect

When the mouse drags over the node, it shows a background in yellow:



Graph#interactionMode

Each interaction device realizes one function. A group of interaction devices coordinates for one work, forming an interaction mode, such as default interaction mode which includes the translate interaction device, node drag interaction device, spot selection interaction device, text hint interaction device and so on. We can combine different interaction devices for satisfying the demand of interaction.

Following several interaction modes are supplied in default:

- Consts.INTERACTION_MODE_DEFAULT - default interaction mode
- Consts.INTERACTION_MODE_SELECTION - frame selection interaction mode
- Consts.INTERACTION_MODE_ZOOMIN - zoom in interaction mode
- Consts.INTERACTION_MODE_ZOOMOUT - zoom out interaction mode

Switch interaction mode

The current interaction mode can be switched by the property of #interactionMode

Example

Switched to frame selection interaction mode

```
graph.interactionMode = Q.Consts.INTERACTION_MODE_SELECTION;
```

Custom interaction mode

The interaction mode can also be completely customized. There are two steps: firstly register a new name of interaction mode, such as "VIEW_MODE", and then set this name to the component of network graph

For example, registers a new interaction mode, and applies the translation, text hint and custom combination of interaction device

```
Q.Defaults.registerInteractions("CUSTOM-MODE", [Q.PanInteraction, Q.TooltipInteraction, {
    onclick: function(evt, graph){
        Q.log(evt);
    }
}]);
graph.interactionMode = "CUSTOM-MODE";
```

Other related properties and methods of interaction

- #enableTooltip :Boolean - whether the prompt text is displayed
- #enableWheelZoom :Boolean - whether the mouse wheel will be used for zooming

Common Properties and Methods

Late call - callLater(call, scope, delay)

After the graph component is created, it will not enter into effect and draw immediately, which will be handled in the next drawing thread. It means the display size of element, canvas range and size can be acquired late only

For example, the canvas range and node size can be acquired via callLater()

```
var graph = new Q.Graph(canvas);
var hello = graph.createNode("Hello", -100, -50);
hello.image = Q.Graphs.server;
var qunee = graph.createNode("Qunee", 100, 50);
var edge = graph.createEdge("Hello\nQunee", hello, qunee);

graph.callLater(function(){
    Q.log('graph bounds: ' + graph.bounds);
    Q.log('hello node bounds: ' + graph.getUIBounds(hello));
});
```

Print results:

graph bounds: -125.5 , -75 , 256 , 167.4

hello node bounds: -125.5 , -75 , 51 , 68.4

For example, delay to call automatic layout, to ensure before layout the component size has been well calculated

```
var layouter = new Q.TreeLayouter(graph);
graph.callLater(function(){
    layouter.doLayout();
    graph.zoomToOverview();
})
```

Acquire the component of mouse point position - #hitTest(evt) → {BaseUI}

For example, the method of hitTest is used to judge if the mouse clicks on the text label

```
graph.onclick = function(evt) {
    var target = graph.hitTest(evt);
    if(target instanceof Q.LabelUI){
        Q.log(target.data);
    }
}
```

Operation results. Clicks the element name by mouse. The text information will be printed via control platform

Update component viewport and redraw the canvas - updateViewport()

Generally the graph component is places at one DIV element. This DIV is so called canvas model. Graph components will be filled fully into this model. By setting CSS and HTML properties, the DIV size may change. Now the size of graph component has to be resized and redrawn

Example

Presses Letter F on the keyboard. Sets CSS style of graph component, to make it fully filled at the browser window

```
var html = graph.html;
graph.html.style.backgroundColor = '#EEE';

graph.onkeydown = function(evt) {
    if(evt.keyCode != 70){
        return;
    }
    if(!graph.oldCSS || html.style.position != 'fixed'){
        graph.oldCSS = {
            position: html.style.position,
            width: html.style.width,
            height: html.style.height,
            left: html.style.left,
            top: html.style.top
        };
        html.style.position = 'fixed';
        html.style.width = window.innerWidth + 'px';
        html.style.height = window.innerHeight + 'px';
        html.style.left = '0px';
        html.style.top = '0px';
        html.style.zIndex = 1000;
    }else{
        html.style.position = graph.oldCSS.position;
        html.style.width = graph.oldCSS.width;
        html.style.height = graph.oldCSS.height;
        html.style.left = graph.oldCSS.left;
        html.style.top = graph.oldCSS.top;
    }
    graph.updateViewport();
}
```


Selectable Filtering, Movable Filtering

Movable Filtering

If the control element is movable - isMovable(item) → {Boolean}

default to obtain the movable property of element, and user can have custom edition:

```
isMovable : function(item) {
 return item.movable !== false;
}
```

For example, judge if the element can be moved via the name of element

```
var canMove = graph.createNode("Q-Node", -100, -50);
var cannotMove = graph.createNode("Node", 100, 50);
var edge = graph.createEdge("Hello\nQunee", canMove, cannotMove);
graph.isMovable = function(item){
    return item.name && item.name.indexOf('Q') === 0;
}
```

Selectable Filtering

What is similar includes whether the control function is selectable isSelectable(item)

- isSelectable(item) → {Boolean} - whether it is selectable
- isSelected(element) → {Boolean} - whether the element is selected

Other properties and methods

Other properties and methods

- Invalid component, redraw canvas -invalidate()
- set the maximum update rate - setMaxFPS(fps)
- traverse visible element - forEachVisibleUI(call, scope)
- Acquire element at mouse point position - getElementByMouseEvent(evt) → {Element}
- Acquire element by name - getElementByName(name) → {Element}
- Move element position - moveElements(elements, dx, dy)
- Export canvas - exportImage(scale, clipBounds) → {Object}
- Whether elements are visible - isVisible(item) → {Boolean}

Example

Exports the canvas to images, and displays at new webpage

```
function exportImage(graph, scale, clipBounds) {
    var imageInfo = graph.exportImage(scale, clipBounds);
    if (!imageInfo || !imageInfo.data) {
        return false;
    }
    var win = window.open();
    var doc = win.document;
    doc.title = "export image - " + imageInfo.width + " x " + imageInfo.height;
    var img = doc.createElement("img");
    img.src = imageInfo.data;
    doc.body.appendChild(img);
}
```

Element Default Style Sheet

Each element can be set with different style property. If the property is not appointed, it will be acquired from the default style sheet of graph components. For example, the system defaults to select blue, and font is in pixel of 12. Of course, users can change these. By setting default style sheet to change the default value

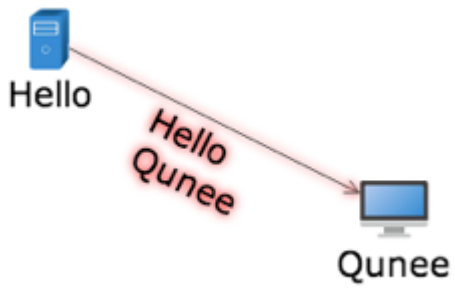
- #styles :Object - default style sheet

Example

Sets element default style sheet. Font size and selected color of this element

```
var graph = new Q.Graph(canvas);

var hello = graph.createNode("Hello", -100, -50);
hello.image = Q.Graphs.server;
var qunee = graph.createNode("Qunee", 100, 50);
var edge = graph.createEdge("Hello\nQunee", hello, qunee);

var styles = {};
styles[Q.Styles.SELECTION_COLOR] = 'red';
styles[Q.Styles.LABEL_FONT_SIZE] = '20';
graph.styles = styles;
```

Operating interface:



Navigation Panel Type

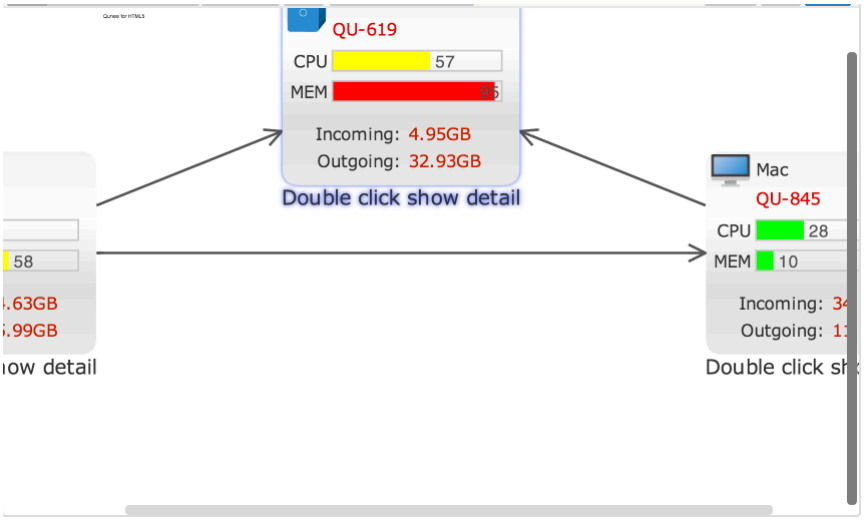
Qunee for HTML5 v1.6 has added rolling bar navigation mode. Plus the navigation button said before, and except the condition of navigation panel, there are three navigation modes. The property of navigationType is provided in Graph and switched, so as to adapt to different application scenes. Defaults to use rolling bar mode

The following codes are used for setting topological graph

```
graph.navigationType = Q.Consts.NAVIGATION_BUTTON;
```

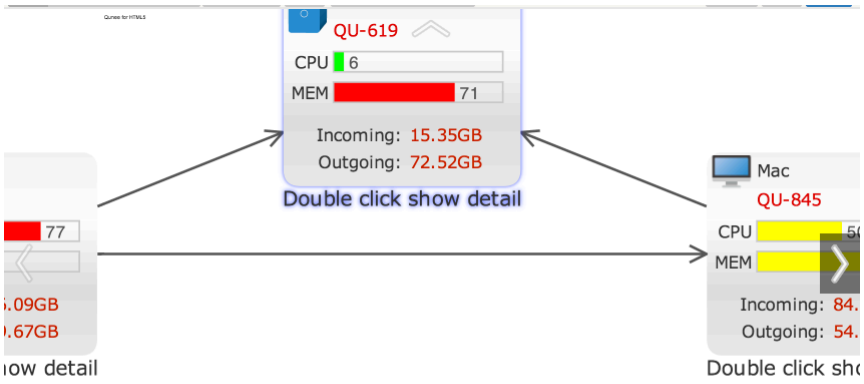
Roll bar mode - Q.Consts.NAVIGATION_SCROLLBAR

The rolling bar mode conforms to traditional navigation mode, while supporting unlimited navigation of canvas range, with good interial effect, which can be used in different modes. Qunee defauts to use this mode

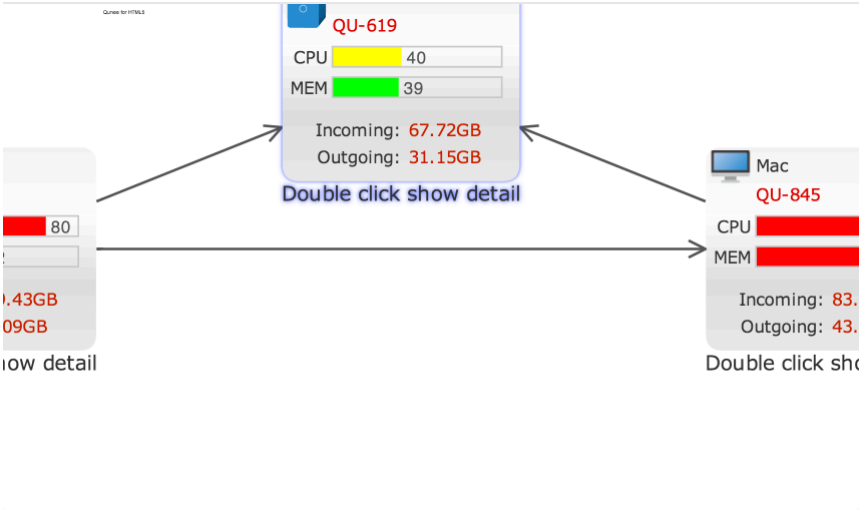


Navigation button mode - Q.Consts.NAVIGATION_BUTTON

The button mode occupies less space. The size of canvas can be marked. It is applicable to used at the scenes in combination with maps



Without navigation panel -Q.Consts.NAVIGATION_NONE



Delayed Rendering

Interface V2.5 release adds delay drawing mechanism can improve the response time interactive zooming experience

#delayedRendering // enable delayed rendering, enabled by default

#pauseRendering = function(pause, force){} //whether to suspend or start drawing

Example of use

In the mouse wheel zoom and touch kneading zoom are enabled for delayed rendering, the following is a pinch zooming relevant code fragment when interacting, stop drawing canvas at the start of kneading event, repainted canvas kneading event is over, reducing the interaction of CPU overhead, shorten the response time interactive

```
Q.PanInteraction.prototype = {
...
    startpinch: function(evt, graph){
        graph.pauseRendering(true);
    },
    onpinch: function(evt, graph){
        this._start = true;
        var dScale = evt.dScale;
        if(dScale){
            var p = graph.globalToLocal(evt.center);
            graph.zoomAt(dScale, p.x, p.y, false);
        }
    },
    endpinch: function(evt, graph){
        graph.pauseRendering(false);
    }
}
```

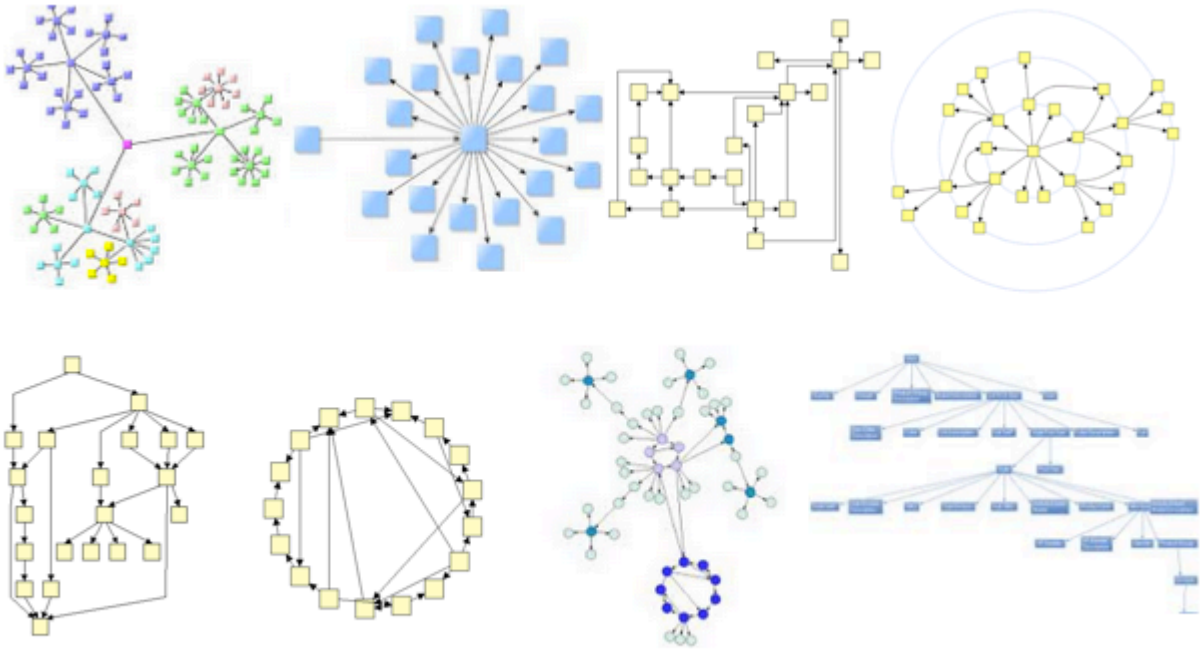
Possible problems

Delay drawing mechanism can effectively shorten interactive response time, but also some code issues
Blank canvas
After opening delay drawing, the canvas at the end of the interactive refresh, which means that during the interaction will appear blank, such as blank canvases appear around narrow
Jitter canvas
When drawing canvas pause, zoom panning effect canvas all realized by CSS transform property, due to different browser support is inconsistent effects that may occur under certain version browser interface jitter problems, such as Firefox V45.0.1, this time you can choose Close canvas delay drawing attributes: graph.delay edRendering = false; to avoid problems.

Automatic Layouter

The automatic layout of graph is an algorithm science. The purpose of layout is to express the structure of graph, generate the layout effect of aesthetics. There are many types of layout. it can be divided into three classes: scatter, tree-form layout and network layout

Three types have several layout algorithm, such us rectangular layout, tree-form layout, bubble layout, organic layout, spring layout, looped layout, layer layout and orthogonal layout etc. Some layout work only to special topological structure. For example, tree layout and bubble layout apply to the tree graph only. Some are general layout, applied to any topological structure. For example, spring layout and organic layout, Qunee realizes only several layout

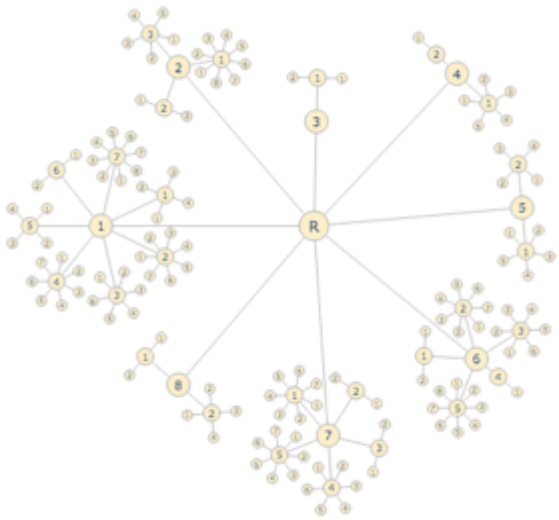


几种常见布局效果，图片来自互联网

- Balloon Layouter
- Spring Layouter
- Tree Layouter

Balloon Layouter

Bubble layout also belongs to tree layout. Tree structure can be distributed via the way of polar coordinates



Layout params

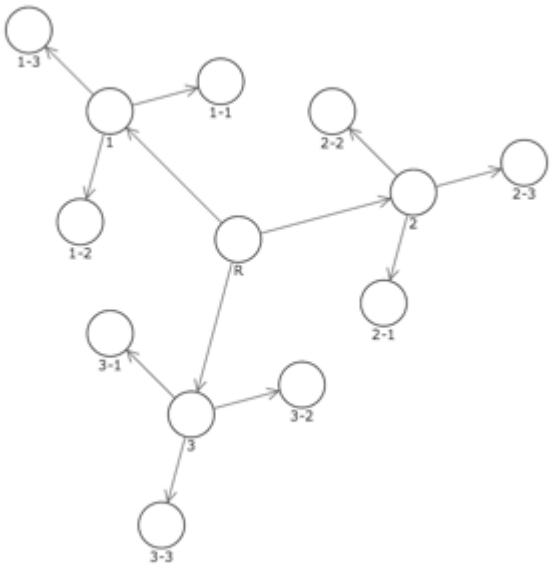
Bubble layout provides the following params:

- #angleSpacing - angle distribution mode
  Support even layout(Q.Consts.ANGLE_SPACING_REGULAR)and distribute by demand(Q.Consts.ANGLE_SPACING_PROPORTIONAL); default to distribute by demand;
- #radiusMode - radius mode
  Supports unified radius(Q.Consts.RADIUS_MODE_UNIFORM)an variable radius (Q.Consts.RADIUS_MODE_VARIABLE). Default to variable radius
- #radius - Min radius length
- #gap - distance among nodes
- #startAngle - start rotate angle

Example

```
var graph = new Q.Graph("canvas");

function createNode(name, from){
    var node = graph.createNode(name);
    node.image = Q.Shapes.getShape(Q.Consts.SHAPE_CIRCLE, 40, 40);
    if(from){
        graph.createEdge(from, node);
    }
    return node;
}
var root = createNode("R");
var i = 0;
while(i++ < 3){
    var node = createNode("" + i, root);
    var j = 0;
    while(j++ < 3){
        createNode("" + i + "-" + j, node);
    }
}

var layouter = new Q.BalloonLayouter(graph);
layouter.radiusMode = Q.Consts.RADIUS_MODE_UNIFORM;
layouter.radius = 100;
layouter.startAngle = Math.PI / 4;
layouter.doLayout({callback: function(){
    graph.zoomToOverview();
}});
```

Operation effect



Spring Layouter

The principle of spring layout is to simulate the physical environment. It is a general layout algorithm realized by the balance of several force, a dynamic layout, which can express people relation chart and dynamic network chart. In Qunee, it is realized by Q.SpringLayouter.

Layout principles

The spring layout is the balance of three force: electrostatic repulsion, elastic force and central gravity. The previous two force are subject to Coulomb law and Hu Ke law. The third force is a balance for controlling the element moving the the center

Coulomb law:

$$F = k_e \frac{qq'}{r^2}$$

Coulomb law:

$$\sigma = E\epsilon$$

Central gravity is in direct proportion to the distance

F = a * D

Layout params

The spring layout has three control factors: spring coefficient, gravity coefficient and repulsion coefficient

#elasticity - the higher the elasticity

coefficient is the shorter the edge zooming would be, with the reference value at 0-10

#attractive - the higher the central

gravity coefficient is, the more intense the overall distribution would be, with the reference value at 0-1

#repulsion - the higher the repulsion coefficient

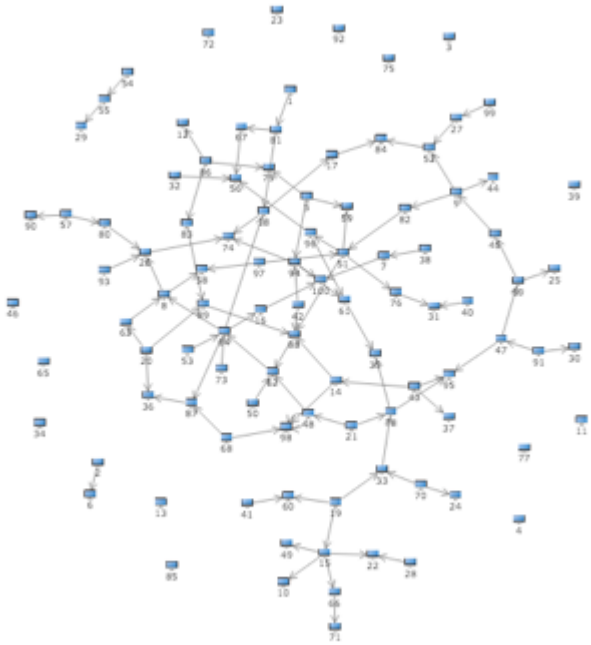
value is, the bigger the distance among nodes would be, with the reference value at 0-100

Example

Simulated data would be used in the following example to express the effect of spring layout

```
var graph = new Q.Graph("canvas");

var nodes = [];
function createNode(name){
    var node = graph.createNode(name);
    node.size = {width: 16};
    nodes.push(node);
    return node;
}
function createRandomEdge(){
    var from = nodes[Q.randomInt(nodes.length)];
    var to = nodes[Q.randomInt(nodes.length)];
    if(from != to){
        return graph.createEdge(from, to);
    }
}
var i = 0;
while(i++ < 100){
    createNode("" + i);
}
i = 0;
while(i++ < 100){
    createRandomEdge();
}

var layouter = new Q.SpringLayouter(graph);
layouter.repulsion = 50;
layouter.attractive = 0.5;
layouter.elastic = 5;
layouter.start();
```

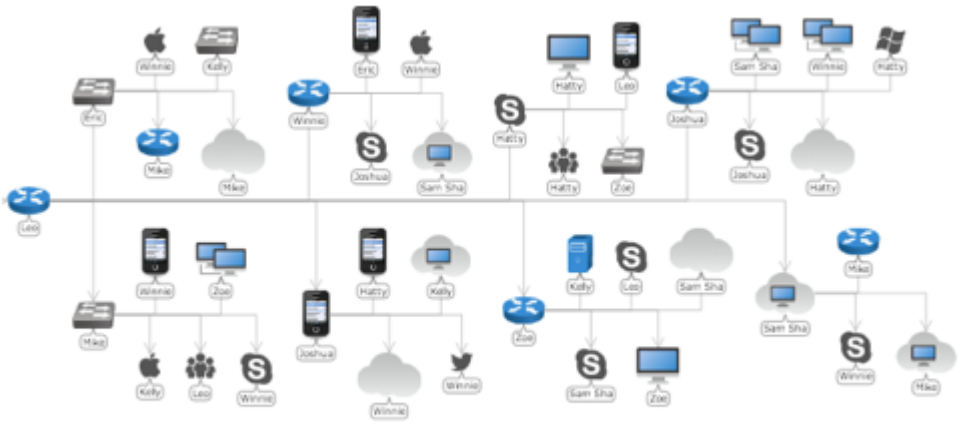
Operation effect



Tree Layouter

Tree graph is a common topological structure, which is generally used to express the layer structure and organization chart and owner-member relationship. The tree structure is usually formed via the set membership, or the topological connection relation

The tree layout is distributed via the traditional branching tree, which may be set in arraying direction of layers, or the arrangement way among the same layer of nodes. They can be combined to realize various tree layout effect



从左向右分布

从下往上分布

Layout params

There are two parameters, separately controlling the layout effect among layers and the effect at the same layer. It can be set for layouter object or node property. The former one will be effective to the entire, and the later one will work at single branch only:

For example, set the global layout direction from the bottom to the top, when following codes can be used:

```
layouter.parentChildrenDirection = Q.Consts.DIRECTION_TOP;
```

#parentChildrenDirection - layout direction among layers

Support directions such as up, down, left and right:

- Q.Consts.DIRECTION_RIGHT - layout rightward
- Q.Consts.DIRECTION_LEFT - layout leftward
- Q.Consts.DIRECTION_CENTER - layout in central middle
- Q.Consts.DIRECTION_BOTTOM - layout downward
- Q.Consts.DIRECTION_TOP - layout upward
- Q.Consts.DIRECTION_MIDDLE - layout in vertical middle

#layoutType - Layout type of adjacent nodes

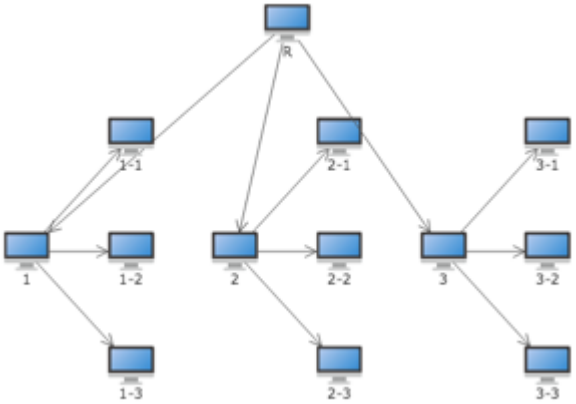
Two types of layout are provided, even and two side distribution:

- Q.Consts.LAYOUT_TYPE_EVEN - even distribution, confirm child layout direction automatically according to direction among layers
- Q.Consts.LAYOUT_TYPE_EVEN_HORIZONTAL - horizontal even layout
- Q.Consts.LAYOUT_TYPE_EVEN_VERTICAL - vertical even layout
- Q.Consts.LAYOUT_TYPE_TWO_SIDE - two-side layout

Example

```
var graph = new Q.Graph("canvas");

function createNode(name, from){
    var node = graph.createNode(name);
    if(from){
        graph.createEdge(from, node);
    }
    return node;
}
var root = createNode("R");
root.parentChildrenDirection = Q.Consts.DIRECTION_BOTTOM;
var i = 0;
while(i++ < 3){
    var node = createNode("" + i, root);
    node.parentChildrenDirection = Q.Consts.DIRECTION_RIGHT;
    node.layoutType = Q.Consts.LAYOUT_TYPE_EVEN_VERTICAL;
    var j = 0;
    while(j++ < 3){
        createNode("" + i + "-" + j, node);
    }
}

var layouter = new Q.TreeLayouter(graph);
layouter.layoutType = Q.Consts.LAYOUT_TYPE_EVEN_HORIZONTAL;
layouter.doLayout({callback: function(){
    graph.zoomToOverview();
}});
```

Operation effect



Style List

The display effect of element can be controlled via the style property in Qunee

Setting of styles

The following codes are used for setting edges and not displaying arrows at terminal end

```
edge.setStyle(Q.Styles.ARROW_TO, false);
```

Style List

The following shows the type style list

ALPHA相关样式

Style Name	Reference Value
ALPHA	The transparency of the element default value: 1

example

```
var node = graph.createNode("hello");
node.setStyle(Q.Styles.ALPHA, 0.5);
var node = graph.createNode("qunee", -20, 10);
node.setStyle(Q.Styles.ALPHA, 0.5);
```

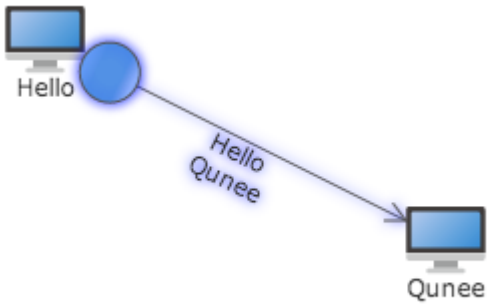
run effect

ARROW_FROM related styles

For setting related styles at the start end arrow

```
var graph = new Q.Graph("canvas");
var hello = graph.createNode("Hello", -100, -50);
var qunee = graph.createNode("Qunee", 100, 50);
var edge = graph.createEdge("Hello\nQunee", hello, qunee);
edge.setStyle(Q.Styles.ARROW_FROM, Q.Consts.SHAPE_CIRCLE);
edge.setStyle(Q.Styles.ARROW_FROM_FILL_COLOR, "#2898E0");
edge.setStyle(Q.Styles.ARROW_FROM_FILL_GRADIENT, Q.Gradient.LINEAR_GRADIENT_HORIZONTAL);
edge.setStyle(Q.Styles.ARROW_FROM_SIZE, {width: 30, height: 30});
```

Operation effect:



Style name	Reference value
ARROW_FROM	Type of start end arrow can be set to Q.Consts.SHAPE_*
ARROW_FROM_FILL_COLOR	Filling color can be set in the following format: #2898E0, rgba(22,33,240,0.5)
ARROW_FROM_FILL_GRADIENT	Filling gradient can use the example of Q.Gradient. For instance, new Gradient(Q.Consts.GRADIENT_TYPE_LINEAR, [Q.toColor(0x8AFFFFFF), Q.toColor(0x44CCCCCC)], [0.1, 0.9])
ARROW_FROM_LINE_CAP	Line top style can be set to Q.Consts.LINE_CAP_*
ARROW_FROM_LINE_DASH	Dotted line style uses array format, such as [2, 5]
ARROW_FROM_LINE_DASH_OFFSET	Offset of dotted line, number type, dynamic modification of this property would realize the flow effect of dotted line
ARROW_FROM_LINE_JOIN	Line inflection point style can be set to Q.Consts.LINE_JOIN_*
ARROW_FROM_OFFSET	Arrow offset supports number type or includes objects of property x, y, such as {x: 80}
ARROW_FROM_SIZE	Default value is: 10 arrow size, supporting value, or containing objects of property width and height, such as {width: 100}
ARROW_FROM_OUTLINE	The thickness of the outer border of the arrow
ARROW_FROM_OUTLINE_STYLE	The style of the outer border of the arrow, can be set as color

ARROW_FROM_STROKE	Line thickness and number type
ARROW_FROM_STROKE_STYLE	Edge style can be set in certain color, such as #2898E0

ARROW_TO related styles

Style name	Reference value
ARROW_TO	Type of arrows in terminal end, which can be set to Q.Consts.SHAPE_*. The default value is true
ARROW_TO_FILL_COLOR	Filling color can be set in the following format: #2898E0, rgba(22,33,240,0.5)
ARROW_TO_FILL_GRADIENT	Filling gradient can use the example of Q.Gradient. For instance, new Gradient(Q.Consts.GRADIENT_TYPE_LINEAR, [Q.toColor(0x8AFFFFFF), Q.toColor(0x44CCCCCC)], [0.1, 0.9])
ARROW_TO_LINE_CAP	Line top style can be set to Q.Consts.LINE_CAP_*
ARROW_TO_LINE_DASH	Dotted line style uses array format, such as [2, 5]
ARROW_TO_LINE_DASH_OFFSET	Offset of dotted line, number type, dynamic modification of this property would realize the flow effect of dotted line
ARROW_TO_LINE_JOIN	Line inflection point style can be set to Q.Consts.LINE_JOIN_*
ARROW_TO_OFFSET	Arrow offset supports number type or includes objects of property x, y, such as {x: 80}
ARROW_TO_SIZE	Default value is: 10 arrow size, supporting value, or containing objects of property width and height, such as {width: 100}
ARROW_TO_OUTLINE	The thickness of the outer border of the arrow
ARROW_TO_OUTLINE_STYLE	The style of the outer border of the arrow, can be set as color
ARROW_TO_STROKE	Line thickness and number type
ARROW_TO_STROKE_STYLE	Edge style can be set in certain color, such as #2898E0

BACKGROUND related styles

```
var graph = new Q.Graph("canvas");
var hello = graph.createNode("Hello");
hello.setStyle(Q.Styles.BACKGROUND_COLOR, "#2898E0");
hello.setStyle(Q.Styles.BACKGROUND_GRADIENT, Q.Gradient.LINEAR_GRADIENT_VERTICAL);
hello.setStyle(Q.Styles.PADDING, new Q.Insets(10, 20));
```

Operation effect

Style name	Reference value
BACKGROUND_COLOR	Filling color can be set in the following format: #2898E0, rgba(22,33,240,0.5)
BACKGROUND_GRADIENT	Filling gradient can use the example of Q.Gradient. For instance, new Gradient(Q.Consts.GRADIENT_TYPE_LINEAR, [Q.toColor(0x8AFFFFFF), Q.toColor(0x44CCCCCC)], [0.1, 0.9])

BORDER related styles

```
var hello = graph.createNode("Hello");
hello.setStyle(Q.Styles.BORDER, 2);
hello.setStyle(Q.Styles.BORDER_COLOR, "#2898E0");
hello.setStyle(Q.Styles.PADDING, new Q.Insets(10, 20));
```

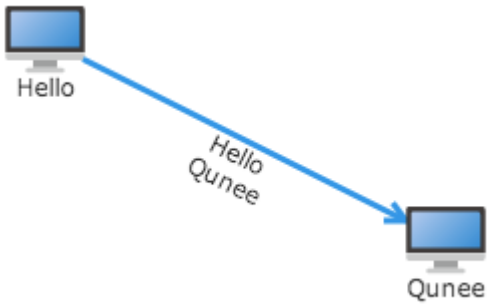
Operation effect

Style name	Reference value
BORDER	Border thickness
BORDER_COLOR	Border color can be set in the following format: #2898E0, rgba(22,33,240,0.5)
BORDER_LINE_DASH	Dotted line style uses array format, such as [2, 5]
BORDER_LINE_DASH_OFFSET	Offset of dotted line, number type, dynamic modification of this property would realize the flow effect of dotted line
BORDER_RADIUS	Fillet supports number type or includes objects of property x, y, such as {x: 10, y: 5}

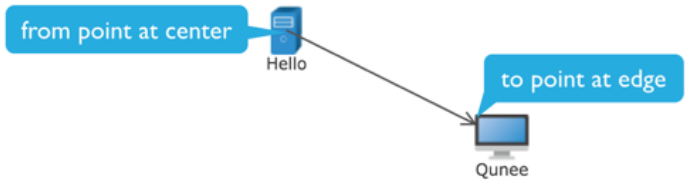
EDGE related styles

```
var hello = graph.createNode("Hello", -100, -50);
var qunee = graph.createNode("Qunee", 100, 50);
var edge = graph.createEdge("Hello\nQunee", hello, qunee);
edge.setStyle(Q.Styles.EDGE_COLOR, "#2898E0");
edge.setStyle(Q.Styles.EDGE_WIDTH, 3);
```

Operation effect



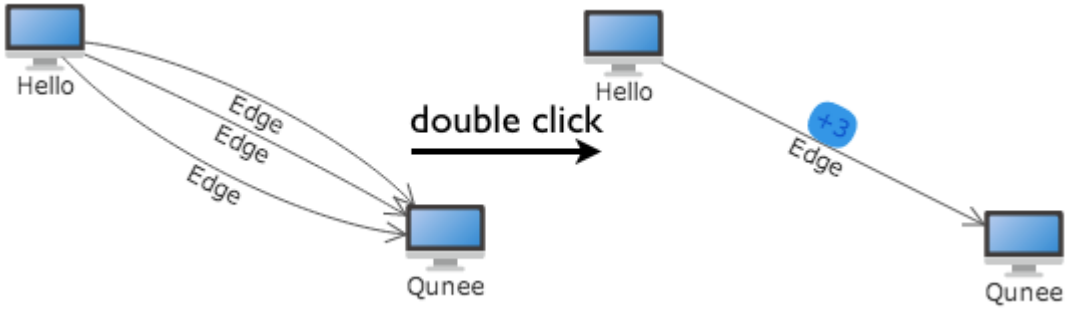
Style name	Reference value
EDGE_COLOR	Default value is #555555. Edge color can be set in the following format: #2898E0, rgba(22,33,240,0.5)
EDGE_CONTROL_POINT	Edge control point position includes objects of property x, y, such as {x: 100, y: 100}
EDGE_CORNER	Default value: round Edge reflection corner can be set to: Q.Consts.EDGE_CORNER_*
EDGE_CORNER_RADIUS	Default value: 8 Round corner size, number type
EDGE_EXTEND	Default value: 20
EDGE_FROM_OFFSET	Offset supports number type or includes objects of property x, y, such as {x: 80}
EDGE_LINE_DASH	Dotted line style uses array format, such as [2, 5]
EDGE_LINE_DASH_OFFSET	Offset of dotted line, number type, dynamic modification of this property would realize the flow effect of dotted line
EDGE_LOOPED_EXTAND	Default value: 10
EDGE_SPLIT_BY_PERCENT	Default value: true
EDGE_SPLIT_PERCENT	Default value: 0.5
EDGE_SPLIT_VALUE	Default value: 20
EDGE_OUTLINE	Edge border width, number type, default to 0
EDGE_OUTLINE_STYLE	Edge border style can be set in certain color, such as #2898E0
EDGE_FROM_AT_EDGE	Edge endpoint at node icon edge, the default value: true see EDGE_TO_AT_EDGE

EDGE_TO_AT_EDGE	Edge endpoint at node icon edge, the default value: true 
EDGE_TO_OFFSET	Offset supports number type or includes objects of property x, y, such as {x: 80}
EDGE_WIDTH	Default value: 1

EDGE_BUNDLE related styles

```
var hello = graph.createNode("Hello", -100, -50);
var qunee = graph.createNode("Qunee", 100, 50);
function createEdge(gap){
    var edge = graph.createEdge("Edge", hello, qunee);
    edge.setStyle(Q.Styles.EDGE_BUNDLE_GAP, gap);
    edge.setStyle(Q.Styles.EDGE_BUNDLE_LABEL_BACKGROUND_COLOR, "#2898E0");
    edge.setStyle(Q.Styles.EDGE_BUNDLE_LABEL_PADDING, 3);
    return edge;
}
createEdge(10);
createEdge(20);
createEdge(30);
```

Operation effect



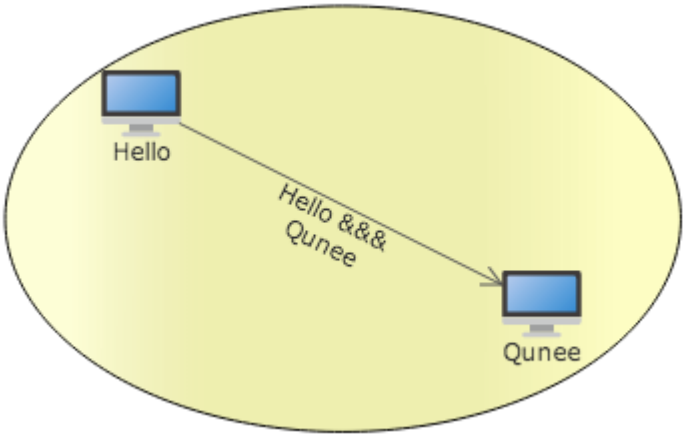
Style name	Reference value
EDGE_BUNDLE_GAP	Default value: 20 Gap, number type
EDGE_BUNDLE_LABEL_ANCHOR_POSITION	Default value: "cb" Aligns position, and supports Q.Position type, such as: Q.Position.CENTER_
EDGE_BUNDLE_LABEL_BACKGROUND_COLOR	Color can be set in the following format: #2898E0, rgba(22,33,240,0.5)

EDGE_BUNDLE_LABEL_BACKGROUND_GRADIENT	Gradient can use the example of Q.Gradient. For instance, new Gradient(Q.Consts.GRADIENT_TYPE_LINEAR, [Q.toColor(0x8AFFFFFF), Q.toColor(0x44CCCCCC)], [0.1, 0.9])
EDGE_BUNDLE_LABEL_BORDER	Border thickness and number type
EDGE_BUNDLE_LABEL_BORDER_STYLE	Border style can be set in certain color, such as #2898E0
EDGE_BUNDLE_LABEL_COLOR	Default value is #075bc5. It can be set in the following format: #2898E0, rgba(22,33,240,0.5)
EDGE_BUNDLE_LABEL_FONT_FAMILY	Font family, such as helvetica arial
EDGE_BUNDLE_LABEL_FONT_SIZE	Font size supports value, unit: pixel
EDGE_BUNDLE_LABEL_FONT_STYLE	Font style, such as: lighter
EDGE_BUNDLE_LABEL_OFFSET_X	Offset x, number type
EDGE_BUNDLE_LABEL_OFFSET_Y	Offset y, number type
EDGE_BUNDLE_LABEL_PADDING	Internal gap supports value or Q.Inserts type, such as: new Q.Insets(1 5)
EDGE_BUNDLE_LABEL_POINTER	Bubble pointer, boolean type, true or false
EDGE_BUNDLE_LABEL_POINTER_WIDTH	Bubble pointer width, number type
EDGE_BUNDLE_LABEL_POSITION	Default value: ct Aligns position, and supports Q.Position type, such as: Q.Position.CENTER_
EDGE_BUNDLE_LABEL_RADIUS	Fillet supports number type or includes objects of property x, y, such as {x: 10, y: 5}
EDGE_BUNDLE_LABEL_ROTATABLE	Whether rotatable, boolean type
EDGE_BUNDLE_LABEL_ROTATE	Rotate angle, number type, such as Math.PI/2

GROUP related styles


```
var hello = graph.createNode("Hello", -100, -50);
var qunee = graph.createNode("Qunee", 100, 50);
var edge = graph.createEdge("Hello &&&\nQunee", hello, qunee);
var group = graph.createGroup();
group.addChild(hello);
group.addChild(qunee);
group.groupType = Q.Consts.GROUP_TYPE_ELLIPSE;
group.setStyle(Q.Styles.GROUP_BACKGROUND_COLOR, Q.toColor(0xCCfcfb9b));
group.setStyle(Q.Styles.GROUP_BACKGROUND_GRADIENT, Q.Gradient.LINEAR_GRADIENT_HORIZONTAL);
```

Operation effect



Style name	Reference value
GROUP_BACKGROUND_COLOR	Default value is rgba(238,238,238,0.80). The color can be set in the following format: #2898E0, rgba(22,33,240,0.5)
GROUP_BACKGROUND_GRADIENT	Gradient can use the example of Q.Gradient. For instance, new Gradient(Q.Consts.GRADIENT_TYPE_LINEAR, [Q.toColor(0x8AFFFFFF), Q.toColor(0x44CCCCCC)], [0.1, 0.9])
GROUP_STROKE	Default value: 1 Edge thickness, number type
GROUP_STROKE_LINE_DASH	Dotted line style uses array format, such as [2, 5]
GROUP_STROKE_LINE_DASH_OFFSET	Offset of dotted line, number type, dynamic modification of this property would realize the flow effect of dotted line
GROUP_STROKE_STYLE	Default value: #000 Edge style can be set in certain color, such as #2898E0

IMAGE related styles

Node image setting


```
var hello = graph.createNode("Hello", -100, -50);
hello.setStyle(Q.Styles.IMAGE_BACKGROUND_COLOR, "#2898E0");
hello.setStyle(Q.Styles.IMAGE_PADDING, 5);
```

Operation effect

Style name	Reference value
IMAGE_ADJUST	Image adjustment mode, support two modes: flip and mirror Q.Consts.IMAGE_ADJUST_FLIP Q.Consts.IMAGE_ADJUST_MIRROR imageAdjust.setStyle(Q.Styles.IMAGE_ADJUST, Q.Consts.IMAGE_ADJUST_MIRROR); 
IMAGE_ALPHA	Image transparency, 0-1
IMAGE_BACKGROUND_COLOR	Color can be set in the following format: #2898E0, rgba(22,33,240,0.5)
IMAGE_BACKGROUND_GRADIENT	Gradient can use the example of Q.Gradient. For instance, new Gradient(Q.Consts.GRADIENT_TYPE_LINEAR, [Q.toColor(0x8AFFFFFF), Q.toColor(0x44CCCCCC)], [0.1, 0.9])
IMAGE_BORDER	Border thickness and number type
IMAGE_BORDER_COLOR	Color can be set in the following format: #2898E0, rgba(22,33,240,0.5)
IMAGE_BORDER_LINE_DASH	Dotted line style uses array format, such as [2, 5]
IMAGE_BORDER_LINE_DASH_OFFSET	Offset of dotted line, value type, dynamic modification of this property would realize the flow effect of dotted line
IMAGE_BORDER_RADIUS	Fillet supports value type or includes objects of property x, y, such as {x: 10, y: 5}
IMAGE_BORDER_STYLE	Border style can be set in certain color, such as #2898E0
IMAGE_PADDING	Internal gap supports value or Q.Inserts type, such as: new Q.Insets(10, 5)
IMAGE_RADIUS	Fillet supports value type or includes objects of property x, y, such as {x: 10, y: 5}

LABEL related styles

```
var text = graph.createText("Qunee for HTML5");
text.setStyle(Q.Styles.LABEL_BACKGROUND_COLOR, "#2898E0");
text.setStyle(Q.Styles.LABEL_BACKGROUND_GRADIENT, new Q.Gradient(Q.Consts.GRADIENT_TYPE_LINEAR,
['#00d4f9', '#1ea6e6'], null, Math.PI/2));
text.setStyle(Q.Styles.LABEL_COLOR, "#FFF");
text.setStyle(Q.Styles.LABEL_BORDER, 0.5);
text.setStyle(Q.Styles.LABEL_PADDING, 4);
text.setStyle(Q.Styles.LABEL_BORDER_STYLE, "#1D4876");
text.setStyle(Q.Styles.LABEL_RADIUS, 0);
text.setStyle(Q.Styles.LABEL_SIZE, new Q.Size(120, 40));
text.setStyle(Q.Styles.SELECTION_COLOR, "#0F0");
```

Operation effect

Style name	Reference value
LABEL_ALIGN_POSITION	Aligns position, and supports Q.Position type, such as: Q.Position.CENTER_TOP
LABEL_ANCHOR_POSITION	Default value: ct Aligns position, and supports Q.Position type, such as: Q.Position.CENTER_TOP
LABEL_BACKGROUND_COLOR	Default value: null Color can be set in the following format: #2898E0, rgba(22,33,240,0.5)
LABEL_BACKGROUND_GRADIENT	Default value: null Gradient can apply with the example of Q.Gradient, such as: new Gradient(Q.Consts.GRADIENT_TYPE_LINEAR, [Q.toColor(0x8AFFFFFF), Q.toColor(0x44CCCCCC)], [0.1, 0.9])
LABEL_BORDER	Default value: 0 Border thickness, value type
LABEL_BORDER_STYLE	Default value: #000 Border style can be set in certain color, such as #2898E0
LABEL_COLOR	Default value: #333 Color can be set in the following format: #2898E0, rgba(22,33,240,0.5)
LABEL_FONT_FAMILY	Font family, such as helvetica arial

LABEL_FONT_SIZE	Font size supports value, unit: pixel
LABEL_FONT_STYLE	Font style, such as: lighter
LABEL_OFFSET_X	Offset x, number type
LABEL_OFFSET_Y	Offset y, number type
LABEL_PADDING	Default value: new Q.Insets(0, 2) Internal gap supports value or Q.Inserts type, such as: new Q.Insets(10, 5)
LABEL_POINTER	Default value: true Bubble pointer, boolean type, true or false
LABEL_POINTER_WIDTH	Default value: 8 Bubble pointer width, number type
LABEL_POSITION	Default value: cb Aligns position, and supports Q.Position type, such as: Q.Position.CENTER_TOP
LABEL_RADIUS	Default value: 8 Fillet supports value type or includes objects of property x, y, such as {x: 10, y: 5}
LABEL_ROTATABLE	Default value: true whether it is rotatable, boolean type
LABEL_ROTATE	Rotate angle, value type, such as Math.PI/2
LABEL_SIZE	Size supports value or contains objects of property width and height, such as {width: 100}
LABEL_SHADOW_BLUR	Shadow blur distance of text label
LABEL_SHADOW_COLOR	Shadow color of text label
LABEL_SHADOW_OFFSET_X	Shadow x offset of text label
LABEL_SHADOW_OFFSET_Y	Shadow y offset of text label

LAYOUT related styles

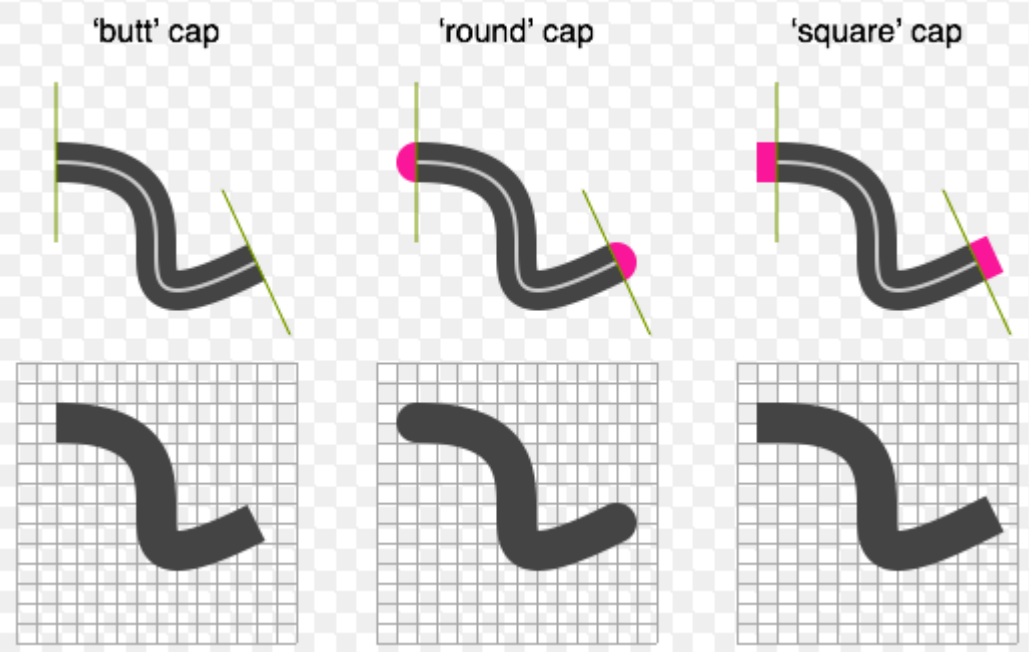
Refer to later examples of LINE related styles

Style name	Reference value

LAYOUT_BY_PATH	boolean，Whether is layouted by certain routines, or proper for ShapeNode and Edge, which react to the child component of element mounting

LINE related styles

Line top style(lineCap) - butt, round, square



Line inflection point style - lineJoin - miter, round, bevel

Line end points and inflection point styles of path


```
 var graph = new Q.Graph("canvas");
function createShape(join, x, y){
   var shape = graph.createShapeNode(join, x, y);
   shape.moveTo(-50, 50);
   shape.lineTo(0, 0);
   shape.lineTo(50, 50);
   shape.setStyle(Q.Styles.SHAPE_STROKE, 10);
   shape.setStyle(Q.Styles.SHAPE_STROKE_STYLE, "#2898E0");
   shape.setStyle(Q.Styles.LAYOUT_BY_PATH, true);
   shape.setStyle(Q.Styles.SHAPE_FILL_COLOR, null);
   shape.setStyle(Q.Styles.LINE_CAP, Q.Consts.LINE_CAP_TYPE_BUTT);
   shape.setStyle(Q.Styles.LINE_JOIN, join);
   return shape;
}
createShape(Q.Consts.LINE_JOIN_TYPE_BEVEL, -150, 0);
createShape(Q.Consts.LINE_JOIN_TYPE_MITER, 0, 0);
createShape(Q.Consts.LINE_JOIN_TYPE_ROUND, 150, 0);
```

Operation effect

Style name	Reference value
LINE_CAP	Line top style can be set to Q.Consts.LINE_CAP_*
LINE_JOIN	Line inflection point style can be set to Q.Consts.LINE_JOIN_*

PADDING related styles

Style name	Reference value
PADDING	For the internal gap of nodes, the gap effect can be seen at the set background or edge

RENDER related styles

Color dyeing, refer to online presentation

dyeing effect

Style name	Reference value
RENDER_COLOR	Color can be set in the following format: #2898E0, rgba(22,33,240,0.5)
RENDER_COLOR_BLEND_MODE	Default value: linear.burn

SELECTION related styles

Example

```
var graph = new Q.Graph("canvas");
var node = graph.createNode("node");
node.setStyle(Q.Styles.SELECTION_SHADOW_BLUR, 10);
node.setStyle(Q.Styles.SELECTION_COLOR, '#8F8');
node.setStyle(Q.Styles.SELECTION_SHADOW_OFFSET_X, 2);
node.setStyle(Q.Styles.SELECTION_SHADOW_OFFSET_Y, 2);
```

Operation selection effect

Bad blur effect under shadow of chrome

Style name	Reference value
SELECTION_BORDER	Default value: 1 Border thickness, number type
SELECTION_COLOR	Default value is rgba(0,34,255, 0.80). The color can be set in the following format: #2898E0, rgba(22,33,240,0.5)
SELECTION_SHADOW_BLUR	Default value: 7
SELECTION_SHADOW_OFFSET_X	Default value: 2 Selected shadow x offset
SELECTION_SHADOW_OFFSET_Y	Default value: 2 Selected shadow y offset

SHADOW related styles

Style name	Reference value
SHADOW_BLUR	Element shadow blur distance
SHADOW_COLOR	Element shadow color
SHADOW_OFFSET_X	Element shadow x offset
SHADOW_OFFSET_Y	Element shadow y offset

example

```
var nodeWithScale = graph.createNode("Node with Scale\nand Render Color", 0, 110);
nodeWithScale.size = {width: 100, height: -1};
nodeWithScale.rotate = -Math.PI / 6;
nodeWithScale.setStyle(Q.Styles.RENDER_COLOR, "#E21667");
nodeWithScale.setStyle(Q.Styles.SHADOW_COLOR, "#888");
nodeWithScale.setStyle(Q.Styles.SHADOW_OFFSET_X, 5);
nodeWithScale.setStyle(Q.Styles.SHADOW_OFFSET_Y, 5);
nodeWithScale.setStyle(Q.Styles.SHADOW_BLUR, 5);
```

result

SHAPE related styles

Style name	Reference value
SHAPE_FILL_COLOR	Color can be set in the following format: #2898E0, rgba(22,33,240,0.5)
SHAPE_FILL_GRADIENT	Gradient can use the example of Q.Gradient. For instance, new Gradient(Q.Consts.GRADIENT_TYPE_LINEAR, [Q.toColor(0x8AFFFFFF), Q.toColor(0x44CCCCCC)], [0.1, 0.9])
SHAPE_LINE_DASH	Dotted line style uses array format, such as [2, 5]
SHAPE_LINE_DASH_OFFSET	Offset of dotted line, value type, dynamic modification of this property would realize the flow effect of dotted line
SHAPE_STROKE	Default value: 1 Edge thickness, value type
SHAPE_STROKE_STYLE	Edge style can be set in certain color, such as #2898E0
SHAPE_OUTLINE	Default value: 0 Outer border width, value type, default to undefined
SHAPE_OUTLINE_STYLE	Outer border style can be set in certain color, such as #2898E0, default to undefined

FAQ

After it is upgraded to v1.4, some problems exist to word alignment

Versions higher than v1.4 supports to take the text as the node main part. Now the anchor position of text will be set directly via the property of Node#anchorPosition, instead of setStyle

```
text.setStyle(Q.Styles.LABEL_ANCHOR_POSITION, Q.Position.LEFT_BOTTOM);
```

revised to

```
text.anchorPosition = Q.Position.LEFT_BOTTOM;
```

Why the icon of SVG can't be loaded under Firefox?

The loading of SVG picture under each browser has difference. In SVG file, certain width-height value must be assigned (such as:<svg width="100" height="100" ...>), which can't be a percentage. Otherwise Qunee will be unable to acquire the original width-height rate. Surely this will generally not be a problem, since SVG, as a drawing tool(such as AI), will be set with corresponding width and height. But if hand-written SVG file, you have to pay attention on this problem

Why the icon of SVG can't be dyed under IE?

In IE browser, you can draw SVG images in Canvas, but unable to acquire the pixel information of SVG images, and thus unable to dye the images, it is the same for seeking node edge. So Qunee recommend to use after SVG file is converted to the code of Canvas

Why the current subnet can't be set?

In Qunee, any element can be set as a subnet, on preconditions that the property enableSubNetwork for modification of element shall be set to true. Otherwise, it can't be set. For example

```
var subnetwork = graph.createNode('SubNetwork');
subnetwork.enableSubNetwork = true;
graph.currentSubNetwork = subnetwork;
```

How to determine whether the Node type Element?

Javascript support instanceof keyword, you can do this type of judgment keyword

```
element instanceof Q.Node
```

the same as edge type

```
element instanceof Q.Edge
```

How to determine whether the node subnet type?

All element types can be set to a subnet, only you need to set enableSubNetwork property is true, to determine whether the node is the subnet type can also be judged by this element

```
if(element.enableSubNetwork){
 Q.log('is SubNetwork');
}
```

