JFlot – Java plugin for Flot

Flot is a pure Javascript plotting library for [jQuery](http://jquery.com). It produces graphical **plots** of arbitrary datasets on-the-fly client-side. JFlot is a java plugin for Flot. It generates JSONs needed for flot. The generated JSONs can be used in any J2EE view envirornments like jsps, html, struts mvc and spring mvc.

What's with the name?

First: it's pronounced with a short o, like "plot". Not like "flawed".So "Flot" rhymes with "plot".

And if you look up "flot" in a Danish-to-English dictionary, some up the words that come up are "good-looking", "attractive", "stylish","smart", "impressive", "extravagant". One of the main goals with Flot is pretty looks.

The construction of flot graph in java script is done by calling **function Plot(placeholder, data\_, options\_, plugins)**

Ex:

var plot = $.plot(placeholder, data , options);

Create a placeholder div to put the graph in:

**<div id="placeholder"></div>**

You need to set the width and height of this div, otherwise the plotlibrary doesn't know how to scale the graph. You can do it inline like this:

**<div id="placeholder" style="width:600px;height:300px"></div>**

Then when the div is ready in the DOM, which is usually on document ready, run the plot function: **$.plot($("#placeholder"), data, options);**

For generating flot graph, the main parameters are data, options and place holder.

JFlot is used to generate placeholder, data and options through java. FlotChart is an abstract class containing placholder, data (flotSeriesBeans) and options (flotGrid,flotSeries, flotLegend, flotXAxisOptions,flotYAxisOptions, flotColors, flotZoom, flotPan).

**protected** List<? **extends** FlotSeriesBean> flotSeriesBeans;

**private** FlotGrid flotGrid = **new** FlotGrid();

**private** FlotSeries flotSeries;

**private** FlotLegend flotLegend = **new** FlotLegend();

**private** FlotAxisOptions flotXAxisOptions = **new** FlotAxisOptions("xaxis");

**private** FlotAxisOptions flotYAxisOptions = **new** FlotAxisOptions("yaxis");

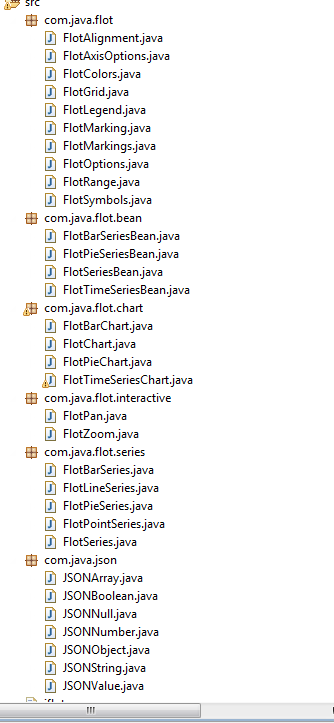
**private** FlotColors flotColors = **new** FlotColors();

**private** FlotZoom flotZoom;

**private** FlotPan flotPan;

**private** String idOfFlotGraph = "placeholder";

Class hierarchy of JFlot :



All flot options like FlotGrid, FlotLegend, FlotAxisOptions, are sub classes of FlotOptions.

FlotOptions is a sub class of JSONObject.

FlotOptions are listed below :

options = {

// the color theme used for graphs

colors: ["#edc240", "#afd8f8", "#cb4b4b", "#4da74d", "#9440ed"],

legend: {

show: true,

noColumns: 1, // number of colums in legend table

labelFormatter: null, // fn: string -> string

labelBoxBorderColor: "#ccc", // border color for the little label boxes

container: null, // container (as jQuery object) to put legend in, null means default on top of graph

position: "ne", // position of default legend container within plot

margin: 5, // distance from grid edge to default legend container within plot

backgroundColor: null, // null means auto-detect

backgroundOpacity: 0.85 // set to 0 to avoid background

},

xaxis: {

show: null, // null = auto-detect, true = always, false = never

position: "bottom", // or "top"

mode: null, // null or "time"

color: null, // base color, labels, ticks

tickColor: null, // possibly different color of ticks, e.g. "rgba(0,0,0,0.15)"

transform: null, // null or f: number -> number to transform axis

inverseTransform: null, // if transform is set, this should be the inverse function

min: null, // min. value to show, null means set automatically

max: null, // max. value to show, null means set automatically

autoscaleMargin: null, // margin in % to add if auto-setting min/max

ticks: null, // either [1, 3] or [[1, "a"], 3] or (fn: axis info -> ticks) or app. number of ticks for auto-ticks

tickFormatter: null, // fn: number -> string

labelWidth: null, // size of tick labels in pixels

labelHeight: null,

reserveSpace: null, // whether to reserve space even if axis isn't shown

tickLength: null, // size in pixels of ticks, or "full" for whole line

alignTicksWithAxis: null, // axis number or null for no sync

// mode specific options

tickDecimals: null, // no. of decimals, null means auto

tickSize: null, // number or [number, "unit"]

minTickSize: null, // number or [number, "unit"]

monthNames: null, // list of names of months

timeformat: null, // format string to use

twelveHourClock: false // 12 or 24 time in time mode

},

yaxis: {

autoscaleMargin: 0.02,

position: "left" // or "right"

},

xaxes: [],

yaxes: [],

series: {

points: {

show: false,

radius: 3,

lineWidth: 2, // in pixels

fill: true,

fillColor: "#ffffff",

symbol: "circle" // or callback

},

lines: {

// we don't put in show: false so we can see

// whether lines were actively disabled

lineWidth: 2, // in pixels

fill: false,

fillColor: null,

steps: false

},

bars: {

show: false,

lineWidth: 2, // in pixels

barWidth: 1, // in units of the x axis

fill: true,

fillColor: null,

align: "left", // or "center"

horizontal: false

},

shadowSize: 3

},

grid: {

show: true,

aboveData: false,

color: "#545454", // primary color used for outline and labels

backgroundColor: null, // null for transparent, else color

borderColor: null, // set if different from the grid color

tickColor: null, // color for the ticks, e.g. "rgba(0,0,0,0.15)"

labelMargin: 5, // in pixels

axisMargin: 8, // in pixels

borderWidth: 2, // in pixels

minBorderMargin: null, // in pixels, null means taken from points radius

markings: null, // array of ranges or fn: axes -> array of ranges

markingsColor: "#f4f4f4",

markingsLineWidth: 2,

// interactive stuff

clickable: false,

hoverable: false,

autoHighlight: true, // highlight in case mouse is near

mouseActiveRadius: 10 // how far the mouse can be away to activate an item

},

com.java.flot:

FlotOptions.java is super of all Flot option classes.

**JSONObject**

**FlotOptions**

**FlotLegend**

**FlotRange**

**FlotMarking**

**FlotGrid**

**FlotAxisOptions**

**FlotZoom**

**FlotPan**

**FlotChart**

**FlotBarSeries**

**FlotPieSeries**

**FlotPointSeries**

**FlotLineSeries**

com.java.flot.bean

**FlotTimeSeriesBean**

**FlotBarSeriesBean**

**FlotSeriesBean**

**FlotPieSeriesBean**

com.java.flot.chart

**FlotOptions**

**FlotOptions**

**FlotChart**

**FlotPieChart**

**FlotTimeSeriesChart**

**FlotBarChart**

com.java.json:

This package contains json related classes. JSONArray, JSONObject, JSONNull, JSONNumber, JSONString are sub classes of JSONValue.

**JSON** (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the [JavaScript Programming Language](http://javascript.crockford.com/), [Standard ECMA-262 3rd Edition - December 1999](http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf). JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

* A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
* An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

An *object* is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).



An *array* is an ordered collection of values. An array begins with [ (left bracket) and ends with ] (right bracket). Values are separated by , (comma).



A *value* can be a *string* in double quotes, or a *number*, or true or false or null, or an *object* or an *array*. These structures can be nested.



A *string* is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.



A *number* is very much like a C or Java number, except that the octal and hexadecimal formats are not used.



Whitespace can be inserted between any pair of tokens. Excepting a few encoding details, that completely describes the language.