dc.js

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

 https://www.facebook.com/tutorialspointindia  https://twitter.com/tutorialspoint

About the Tutorial

DC.js is a charting library built on top of D3.js and works natively with crossfilter, which is another popular JavaScript library used to explore millions of records in a short period on the client side. DC.js is a JavaScript library used to make interactive dashboards in JavaScript. This tutorial will give you a complete knowledge on the DC.js framework.

This is an introductory tutorial, which covers the basics of DC.js and explains how to deal with its various modules and sub-modules.

Audience

This tutorial is prepared for professionals who are aspiring to make a career in online data visualization. This tutorial is intended to make you comfortable in getting started with the DC.js framework and its various components.

Prerequisites

Before proceeding with the various types of concepts given in this tutorial, it is being assumed that the readers are already aware about what a Framework is.

In addition to this, it will be very helpful, if the readers have a sound knowledge on HTML, CSS, JavaScript and D3.js.

Copyright and Disclaimer

Table of Contents

1.DC.JS – INTRODUCTION

DC.js is an excellent JavaScript library for data analysis in the browser, mobile devices and ultimately helps in creating data visualization. Data visualization is the presentation of data in a pictorial or graphical format. The primary goal of data visualization is to communicate information clearly and efficiently via statistical graphics, plots and information graphics. Data visualizations can be developed and integrated in regular web and even mobile applications using different JavaScript frameworks.

What is DC.js?

DC.js is a charting library for exploring large multi-dimensional datasets. It relies on the D3.js engine to render charts in a CSS-friendly SVG format. It allows complex data visualization to be rendered and has a designed dashboard having Bar Charts, Scatter Plots, Heat Maps, etc. DC.js is built to work with **Crossfilter** for data manipulation. DC.js enables a single (large) dataset to be visualized with many interconnected charts with an advanced auto-filtering option.

Why Do We Need DC.js?

In general, data visualization is quite a complex process and carrying it out on the client side requires extra skill. DC.js enables us to create almost any kind of complex data visualization using a simpler programming model. It is an open source, extremely easy-to-pick-up JavaScript library, which allows us to implement neat custom visualizations in a very short time.

DC.js charts are data driven and very reactive. In addition, it delivers instant feedback to user interaction using the **Crossfilter Library**.

DC.js Features

DC.js is one of the best data visualization framework and it can be used to generate simple as well as complex visualizations. Some of the salient features are listed below:

- Extremely flexible.
- Easy to use.
- Fast rendering of the charts
- Supports large multi-dimensional datasets.

- Open source JavaScript library.

Dc.js Benefits

DC.js is an open source project and it requires lesser code when compared to others. It comes with the following benefits:

- Great data visualization.
- Performs graphical filtering.
- Fast creation of charts and dashboards.
- Creation of highly interactive dashboards.

In the next chapter, we will understand how to install D3.js on our system.

2.DC.JS – INSTALLATION

In this chapter, we will learn how to setup the DC.js development environment. Before we start, we need the following components:

- DC.js library
- Editor
- Web browser
- Web server

Let us go through the steps one by one in detail.

DC.js Installation

DC installation is very easy to set up. Follow the below steps to install *DC* on your machine.

Download DC Library

DC is an open-source library; use the link https://github.com/dc-js/dc.js/releases to download the file.

Download here

Download the latest version of the DC file. (As of now, the latest version is 2.0.2.). After the download is completed, unzip the DC folder and paste it to your project's root folder or any other folder, where you want to keep all your library files.

The sample HTML page is as shown below.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <script src="js/d3.js"></script>
    <script src="js/crossfilter.js"></script>
    <script src="js/dc.js"></script>
</head>
<body>
```

```
<script>
    // write your dc code here..
</script>
</body>
</html>
```

DC is a JavaScript code, so we have to write all the DC codes within the "script" tag. We may need to manipulate the existing DOM elements, hence it is advisable to write the DC code just before the end of the "body" tag.

DC.js Editor

We will need an editor to start writing the code. There are some great IDEs (Integrated Development Environment) with support for JavaScript such as –

- Visual Studio Code
- WebStorm
- Eclipse
- SublimeText

These IDEs provide intelligent code completion as well as support some of the modern JavaScript frameworks. If we do not have any fancy IDE, we can always use a basic editor such as Notepad, VI, etc.

Web Browser

DC.js works on all browsers except IE8 and lower.

Web Server

Most browsers serve local HTML files directly from the local file system. However, there are certain restrictions when it comes to loading external data files. In the subsequent chapters of this tutorial, we will be loading data from external files such as CSV and JSON. Therefore, it will be easier for us, if we set up the web server right from the beginning.

We can use any web server, which we are comfortable with. For example – IIS, Apache, etc.

Viewing a Page

In most cases, we can just open the HTML file in a web browser to view it. However, when loading external data sources, it is more reliable to run a local webserver and view the page from the server (http://localhost:8080).

3.DC.JS – CONCEPTS

DC.js is simple and easy for most front-end developers. It enables building basic charts quickly, even without any knowledge of D3.js. Before, we start using DC.js to create visualization; we need to get familiar with web standards. The following web standards are heavily used in D3.js, which is the foundation of DC.js for rendering charts.

- Hypertext Markup Language (HTML)
- Document Object Model (DOM)
- Cascading Style Sheets (CSS)

Let us understand each of these web standards in detail.

Hypertext Markup Language (HTML)

As we know, HTML is used to structure the content of the webpage. It is stored in a text file with the extension ".html".

A typical basic HTML example looks like as shown below:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>

</body>
</html>
```

Document Object Model (DOM)

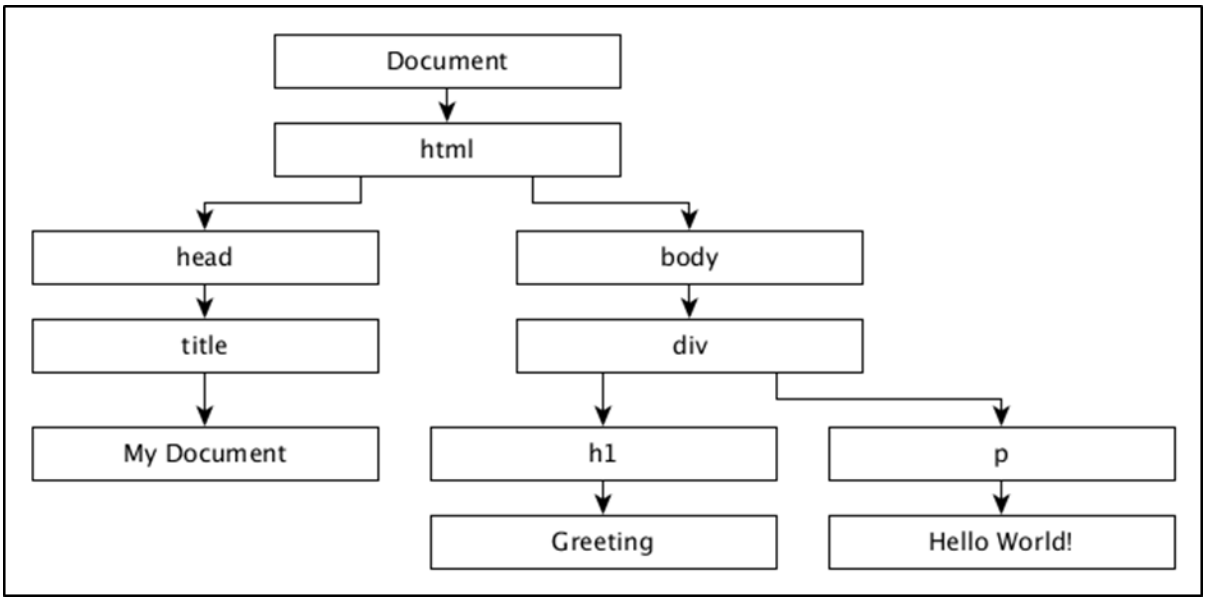
When a HTML page is loaded by a browser, it is converted to a hierarchical structure. Every tag in HTML is converted to an element / object in the DOM with a parent-child hierarchy. It

makes our HTML more logically structured. Once the DOM is formed, it becomes easier to manipulate (add/modify/remove) the elements on the page.

Let us understand the DOM using the following HTML document:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>My Document</title>
    </head>
    <body>
        <div>
            <h1>Greeting</h1>
            <p>Hello World!</p>
        </div>
    </body>
</html>
```

The document object model of the above HTML document is as follows:



Cascading Style Sheets (CSS)

While HTML gives a structure to the webpage, CSS styles make the webpage more pleasant to look at. CSS is a style sheet language used to describe the presentation of a document written in HTML or XML (including XML dialects such as SVG or XHTML). CSS describes how elements should be rendered on a webpage.

JavaScript

JavaScript is a loosely typed client side scripting language that executes in the user's browser. JavaScript interacts with html elements (DOM elements) in order to make the web user interface interactive. JavaScript implements the ECMAScript standards, which includes core features based on ECMA-262 specification as well as other features, which are not based on ECMAScript standards. JavaScript knowledge is a prerequisite for DC.js.

Components

DC.js is based on two excellent JavaScript libraries, which are –

- Crossfilter
- D3.js

Crossfilter

Crossfilter is a JavaScript library for exploring large multivariate datasets in the browser. It is used for Grouping, Filtering, and Aggregating tens or hundreds of thousands of rows of raw data very quickly.

D3.js

D3.js stands for Data-Driven Documents. D3.js is a JavaScript library for manipulating documents based on data. D3 is Dynamic, Interactive, Online Data Visualizations Framework and used in large number of websites. D3.js is written by **Mike Bostock**, created as a successor to an earlier visualization toolkit called **Protovis**. D3.js is used on hundreds of thousands of websites.

4.INTRODUCTION TO CROSSFILTER

Crossfilter is a multi-dimensional dataset. It supports extremely fast interaction with datasets containing a million or more records.

Basic Concepts

Crossfilter is defined under the crossfilter namespace. It uses semantic versioning. Consider a crossfilter object loaded with a collection of fruits that is defined below:

```
var fruits = crossfilter([

  { name: “Apple”, type: “fruit”, count: 20 },
  { name: “Orange”, type: "fruit”, count: 10 },
  { name: “Grapes”, type: “fruit”, count: 50 },
  { name: “Mango”,  type: “fruit”, count: 40 }
]);
```

If we need to perform the total records in a group, we can use the following function:

```
var count = fruits.groupAll().reduceCount().value();
```

If we want to filter by a specific type:

```
var filtering = fruits.dimension(function(d) { return d.type; });
filtering.filter(“Grapes”)
```

Similarly, we can perform grouping with Crossfilter. To do this, we can use the following function:

```
var grouping = filtering.group().reduceCount();
var first = grouping.top(2);
```

Hence, Crossfilter is built to be extremely fast. If you want to recalculate groups as filters are applied, it calculates incrementally. Crossfilter dimensions are very expensive.

Crossfilter API

Let us go through the notable Crossfilter APIs in detail.

- **crossfilter([records]):** It is used to construct a new crossfilter. If the record is specified, then it simultaneously adds the specified records. Records can be any array of JavaScript objects or primitives.
- **crossfilter.add(records):** Adds the specified records to the crossfilter.
- **crossfilter.remove():** Removes all records that match the current filters from the crossfilter.
- **crossfilter.size():** Returns the number of records in the crossfilter.
- **crossfilter.groupAll():** It is a function for grouping all records and reducing to a single value.
- **crossfilter.dimension(value):** It is used to construct a new dimension using the specified value accessor function.
- **dimension.filter(value):** It is used to filter records for dimension's match value, and returns the dimension.
- **dimension.filterRange(range):** Filters records for dimension's value that are greater than or equal to range[0], and less than range[1].
- **dimension.filterAll():** Clears any filters on this dimension.
- **dimension.top(k):** It is used to return a new array containing the top k records, according to the natural order of this dimension.
- **dimension.bottom(k):** It is used to return a new array containing the bottom k records, according to the natural order of this dimension.
- **dimension.dispose():** It is used to remove the dimension from the crossfilter.

In the next chapter, we will understand in brief about D3.js.

5.INTRODUCTION TO D3.JS

D3.js is a JavaScript library used to create interactive visualizations in the browser. The D3 library allows us to manipulate elements of a webpage in the context of a dataset. These elements can be HTML, SVG, or Canvas elements, and can be introduced, removed, or edited according to the contents of the dataset. It is a library for manipulating DOM objects. D3.js can be a valuable aid in data exploration. It gives you control over your data's representation and lets you add data interactivity.

D3.js is one of the premier framework when compared to other libraries. This is because; it works on the web and data visualizations and is of enterprise grade. Another reason is its great flexibility, which enables developers around the world to create many advanced charts. Also, it has extended its functionality to a great extent.

Let us understand the basic concepts of D3.js, which are as follows:

- Selections
- Data join
- SVG
- Transition
- Animation
- D3.js API

Let us understand each of these concepts in detail.

Selections

Selections is one of the core concept in D3.js. It is based on the CSS Selector concept. Those who have used and are aware of JQuery already can easily understand the selections. It enables us to select the DOM based on CSS selectors and then provide options to modify or append and remove the elements of DOM.

Data Join

Data join is another important concept in D3.js. It works along with selections and enables us to manipulate the HTML document with respect to our dataset (a series of numerical values). By default, D3.js gives dataset the highest priority in its methods and each item in the dataset corresponds to a HTML element.

SVG

SVG stands for **Scalable Vector Graphics**. SVG is an XML based vector graphics format. It provides options to draw different shapes such as Lines, Rectangles, Circles, Ellipses, etc. Hence, designing visualizations with SVG gives you more power and flexibility.

Transformation

SVG provides options to transform a single SVG shape element or group of SVG elements. SVG transform supports Translate, Scale, Rotate and Skew.

Transition

Transition is the process of changing from one state to another of an item. D3.js provides a transition() method to perform transition in the HTML page.

Animation

D3.js supports animation through transition. Animation can be done with the proper use of transition. Transitions are a limited form of key frame animation with only two key frames: **start** and **end**. The starting key frame is typically the current state of the DOM, and the ending key frame is a set of attributes, styles and other properties you specify. Transitions are well suited for transitioning to a new view without a complicated code that depends on the starting view.

D3.js API

Let us understand some of the important D3.js API's methods in brief.

Collections API

A collection is simply an object that groups multiple elements into a single unit. It is also called as a container. It contains Objects, Maps, Sets and Nests.

Paths API

Paths are used to draw rectangles, circles, ellipses, polylines, polygons, straight lines and curves. SVG Paths represent the outline of a shape that can be stroked, filled, used as a clipping path, or any combination of all three.

Axis API

D3.js provides functions to draw axes. An axis is made of lines, ticks and labels. An axis uses scale, thus each axis will need to be given a scale to work with.

Zooming API

Zooming helps to scale your content. You can focus on a particular region using the click-and-drag approach.

Delimiter-Separated Values API

A delimiter is a sequence of one or more characters used to specify the boundary between separate, independent regions in plain text or other data. A field delimiter is a sequence of comma-separated values. In short, the delimiter-separated values are Comma-Separated Values (CSV) or Tab-Separated Values (TSV).

6.DC.JS – MIXINS

Mixin is an **abstract functional object** having a lot of pre-defined functions, which can be mixed and used in JavaScript classes. However, they cannot be used as stand-alone. For example, DC.js has a **Mixin** and **dc.baseMixin**, which cannot be used as it is, but is used by all the *DC* chart classes such as the Line Chart, Pie Chart, etc. DC.js has a limited set of useful Mixins to create charts easily and efficiently. They are as follows –

- **baseMixin:** baseMixin provides common functionality for all type of charts. It integrates crossfilter and d3.js JavaScript library and provides a simple set of function to create charts with limited knowledge of D3.js.
- **capMixin:** capMixin provides grouping support for the data elements below a limit (cap).
- **colorMixin**: colorMixin provides color support for the charts.
- **marginMixin:** marginMixin provides margin support for the charts.
- **coordinateGridMixin:** coordinateGridMixin provides coordinate support for the charts.
- **stackMixin:** stackMixin provides stacking support using the **d3.layout.stack**.
- **bubbleMixin:** bubbleMixin provides bubble support for the charts.

Let us understand all these mixins in detail in the subsequent chapters.

7.DC.JS – BASEMIXIN

baseMixin provides basic methods needed to create any type of a chart. It ranges from setting the width of the chart to advanced filtering of the chart.

General Chart Options

The **basicMixin** provides many chart methods to get / set the properties of the charts. They are as follows,

- **chartID() –** Returns the internal numeric ID of the chart.
- **chartGroup([chartGroup]) –** Gets or sets the group to which the chart belongs. In DC.js, charts can be grouped into a single set. All charts in a group are expected to share the same Crossfilter dataset. They are rendered as well as redrawn simultaneously.

```
mychart.chartGroup('dashboard');
```

- **minWidth([minWidth]) –** Sets the minimum width of the chart.

```
mychart.minWidth(300);
```

- **width([width]) –** Gets or sets the width of the chart.

```
mychart.width(600);
```

- **minHeight([minHeight]) –** Gets or sets the minimum height of the chart.

```
mychart.minHeight(300);
```

- **height([height]) –** Gets or sets the height of the chart.

```
mychart.height(300);
```

- **title([titleFunction]) –** Gets or sets the title function. Title is the SVG Element's title of the child element in the chart (e.g. a single bar in a bar chart). Title in the charts are represented as tooltip in the browser.

```
mychart.title(function(data) {
```

```
    return d.key + ': ' + d.value;
});
```

- **label(labelFunction[??]) –** Similar to the title() method, but it sets the label instead of the title.

```
mychart.label(function(data) {
    return d.key + ': ' + d.value;
});
```

- **options(opts) –** Sets any chart option using the JavaScript object. Each key represents the corresponding method available in the charts and the matched method will be invoked with the relevant value.

```
mychart.options({
    'width' : 300,
    'height' : 300
});
```

Here, width() and height() method will be fired with the specified value.

- **legend([legend]) –** Attaches a legend to the chart. The legend can be created using the **d3.legend()** method.

```
mychart.legend(
    dc.legend()
        .x(500)
        .y(50)
        .itemHeight(12)
        .gap(4))
```

- **anchor(parent[??]) –** Sets the root SVGElement to be either an existing chart's root or any valid D3 single selectors. Optionally, the chart group can also be set using the second argument.
- **anchorName() –** Gets the DOM ID of the chart's anchored location.
- **svg([svgElement]) –** Returns the SVGElement of the chart.
- **resetSvg() –** Resets the SVG container in the DOM.

- **root([rootElement]) –** Gets the root container of the chart.

Data Options

basicMixin provides methods to set the data for the charts. The data is set as Crossfilter dimension and group. In addition, it provides an option to get the underlying dataset.

- **dimension([dimension]) –** Sets or gets the dimension of the chart. A dimension is any valid Crossfilter dimension.

```
var mycrossfilter = crossfilter([]);
var ageDimension = mycrossfilter.dimension(dc.pluck('age'));
mychart.dimension(ageDimension);
```

- **group(group[??]) –** Sets or gets the group of the chart. A group is any valid Crossfilter group. The group can be named using the second argument to use it later in the code.

```
var mycrossfilter = crossfilter([]);
var ageDimension = mycrossfilter.dimension(dc.pluck('age'));
mychart.dimension(ageDimension);
mychart.group(ageDimension.group(crossfilter.reduceCount()));
```

- **data([callback]) –** Sets the data callback and enables us to get the underlying chart's data set.

```
// get all groups
mychart.data(function (group) {
    return group.all();
});

// get top five groups
mychart.data(function (group) {
    return group.top(5);
});
```

- **keyAccessor([keyAccessor]) –** Gets or sets the key accessor function. It is used to retrieve the key from the underlying Crossfilter group. The key is used for slices in

a pie chart and x-axis in the line / bar chart. The default **key accessor** function is as follows:

```
chart.keyAccessor(function(d) { return d.key; });
```

- **valueAccessor([valueAccessor]) –** Gets or sets the value accessor function. It is used to retrieve the value from the underlying Crossfilter group. The value is used for slice size in the pie chart and y-axis position in the line / bar chart. The default **value accessor** function is as follows:

```
chart.valueAccessor(function(d) { return d.value; });
```

- **ordering([orderFunction]) –** Gets or sets an ordering function to order ordinal dimension. By default, a chart uses **crossfilter.quicksort.by** to sort the elements.

```
_chart.ordering(dc.pluck('key'));
```

Filter Options

Filtering is one of the highlights of DC.js. We can apply one or more filters directly on the chart object using the filter() method and call chart's redrawGroup() or dc.redrawAll() method to see the filtering effect on the chart. By default, a chart object takes one or more filters using the filter() method, applies it on the underlying Crossfilter() data set, gets the filtered data from the Crossfilter and redraws the charts using the filtered data. DC.js provides the following methods to handle filtering in the chart.

Filter([filter])

Gets or sets the filter for the chart. If a supplied filter is new, then it will be added to the chart's filter collection and applied on the underlying dataset. If the filter supplied is already available in the chart's filter collection, then it will remove the filter and do the relevant filtering on the underlying data. In short, filter method will toggle the supplied filters.

```
mychart.filter(10);
```

To remove all filters, call the filter method with **null** value. The filter may be any one of the following items:

- **null –** Chart will remove all the filters previously applied.
- **single value –** Chart will call the underlying Crossfilter's filter method and send the supplied value.

- **dc.filters.RangedFilter –** It accepts two values, low and high. Chart will filter out all the data, except the value in the range between low and high value.
- **dc.filters.TwoDimensionalFilter –** It accepts two-dimensional values that are used in the heat map.
- **dc.filters.RangedTwoDimensionalFilter –** It is similar to the dc.filters.RangedFilter, except that it accepts a two-dimensional value only used in scatter plots.

hasFilter([filter])

Checks whether the supplied filter is available or not in the chart.

replaceFilter([filter])

Replaces the current filter of the chart with the supplied filter.

filters()

Returns all current filters associated with the chart.

filterAll()

Clears all filters associated with the chart.

filterHandler([filterHandler])

Gets or sets a filter handler function. Filter handler function is used by the chart to filter the underlying dataset using the filter. Chart has a Default Filter Handler Function and it can be replaced by a Custom Filter Handler Function using this method. The default filter handler is as follows:

```
 chart.filterHandler(function (dimension, filters) {
        if (filters.length === 0) {
                // the empty case (no filtering)
                dimension.filter(null);
        } else if (filters.length === 1 && !filters[0].isFiltered) {
                // single value and not a function-based filter
                dimension.filterExact(filters[0]);
        } else if (filters.length === 1 && filters[0].filterType ===
'RangedFilter') {
                // single range-based filter
                dimension.filterRange(filters[0]);
```

```
        } else {
                // an array of values, or an array of filter objects
                dimension.filterFunction(function (d) {
                        for (var i = 0; i < filters.length; i++) {
                                var filter = filters[i];
                                if (filter.isFiltered && filter.isFiltered(d)) {
                                        return true;
                                } else if (filter <= d && filter >= d) {
                                        return true;
                                }
                        }
                        return false;
                });
        }
        return filters;
});
```

hasFilterHandler([hasFilterHandler])

Gets or sets a has-filter handler function. This function is used by the chart to check whether a filter is available in the chart's filter collection or not. The default has-filter handler is as follows:

```
chart.hasFilterHandler(function (filters, filter) {
        if (filter === null || typeof(filter) === 'undefined') {
                return filters.length > 0;
        }
        return filters.some(function (f) {
                return filter <= f && filter >= f;
        });
});
```

addFilterHandler([addFilterHandler])

Gets or sets the add-filter handler function. This function is used by the chart to add the filter into the chart's filter collection. The default add-filter handler is as follows:

```
chart.addFilterHandler(function (filters, filter) {
```

```
        filters.push(filter);
        return filters;
});
```

removeFilterHandler([removeFilterHandler])

Gets or sets the remove-filter handler function. This function is used by the chart to remove the filter from the chart's filter collection. The default remove-filter is as follows:

```
chart.removeFilterHandler(function (filters, filter) {
        for (var i = 0; i < filters.length; i++) {
                if (filters[i] <= filter && filters[i] >= filter) {
                        filters.splice(i, 1);
                        break;
                }
        }
        return filters;
});
```

resetFilterHandler([resetFilterHandler])

Gets or sets the reset-filter handler function. This function is used by the chart to reset the chart's filter collection. The default reset-filter is as follows:

```
function (filters) {
        return [];
}
```

filterPrinter([filterPrinterFunction])

Gets or sets the printer-filter function. This function is used by the chart to print the filter information.

commitHandler()

Gets or sets the commit handler. The purpose of the commit handler is to send the filtered data to the server asynchronously.

Event Options

DC.js defines a limited set of events to do some functionalities such as Filtering, Zooming, etc. The list of events defined in the DC.js are as follows:

- **renderlet -** Fired after transitions are redrawn and rendered.
- **pretransition -** Fired before the transitions start.
- **preRender -** Fired before the chart rendering.
- **postRender -** Fired after the chart finishes rendering including all the renderlet's logic.
- **preRedraw -** Fired before chart redrawing.
- **postRedraw -** Fired after the chart finishes redrawing including all the renderlet's logic.
- **filtered -** Fired after a filter is applied, added or removed.
- **zoomed -** Fired after a zoom is triggered.

basicMixin provides a method, **on(event, listener)** to set the callback function for all the above defined events.

- **on(event, listener) -** Sets the callback or listener function for the specific event.
- **onClick(datum) -** It is passed to **D3** as the **onClick** handler for each chart. The default behavior is to filter on the clicked datum (passed to the callback) and redraw the chart group.

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint.com**