ng-info-chart: Postprint of Interactive Visualization Components Based on Custom HTML Tags

Authors: Chen Ting, Wang Xiaomei, Lü Weimin

Date: 2017-10-11T00:00:00+00:00

Abstract

Abstract

Objective: To design and implement the ng-info-chart visualization component based on the Model-View-Controller (MVC) front-end AngularJS framework.

Background: Sophisticated intelligence analysis platforms often require combining multiple complex visualization charts to display analytical results, necessitating more effective methods for constructing web-based intelligence analysis visualization charts that support extensive interactive operations.

Method: The ng-info-chart integrates multiple visualization charts by uniformly encapsulating them through AngularJS custom directive tags, enabling direct invocation of rendering methods in web pages via custom HTML tags.

Results: As the research team's intelligence analysis projects have matured, the ng-info-chart visualization component has integrated 11 types of visualization charts from 5 third-party visualization libraries, supporting mainstream desktop browsers including IE9+ and Firefox.

Conclusion: By leveraging this visualization component to implement functionalities such as asynchronous data acquisition, automatic detection of data changes, and real-time chart rendering, the development of complex visualization charts in intelligence analysis systems has been significantly simplified.

Full Text

ng-info-chart: An Interactive Visualization Component Based on Custom HTML Tags

Chen Ting[1], **Wang Xiaomei**[1], **Lv Weimin**[1,2]
[1](National Science Library, Chinese Academy of Sciences, Beijing 100190,

China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract:
[**Objective**] This research designs and implements ng-info-chart, a front-end visualization component based on the MVC framework AngularJS. [**Context**] Effective intelligence analysis platforms often require multiple complex visualization charts to present analytical results, necessitating more efficient methods for constructing sophisticated, interactive web-based intelligence analysis visualizations. [**Methods**] The ng-info-chart component integrates multiple visualization charts through AngularJS custom directive tags, enabling direct invocation of drawing methods via custom HTML tags. [**Results**] The component has evolved through continuous development in the research team' s intelligence analysis projects and currently integrates 11 visualization chart types from 5 third-party libraries, supporting mainstream desktop browsers including IE9+ and Firefox. [**Conclusions**] The visualization component implements asynchronous data fetching, automatic detection of data changes, and real-time chart rendering, significantly simplifying the development of complex visualizations in intelligence analysis systems.

Keywords: Visualization; Front-end MVC; Directive

2.1 Visualization Requirements in Intelligence Analysis

Information visualization not only displays multidimensional non-spatial data through views to deepen users' understanding of data meaning but also uses intuitive graphical representations to guide service processes, thereby improving service efficiency and effectiveness [3]. Modern intelligence analysis increasingly demands more sophisticated visualization charts and higher operability. For instance, the Global Open Access Development platform requires its open access data map visualization to support direct data filtering, calculation, retrieval, and show/hide operations within the map itself. In the Major Science and Technology Trends Mapping project, multidimensional comparison charts must enable dynamic data filtering through timeline dragging and importance scoring, while allowing switching between statistical modes and coordinate axes based on different analytical requirements. Similarly, the Research Competitiveness Omnidirectional Evaluation System employs force-directed layouts to display papers and projects, allowing users to adjust clustering parameters, force coefficients, and filter analysis objects directly on the page, with the system generating final visualization results based on user interactions. Although numerous JavaScript-based visualization components are available, integrating multiple libraries presents significant development challenges and high maintenance costs. Developing a fully functional visualization analysis platform that supports extensive user interaction requires substantial human and material resources, creating an urgent need for a loosely coupled, modular development approach for visu-

alization charts. This study conducted comprehensive research on commonly used visualization libraries and front-end frameworks, leading to the design and initial implementation of the ng-info-chart visualization component library.

2.2 Common Web-based Visualization Libraries

GitHub' s programming language trend analysis demonstrates rapid growth in JavaScript usage for web development [4], while JavaScript-based front-end visualization libraries have also developed quickly, becoming the preferred choice for web visualization [5] and offering many excellent tools and charts. Meanwhile, technologies such as Flash, Silverlight, and Java Applet have gradually faded from developers' view. In web-based visualization development, multiple JavaScript tools and packages are available for different analytical needs. However, complex intelligence analysis platforms often require more than a single tool, necessitating the integration of multiple libraries. For highly specialized visualization needs that cannot be met by existing charts, 底层 customization is required. Table 1 compares commonly used visualization tools for intelligence analysis platform development, listing six tools and libraries that meet requirements ranging from simple statistical charts to advanced visualizations like force-directed graphs, time-series analysis, and geographic information maps. The third column shows statistics from a programmer Q&A website regarding visualization tool inquiries, revealing that Highcharts.js and D3.js have far more active communities than Echarts.js and Datav.js, leading us to prioritize these two libraries in our integration efforts.

2.3 Front-end Development Framework AngularJS

The conventional development pattern using jQuery with Ajax for asynchronous data updates followed by third-party visualization library invocation produces overly complex code that cannot meet the demands of sophisticated, interaction-rich intelligence analysis platforms. Our research required an efficient front-end visualization development approach. After investigating mainstream front-end MVC frameworks, we determined that despite the proliferation of such frameworks in recent years, AngularJS [6], released by Google, stands as the most influential. When selecting an open-source technology framework, community activity serves as a crucial evaluation criterion alongside technical merits. As shown in Figure 1 [Figure 1: see original paper], AngularJS far surpasses other frameworks in both developer attention and technical community activity, making it our framework of choice for developing the visualization component.

AngularJS is a pure client-side JavaScript framework developed and maintained by Google for extending and enhancing HTML functionality, simplifying development and testing complexity, and specifically designed for building robust web applications [8]. The framework provides numerous advanced features: MVC architecture, two-way data binding, routing, modular management, custom directives, dependency injection, and front-end HTML templates. Among these, AngularJS' s unique "custom directives" and "two-way data binding" capabilities

particularly suit the technical requirements of complex, interactive intelligence analysis charts, substantially simplifying Ajax data operations and visualization method invocations triggered by user interactions. Additionally, the framework' s "modular management" and "dependency injection" features provide advanced language characteristics that compensate for native JavaScript' s limitations in modular invocation.

(1) Custom Directives
Custom directives are AngularJS framework features that allow developers to extend HTML tags and bind dynamic data to them [9]. In visualization component design, different visualization libraries can be encapsulated into distinct HTML tags through custom directives, enabling direct invocation of visualization methods via custom tags in HTML pages.

(2) Two-Way Data Binding
Two-way data binding is perhaps one of AngularJS' s most important features that cannot be easily implemented with jQuery alone. It enables binding between properties in a private scope and DOM attribute values [10]. In typical web applications, most front-end code involves DOM element listening and data retrieval/update operations. Whether data is pre-saved in controllers, fetched from APIs, or modified by user operations, bound data automatically synchronizes across all application layers, including data invoked in custom tags, making it highly suitable for developing visualization charts with extensive interactions.

---

3. ng-info-chart Visualization Component Design and Implementation

3.1 Challenges in Complex Visualization Chart Development Developing interactive visualization charts using native JavaScript or jQuery typically involves multiple steps: Ajax data fetching, binding visualization libraries to div elements, rendering visualization data, re-binding interaction events (such as user clicks and drag operations) to Ajax data fetching, updating visualization data, removing existing charts, and re-rendering new visualizations. Complex charts often require calling multiple visualization datasets and libraries on the same page or binding visualization data to multiple user operations, leading to difficult front-end JavaScript development, chaotic management of external dependencies, and challenging code maintenance.

3.2 Visualization Component Design Addressing these challenges, the ng-info-chart visualization component leverages AngularJS' s unique "custom directives" and "two-way data binding" features to implement six key functions that simplify visualization chart development:

(1) Modular Encapsulation of Visualization Libraries: The component employs modular encapsulation using AngularJS' s module management capabilities, packaging third-party visualization methods into separate component

modules (AngularJS Modules). This allows developers to load only required modules when invoking components to draw charts, rather than loading all visualization components, thereby improving runtime performance.

(2) Dynamic External Resource Loading: When analysis systems require multiple visualization libraries, developers must otherwise load numerous library files in the page, where version and loading sequence issues frequently cause rendering errors. The visualization component moves library and resource loading from the page into unified component management, requiring developers to load only the component file infochart.js instead of managing multiple resources.

(3) Defining Chart "Custom Directives" : Rather than monitoring div element IDs to bind visualizations to page elements, the component encapsulates different visualization libraries into distinct "custom directives." For example, a force-directed graph developed with D3.js is encapsulated as the "ng-info-force" directive with defined visualization data inputs, enabling direct invocation through custom tags in HTML pages. This simplifies chart-to-element and data binding while improving code readability and maintainability.

(4) "Two-Way Data Binding" for Visualization Data: Visualization data is stored in the AngularJS Model layer. Whether users modify data or APIs update it, the Model layer automatically refreshes with the latest data, which the visualization component automatically inherits. Developers no longer need to repeatedly map updated data to visualization methods.

(5) Encapsulation of Common Chart Drawing Methods: Frequently used drawing methods from multiple intelligence analysis platform libraries and common tools like click and hover interactions are encapsulated within directives for convenient invocation.

(6) Automatic Data Change Detection and Chart Refresh: The component monitors data changes and automatically refreshes visualization charts when user operations modify data or data interfaces provide updates.

The component invocation flow is illustrated in Figure 2 [Figure 2: see original paper]. Visualization chart drawing methods are uniformly encapsulated through custom directives, with each directive including methods for calling external visualization library resources and monitoring data changes. Custom directives are packaged in different Angular modules according to their underlying third-party libraries and uniformly named after the visualization chart type. For example, the custom directive encapsulating the heatmap method from Highcharts.js is named ng-chart-heatmap. To invoke the chart drawing, developers first inject the Highcharts drawing module via dependency injection, then introduce the tag in the HTML page to render the heatmap.

3.3 Encapsulating Visualization Libraries in Custom Directives To systematically package multiple third-party visualization libraries within AngularJS custom directives, component development follows AngularJS framework

best practices [9]. The component structure is shown in Figure 3 [Figure 3: see original paper]. The bottom layer comprises multiple third-party visualization libraries, with commonly used chart methods from intelligence analysis platforms encapsulated in AngularJS Services (Service Factories). These can be injected into custom directives when needed. The directives uniformly define custom tag names and corresponding drawing data, and monitor data changes by calling the `$watch` command to monitor data updates and automatically refresh the visualization.

Figure 3 illustrates the ng-info-chart visualization component structure using the heatmap as an example, showing custom tag encapsulation JavaScript code.

(1) **Modular Encapsulation of Visualization Chart Methods:** Third-party visualization chart drawing methods are encapsulated into Factories within the AngularJS framework for invocation in custom tags. One module can encapsulate multiple Factories, and a Factory can contain multiple drawing methods. Methods from the same library can be encapsulated in a single Factory. The core Factory code is shown below:

```
/* Use Factory to encapsulate some visualization charts from Highchart */
var chart = angular.module("chart", []);

/* Heatmap chart */
chart.factory('highchart', function(){
    return {
        heatmap: function(container, data){
            /* Highchart drawing method here */
        },
        /* Encapsulate more Highchart drawing methods */
        barline: function(container, data){
            // ...
        }
    };
});
```

(2) **Encapsulating Custom Extended Tag :** The tag is encapsulated in the infoChart.highcharts module, introducing the Highchart Heatmap drawing method from the previously encapsulated Chart drawing module. Similarly, one module can contain multiple tags. The core code is as follows:

```
/* Define new module infoChart.highcharts, encapsulate custom extended tags */
var infoChartHighcharts = angular.module("infoChart.highcharts", ['highcharts']);

/* Define 'custom html tag' - infoChartsHeatmap, introduce the corresponding drawing method
infoChartHighcharts.directive('infoChartsHeatmap',
    function($window, highchart){
        return {
```

```
            restrict:'EA',
            template:"<div class='myCharts'></div>", // Define HTML container in tag
            scope:{
                chartData:'=' // Get data from tag attribute chartData
            },
            link: function(scope, elem, attrs){
                // Use $watch method to monitor data changes, automatically update chart if
                scope.$watch('chartData', function(nv){
                    var container=elem.find('div');
                    highchart.heatmap(container, nv);
                });
            }
        };
    }
);
```

https://docs.angularjs.org/guide/services.

3.4 ng-info-chart Visualization Component Invocation Compared with direct invocation of third-party visualization libraries, drawing charts through the ng-info-chart component is more straightforward to implement. Figure 4 [Figure 4: see original paper] illustrates the component invocation process using a Motion Chart as an example. The Google visualization module is called through AngularJS dependency injection (position 1 in Figure 4(a)); visualization data is fetched from the backend data interface via Ajax services and stored in the Model $scope.motion_{data}, as shown at label 2 in Figure 4(a). At this point, developers only need to use the custom tag ° in the HTML page to call the visualization component, as shown at label 3 in Figure 4(a), and associate the visualization data required by the component with the data stored in the Model through the tag attribute 'data=motion_{data}' . The component will automatically draw the multidimensional dynamic chart at the tag position in the HTML page, as shown in Figure 4(b). Afterward, whether the data interface updates or user interaction operations modify the visualization data, the visualization component will automatically monitor data changes and dynamically regenerate the visualization chart. Compared with traditional development patterns that call third-party visualization libraries, using ng-info-chart makes the development code more concise and readable, development more efficient, and especially easier for later maintenance.

Figure 4: Visualization Component Invocation Process Example

Wherein, label 1 in Figure 4(a) shows the introduction of the required drawing method module through dependency injection; label 2 shows calling Ajax asynchronous services in JavaScript to fetch drawing data and store it in the Model $scope.motion_{chart} (position 2). Position 3 shows the invocation of the visualization component in the HTML page through the custom HTML tag to load the visualization data from the Model. Through these three steps, the

component calls the underlying third-party visualization library to render the multidimensional comparison chart.

The ng-info-chart visualization component originated from practical requirements for complex visualization charts in previous intelligence analysis platforms. Since early 2014, it has integrated 11 visualization chart types from 5 third-party libraries (see Table 2), including basic analytical charts such as pie charts and bar charts, multidimensional comparison charts like bubble charts and Motion charts, geographic location analysis charts based on Google map, and relationship-revealing force-directed graphs and streamgraphs. The component has been applied in multiple intelligence research projects (see Figure 5 [Figure 5: see original paper]), including interactive maps in the Open Access platform, multidimensional comparison bubble charts in the Major Science and Technology Trends Panorama project, force-directed graphs in the Research Competitiveness Omnidirectional Evaluation project, and streamgraphs in the Science Structure Map web version. Compared with direct invocation of third-party libraries in native JavaScript or jQuery, using the ng-info-chart component significantly reduces front-end code, improves development efficiency, and produces cleaner, more readable code that is easier to maintain. The component continues to evolve, gradually supplementing more visualization components according to different analytical needs. Current components support mainstream desktop browsers including IE9+, Chrome, and Firefox.

References

[1] Chen C. CiteSpace: Visualizing Patterns and Trends in Scientific Literature [CP/OL]. [2015-08-10] http://cluster.cis.drexel.edu/~cchen/citespace/.
[2] Batagelj V, Mrvar A. Pajek—Analysis and Visualization of Large Networks[A] // Graph Drawing [M]. Springer-Verlag Berlin Heidelberg, 2004:77-103.
[3] Kirk A. Data Visualization: A Successful Design Process [M]. Packt Publishing, 2012.
[4] La A. Language Trends on GitHub [EB/OL]. [2015-08-30]. https://github.com/blog/2047-language-trends-on-github.
[5] Zhao Yue, Chen Zhiwei, Cai Shuhui, et al. Web Visualizing for Large Mount of Scientific Computing Data [J]. Modern Computer, 2012(5): 3-6.
[6] Darwin P B, Kozlowski P. AngularJS Web Application Development [M]. Packt Publishing, 2013.
[7] Uri Shaked. AngularJS vs. Backbone.js vs. Ember.js [EB/OL]. [2015-08-01]. https://www.airpair.com/js/javascript-framework-comparison.
[8] Jain N, Mangal P, Mehta D. AngularJS: A Modern MVC Framework in JavaScript [J]. Journal of Global Research in Computer Science, 2015, 5(12): 17-23.
[9] Green B, Seshadri S. AngularJS [M]. O' Reilly Media Inc., 2013.
[10] Lerner A. Ng-book: The Complete Book on AngularJS[M]. Fullstack.io,

2013.

Author Contributions

Chen Ting: Designed the component technical solution, wrote program code, drafted the manuscript;
Wang Xiaomei: Designed the component technical solution, revised the manuscript;
Lv Weimin: Researched visualization solutions, wrote program code.

Conflict of Interest Statement

All authors declare no conflict of interest.

Supporting Data

Supporting data is self-archived by the authors, E-mail: chent@mail.las.ac.cn.
[1] Chen Ting. nginfochart-master.zip. Visualization component project source code repository link. https://github.com/jy00295005/nginfochart.
[2] Chen Ting. motion_{chart}.csv. Visualization data for dynamic comparison chart in Figure 4. https://github.com/jy00295005/nginfochart/blob/master/motion_{chart}.csv.

Received: 2016-02-18
Revised: 2016-03-22

Note: Figure translations are in progress. See original paper for figures.