Bachelor’s thesis

Degree programme: Information Technology

Study Group: NINFOS12

Completion year of the thesis 2016

Yang Xiao

WEB FRONT-END ARCHITECTURE WITH NODE.JS PLATFORM

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme | Information Technology

Completion year of the thesis 2016 | Total number of pages 45

YANG XIAO

WEB FRONT-END ARCHITECTURE WITH NODE.JS PACTFORM

Along with the rapid development of information science technology, Web standards have been constantly improved and the demand for the user experience of product rises. As a result, Interactive Web applications are more carefully designed and front-end technology becomes a concern for many companies.

Based on the research of front-end architecture in this thesis, a high performenced front-end architeture was proposed and was validated in practice. A carefully designed interactive application Mobile Assistant was taken as an example to demonstrate that. The feasibility and rationality of the proposed method are verified in the development work of this thesis project. The front-end architecture proposed in this thesis gives a good solution idea to the problem of complex interaction enterprise Web applications.

KEYWORDS:

Node.js, front-end, Grunt, MVC, Sea.js, backbone.js

CONTENTS

APPENDICES

FIGURES

PICTURES

LIST OF ABBREVIATIONS (OR) SYMBOLS

Abbreviation	Explanation of abbreviation (Source)
MVC	Model–View–Controller
SPA	Single Page Application
CMD	Common Module Definition
NPM	Node Package Manager
JSON	JavaScript Object Notation
URL	Uniform Resource Locator
DOM	Document Object Model
API	Application Programming InterfaceI
PubSub	Publish/Subscrib Pattern
XML	Extensive Markup Language
W3C	World Wide Web Consortium
FIFO	First In First Out
SPI	Single Page Interface
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
AMD	Asynchronous module definition
UI	User Interface

1 INTRODUCTION

1.1 Research Background

According to the latest study, web traffic amounts to 60% of all Internet traffic(Cantelon, 2015), which means that websites have become a kind of mechanism for data organizing and passing. Web communication, no doubt, is the core data transmission intermediary amount Internet data traffic.

In early days of web development, HTML, as the core of web technology, could only display simple and rough text-based webpages with complex later maintenance or update. In addition, static documents were the base of web module in the early stage, which was the foundation of a webpage and presented the state of webpage stored in the server. Since CSS 2.0 has become the standard, web front-end developers can separate the web content from its performance, which makes web constructing and later maintenance easier. Therefore, user interfaces are restricted and integrated, based on CSS＋DIV as the main development technology.

After 2000, along with the fast development of web browser technology, JavaScript was becoming a main web development language, and web front-end technologies were more developed. When Web 2.0 progressed further, web related products became a key link in the global high-tech value chain. Such as YouTube, Twitter, Facebook, and web games appeared on the scene. With a growing number of web users boosting demand and strict requirements for functionality and appearance of web pages, optimizing the performance of website was one of the top priorities for future development in this area. Under such circumstances, the JavaScript language was acquired great importance and was assigned major responsibilities. Web development technology marked by JavaScript language was a good indicator that this trend will accelerate. Before Web application became popular, for front-end development, main libraries such as YUI or jQuary did not exist and there was no library that can be called front-end framework. However, due to the particularity of the Web front-end development course and development method, it is hard to meet the demand of diversifying developing technologies. Even thought there were a series of front-end mode deriving from MCV along with SPA concept have been appearing, relaying on only one specific developing framework is insufficiennt to solve practical problem.

In the last few years, the emergence of increasingly diverse interactive web application generates thousands of JavaScript codelines for a small web application. For now, JavaScript is not just a general-purpose scripting language, but also a language that can replace flash/flex to implement traditional client RIA and at least a majority of the features(Helm, 2012). A popular development mode is being used: almost all work of UIs is processed through the front-end side, which is the focus of entire development. And for backend, the only task is to support the json data resource. In this case, old frameworks are being updated and new open source libraries are emerging, which has has a profound influence on modern frontend development patterns. Architecture oriented framework development becomes the future direction.

1.2 Current state of research

To facilitate the development using JavaScript, some JavaScript libraries have been created, like JQuary. "Library is a pre-written code that developers simply just need to include and invoke the code pattern in their source code. It solves problems like repeated using certain code pattern and module or too complex to wrap combinations."(Rivest, 2006) In addition, by using frameworks, code patterns will be associated according to a standard form. With the combination of preceding two technologies, like the MVC module, we can transform tightly coupled component to an extensible and modular and loosely coupled core business code. For example, the Pub/Sub pattern, it brings out the "Control" part to decoupling "Module" and "View".

Because of the advantages mentioned above, the SPA concept becomes popular. SPA is acronym for Single-page Application. The most prominent technique currently being used to implement the concept is Ajax that is derived from MVC mode. SPA has advantages such as less server requests, fast reaction and good practicability. However, it also brings out problem for developers because it makes front-end architecture more complex. This makes front-end developers to have to face the back-end issues, for example how to logically decouple the code. But now, for JavaScript development aspect, there is no effective solution to solve back-end issue but including MVC module in front-end development.

By convention, when dealing with large scale SPA, developers usually split it into small components. Templates, patterns and script constitute a small component. So how can we split SPA according to web loading behavior, that is, HTML code loading first, then strategically loading CSS and JavaScript file? This is the problem that obviously can't be solved using the MVC frame.

1.3 Research content and goal

Based on the research background and status above, we build a front-end architecture and make it more likely to back-end development. This idea provides more possibilities to solve problems that front-end frameworks cannot handle. So we can consider using some external tools to find a solution.

The purpose of this thesis is to implementing Sea.js, SASS, Backbone.js with MVC pattern, and Grunt task runner based on Node.js platform to reach the following objectives:

- To build a reusable front-end development workflow.
- Creating a standard UI components library.
- Implanting a broadcast mechanism and decoupling between modules.
- Perfecting the automation system to eliminate the manual management.

We need to combine framework, tools, process specification, and text files together to optimize efficiency of front-end technologies.

2 INTRODUCTION OF MAIN DEVELOPMENT TOOLS AND CORE TECHNOLOGIES

Firstly, kennel technologies and overall technology framework we used to build the high performance front-end web architecture will be introduced in this chapter. And some of the common problems that developers may encounter when testing HTML applications and how to deal with them will be talked in the later part.

2.1 Node.js

Node.js is a cross-platform runtime environment that is built on Google Chrome’s V8 engine for developing server-side Web application. Under this kind of runtime environment, JavaScript can run out-of- browser in a server side environment and load exactly the same way as languages like Ruby, Python and PHP. In other word, JavaScript can be coded as a back-end language by using Node.js and it even can do better then those conceptual back-end languages such as Rube or PHP.

Node.js does not like other open-source libraries for web development, Node.js is more lightweight and easier to built. Problems like processing data at a massive scale and requirement for capability to respond in real time have been take into full consideration when creating this environment at the beginning. “By abandoning traditional design idea that relay on multithreaded environment to implement high concurrency, complex queries, Node.js operates on a single thread, using non-blocking I/O calls and event-driven networking.”(Wilton, 2010) In the meantime, those features not only make a dramatic improvement in performance but also reduce the complexity of the multithreaded application design.

The Node.js appearance has brought new hope for back-end development using JavaScript. Meanwhile, CommonJS has been used into modules and packages. Especially bringing asynchronous workflows into the business tier, JavaScript has the non-blocking feature of making itself stand out amoung all the interpreted language modules.

Above all, Node.js is a runtime environment that can quickly build network server and web. The advantages of Node.js are as following:

- Built on currently the fastest Google Chrome V8 engine.
- Single thread concept eases development processing.
- Event loop module
- It is an interpreted language. Developer can code on cross-platform and use features such as closures and anonymous functions freely.

2.2 Introduction to Sass

As anybody who has ever done front-end development knows, CSS technologically is not a programing language. Although it can be used to develop web styling, it is not programming. In other words, CSS is more like a tool for web designers, but not for developers. In the eyes of the developer, CSS is an intricate tool, because it has no variable or statement, just lines of description of the target object. And not only that, because of the layer limitation of selector, the priority management for CSS selector is really time-wasting and have to write complex coding pattern repeatedly. To solve this problem, programming elements have been adding to CSS language, which is called “CSS processor”. The basic idea of CSS processor: programmers can use a specialized language for developing front-end web styling, and then the code will be compiled to normal web styling CSS file.

SASS is one of a CSS development tool. It frees developers from repeating simple code to makes them more productive. SASS provides lots of common syntaxes for developing CSS web styling, which make development and later maintenance more elegant and effective.

2.3 Introduction to Sea.js

Different form JavaScript frameworks like JQuary, Sea.js will not extend the language encapsulation feature, but make front-end code a JavaScript loadable modules framework to load modules in turn by following the CommonJS standard rules. In this way, front-end developers do not need to worry about the dependency between burdensome front-end JavaScript files and target object. Common problems like tangling and scattering of code in JavaScript programming can be solved efficiently by introducing Sea.js. Meanwhile, it also makes JavaScript source code more readable and easier to maintain. In general, we can invoke Sea.js to encapsulate a mess of JavaScript code into multiple modules that have private logical address space, than place as much core function into modules as possible. To be sure, in this way, we use Sea.js to providing utilities to handle module loading and management efficiently.

2.4 Introduction to Backbone.js

The Model-View-Controller (MVC) pattern is a popular and widespread software design pattern that is suitable for interactive system at present. But as all developer know, MVC pattern are mostly used in back-end development, front-end developers care less about it.

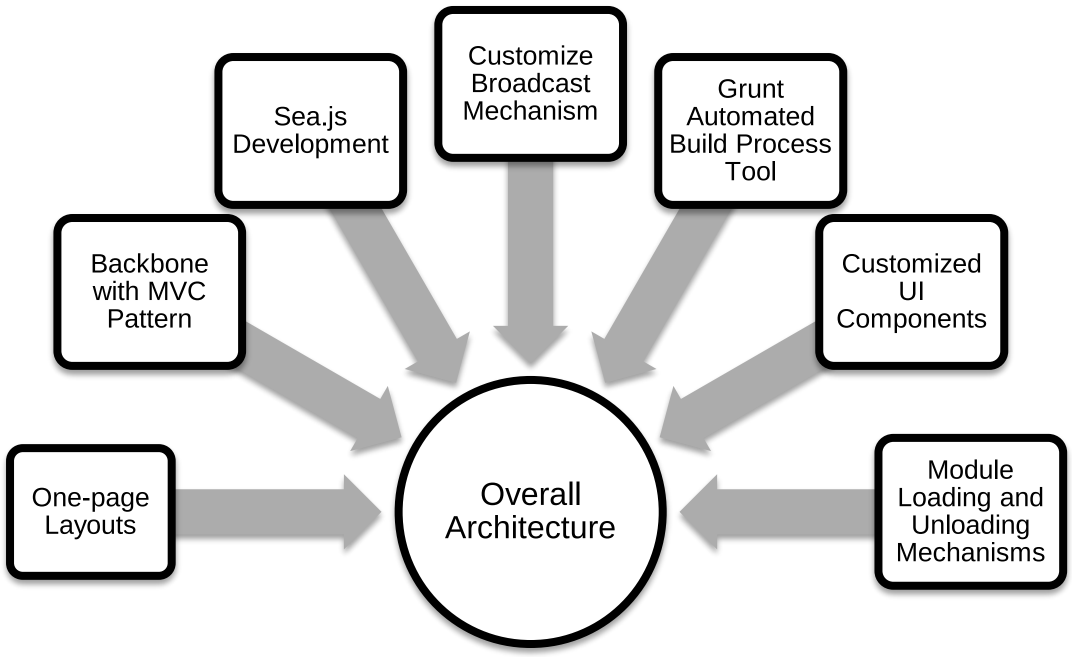
Backbone.js, as previously outlined is rare front-end JavaScript framework that provides a MVC pattern. In one word, Backbone.js is a Web-application framework that includes everything that needs to create key-value binding and database-backed custom events web applications using the Model-View-Control pattern. Backbone.js can define the name of the class, its attributes, and its operations, like programing in Java, and separates the data and logic, thus simplifying the development process and the technologies needed to create the application and getting less clutter and more memory space.

To put it simply, Backbone.js is a front-end MVC framework that has the same powerful management feature as JavaScript. Here is the simple introduction of Backbone.js's main features:

- Module: it is for getting or setting the value of data elements.
- Collection: it is a collection of a certain module type.
- View: It is the interaction presentation between operating module and Dom, and external presentation of an application as user interface.
- Controller: It acts as facilitator tor keep business logic completely separate control and logic.

2.5 Overall architecture and technologies

This thesis comprehensively expounds the overall web front-end architecture in the Node.js platform and the details of technologies. See picture 1:


Customize Broadcast Mechanism
Sea.js Development
Grunt Automated Build Process Tool
Backbone with MVC Pattern
Customized UI Components
One-page Layouts
Overall Architecture
Module Loading and Unloading Mechanisms

Picture 1. Front-end architecture

On the Node.js platform, Sea.js technology leads modular development implementation can solve naming conflicts and the problem of file dependencies problems. Moreover, Backbone technology can achieve the MVC module implementation for font-end code. In addition, observer pattern leads to customize broadcast mechanism implementation to solve the high coupling module development problems effectively. The consistency of module development is achieved by a customized module loading and unloading mechanisms and the requirements of their individual application UI components are satisfied by customized UI components. Moreover, developers always complain about some schemas or maps tend to tie up programmers with works that are not related to programming. The front-end automation was achieved by Grunt automated build process tool, allowing developers to focus more on the code development. This leads to efficient code management and efficient collaboration development.

2.6 Combining Backbone to implement Single Page Interface

Single Page Interface, or more exactly an entire application accessed via a single URL and loading a single HTML file. By showing only those fields and options that are relevant to the runtime phrase and hiding the irrelevant contents, HTML single page sticks on one interface during every user operation. There are main reasons using Single Page Interface:

- Less server request, reducing network traffic and resource consumption.
- UI will be more responsive. There is only displaying and hiding strategy, but no loading process needed.
- Providing a seamless user experience that doesn't require a page reload.

Consequently, every interface has its advantages and disadvantages. Here are key challenges we meet:

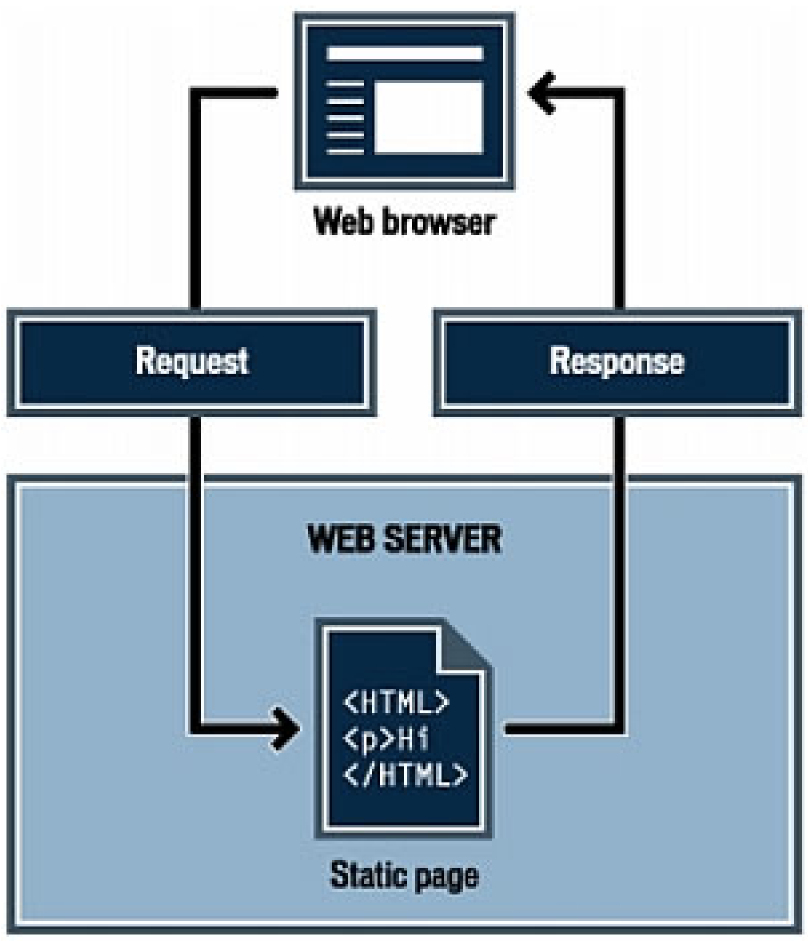
- SPI makes front-end work too heavy.
- Causes problems like logical decouple or code association for back-end.

By implementing MVC pattern and Backbone.js, these problems can be solved. The main line of thought is that the basic HTML document frame has to be defined first, and then invoke the backbone.js framework with a MVC pattern. After deploying and modifying the whole interface using JavaScript, finally more data can be accessed from web server. Thus an integrate operating loop is formed.

2.6.1 How to define the basic HTML documentation framework

Generally, the overall web page will be separated into three main parts: header, content (that also called body), and footer. This is the base frame for a web application. Developers can classify the content as well as defining structures for each part according to application requirements. Taking Google Chrome as an example, in most cast, Web content will be separated to left content and right content, or left menu and right content.

Since Single Page Interface will be built, it also has to display a new page as a functional website. Here we need to invoke Backbone.js to implement built-in support for embedded templates. The original multi-page contents will be separated and put into deferent templates. When users switch a web page, only the content template is replaced. So in fact, the initial layout of the upper frame of HTML code is still the same(Pic.2).



Picture 2. SPI workflow(Shimizu, 2006)

2.6.2 Using JavaScript to modify page

As mentioned above, different template contents are used to display multi-page interfaces. The question is how to achieve the optimized loading algorithm and the highly effective content switching.

The traditional way is binding responsive user interface controls with the event, and the event hander will switch from one interface to another. In the traditional method, the above handling could become even more cluttered, especially if we use the same method to deal with different templates as dealing with the normal page. “Developers will spend a plenty of time struggle with event callback logic or ’an inverted pyramid’ code.”(Leeds, 2001) If a traditional method is used to build a Single Page Interface, it leads to an inevitable code part for storing variables adding to application logic. This situation can be handle when there is a small amount of code, but the code will be difficult for maintenance when a project grows in size, which also makes a small modification on the main logic which will affect corresponding data display logic. Usually, the stronger thecoupling of code is, the less elasticity of product design process will be.

Backbone.js can solve this kind of code coupling problem. It decouples the code by providing frame and module for both the control layer and display layer. We can use templates from Bcakbone.js to associate the contents mentioned above. Another powerful tool “router” is used to implement toggling between different templates of a program, which achieve separation between business control and interface content display.

The concrete realization method is that different templates will be bound with different hash tagged patters and provide linkable, bookmarkable, and shareable URLs to meaningful locations within a web page. We can use multiple ways to link URLs to the specified actions and event by invoking *Backbone.Route*. This one of “route” function works as Controller, and the page markup and data loader module could be segregated ideally.

2.6.3 Data transfer with server and Interface modification

Speaking of data, defining the data format of each individual transmission in the communication should be taken into consideration very first. JSON format is used here for data transfer. JSON is a lightweight data-interchange format, which It is easy for humans to read and write and also easy for machines to parse and generate. It makes sense that a data format that is interchangeable with programming languages.

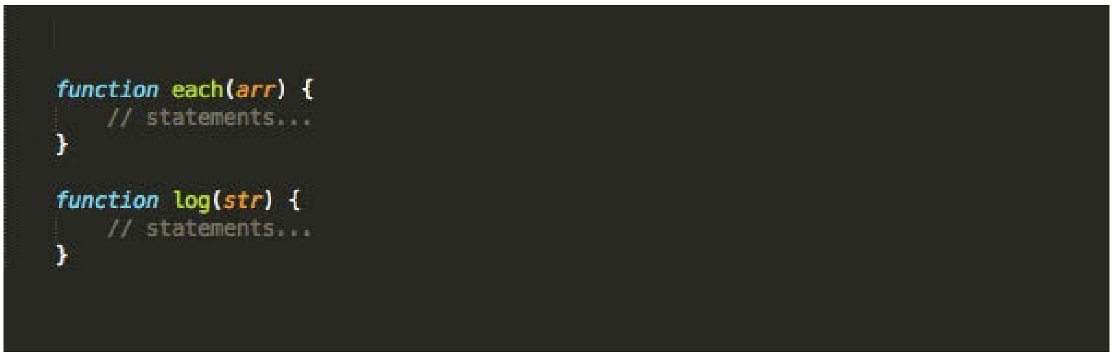
We use Mode and Collection functions from Backbone.js process JSON formatted data transferred from web server, so View function lefted is used to display the data. This is how Backbone.js MVC pattern works. A module normally contain a state of the application, like data in JSON and element in Backbone, is working for managing and accepting state/data changes. Not only this, module function can also coordinate state/data content, and it continues to be beamed to web server to achieve data synchronization. This flexible and efficient method is easy to manage and modify data uniformly.

2.7 Using Sea.js to implement modular development

As mentioned above, Boasbone.js is introduced to achieve MVC pattern in Front-end development, and MVC pattern strategy is separating the application to M, V and C three parts. This lead to a number of extra files will be generated using the framework useded in this thesis. So the adoption of this pattern means that developers have to deal with more files, and more difficulties of file dependency and management. Moreover, some name collisions between functions and classes will arise at software development practices.

2.7.1 Name Conflicts

In software project development, most of JavaScript developers are used to abstracting out independent function into several corresponding functions. See figure 1:



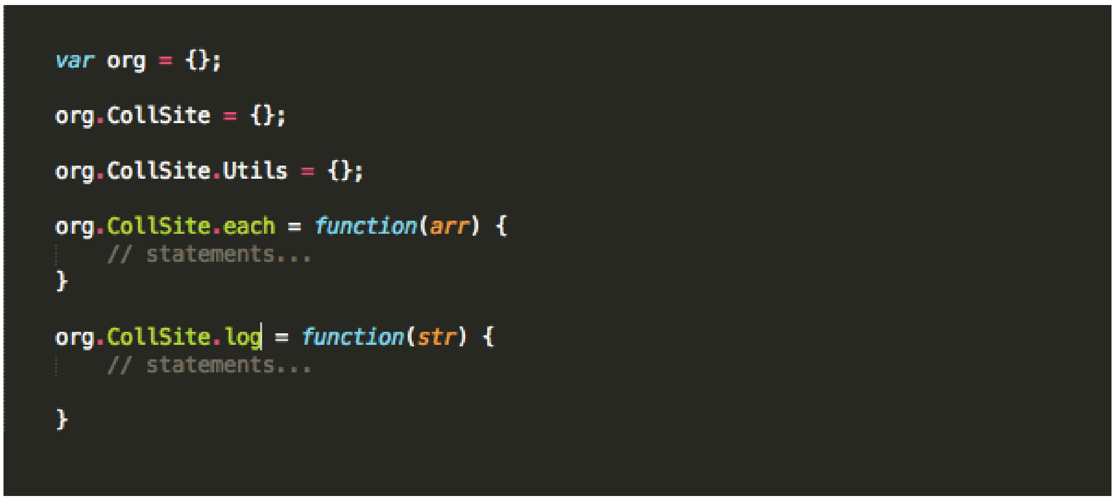
```
function each(arr) {
    // statements...
}

function log(str) {
    // statements...
}
```

Figure 1: Abstracting functions

In module development, these functions will be encapsulated and packaged in a util.js file, then developers have to decide whether to invoke them. This way is entirely feasible in small projects or small developer teams. When project grows larger, there a problem arises and, one or more of the following issues may occur:

a. “I want to define a each method, but util.js package was already invoked once, so I have to create a new function and mane it *eachObject*.”
b. “I defined a log method, but why the other developer’s code is reporting error?”

Faced with a comparable problem, We can imitate the solution in Java language by using namespaces: a base set of tags should be defined for whatever the application needs, and then allow developer to add their own data into the file, in their own namespace, without disturbing the DOM tree. This will change the code as shown following(Fig.2):



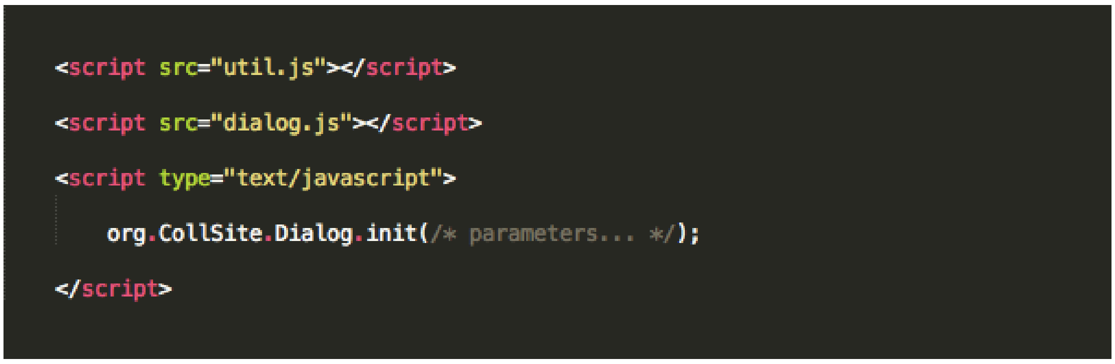
```
var org = {};

org.CollSite = {};

org.CollSite.Utils = {};

org.CollSite.each = function(arr) {
    // statements...
}

org.CollSite.log = function(str) {
    // statements...

}
```

Figure 2. Implementing NameSpace

Thus by using namespaces, the conflicts are solved once and for all. Furthermore, another question came up: we invoke a really simple function with a very long namespace need to be memorized.

2.7.2 File Dependency

After solving the name conflict above, we can start developing a common component in the UI layer based on util.js. There is no need for other developers in the project to repeat the same piece of code. For example like the common component named dialog.js, the way of invoking it is very easy(Fig.3):



```
<script src="util.js"></script>

<script src="dialog.js"></script>

<script type="text/javascript">

    org.CollSite.Dialog.init(/* parameters... */);

</script>
```

Figure 3. Invoking dialog.js

As a matter of fact, when other programs calling dialog.js, there was always error reported. After debugging, the problem was found that util.js is not invoked in the dialog.js package. The file dependent bug in a small piece of code can cause inconvenience. Especially when the project grows larger, structure becomes more complex, dependency in multiple files will bring more problems. Here are some common problems in Web development:

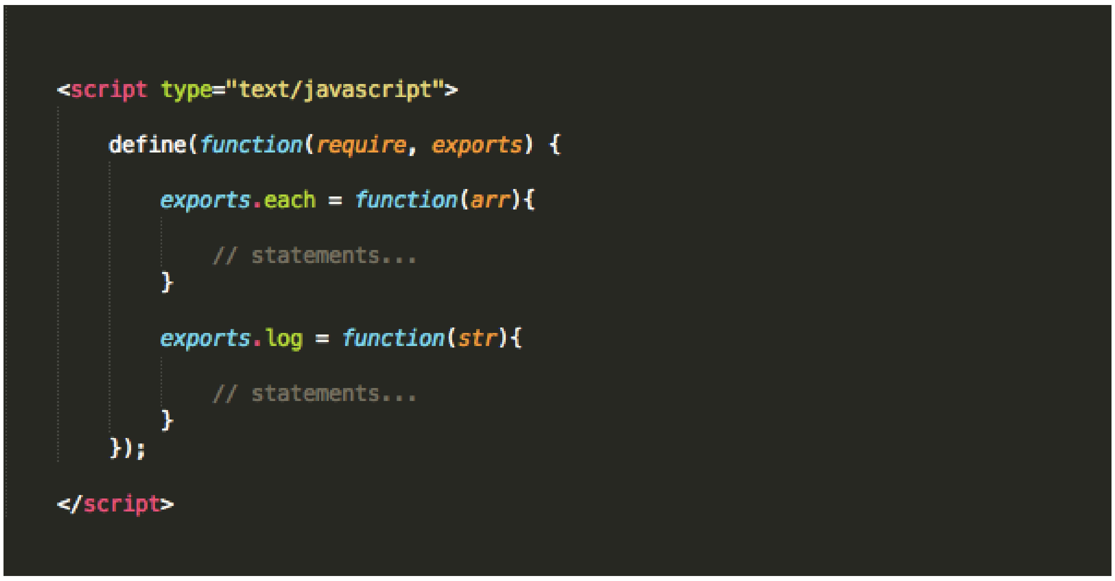
- Common components update the front-end foundational class library, which can make it more difficult for whole web application upgrades.
- The business layer have to call a new common component, but hundreds of lines extra code have to be written.
- Only the oldest version of the class library can be used to extend their built-in functionality.
- The front-end code conflict makes it difficult to merge the functional business layer.

The key cause of the problems above is that file dependency has not received enough recognitions and not been effectively resolved. A few years back, developers only could ensure file independency between front-end pages and scripts by testing. A naive approach is unlikely to succeed for large projects or complex structures, however, it is working on small web applications. Being stranded on file dependent problem is a very common situation in project development.

Name conflict and file dependency are classic problem in front-end development. The next section discusses how to solve these problems by using Sea.js as a framework for developing and deploying modular reusable JavaScript programs.

2.7.3 Sea.js for developing modular

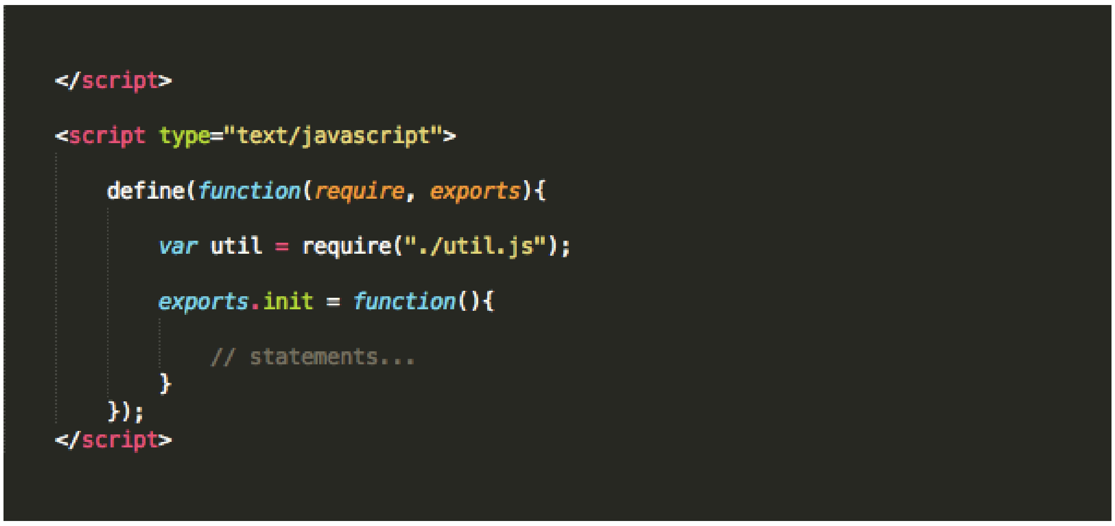
Sea.js is quite a mature open source project. Programming based on Sea.js framework has higher development rate, fewer opportunities to be wrong, and is more convenient to maintain. Developers should always follow the CMD (Common Module Definition) while using Sea.js for developing modular. An independent file is treated as a module. So the code mentioned above becomes like this(Fig.4):



```
<script type="text/javascript">
    define(function(require, exports) {
        exports.each = function(arr){
            // statements...
        }
        exports.log = function(str){
            // statements...
        }
    });
</script>
```

Figure 4. Using Sea.js developing module

Meanwhile, a unified interface is provided through the export parameter. In this way, dialog.js can be modified like this(Fig.5):

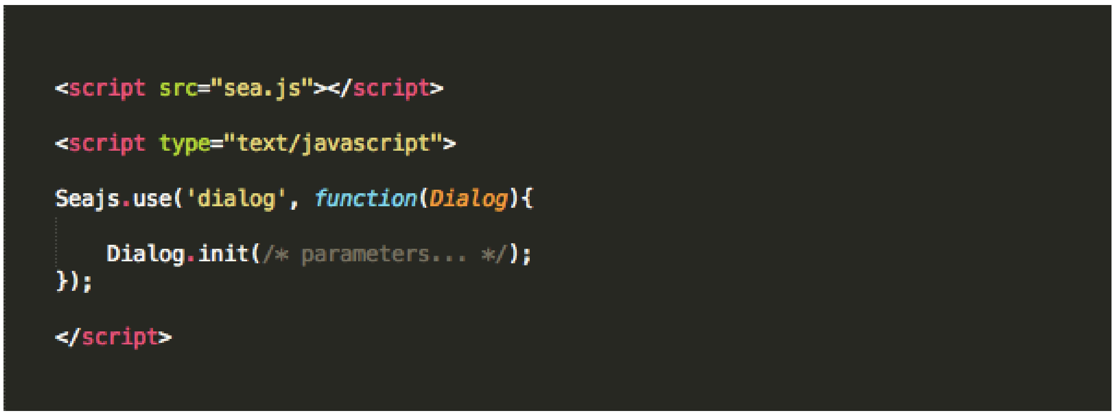


```
</script>
<script type="text/javascript">
    define(function(require, exports){
        var util = require("./util.js");
        exports.init = function(){
            // statements...
        }
    });
</script>
```

Figure 5. Modifying dialog.js

According to the updates above, we can get a unified common interface through the export provided in util.js file by a line of code- “*require("./util.js")*”. Here “require” works as a key word grammar that Sea.js provides for JavaScript language. In other word, the key word “require” can be also used to expose interfaces in other module.

By using Sea.js, our front-end developer can program like Back-end developers. It makes JavaScript language work like Python, Java, or C# languages that can use function like “include” and “import”. So we can easy invoke dialog.js file in a page, for example(Fig.6):



```
<script src="sea.js"></script>

<script type="text/javascript">

Seajs.use('dialog', function(Dialog){

    Dialog.init(/* parameters... */);
});

</script>
```

Figure 6. "use" keyword

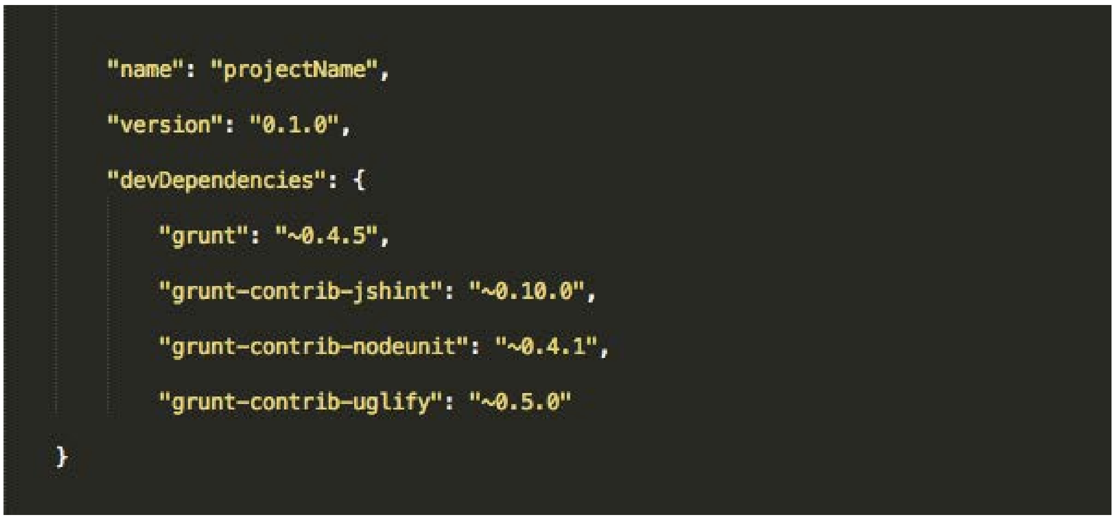
When using Sea.js, we have to import Sea.js file first, then use *Seajs.use* to import any common component needed in the page. To sum up, there are many benefits using Sea.js:

- Solving the name conflict problems once and for all. Using the exports exposed interface, there is no need to use namespace and global variables.
- Solving the file dependent problem. By using key word “require ” to import the depended files, Sea.js can handle the dependencies automatically.
- Version control for modules. By executing build tools and using hostname aliases, Sea.js can make version control much more easer.
- Improving code maintainability. Sea.js make the files have single responsibility that helps a lot for version control.
- Sharing module in crossed environment. Sea.js has the similar CMD as Node.js. So Sea.js module can be developed on Node.js platform in crossed server and browser.

2.8 Using Grunt for automated builds management

If we want to install Grunt task runner in our project, there are two files named package.json and Gurntfiles.js should be included in our source folder. The kernel reasonable configuration of these two files.

File package.js is used for npm to storing project configuration metadata. Here is the most important part of configuration in “devDependencies”(Fig.7):



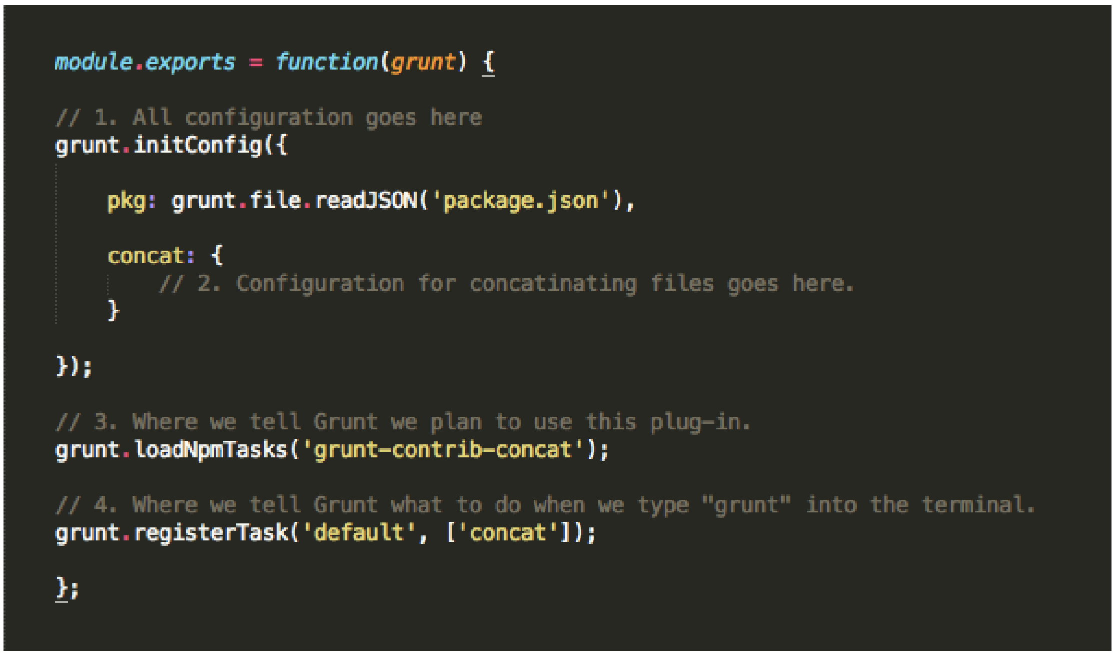
```
    "name": "projectName",
    "version": "0.1.0",
    "devDependencies": {
        "grunt": "~0.4.5",
        "grunt-contrib-jshint": "~0.10.0",
        "grunt-contrib-nodeunit": "~0.4.1",
        "grunt-contrib-uglify": "~0.5.0"
}
```

Figure 7. Grunt configuration

The configuration displayed above shows how npm, the management tool we need, manage the tool packages.

After configuring package.json, we can get in the content by enter some commend line in administration console. Typing in npm install, npm will install the needing tool packages and plug-ins according configuration in *devDependencies* under the project content.

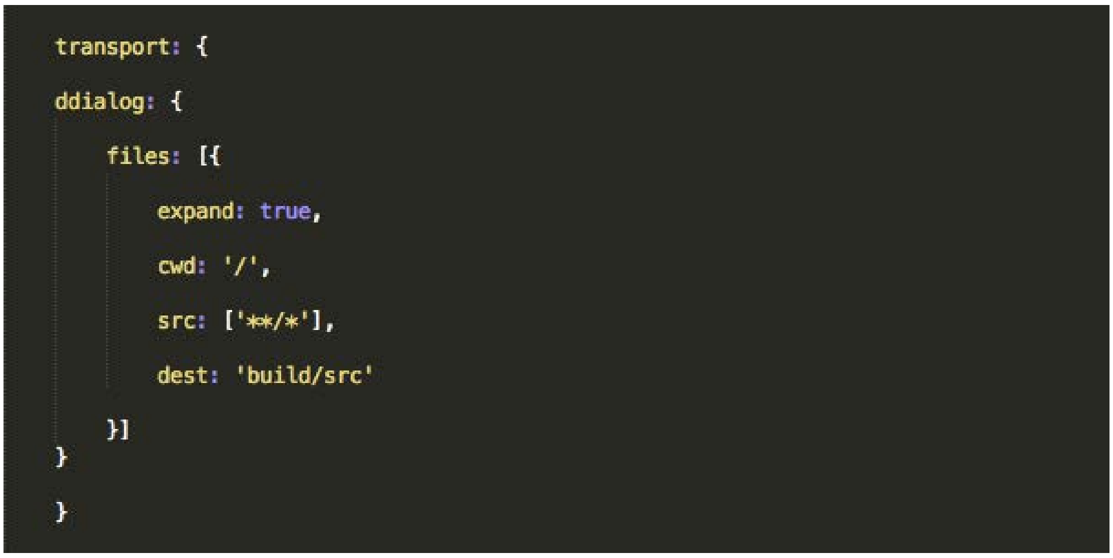
And now we have our tool packages installed. To use them effectively implement automated build process, another file Grunt.js need to be configured like as following in Figure 8:



```
module.exports = function(grunt) {

// 1. All configuration goes here
grunt.initConfig({

    pkg: grunt.file.readJSON('package.json'),

    concat: {
        // 2. Configuration for concatinating files goes here.
    }

});

// 3. Where we tell Grunt we plan to use this plug-in.
grunt.loadNpmTasks('grunt-contrib-concat');

// 4. Where we tell Grunt what to do when we type "grunt" into the terminal.
grunt.registerTask('default', ['concat']);

};
```

Figure 8. Confuguring automated build tesks.

We can see that there is main configuration in *grunt.initConfig*, and we can read the defined values in package.json by typing line *pkg: grunt.file.readJSON('package.json')* in latter configuration tasks. Here is some basic task configuration(Fig.9):

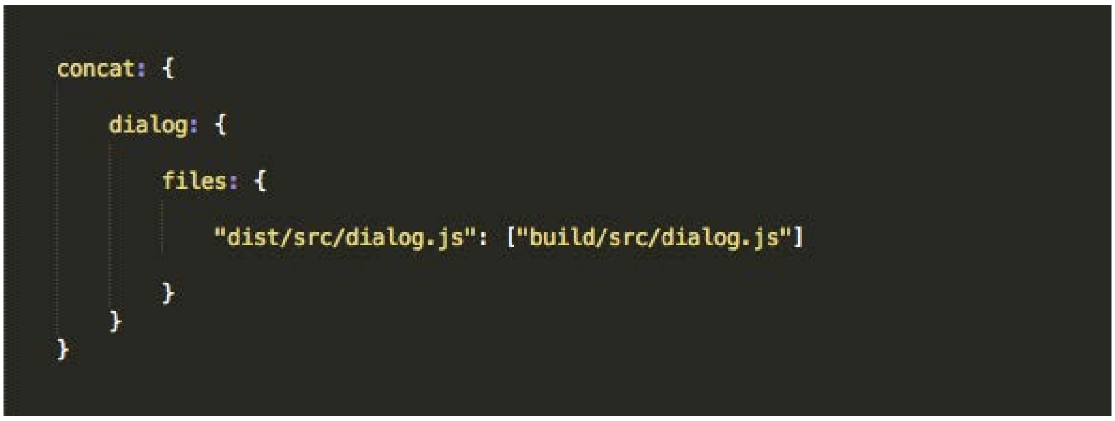


```
transport: {

ddialog: {

    files: [{

        expand: true,

        cwd: '/',

        src: ['**/*'],

        dest: 'build/src'

    }]
}

}
```

Figure 9. Configuring task information

Like the code above shows, *transport* is a task that we configured. It is one of a Grunt building plug-ins. The function of this plug-in is extracting dependencies between modules and set a unique module ID for them. So *dialog* is one of the targets in our *transport* configuration task. In *files* we define which file will be the source file to execute in our task, and which file directory will be the destination of resulting file generated after the task.

There is another task for concatenation I will introduce as follow(Fig.10):

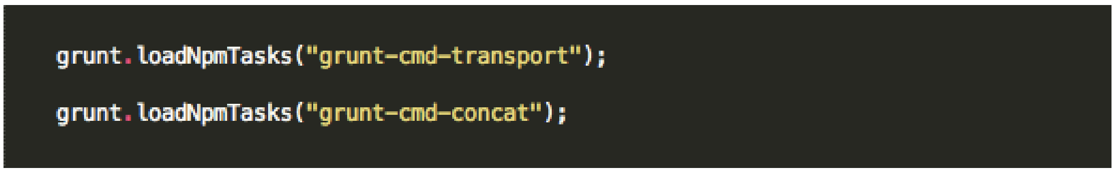


```
concat: {

    dialog: {

        files: {

            "dist/src/dialog.js": ["build/src/dialog.js"]

        }
    }
}
```

Figure 10. Concatenating dependent module

The function of *concat* is concatenating dependent modules. In line *"dist/src/dialog.js": ["build/src/dialog.js"]* is another way to define source file and destination that explained above.

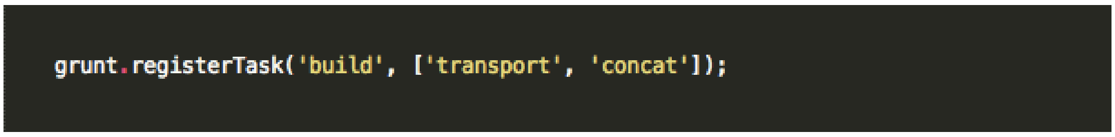
Then we can loading the needed plug-in in Grunt(Fig.11):



```
grunt.loadNpmTasks("grunt-cmd-transport");

grunt.loadNpmTasks("grunt-cmd-concat");
```

Figure 11. loading plug-in

And we configure the commend line to execute the tasks(Fig.12):



```
grunt.registerTask('build', ['transport', 'concat']);
```

Figure 12. Excuting task

So an integrated Gruntfile.js will look like the following:

```
module.exports = function(grunt) {
grunt.initConfig({
    pkg: grunt.file.readJSON('package.json');
transport: {
  dialog: {
    files: [{
        expand: true,
       src: ['**/*'],
 grunt.loadNpmTasks("grunt-cmd-transport");
grunt.loadNpmTasks("grunt-cmd-concat");
grunt.registerTask('build', ['transport', 'concat']);
```

After all configuration steps in Grunt.js file, we can enter the content by typing commend lines *grunt build* in console window. This one single line can implement extraction and concatenation of relevant dependent files.

The workflow described in this session is just a simple introduction. In real project, developers can configure package.json and Gruntfile.js according project needs.

2.9 Implementing a broadcast mechanism

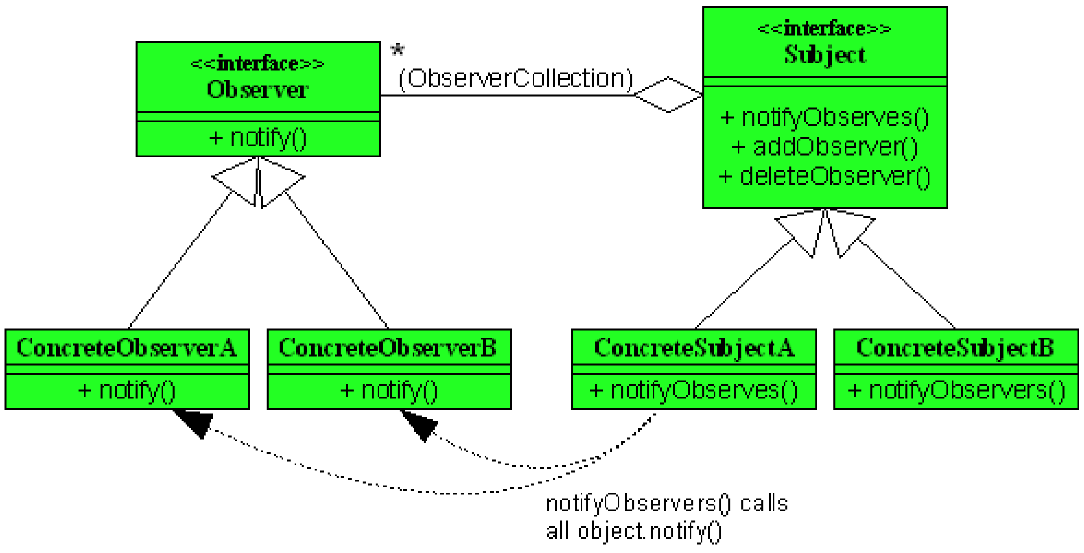
One interesting difficulty that every modular development faces is how to decouple. This section employs the broadcast mechanism to implement communications between modules then achieve low coupling among them. The underlying principle modules communicate by sending broadcast notification events and receiving broadcast events, so the sender only sending evens, but does not care about whether they will be subscribed or not. Even though broadcast event is a kind of custom events, but a different form of custom events, the broadcast event is not bound to any set of principals. It is implemented based on the Observer design pattern. The demarcation between modules is clearly defined by the Observer design pattern, which increases the reusability and maintainability of the code.

2.9.1 Observer design pattern

In the Observer design pattern, observers and the observed are separated efficiently. There are multiple ways to implement the Observer design pattern, but pattern should have two objects pointing to observers and the observed. There is “observe” logic between two objects: when observer is defined, the observed also vary accordingly. Imagine that front-end user interface is the observer and business data will be the observed. When business data is changing, the user interface will also send a corresponding request, for example, like application updates. Also, when web application is updated, the content that webpage displays is also need some modification, the business data will remain the same.

2.9.2 Implementing an Observer design pattern

According to different application requirements the Observer design pattern can also transform into publish-subscribe pattern, model-view patter, source-listener pattern, etc. This thesis uses the “register-notify-deregister” pattern, which is also one kind of publish-subscribe patter. See Picture 3:



Picture 3. Observer design pattern(Payne, 2010)

The Observer will be registered as one of Subjects, and the Observed will uses a Container to store corresponding Observer. If the Observed is changed, Container that all the preceding Observers registered will be notified the modifications and react accordingly. If there is no need to observe any Observable, the Observer will be notified to cancel the observation, and the Observed will be removed from the Container and Subject list.

Without knowing any specific information, the Observed will register the Observer in an observable container through an interface provided by Observer. The benefit of working in this way is that the same interface can be used repeatedly when there is other Observer exists in application. In Observer design pattern, mapping becomes one-to-many relationship: When one object varies, objects relying on it will get notifications and be updated automatically; when the Observed is updated, it will broadcast the notifications to all the corresponding Observer. Basing on the interface, the pattern can significantly increase programming flexibility. Meanwhile, the publish/subscribe message pattern to provide one-to-many message distribution and decoupling of applications. ”Defining a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.”(Vinoski, 2010)

2.10 Customized UI Components

2.10.1 Reasons for customizing UI components?

In front-end development, UI components provided by native JavaScript language are only button, input field, menu and drop-down list. The high-speed growth of front-end developing technologies leads to more varied and personalized productions, which urged the developers to develop new UI components.

It is a common situation that developers use any plug-ins they want in their web application once they know how it works. Developer only needs to import a third party library, and invoke the plug-ins, without knowing how it was developed. But mostly, only a single function in the library is needed but we have to import the whole library. For example like Bootstrap library, when a multi-functional button is needed, I have to import a whole library and my layout design is automatic replaced with a Bootstrap layout. It not only increases the size of my application, but also affects my design.

So it is a good choice to develop a customized UI component, to break the limitation of native JavaScript and define our own UI components. It also helps us knowing insight into internal implementation details of JavaScript language.

2.10.2 How to implement customize UI components

As we analyzed above, it is necessary to develop customized UI components, but not all the component is need to customizing.

For the UI component that repeatedly used and will be used in later extensions, we need customize them. They will be encapsulated together and developed as one module to fit the modular development framework described in this thesis.

Firstly, the modular development framework requirement should be followed when we developing our customize UI components to keep a united layout design. Then we encapsulate the components and keep the application flexibility at the same time.

For example, we need customized a component that has HTML5 drag-and-drop functionality, so here is the logic: UI effects are changed together at the same time after Drag-Enter mouse event, trigger a callback event, and restore the original effects after Drop event, then trigger a callback event again. In this example, the event trigger time is customized fixed, but we can set a default value when callback event is triggered. So this component can be called anytime and configured on callback event. But we need to be careful with garbage collecting operating that as soon as running period of the component completes its work, it is destroyed and its memory is returned to the operating system to avoid out-of-memory exception.

2.11 Summary

This chapter has a simple introduction to the about main developing tools and overall technology framework in front-end architecture design. Then it has some discussion any analysis about kernel technologies. Firstly, Node.js platform is introduced, and analyzed its advantages that are the reason why choosing Node.js as a development platform. A CSS preprocessing tool--SASS, modular development tool--Sea.js, and framework implement MVC patter—Backbone.js are introduced next, consecutively. After that, the overall front-end framework was outlined, also, how to SPI implemented by Backbone.js using MVC pattern was clearly described. Then, we analyzed how to using Sea.js solve name conflict and file dependency problems in detail. In order to make developers focus more on code, task runner Grunt is used as an automated builder, which was analyzed in detail. Meanwhile, to solve coupling problem, a solution using broadcast mechanism was proposed. In order to reduce complexity, and enable maintainability and extendibility, while still providing the functionality, the last part in this section analyzed the customized UI components in detail.

3 IMPLEMENTATION AND DESIGNING OF WED FRONT-END ARCHITECTURE IN A PRACTICAL APPROACH

The high-performance front-end architecture proposed in this thesis was implemented in "Diandian Mobile Assistant" as an example to verify practical meaning of it.

3.1 Basic introduction of "Diandian Mobile Assistant"

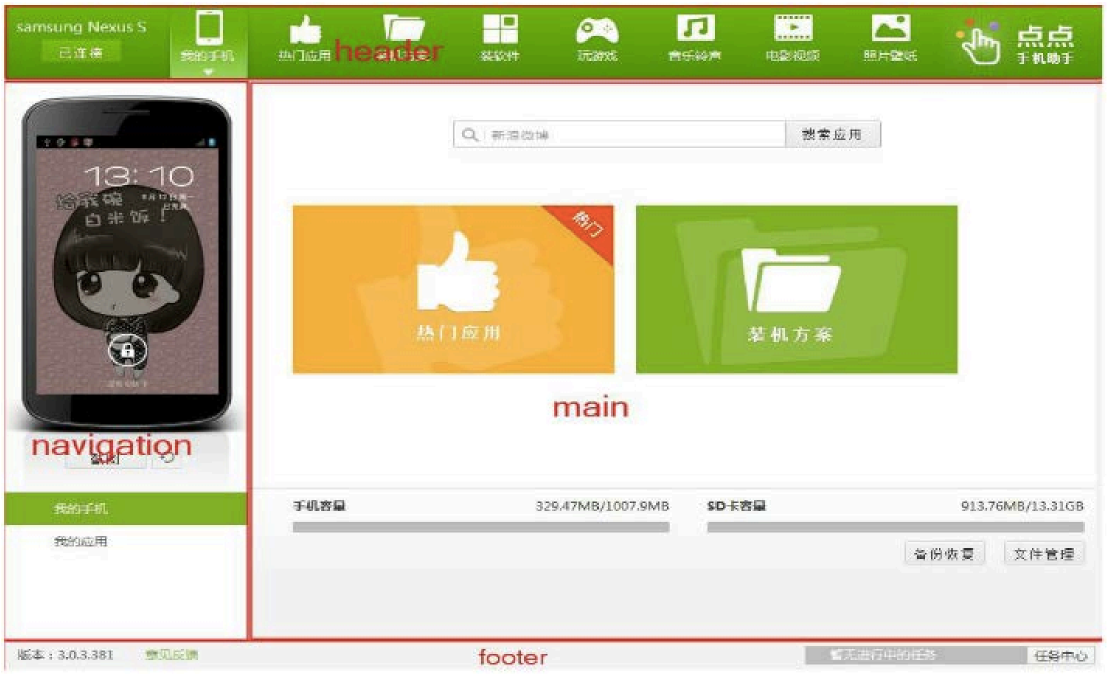
In China, Google Service cannot load on Android OS, so some software companies holds the opportunity. Many well-known publishers were launched similar applications for Android mobile users, and "Diandian Mobile Assistant" is one of a product that I take part in the development process in my last internship. "Diandian Mobile Assistant" is a desktop application for mobile running on Windows Operating System. This application was developed using Chromium Embedded Framework+Web, which has advantages of high development speed and stable running. This application has eight main functions: My Mobile, Popular Application, File List, App Install, Games, Music, Video, Photo.

3.2 Front-end architecture developing environment of "Diandian Mobile Assistant"

The front-end developing environment of "Diandian Mobile Assistant" is the same as front-end architecture proposed above. In Node.js platform, using Sass as a highly efficient tools for developing CSS, and using Sea.js to implement file management and modular development. Also, Grunt was used to automate the developing process.

3.3 SPI implementing

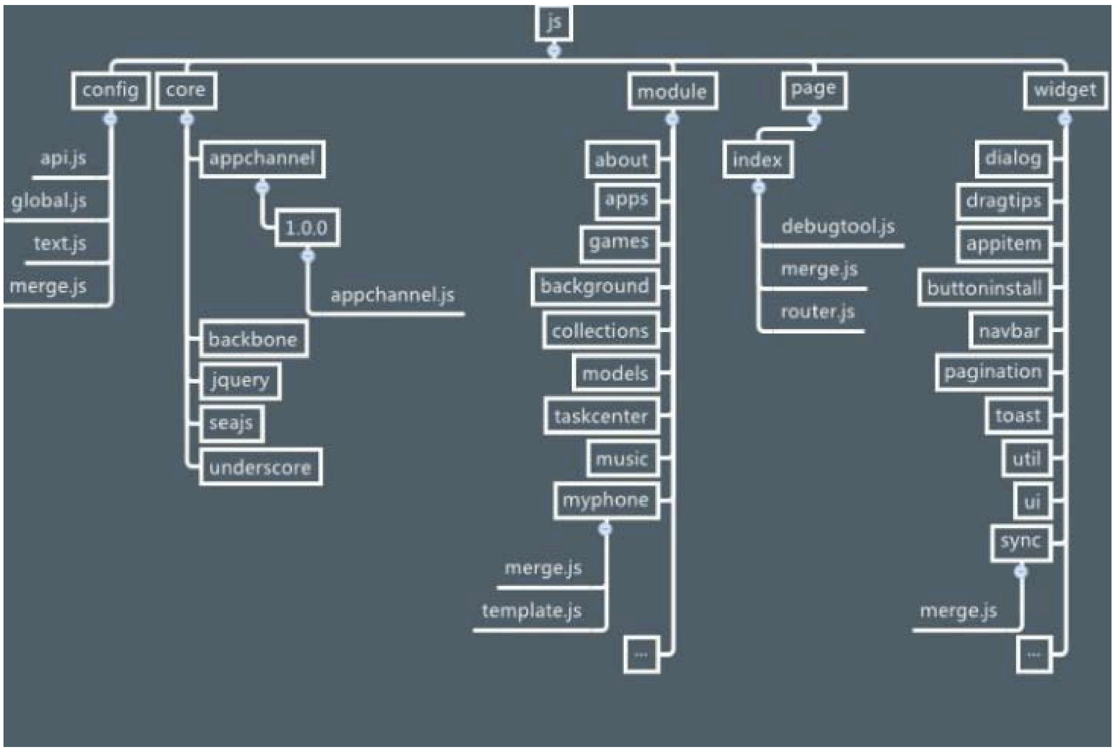
The whole application has one only single page interface. The layout is separated into upper, middle, and bottom parts, and the middle part was divided into left and right panes. So the whole interface has header, left menu, main content, and footer, fore functional area in total. If one functional area modifies another one, a broadcast mechanism needs to be used to send notification. Here is the SPI of "Diandian Mobile Assistant"(Pic.4):



Picture 4. Basic UI layout

3.4 Directory structure

The directory structure of “Diandian Mobile Assistant” architecture is very typical. See picture 5:



Picture 5. JavaScript directory structure

According to the directory structure of overall architecture (JavaScript in main), we can see that JavaScript is divided into five subdirectories: config, core, module, page, widget. Under config directory, there stored interface api.js that back-end will data through, file global.js that store all the global variables in the architecture, differed language version file called text.js, and the entry file for the directory named merge.js. Third party libraries imported like JQuery, Sea.js, and Backbone.js are stored in core. Under module directory, as the name suggests, there are functional modules stored, like the different main function module are stored under the same directoryas the name suggests. And our architecture is a single page interface, so under page directory, with no doubt, stored entry file for whole JavaScript. Also, the merge.js file, router.js file that control backbone control layer, and debugtool.js file for testing are also included in it. The widget directory left is the directory that stored the customized UI components, like dialog box, tab, loading progress bar, etc.

3.5 Implementing Modular development

As it was mentioned above, we are using Sea.js to implement modular development. Let make music function module as an example.

From the code in Appendix 1, we can use:

```
define(function(require, exports, module) {
  // JavaScript code goes here
}
```

as a Interface to define a module, and use *require* interface to invoke the needed module. For example, if a Backbone framework needs to be imported in to the main code, it implements like this:

```
var Backbone = require('backbone');
```

After importing the framework library module, we can invoke any corresponding open interface. For example in a music function module, we invoke an *extend* function from view *object* in Backbone frame work, and assign a value to Music module like following:

```
var Music =Backbone.View.extend({
      //….
  });
```

We configure the interface of module first, then updates the properties file created in the first step with a key-value pair that provides the corresponding behavioral definition interface. And here is binding event interface for music module:

```
var Music =Backbone.View.extend({
       //binding event
```

```
        bindEvent:function(){
          // …
                },
                //there are interfaces of other function modules going on here
      });
```

At last, though *export* interface, we can configure our customized UI component like this:

```
    Module.exports = Music;
```

After all the steps above, our music function module is well defined.

3.6 MVC pattern in module development

In order to associate the front-end side code, and implement model, view, and control logic separately like back-end side code, we use Backbone framework mentioned in last chapter to implement MCV pattern. And Music function module is as took as an example.

3.6.1 Model file of Music function module

Here is the code detail:

```
/*
        Backbone.Model For Music
    */
    define(function(require,  exports,  module){
    // import resources
    var Backbone = require('backbone');
    // Music Model
    var MusicModel = Backbone.Model.extend({
});
    // export Music Model
    module.exports = MusicModel;
});
```

Because that Model is also a module file, so we still use Sea.js file module to combine Backbone framework and Model part of Music function.

3.6.2 Collection file of Music function module

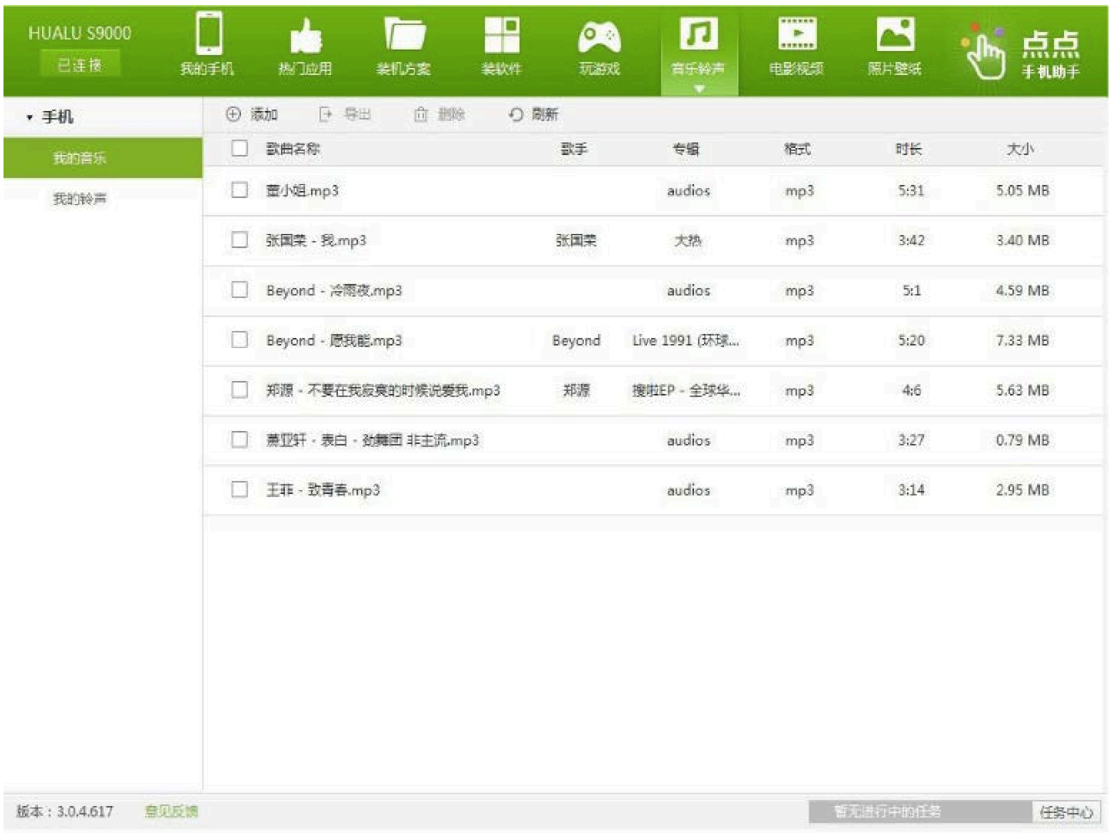
More code detail can be found in Appendix 2.

Similarly, the Collection part of Music function module is also a file module, so Backbone framework is also invoked in this code pattern. To implement the relevant function of Music module, we customized corresponding interface, for example: Using *toggleChecked* to obtain a selected Model interface, using *allChecked* to obtain all selected Model interface.

3.6.3 View file of Music function module

More code detail can be found in Appendix 3.

We fill the profile data that obtain by using *fatchData* function from back-end into Collection file, and render the data on Music module interface by calling *renderMusic* function. So here is a example of Music model interface after rendering:



	歌曲名称	歌手	专辑	格式	时长	大小
☐	董小姐.mp3		audios	mp3	5:31	5.05 MB
☐	张国荣 - 我.mp3	张国荣	大热	mp3	3:42	3.40 MB
☐	Beyond - 冷雨夜.mp3		audios	mp3	5:1	4.59 MB
☐	Beyond - 愿我能.mp3	Beyond	Live 1991 (环球...	mp3	5:20	7.33 MB
☐	郑源 - 不要在我寂寞的时候说爱我.mp3	郑源	搜啦EP - 全球华...	mp3	4:6	5.63 MB
☐	萧亚轩 - 表白 - 劲舞团 非主流.mp3		audios	mp3	3:27	0.79 MB
☐	王菲 - 致青春.mp3		audios	mp3	3:14	2.95 MB

Picture 6. Music function interface

3.7 Implementation of broadcast mechanism

In the previous chapter, we talked about implementing broadcast mechanism by using Observer design pattern to solve coupling problems. Since broadcast mechanism can build a decoupled communication between modules, it will improve usability and maintainability of the application. In Observer design pattern elements like observer, the observed, and subject are all object, and JavaScript is an object-oriented language. So we use object as a function to present all the data and code patterns in front-end side. Here we use FIFO strategy to store subjects in object queue: the first element that is removed form the collection is always the first added one. Full code can be found in Appendix 3.

From the code we can find that the observers will be adding to an list object, see:

```
this.attachFunctionList = {};
```

Through attach interface, observers that corresponding to observables will be put in the list, so we can listen message. In the same way, detach interface will remove the corresponding observer from the queue and decouple the message.

```
Broadcast.attach('system:connectStatusChange',
                this.onConenctChange, {
                                scope:this,
                                isAsync:true
                });
Broadcast.detach('system:connectStatusChange', this.onConenctChange);
```

3.8 Implementation of customized UI components

As it was mentioned that developing customized UI component is necessary process. Here we take a customized dialog box as an example, and the code will be given in Appendix 4.

Fron the code we can find that UI component is also treated as a file module, so it also used Sea,js development framework. We first encapsulate the common part of the UI, so other develops only need to configure some parameters. In the dialog box code above, the basic layout is fixed, and only setTemplate interface can be the entry to customize the dialog box. So all the dialog box in this application will have title, body, and footer. But some parameters like size, position, mouse click event can be configures differently by using interface like setWidth, setPosition, and setMouthPosition.

3.9 Implementation of automated builder

A task runner Grunt was introduced as an automated build management. And the automated build tasks “Diandian Mobile Assistant” need are JavaScript grammar check, file merging, file compression, Sass tool, and Sass file listener. We also need to configure properties of application like version number, model name, alias, etc. And here is the configuration in package.json:

```
{
        "family": "dolphin",
        "name": "diandian",
        "version": "9.0.0",
        "spm": {
                "alias": {
                        "jquery": "jquery",
                        "backbone": "backbone",
                        "underscore": "underscore",
                        "appchannel": "appchannel"
                }
        },
        "devDependencies": {
                "grunt": "~0.4.1",
                "grunt-contrib-jshint": "~0.6.0",
                "grunt-cmd-transport": "~0.2.4",
                "grunt-cmd-concat": "~0.2.2",
                "grunt-contrib-uglify": "~0.2.0",
                "grunt-contrib-clean": "~0.4.0",
                "grunt-contrib-sass": "~0.3.0",
                "grunt-contrib-watch": "~0.4.4",
                "grunt-contrib-cssmin": "~0.6.1",
                "grunt-filetransform": "~0.1.0",
                "grunt-contrib-compress": "~0.5.2"
```

```
		}
	}
```

After configure the package.json, Grunt.js must be configured to set automated build tasks. Since the basic framework of Grunt configuration is already introduced in last chapter, here is more detailed configuration code for obtaining file dependency task:

And for file compression:

```
	concat : {
		main : {
			options : {
				paths:['output/.tmp'],   include:'all'
			},
			files : [{
				expand:true,
				cwd: 'output/.tmp',
				src:['**/*.js',   '!**/*-debug.js'],
				dest:'output/<%=pkg.version%>/js'
			}]
		}
	}
```

The configuration of other tasks is really similar to the two example above. More syntax structure for configuring Grunt can be found in their official documentation.

3.10 Summary

This chapter analyzed more piratical technologies based on front-end architecture proposed on chapter two, and take “Diandian Mobile assistant” as an example talked about implementation of SPI design, MVC pattern, module development, broadcast mechanism, customized UI components, and automated task building. Finally the effectiveness and reliability ware validated through the examples.

4 FRONT-END PERFORMANCE OPTIMIZATIONS

Internet is rapidly developing nowadays, many enterprise users and single users connect to it, and net information resources increase greatly too, web browser has become one of the major carriers that people capture information. Front-end performance issue has become very austere and received more and more attention. Developers optimize front-end performance by optimizing code. Sometimes a small change will make big different, such as decreasing wait times for your users. This chapter will have a discussion about optimizing front-end performance in two aspects.

4.1 Reducing the number of HTTP requests

HTTP is a connection between web server and client side through Internet to implement data receiving and sending. In fact, when web browser communicating with web server, there are many events ware processing in that few millisecond.

With a normal process flow of HTTP request goes like this: a URL is entered in the address bar of your browser, browser will connect the server this URL is pointed and sent a request to the server, then server will make a reply to the request, after browser revive the reply, it can start to parse the data sent from server. Especially when the web page contains los of photos, CSS files, or JavaScript files, connection between web server and browser will continuously build and release. This inevitably results in a waste of resources and a heavy performance burden on a web application. In a static internet speed, time spend on downloading a 100KB file is less then downloading two 50KB files. So reducing HTTP request is a kernel way to optimize front-end performance.

The front-end architecture discussed in this thesis can effectively reduce HTTP request. Firstly, the SPI design means that there is only one THML file requested from server. Different from traditional web design, one click will request a HTML file. Secondly, the CSS, JavaScript file compression technology provided by Grunt can greatly reduce size of files. In

"Diandian Mobile Assistant", it only contains one JavaScript file and one CSS file, both are compressed.

4.2 Using JSON format for data transfer

JSON is a lightweight data-interchange format, and also the data-interchange format for native JavaScript, which means that it won't use and API or tool package during parsing process by JavaScript.

Traditional Web services are based on XML. But compare to serialization of XML, size of file in JSON format is smaller after serialization. So famous websites, like Facebook, are all using JSON as their data-interchange format.

JSON has two data structure: objects and arrays. Object works internally with Key/Value pairs that defined between a pair of braces. Keys are followed by a colon, and Key/Value pairs are separated by a comma. Code looks like this:

```
var obj = {"name": "Yang ", "age": "23", "location": "Turku" }
```

Arrays are collections of values. It must be enclosed in brackets, and also separated by a comma.

```
var jsonList =[{{"name": "Yang ", "age": "23", "location": "Turku" },
              {"name": "Username", "age": "unknown", "location": "Turku"}];
```

Operations in JSON are convenient and high efficient. In implementation of MVC pattern using Backbone.js and data processing in Model and Collection mention in this thesis are all using JSON data format.

4.3 Summary

This chapter discussed about optimizing front-end performance based on our front-end architecture, also presented two solutions and their respective use cases. The first one is reducing HTTP requests, and using JSON data format is the second. Both practical solutions have the benefits of high-efficiency during the whole project development, maintenance, and management period.

5 CONCLUSION

The main research goals aspects in this thesis were:

- Research on high performance front-end architecture based on Node.js platform.
- A deep study of Sea.js, and implement module developing in practical project work.
- Through multiple verifications, implementing Grunt automatically building technologies in practice.
- Combining technologies mentioned in this thesis and implementing them in "Diandian Mobile Assistant", I took part in module developing for functions, which are App Management, Music and Photo.

The front-end web architecture proposed in this thesis was used in project, so the technologies used in the project complement each other and operate seamlessly, and realize advantageous complementarities and promote common development. It does show its efficiency and high performance during designing, developing, and maintaining periods. Meanwhile, the reducing of HTTP requests, file compression, data exchanging in JSON format, and automated building management optimized the advantages of this front-end wed architecture.

The development of HTML5 and CSS3 technologies greatly improves the existing networking platforms. New HTML5 web page elements like header, section, figure, and footer that are more semantical and readerable. Moreover, the use of Web Socket technology creates server updates data and pushed them in UI, so users do not need to refresh webpage frequently. HTML5 makes interaction between video, music, picture and computer more standardized. While integrating new technologies and new methods to achieve better performance, we always need to follow the standards. New technologies, like Ajax, jQuery, YahooUI, have become really popular in front-end development, and they are all standardized derived application based on the W3C standard. To develop web applications which are more creatively designed, optimized program code, user-friendly, compatible with more browsers, developers need to forge ahead and make continued efforts.

REFERENCES

Backbone.js. 2014. http://backbonejs.org/. Consulted 11.4.2014

Cantelon M, Holowaychuk T. 2001.Java Servlet Programming 2nd Edition. California: O'Reilly Media.

Cormen T, Leiserson C, Rivest R. 2006. Introduction to Algorithms. 2nd ed. Massachusetts: MIT Press.

Coyier C. 2012. Sass vs. LESS. http://css-tricks.com/sass-vs-less/. Consulted 31.5.2014

Crawford W. 2010. High Performance JavaScript. California: O'Reilly Media.

Gamma E, Helm R, Johnson R. 2012. Design patterns elements of reusable object-oriented software. New York: Adison-Wesley.

Grunt. http://gruntjs. Com Consulted 23.8.2010.

IDC. Worldwide Quarterly Mobile Phone Tracker. http: //www.idc.com/ Consulted 22.5.2005

Leeds C. 2009. Designing Web Navigation. California: O'Reilly Media.

Payne J. 2010. Professional JavaScript for Web Developers. New Jersey: Wiley Publishing, Inc.

Sass. 2014. Sass Basics. http://sass-lang.com/guide .Consulted 19.4.2014

Sea.js. http://seajs.org/docs/. Consulted 12.4.2012.

Tilkov S, Vinoski S. Node. js: Using JavaScript to buildhigh-performance networkprograms. http://www.slideshare.net/hustwj/nodejs-using-javascript-to-build-highperformance-network-programs. Consulted 3.11.2010.

Tateno M, Shimizu S, Arakawa Y, 2006. Construction of overlay network considering user preference in peer-to-peer systems[J]. TechnicalReport of IEICE.

W3C Recommendation. 2006. Extensiblle Markup Language(XML)1.0. 5th ed. https://www.w3.org/TR/REC-xml/. Consulted 26.11.2008.

Wilton P, McPeak J. 2010. Begginning JavaScript. 5th ed. New Jersey: Wiley Publishing, Inc.

Wong C. 2006. HTTP Pocket Reference-Hypertext Transfer Protocol. California: O'Reilly Media.

},

Appendix 1. Using Sea.js to implement Music function modular development

```
On this define(function(require, exports, module) {
     // import resources
    var Backbone = require('backbone'),
                    $ = require('jquery'),_ = require('underscore'),
                    AppChannel= require('appchannel'),
                    API=require('config/api'),Broadcast=
require('widget/broadcast/merge'),
                    Dialog =require('widget/dialog/merge'),
                    MusicItem =require('widget/musicitem/merge'),
                    musicSync = require('./sync'),
                    Collections = require('module/collections/merge');
    var Music =Backbone.View.extend({
                el:$($('#mod-music').html()),
      moduleName:'music',
    //binding event
      bindEvent:function(){
                  this.$el.on('click.music',
                             '.music-manage-list-content .checkbox',
                             this.toogleCheckOne).on('click.music',
                             '.list-head .checkbox',
                             this.toogleCheckAll).on('click.music',  '.operation .import-button',
                             this.importMusic).on('click.music',  '.operation .delete-button',
                             this.deleteMusic).on('click.music', '.operation .export-button',
                             this.exportMusic);
                },
                //there are interfaces of other function modules going on here
    });
```

Appendix 2- Collection file of Music function module

```
/*
Backbone.Collection For Music
*/
define(function(require, exports, module) {
// import resources
var Backbone = require('backbone');
var _ = require('underscore');
var Model = require('module/models/merge');
// Music Collection
var MusicCollection = Backbone.Collection.extend({
        model : Model.MusicModel,
        parse : function(response){
                        console.log(response);
                        var json;
                        var results;
                        if(typeof(response) === 'string'){
                                        json = eval('(' + response + ')');
                        }
                        else{
                                        json = response;
                        }
                        results = json.ret.result;
                        _.each(results, function(model){
                        var filePath = model.filePath;
                        if (undefined === filePath){
                        filePath = model.path;
                                                        }
        model.forma=    filePath.substring(filePath.lastIndexOf('.') + 1);
```

```
                                        });
                                        console.log(results);
                                return results;
                                        },
/*
update the checked attribute of MusicModels in this collection
*/
toggleChecked : function(checked){
        this.map(function(model){
                        model.set({
                                        checked : checked
                        });
                                        return model;
                                        });
                                        },
/*
get all the checked MusicModels
*/
allChecked : function(){
        return this.filter(function(model){
                        return model.get('checked');
                        });
                        }
}) ;
// export MusicCollection
module.exports = MusicCollection;
});
```

Appendix 3. View file of Music function module

```
efine(function(require, exports, module) {
                // import resources
                var Backbone = require('backbone'),
                                $ = require('jquery'),
                                _ = require('underscore'),
                                AppChannel= require('appchannel'),
                                API = require('config/api'),
                                Broadcast = require('widget/broadcast/merge'),
                                Dialog =require('widget/dialog/merge'),
                                MusicItem =require('widget/musicitem/merge'),
                                musicSync = require('./sync'),
                                Collections = require('module/collections/merge');

                var Music = Backbone.View.extend({
                                el:$($('#mod-music').html()),
                                moduleName:'music',
                                //Collect data filling
                                fetchData:function(){
```

```
                        var self = this;
                        self.resetState();
                        self.Collection.fetch({
                                success:self.__success
                        });
                },
                //rander Interface
                renderMusic:function(){
                        models = this.MusicCollection.toArray();
                        models.forEach(function(model){
                                model.set({checked:0});
                                });
                        MusicItem.loadByData(models);
                        this.$el.find(".music-manage-list-
content").find("ul").empty().append(MusicItem.els);
                },
        });
                        module.exports= Music;
        });
```

Appendix 4. Implementation of broadcast mechanism

```
define(function(require, exports, module){
        var Notify=function(){
                this.attachFunctionList={};
        };
        Notify.prototype={
                /*
                Message listener implementation
                @param {String} notifyName
                @param {Function} processFunction
                @param {Object} [options]
                @param {Object} options.scope
                @param {Boolean} options.asynchronous
                */
                attach:function(notifyName,processFunction,options){
                        var _temp,
                        opt= options ||{};

        if('string'===typeofnotifyName&&'function'===typeofprocessFunction){

        _temp=this._processFunction(notifyName);
                                if(!_temp){
                this._addNotify(notifyName);
                                }
        this._processFunction(notifyName).push({
                        fun:processFunction,
                        scope:opt.scope,
                        isAsync:opt.isAsync
                        });
                        }
                        return this;
                },
                /*
                decoupling notify
                @param {Object} notifyName
                @param {Object} processFunction
                */
                detach:function(notifyName, processFunction){
                        if('string' !==typeof notifyName){
                                return;
                        }
                        if('function' === typeof processFunction ){
                        this._processFunction(notifyName,
processFunction, true);
                        }
                        else{
                this._processFunction(notifyName,true);
                        }
                        return this;
                },
                _addNotify:function(notifyName){
                        this.attachFunctionList[notifyName]=[];
                },
        };
        module.exports=new Notify();
});
```

Appendix 5. Implementation of customized UI components

```
define(function(require, exports, module){
        var Template = require('./template'),
                $ = require('jquery'),
                html=Template.dialog,
                shim=Template.shim,
                …
                // size of dialog window
                bodySize={……};
                var Dialog=function(con){
                        var config = con || {};
                        ……
                        //store cache
                        this.cache={};
                        //displayed times
                        this.showTimes=0;
                        this.isDrag = config.isDrag;
                        this.options=config;……this.init();
                };
                Dialog.prototype={
                init:function(){
                this.bindButton();
                ……
                this.setCenter();
                this.setEvent('close',this.hide,this);},
                //dialog bow module
                setTemplate:function(title,body,foot){
                        ……
                },
                isShow: function(){
                        return this._isShow;},
                        ……
                        //destroy closed dialogbox
                        destroy:function(){
                                ……
                        }
                };
                module.exports=Dialog;
        });
```