Priority Based Loading Of HTML Elements For Gecko Rendering Engine

Sriram Baskaran, Shankar Narayanan S G S, Vishnu Charan V, Govind Sudarsan
Department of Computer Science and Engineering
Meenakshi Sundararajan Engineering College
Chennai, India

Sandhya.M. K.
Faculty at Department of Computer Science and Engineering
Meenakshi Sundararajan Engineering College
Chennai, India

IJERT

Abstract

At present, the loading of elements in a HTML document is based on a particular order, starting from <body> to </body>. So this makes the loading of all the pages similar. Therefore an attempt to load a particular element first in the page is not possible. In order to fix this, we suggest going in for "Priority Based Loading of HTML Elements for Gecko Rendering Engine". In this, we propose that the element should be loaded based on the priority value which is set by developer. This has been proposed for Gecko Rendering Engine as the Layout process differs in differing rendering engines. The browsers look for the priority attributes of any element and load the elements in that order (i.e.) the element with the higher priority is loaded first followed by the elements with lower priority. This is intended to be added to the HTML API. On successful implementation of this, developers would be able to load the heavier elements, such as image, video, advertisements etc., at the last. Thereby the user can be provided relevant information at the correct time.

1. Introduction

All of us want the things to be as quick as possible. Be it the lift in a skyscraper or a restart of a computer. Technology has become so advanced that people actually want things at their hand's reach or at least available as soon as they need it. This is mainly because of the advent of computers and the internet. The internet is provided by different ISP present all across the world. Each ISP has established policies that allow limited download speed in order to provide a fair usage to all its customers. This restricts the speed of download of websites.

The main use of the internet is to share the resources available with a person and hosting it in the public for all to see and use it. Developers who create these applications will want their product to be used by all the users. All these applications are hosted online at different web servers given by different vendors. Most of the resources that are hosted online do not provide the relevant information at first; it makes the user irritated and hence they aren't given the correct information. The user either gets it slow or doesn't get the necessary information at the first. The causes for the problems are either the plan of internet that the user chooses or the speed of the network that the user is connected to.

The developer of these resources does not have control over the order or loading of the HTML elements in the website. This paper proposes a concept of Priority Based Loading of HTML Elements where the new attribute called "priority" is added to the HTML elements which modify the order in which the HTML elements are loaded in the browser.

2. Existing System

The existing system is a browser that loads the HTML elements in the First In First Out Order (FIFO), a concept similar to the implementation of a queue. The process of loading of browser elements starts with the browser requesting certain resources from the internet. The resource can be present at any server located all around the world. The request at the Application Layer will be the HTTP Request with the following format

```
GET / HTTP/1.1
Accept:*/*
Accept-Language: en-gb
Host: www.example.com
Connection: Keep-Alive
```

The response from the server to the browser at the Application Layer will be a HTTP Response. The

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
Date: Mon, 04 Oct 2004 12:04:43 GMT
X-Powered-By: ASP.NET
X-AspNet-Version: 1.1.4322
Cache-Control: no-cache
Pragma: no-cache Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 8307
<html>
 <head>
...
```

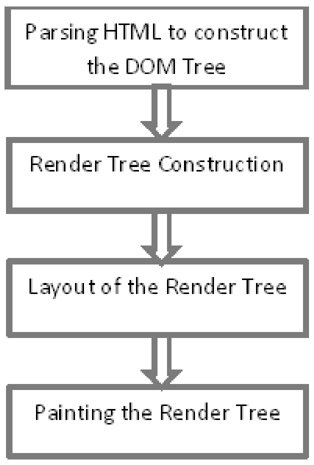

Parsing HTML to construct the DOM Tree
Render Tree Construction
Layout of the Render Tree
Painting the Render Tree

Figure 1. Basic Flow of Rendering

Once the browser gets the HTML response, the browser starts parsing the HTML response. The HTML standards have been defined by a XML schema which provides the necessary format for the HTML document. The browser generates a parse tree which contains every element, attributes and the content as a tree node. The parse tree is generated by the browser. All the errors in the HTML code, mostly syntactical, are found here and are ignored. The parse tree enables the browser to know the elements present, attributes for each of the elements present, and the styles that are added to each of the element.

```
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
```

HTTP Response will contain a response status code which returns the status of the HTTP Response. The packet data will be the HTML document which is then rendered by the browser.

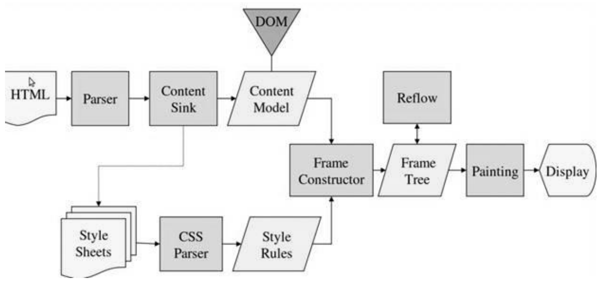


Figure 2. Gecko Rendering Engine Main Flow

IJERT

From the parse tree that is generated, a render tree is created by the browser. The elements of the render tree are generated by choosing each element of the parse tree and applying the attributes and styles for it. If an external style sheet is referenced, the browser sends the request for the stylesheet and the style rules are generated. Once the styles are applied to the elements, the frame constructor generates the layout using the Reflow module and the elements are painted on the browser from the frame tree.

Figure 2 provides the flow of loading and rendering of HTML elements in Gecko Rendering Engine.

3. Browser Architecture

The browser's main components are:

The user interface – Every part of the browser display except the main window where you see the requested page.

The browser engine – The interface for querying and manipulating the rendering engine.

The rendering engine – Responsible for displaying the requested content.

Networking – Used for network calls, like HTTP requests. It has platform independent interface and underneath implementations for each platform.

UI backend – Used for drawing basic widgets like combo boxes and windows. It exposes a generic interface that is not platform specific. Underneath it uses the operating system user interface methods.

JavaScript interpreter – Used to parse and execute the JavaScript code.

Data storage – This is a persistence layer. The browser needs to save all sorts of data on the hard disk, for examples, cookies.

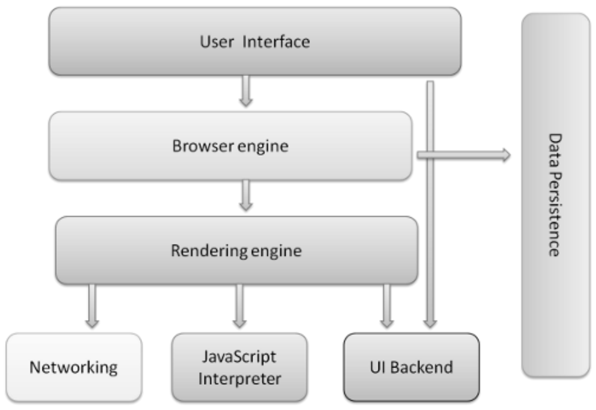


Figure 3. Browser Architecture

4. Priority Based Loading

The proposed system makes slight changes to the basic flow of rendering by bringing in the concept of priority. According to this, every block element in the HTML document will have an added attribute called priority. The priority attribute will take values from 0-10 with 10 having the highest priority. The priority will be given to block elements such as <div>, <p>, <h1> to <h6>, <body>. If a particular element is not given any priority then a default priority 0 is assigned to that element. The proposed system uses Priority Heap to achieve priority based loading.

IJERT

4.1 Priority Heap

The priority heap is a data structure that will contain the node with the highest priority as the root node. The elements in the heap are filled in the order of maximum priority. The heap is represented as a Binary Search Tree with elements having the two properties, viz., Order and Shape [2].

The ORDER property:

For every node n, the value in n is greater than or equal to the values in its children (and thus is also greater than or equal to all of the values in its sub-trees).

The SHAPE property:

1. All leaves are either at depth d or d-1 (for some value d).
2. All of the leaves at depth d-1 are to the right of the leaves at depth d.
3.
   a. There is at most 1 node with just 1 child.
   b. That child is the left child of its parent, and
   c. It is the rightmost leaf at depth d.

4.2 Render Process using Priority Heap

The priority attribute will provide the browser enough information to reorder the loading of elements and load them into the browser. The flow of rendering changes with the new attribute added to the HTML elements. The rendering starts by constructing the parse tree. The render tree is constructed from the parse tree that is generated and the styles of the elements are added to the render tree. The styles are fetched in separate threads if needed by sending HTTP request for them. Next to the render tree construction is the priority heap generation.

After the render tree construction, a priority heap needs to be constructed from the render tree. The priority heap will have the element with the highest priority at the root and the layout construction starts from that. The layout of the document also called "reflow" in terms of Gecko will be done by taking the root element from the priority heap. The flow of rendering in the proposed way is given below.

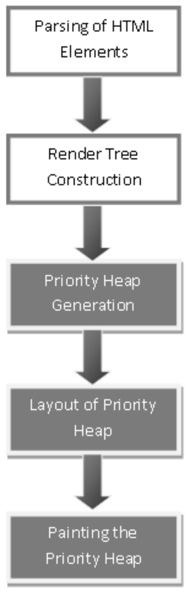


Figure 4. New Flow of Rendering Process

4.3 Rules of Reflow

The following are the rules that need to be followed when constructing the layout of the HTML elements.

1. All the elements are loaded based on the value of the priority. The value of a priority of parent element will be given to the child element unless a priority is explicitly defined for the child element; in such case the rule 2 is followed.
2. When a child element will have priority defined and the parent element doesn't have a priority

value set, then the element is loaded with the default layout attributes and when the parent reaches the heap's root the element the dirty bit system is followed and the attributes are changed based on the value of the parent's layout attributes

3. When removing elements from the heap for layout or reflow, the conflict of two children having same priority which is higher than the parent is resolved by giving preference to the block element.
    3.1. If both the children are block elements then the element which appears at the first in the original document will be chosen.

5. Implementation

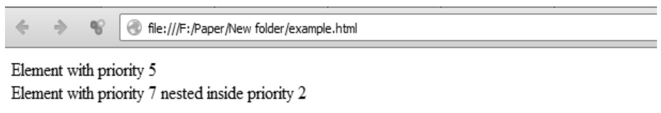
The whole process has to be implemented at the rendering engine of the browser. The rendering engine contains different modules that allow the browser to properly select each element and add its necessary styles and then paint it in the browser area.

5.1 DOM Tree Construction

As we have already mentioned the render tree has to be converted into a priority heap which contains all the block elements based on the priority of each element. Once the render tree is constructed, all the elements from the tree are taken in FIFO manner and populated in a heap. The heap is constructed based on the priority value of each node. Consider the following example

```
<html>
<head>
   <title>Priority Based Loading </title>
</head>
<body>
   <div priority="5" id="Priority5">
      Element with priority 5
   </div>
   <div priority="2" id="Priority2">
      <div priority="7" id="Priority7">
         Element with priority 7 nested inside priority 2
      </div>
   </div>
</body>
</html>
```

The output of this HTML document is given below.


file:///F:/Paper/New folder/example.html

Element with priority 5
Element with priority 7 nested inside priority 2

Figure 5. Output for the HTML Example

The whole concept of Priority based loading is that the final layout is exactly the layout of the designed HTML document. The order in which the elements are loaded can be changed based on the priority attribute.

The DOM tree is constructed from the HTML document and the tree is parsed based on the DOM specification given by W3C [4].

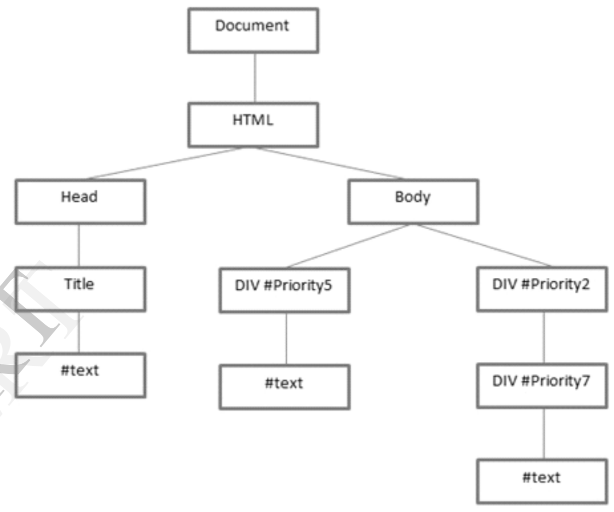


Figure 6. DOM Tree Output for the Example

The DOM tree is essential to parse the document as it can find all the errors in them.

The DOM starts with Document as root and all the enclosing tags as its children. This goes on until the leaves are reached. The DOM tree for the example is given above. The DOM is represented in HTML tags for better understanding.

5.2 Render Tree Generation

The next step is the construction of the Render Tree. The Render tree is constructed only for the <body> and its children and not the <head> section [1]. The render tree for the example is as given below.

IJERT

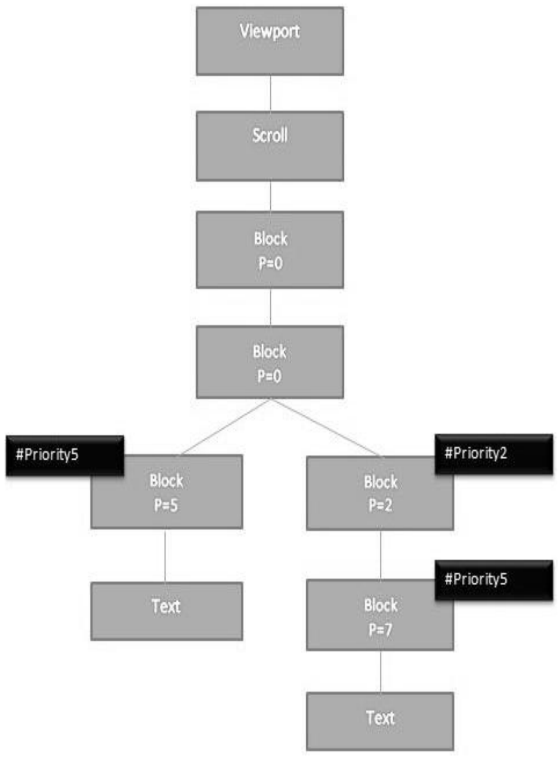


Figure 6. Render Tree Output for the Example

5.3 Construction of Priority Heap

The implementation till now has already been done in the current web browsers. The next step is the construction of the priority heap (the proposed system). All the block elements and organize them in priority way. The priority heap starts from the <body> tag.

5.4 Steps of Construction of Priority Heap

The priority heap is constructed as follows.

1. Select the <body> tag as the first node of the heap and add it to the heap. Since the priority value of <body> is not set, it is taken as zero by default.
2. The render tree is read in depth first order. The leftmost child, #Priority5, is taken as the next node and added to the heap. Since the priority is greater than the parent, the elements are swapped. The heap is balanced now.
3. The next element will be the child of the #Priority5 element. Since no priority is set for the text, it takes the value of the parent as its priority value. This node is placed as a child of the the root and no swapping is needed as the heap is balanced.
4. The same process is continued untill the final priority heap is generated.

The final priority heap that is generated is given below.

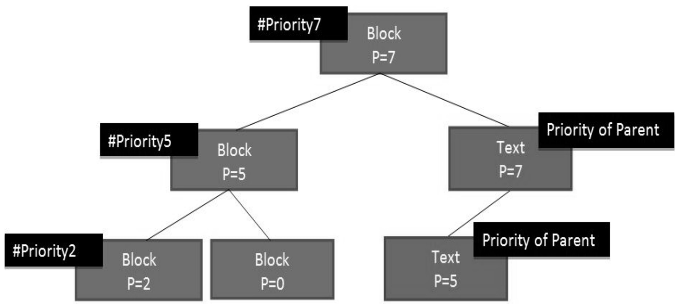


Figure 7. Priority Heap Output for the Example

5.5 Layout or Reflow

The layout of the elements start with by reading the heap's root node, with the root node having the HTML element with the highest priority. The next higher priority node will replace the node removed and the heap will rearrange accordingly.

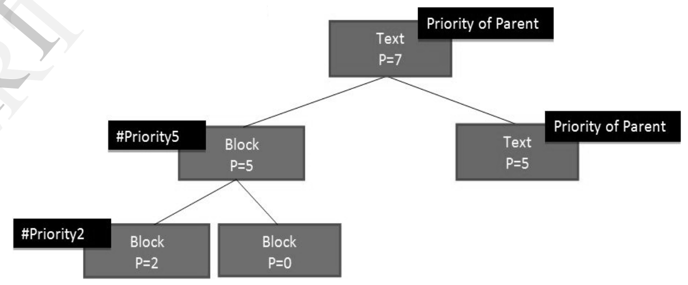


Figure 8. Priority Heap (Root Element Removed)

In the given example, the element #Priority7 will get loaded first. The node will be removed and the heap will rearrange and have the Text "Element with priority 7 nested inside priority 2" element with loaded.

The next element to load is the #Priority5 element and its text. The conflict in the heap during removal is removed by choosing the block element over the Text element.

This process continues until all the elements are loaded and the layout is designed. The order in which the layout is generated is governed by the priority of the element given. Once the layout process has started the painting can start simultaneously as and when the elements are popped from the heap.

IJERT

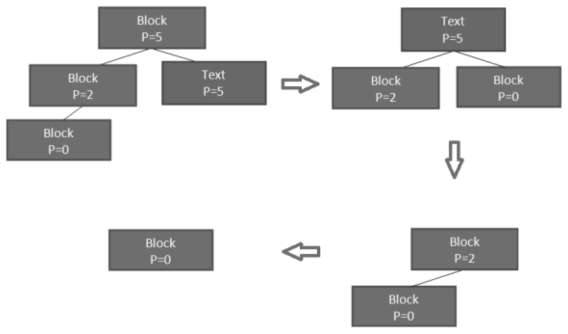


Figure 9. Priority Heap Removal of Elements

5.6 Painting

Once the layout or reflow is finalised the UI Backend of the browser will provide the necessary User Interface Elements to paint the layout defined. The painting process occurs in a separate thread and all the elements that are laid out are painted immediately.

6. Conclusion

The "Priority Based Loading of HTML elements" enables the developer to decide the information that needs to be loaded or rendered initially. This concept once implemented will load the relevant blocks before the other blocks based on the priority value set. This will help the users of such HTML pages to view content efficiently even in slow internet connection environment as the relevant information is loaded first and the irrelevant is loaded at the end.

IJERT

7. Future Enhancements

Some of the future enhancements are

- Extending this concept to other Rendering engines such as Webkit which is popularly used by Chrome, Safari and Opera, which can further be made into a HTML Standard.
- Implementing a new CSS rule "priority", similar to z-index which helps the developer to use it as a style attribute but the style should be set only for block elements.
- Using newer data structures to return data more efficiently to the painting module.

8. Acknowledgements

We would like to thank our parents for providing us with valuable education. We would also thank all the authors of our reference who provided us with in depth knowledge in the field. We would also like to thank our Principal, Head of the Department and the Staff for providing us with necessary resources to help us work on this.

9. References

[1] http://taligarsiel.com/Projects/howbrowserswork1.htm
[2] http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html
[3] http://www.w3.org/DOM/DOMTR
[4] http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109/idl-definitions.html